15th IFK
Proc. of the 15th International Fluid Power Conference <Symposium/Conference>
<DOI>

Vertical Integration of Energy Data – an Architectural Evaluation of Asset Administration Shell (AAS)

M.Sc. Fabian Geibel[1], Dr-Ing. Karin Tischler[2]

[1]*Bosch Rexroth AG, Ulm, fabian.geibel@boschrexroth.de*

[2]*Bosch Rexroth AG, Lohr am Main, karin.tischler@boschrexroth.de*

Abstract.

Understanding the energy consumption of a process is key to optimize the economic and ecological performance. Today's data integration process still comes with manual effort especially during the initial set-up. The Asset Administration Shell (AAS) is a framework of open standards for normalized, referenced and machine-readable information that enables the exchange and collaboration among different players within a broad ecosystem. This research investigates the technological aspects of the AAS and provides a detailed concept on how to implement an AAS architecture in a practical energy monitoring use case. The test results show that AAS has the potential to reduce the manual effort during the vertical integration of energy data. Nevertheless, AAS needs to be adopted by a broad range of hardware and software suppliers to scale this approach. A reflection regarding the impact of Artificial Intelligence (AI) reveals further potential.

Keywords. Asset Administration Shell (AAS), Energy Data Integration

1. MOTIVATION

The vertical integration of energy data from machines into IT applications is crucial for energy monitoring and reporting. Furthermore, it is the basis for use cases with higher economic and ecological impact like optimizing energy consumption, managing systems, quantifying the efficiency of applied measures and various other applications within industrial settings to achieve an optimized operation.

The integration of energy data is a typical task of machine operators. While the general procedure is well-known it comes with high manual effort. Each machine has different interfaces and individually specified data. There is an continuously increasing number of machine types that can be rather complex and specific. However, the energy consumption should be provided as a meaningful dataset. Currently, each machine builder or sensor

supplier provides his own solution(s) that must be integrated. The initial set-up of the communication services includes the search for the right interface description and the extraction of the configuration information. Due to supplier- and device-specific variations of common standards the configuration and mapping has to be repeated for each interface and in many applications for each datapoint again. This work is evaluating how the concept of the Asset Administration Shell (AAS) can address these challenges and supports the vertical integration of energy data.

Once the integration is established, the understanding of the data is key to generate benefit. Therefore, the data itself has to be provided with meta-data e.g., regarding origin, type and quality. Even more value comes with context information, like measurement location and process details or product information. This work also shows how AAS can support the combination of such information.

2. CONTEXT

This research focuses on the evaluation of AAS technology applied to the vertical integration of energy data in the context of established standards. A hydraulics demonstrator of an injection molding machine from Bosch Rexroth [1] serves as an example and the basis for the implementation (Figure 1). The hydraulics demonstrator includes a physical energy meter as well as a database service. The concrete components and the implementation are described in chapter 5.

Figure 1: Hydraulics demonstrator of an injection molding machine – Bosch Rexroth AG [1].

3. AS-IS PROCESS AND CHALLENGES

Performing energy management strongly relies on historic time-series data. A first major step to establish the functionality is the configuration of the communication between the participants. In the context of this research, the data is related to a physical system, a machine or device, referred to as an *Asset*. Assets can contain further Assets e.g., an energy meter for monitoring. Services that allow an interaction with the data are classified as an *Asset service* if located on a physical part of the Asset (e.g., an OPC UA [2] server running on the PLC of the machine), or as *Asset related service* running on another machine or IT system (e.g., an influxDB [3] database running on a separate device).

This research investigates the processes and challenges based on two tasks (Figure 2):

- Task 1: Store real-time data (RT) in a database (DB).
- Task 2: Read time-series data (TS) from a database (DB).

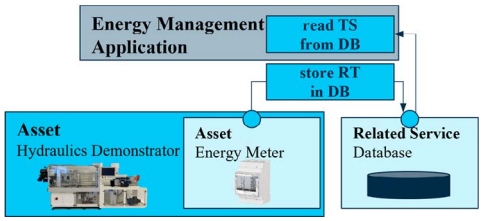


Figure 2: Overview task 1 and task 2. Task 1: Store real-time data (RT) in a database (DB). Task 2: Read time-series data (TS) from a database (DB).

After defining the term "Communication interfaces of participants" in chapter 3.1, the two main tasks and their processes are described in chapter 3.2 and chapter 3.3. Based on today's processes the main challenges are stated in chapter 3.4. A simplified example in chapter 3.5 shows how the configuration effort for the vertical integration of energy data can be assessed.

3.1. Communication interfaces of participants

Each participant (data sources and data sinks) provides a communication interface. The interface follows specific protocols and offers data in a structured way. Descriptions of the interfaces are needed for the configuration of the communication between two participants.

Elements of the interface description are e.g., for an OPC UA [2] communication: the endpoint URL, the node ID or browse path, credentials and data types. For databases e.g., influxDB providing a SQL interface [3], additional information about tables, tags and fields are required. Additionally, further meta-data like units, valid ranges, applicable actions are required for a correct processing of the data.

3.2. Task 1: Store real-time data in a database

The task "Store real-time data in a database" is realized by a service with three computing steps (Figure 3). An initial manual configuration of the service to communicate with the interfaces is necessary. The service then runs in a loop to read from the source, process the data and write the result to the sink.

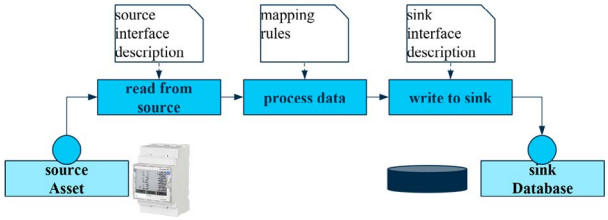


Figure 3: Task 1: Store real-time data (RT) in a database (DB) (dashed line: Initial configuration, continuous line: Running data process).

The overall process is shown in Figure 4. The user searches for the interface descriptions of the data source and the data sink from the providers or integrators of the source system (Asset) and the sink system (database). The user interprets the description and configures the runtime service for each source-sink element pair that needs to be mapped. This includes the set-up of a runtime service to communicate to the interfaces of the source and the sink (chapter 3.1).

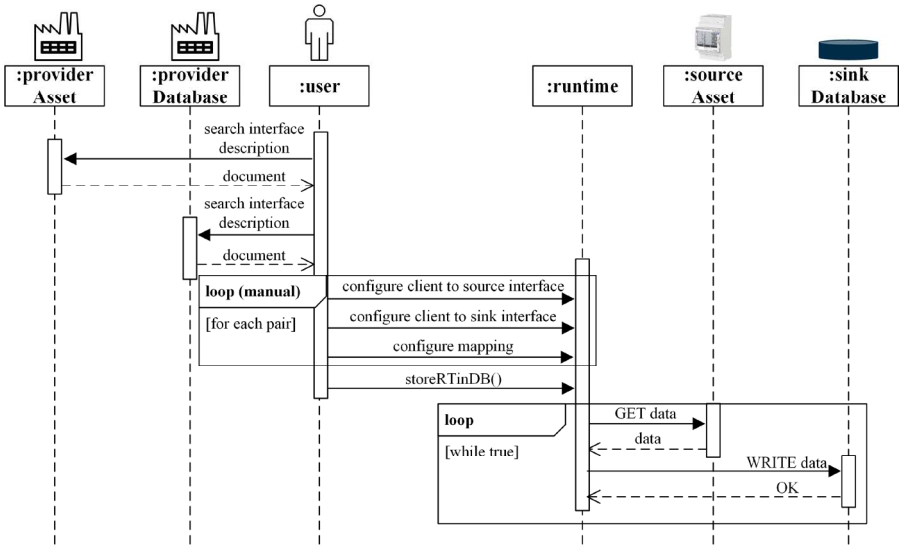


Figure 4: Process to initiate and perform task 1: Store real-time data (RT) in a database (DB).

3.3. Task 2: Read time-series data from a database

The task "Read time-series data from a database" is realized by one computing step, part of a runtime e.g., an energy management application (Figure 5). An initial configuration of the service is necessary to communicate to the interface of the source (chapter 3.1). The application defines how often the service performs the task.

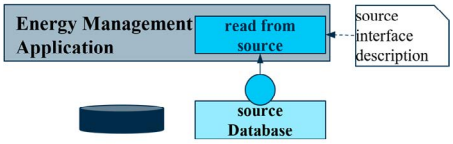


Figure 5: Task 2: Read time-series data (TS) from a database (DB) (dashed line: Initial configuration, continuous line: Running data process).

The overall process is shown in Figure 6. The user searches for the right interface description of the data source at the provider or integrator of the source system. The user interprets the

description and configures the runtime service. This includes the set-up of a client to communicate to the interface of the source (chapter 3.1).

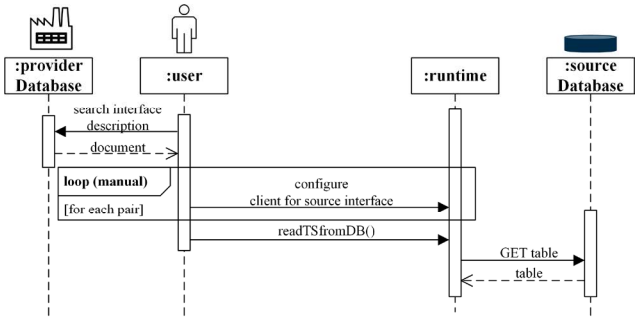


Figure 6: Process to initiate and perform task 2: Read time-series data (TS) from a database (DB).

3.4. Challenges

The manual steps during the integration process of energy data are identified facing the following four challenges:

Challenge 1: Find the right interface description for the Asset.

The interface description usually is not directly linked to the Asset. Typically, the interface description is provided as part of the product documentation. The documentation is provided via different and supplier-specific channels like printed manuals or on the manufacturer's website. In some cases, a detailed interface description can be retrieved from a webserver on the Asset itself. The identification of the information source with the correctly associated file and version might be difficult.

Challenge 2: Use the interface description to configure the service.

The interface description is often provided in a non-machine-readable way, such as tables in different formats or with a vendor specific variation of a standard. The configuration information is manually transferred into the services which is time-consuming and error-prone.

Challenge 3: Decouple and reuse code and configuration.

The configuration of the services that communicate to the interfaces and the mapping of the data between the interfaces is done directly inside the code or with a service specific configuration file. Code and vendor specific configuration information can't be reused easily. Additionally, different protocols need different connector services.

Challenge 4: Understand the data.

Meta-data and context information to understand the data coming from the interfaces, like semantic descriptions, measurement location, parameters and detailed data of the production process or the Asset itself are needed for the right interpretation of the energy data. For example, an increase of energy consumption might be related to a higher demand of the process, different production material, changes in environmental conditions, inappropriate machine configuration or machine wear. Depending on the correct understanding, it is possible to derive suitable measures or classify a change “as expected”.

3.5. Evaluation example of the implementation effort

The following example shows how an assessment of the implementation effort for the vertical integration of energy data is possible (Figure 7). The effort must be quantified individually for each use case and set-up. The example consists of several Assets (A) (e.g., machines on a shopfloor), that offer the same amount of data points (D) (e.g., actual power consumption) via the same communication interface standard (e.g., OPC UA). Every data point from every Asset needs to be integrated into several user applications (U) (e.g., for energy management). The description of the communication interface is stated in each Asset’s documentation.

The overall configuration effort (c) is based on the number of Assets (a), the number of user applications (u) and the number of data points per Asset (d). The effort is exemplarily defined as $c=a*d*u$. This example visualizes that with a rising number of Assets, data points and user applications, the configuration effort rises. In case of different communication interface standards, additional effort would be necessary.

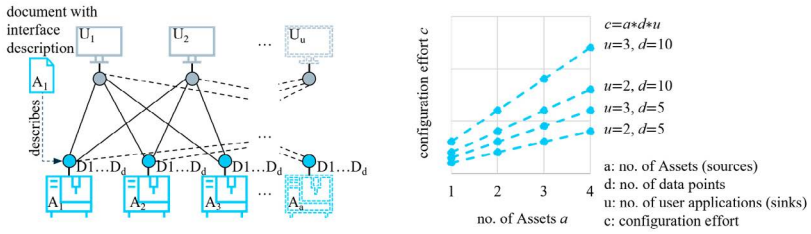


Figure 7: Integration example (left) and configuration effort (right) – providing configuration via documents.

4. AAS TECHNOLOGY TO ADDRESS THE CHALLENGES

AAS technology is characterised by open-source specifications, software development kits (SDK), simple interfaces and a community driving the interoperable data exchange standard. IEC 63278 [4] defines the *Asset Administration Shell* (AAS). The AAS consists of several open-source specifications and components relevant for this research (Figure 8):

- Asset: A virtual or physical product, machine or simple system of interest. The *Asset ID* is the globally unique identifier (ID) of an Asset.

- Asset Administration Shell [AAS]: Digital representation of an Asset. Each AAS is linked to one Asset by the Asset ID and reference to one or many *Submodels* [SM].
- Submodel [SM]: A data structure that is usually based on a *SubmodelTemplate* [SMT]. [SM] exist dynamically in a runtime as an object or statically as a .json or .xml format. Via the *AAS interface*, client applications are able to interact with a [SM] on a server. For this interaction, each [SM] has a unique *Submodel ID*. Each [SM] contains one or many *SubmodelElements* [SME].
- SubmodelElement [SME]: A structured data element. [SME] types relevant for this research are:
  - *SubmodelElementCollection* [SMC]: contains [SME]
  - *SubmodelElementList* [SML]: contains [SME] of the same type and definition
  - *Property* [PROP]: single value data element
  - *ReferenceElement* [REF]: defines a logical reference to another element
  - *RelationshipElement* [REL]: defines a relationship between two elements, contains two [REF] First and Second
- SubmodelTemplate [SMT]: A specified data structure. Each [SMT] has one unique *semantic ID*. For this research the two IDTA Submodel Templates *AssetInterfacesDescription* (AID) and *AssetInterfacesMappingConfiguration* (AIMC) are used.
  - [SM] AssetInterfacesDescription (AID) (IDTA 02017 [5]): It contains the description of the communication interfaces. Besides general information (URL, protocol type), a description of a specific property like the href or browse path, the unit, the data type or min and max values.
  - [SM] AssetInterfacesMappingConfiguration (AIMC) (IDTA 02027 [6]): It contains several [REL], that reference a property from the source [SM] AID to a property of the sink [SM] AID.
- AAS interface: A server hosting [AAS] or [SM] provides a specified REST API. For this research, the [SM] repository API with the following method is used: GET {URL}/submodels/{[SM]ID64}. Where the {URL} is the server address of the [SM] repository and the {[SM]ID64} is the base64 encoded [SM]ID.
- AAS user application: An application that communicates with the AAS or SM via the AAS interface.

[AAS] and [SM] data can be exchanged as a static file, a .json, .xml or within the .aasx package format (AAS Type 1) or, hosted on a server, via an API (AAS Type 2).

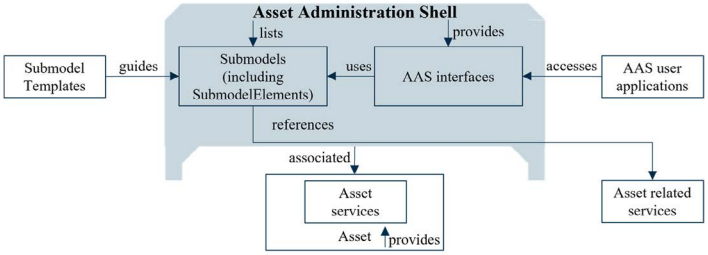


Figure 8: Relevant components of the Asset Administration Shell (based on IEC 63278 [4]).

4.1. Concept to apply AAS

The following concept is developed to apply the AAS technology in regards to the four challenges of the vertical data integration.

Challenge 1: Find the right interface description for the Asset.

The [AAS] is linked via the Asset ID to one Asset and refence the [SM] containing the information describing the Asset. An [SM] based on an [SMT] contains the semantic ID of the [SMT]. The AAS standard includes discovery, registry and repository specifications. Knowing the Asset ID and discovery URL (e.g., from the product type plate following the ID-Link standard (IEC 61406 [10])) as well as the semantic ID of the [SMT] AID, the [SM] AID can be retrieved automatically.

Challenge 2: Use the interface description to configure the service.

The [SM] AID contains all data relevant to configure a service to communicate with an Asset or Asset related interface. For each protocol, the [SM] structure follows a standard. The [SM] AID and its [SME] are accessed in a machine-readable way. Once the logic of the [SMT] is integrated in an application, the configuration process can be automated.

Challenge 3: Decouple and reuse code and configuration.

The communication interfaces can be described via [SM] AID and the mapping of the source-sink relationship via [SM] AIMC. With this concept, the configuration is provided in a neutral, standardised .json format and can be used by several applications.

Challenge 4: Understand the data.

The [AAS] represents with the [SM] different views on the Asset along its lifecycle. AAS technology is capable of structuring information that describes the Asset like provision of documentation or provision simulation models. Another aspect are relationships between Assets e.g., the connection of a valve with a pump. Here the RelationShipElement [REL] and ReferenceElement [REF] of the AAS are used. Furthermore, the context of different perspectives like production data of a part manufactured on a machine can be structured as well. In case of timeseries data, the time stamp is the main reference to synchronize different data sources.

This research is conducted as part of the funded project Fluid 4.0 [7][6]. Components to automate the communication channel between a runtime and an Asset interface are currently under development. An AAS API client [8], as well as a standard parser [9], including a handler for the [SM] AID and [AIMC], have already been published.

5. AAS TEST SET-UP, IMPLEMENTATION AND PROCESS

To evaluate the vertical integration of energy data by using an Asset Administration Shell (AAS) architecture, a test set-up was designed and implemented. The building blocks of the architecture are described in chapter 5.1. The processes to initiate and perform the two tasks "Store real-time data in a database" and "Read time-series data from a database" are described in chapter 5.2.

NOTE: for this research all software components are running on a local Windows 11 machine.

5.1. Building blocks

The AAS architecture consists of three building blocks, "Asset", "AAS Infrastructure", "Runtime" (Figure 9) that are described in more detail below.

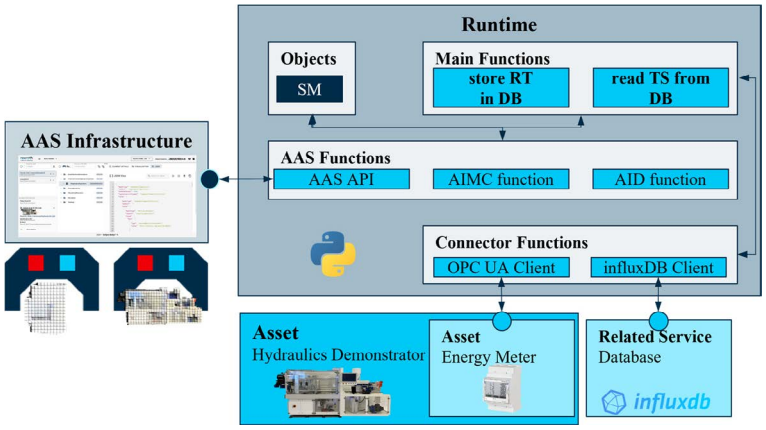


Figure 9: AAS architecture consists of three building blocks: "Asset" (bottom), "AAS Infrastructure" (left), "Runtime" (centre).

Asset

An hydraulics demonstrator of an injection molding machine from Bosch Rexroth [1] is modelled as the main Asset. The hydraulics demonstrator has an energy meter integrated, modelled as another Asset. The energy meter is providing real-time data of voltage, current and power via an OPC UA [2] interface (Figure 9, bottom). The OPC UA server is defined as an Asset service.

For the detailed testing of the AAS implementation, an additional virtual environment was established. In a local python environment, the OPC UA server of the energy meter is running with simulated data.

An InfluxDB 3 Core [3] database is storing the real-time data from the energy meter and providing the time-series data via a SQL interface. The influxDB is not part of the demonstrator and thus an Asset related service.

AAS Infrastructure

The AAS infrastructure is the Eclipse BaSyx Java implementation [11] running inside a local Docker [12] environment (Figure 9, left) The [AAS] are initialised by .aasx package files. There is one [AAS] for the hydraulics demonstrator and one for the energy meter. The AAS infrastructure offers several APIs. For this research, only the following AAS SM repository API method is used: GET {URL}/submodels/{[SM]ID64}. The return is a .json payload.

Design Decision: There is an API method to retrieve a specific [SME] inside the [SM] by a specific path: GET {URL}/submodels/{[SM]ID64}/submodelelements/{PATH}. To keep the number of requests between the runtime and the AAS Server to a minimum, the [SM] itself is retrieved as a single .json payload.

Figure 10 shows the concrete relationship between the Assets, the [AAS] and the [SM]. The [AAS] Hydraulics Demonstrator contains the [SM] AID to describe the influxDB interface. The [AAS] Energy Meter contains the [SM] AID to describe the OPC UA interface. Furthermore, the [AAS] Hydraulic Demonstrator contains the [SM] AIMC that refers to [SMC] inside the two [SM] AID.

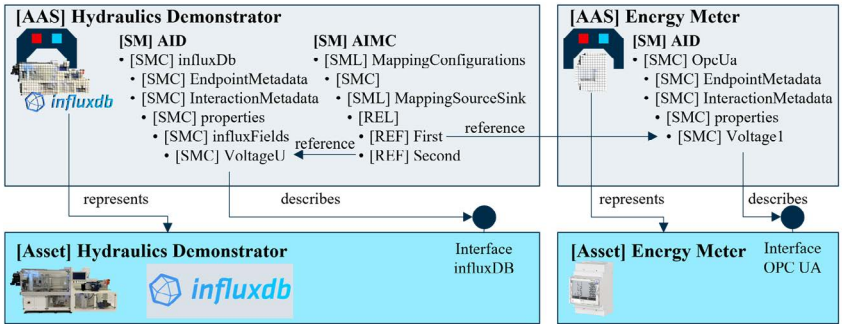


Figure 10: Relationship between Assets, AAS and Submodels.

Runtime

The runtime is implemented with Python. It contains three function modules “Main Functions”, “AAS Functions”, “Connector Functions” and the [SM] object storage (Figure 9, centre).

The functions and the object storage are described in more detail.

Main functions that fulfil the main tasks:

- Store real-time data in a database: this function starts the process to read from the source (OPC UA server) and writes to the sink (influxDB).
- Read time-series data from a database: this function starts the process to read from the source (influxDB).

AAS functions to interact with [AAS] and [SM] repository API and [SM] specific functions.

- AAS API: client to interact with the [AAS] and [SM] repository API. Relevant for this research: GET {URL}/submodels/{[SM]ID64}. The status code and the .json payload is returned.
- AIMC function: process the [SM] AIMC to set up two mapping vectors with information from the source and sink [SM] AID.
- AID function: process the [SM] AID to configure the clients.

Connector functions to set-up communication to the interfaces:

- OPC UA client: service to read data from an OPC UA server.
- influxDB client: service to write to and read from an influxDB.

SM object storage to store [SM] as a python object:

- The .json payload, returned from the [SM] repository server, is transformed via the AAS Core Works Python SDK [13] to a python object. The runtime interacts with this object instead of performing several API calls to the [SM] repository server.

5.2. Processes

NOTE: it is assumed that the suppliers or integrators, that set-up the interface and data structure offered by these interfaces, provide the interface descriptions as [SM] AID, as well as the URL to the corresponding repository and the ID of the [SM].

Task 1: Store real-time data in a database

The overall process to “store real-time data in a database” is described in Figure 11. The user effort for the configuration is limited to define the [SM] AIMC. That incudes to add for each mapping a [REL] element. The [REF] First points to the [SMC] within the [SMC] influxFields containing the description of the source element (e.g., [SMC] VoltageU in Figure 10). The [REF] Second points to the [SMC] within the [SMC] properties containing the description of the sink element (e.g., [SMC] Voltage 1 in Figure 10). To initiate the computing process, the user calls the runtime main function “Store Real-Time Data in Database” with the ID of the [SM] AIMC, the [SML] index of the [SMC] containing the concrete mapping and the URL of the corresponding repository.

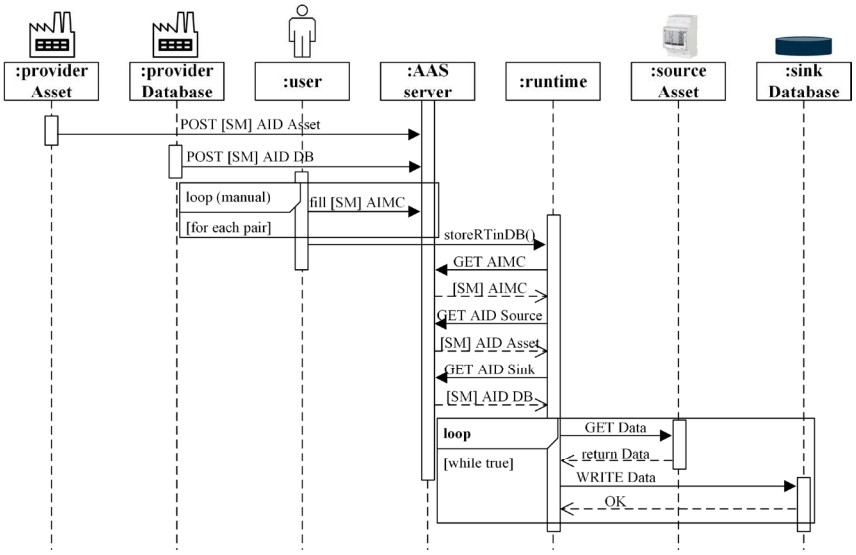


Figure 11: Process to initiate and perform task 1 with AAS: Store real-time data (RT) in a database (DB).

Task 2: Read time-series data from a database

The overall process to "Read time-series data from a database" is described in Figure 12. To initiate the computing process, the user calls the runtime main function "Read Time-Series Data from Database" with the ID of the [SM] AID, the name of the [SMC] for one concrete interface of the Asset (e.g., "OpcUa" in Figure 10) and the URL of the corresponding repository.

NOTE: it is assumed that all properties in the [SMC] for one concrete interface need to be integrated.

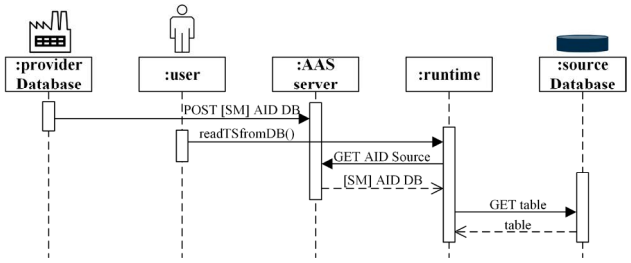


Figure 12: Process to initiate and perform task 2 with AAS: Read time-series data (TS) from a database (DB).

6. EVALUATION

The four challenges (chapter 3) of the vertical integration of energy data are evaluated based on the implemented test set-up of the Asset Administration Shell (AAS) architecture (chapter 5) in chapter 6.1.. It is important to note that the AAS is used in this evaluation to provide the static configuration information. The AAS is not used to handle the dynamic data directly. The data handling is done by runtime clients designed to communicate with the specific interface standards (e.g., OPC UA) directly. In chapter 6.2, the example from chapter 3.5 used again to show the potential of the AAS approach.

6.1. Evaluation of challenges

The evaluation focusses on the manual steps that a user needs to perform for the configuration and the interpretation of a communication interface. For each challenge, the limitations of the implementation are shown as well. The progress in the field of Artificial Intelligence (AI) will lead to new solutions in IoT applications. These opportunities in combination with the developed AAS approach are discussed as further potential.

Challenge 1: Find the right interface description for the Asset.

The process is reduced from searching the interface description in different supplier documents and websites to the search for the [SM] ID and URL of the Submodel repository. Furthermore, when [AAS] discovery and registry mechanisms are implemented, the Asset ID (following the ID-link standard) is sufficient to find and choose the right [SM]. The Asset ID can be encoded e.g., in the QR-code on the Asset.

Further potentials:

To scale this approach, it is necessary that a broad range of vendors create and provide AAS for their Assets. If vendors offer an AAS API, customer applications could retrieve the AAS automatically. As this is not yet common standard the scaling is an open question.

An AI approach could help transforming the data from a vendor specific standard, or a document to the AAS specification. This would reduce the initial effort for each participant. Otherwise, AI might search for the interface descriptions on websites and in documents – even without AAS, but with a risk of misinterpretation and recurring controlling efforts by a user in each case.

Challenge 2: Use the interface description to configure the service.

The process is reduced from taking over the configuration information manually into the application to a fully automatic configuration. This is due to the standardised machine-readable and vendor-independent .json format for the [SM] AID.

Further potentials:

Consuming applications could provide a ready-to-use service that configures the communication to the Asset services automatically based on the [SM] AID and [SM] AIMC.

The AAS specifications can be developed further to describe more different kinds of interfaces.

An AI approach would be suitable to take over the mapping process providing the [SM] AIMC, based on the source and sink [SM] AID.

Challenge 3: Decouple and reuse code and configuration.

Redundant integration effort is reduced significantly as a typical use case set-up and software services can be reused easily. Once the interface descriptions and the mapping are provided as [SM] AID and [SM] AIMC and the software can interpret the standard structure, the complete content can be transferred to different Assets. Changing or upgrading a single Asset within an IT infrastructure is possible with little implementation effort.

Further potentials:

The development of further open-source components for AAS is ongoing [8], [9], [11], [13], [14]. This enables different users and applications to adopt the AAS technology faster in their use cases.

Challenge 4: Understand the data.

The AAS provides easier access to structured information beyond the basic sensor measurement values. Information like units, valid ranges, accuracy, semantic IDs and additional descriptions can be used by different applications and use cases. Client applications are able to retrieve additional information when needed.

The AAS forms a framework in which the information of different domains is structured semantically and handled together. Therefore, the enrichment of a single use case with other perspectives can be conducted easily. The access via an API is also key for the interaction with AI-based applications. To gain deeper insights, information about the Assets, the manufacturing process or the production plan shall be combined.

Further potentials:

Today, [SMT] are able to describe location and relation information (topology) as well as process data and information about the Asset that is currently produced. By this, a big data ecosystem is evolving over time that supports information sharing. A concept how Assets, Submodels, AAS infrastructure and applications shall be structured and interact efficiently is needed, as well as the associated data ownership and privacy. Deriving [SMT] fitting to specific use cases would lead to data redundancy (also to other established standards like OPC UA companion specifications). To keep the framework efficient and comply with the single source of truth principle is still a challenge for cross-vendor interoperability.

6.2. Evaluation example of the implementation effort – AAS compared to document

To demonstrate the different integration effort when the communication interface description is provided as the AAS [SM] AID instead of a document for each Asset, the

integration example from chapter 3.5 is used again. When implementing the logic to process the [SM] AID, not every data point (D) needs to be configured manually anymore. For this reason, the number of data points (d) has no impact on the configuration effort (c). However, there is an initial effort to integrate the logic behind the [SM] AID. Therefore, to provide a meaningful comparison between [SM] AID and the document-based configuration effort, the initial configuration effort for the [SM] AID implementation has a positive offset of c_0.

The overall configuration effort (c) changes from $c=a*d*u$ (document-based interface description), to $c=a*u+c_0$ (AAS [SM] AID based interface description). This example visualizes that the integration effort rises with a higher number of Assets (a) and user applications (u), but more slowly by using the AAS [SM] AID in comparison to use a document-based interface description. Nevertheless, the break-even in the vertical integration of energy data is depending on the use case specific effort for the implementation. When software vendors start to implement the logic behind the AAS [SM] AID, the offset c_0 is assumed to decrease and the break-even is reached earlier.

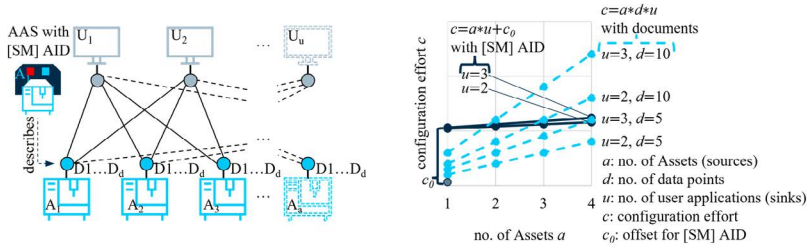


Figure 13: Integration example (left) and configuration effort (right) – providing configuration via [SM] AID (continuous lines) and via documents (dashed lines).

7. SUMMARY

This research addresses the significant manual effort involved in the vertical integration of energy data from diverse industrial Assets, a crucial bottleneck for optimizing economic and ecological performance. We investigate the Asset Administration Shell (AAS) as a framework to standardize and streamline this process. Our methodology involves developing and testing a practical AAS architecture for energy monitoring, focusing on two key tasks: “Store real-time data in a database” and “Read time-series data from a database”. Key findings demonstrate that AAS significantly reduces manual configuration and interpretation efforts by providing machine-readable, vendor-independent interface descriptions and enabling the reuse of code and configuration across different Assets. The AAS framework also facilitates a better understanding of data through structured metadata and contextual information. The main conclusion is that while AAS holds substantial potential to automate energy data integration, its widespread adoption by hardware and software suppliers is essential for scalability. The study also highlights promising future directions, particularly integrating Artificial Intelligence to further enhance configuration and data interpretation within the AAS ecosystem.

8. ACKNOWLEDGEMENT

This work has been funded by the European Union (EU) and supported by the German Federal Ministry for Economic Affairs and Energy (BMWE) in the project Fluid4.0 – Revolutionizing Fluid Engineering: Implementation of digitalization for fluid power 4.0 in the cross-industry and cross-manufacturer data space using asset administration shells, submodels and demonstrators (project number 13IK039L). The views and opinions expressed are solely those of the authors and do not necessarily reflect the views of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

9. REFERENCES

[1] Bosch Rexroth AG, 'Virtual World – injection molding machine', accessed Dec. 2025 on home page

[2] OPC Foundation, 'OPC Unified Architecture - Part 1: Overview and Concepts, OPC 10000-1', Version 1.05.04, Nov. 2024

[3] InfluxData Inc, 'influxDB 3 Core – Documentation', accessed Sep. 2025 on vendor home page

[4] International Electrotechnical Commission (IEC), 'IEC 63278-1 – Asset Administration Shell for industrial applications – Part 1: Asset Administration Shell structure', edition 1.0, Dec. 2023

[5] Industrial Digital Twin Association (IDTA), 'IDTA 02017 – Asset Interfaces Description', Version 1-0, Jan. 2024

[6] Industrial Digital Twin Association (IDTA), 'IDTA 02027 – Asset Interfaces Mapping Configuration', Version 1-0, Jun. 2024

[7] Fluid 4.0, 'Project – overview, accessed Dec. 2025 on home page

[8] D. Klein – :em engineering methods AG, 'AAS HTTP Client – PyPi package', accessed Dec. 2025 on PyPi

[9] D. Klein , 'AAS Standard Parser – PyPi package', accessed Dec. 2025 on PyPi

[10] Deutsches Institut für Normung (DIN), 'IEC 61406-1:2022 Identification Link – Part 1: General requirements', Dec. 2023

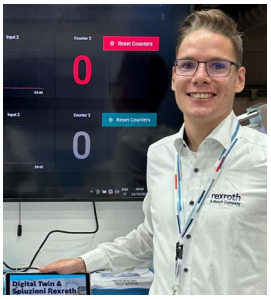
[11] Eclipse BaSyx, 'BaSyx WiKi – Quick Start Guide', accessed Sep. 2025 on home page

[12] Docker Inc., 'Docker Desktop', accessed Oct. 2025 on home page

[13] aas-core3.0-python AUTHORS, 'aas-core-works - aas-core3.0-python', accessed Sep. 2025 on GitHub

[14] Fluid 4.0, 'GitHub Project – open-source components', accessed Oct. 2025 on GitHub

10. BIOGRAPHIES



Fabian Geibel received his master's degree in mechanical engineering from RWTH Aachen University in 2019. After working as a research assistant at the Institute of Power Plant Technology, Steam and Gas Turbines in Aachen, he started a graduate program at the Bosch Rexroth AG. His profession is in the field of standardised data exchange and the application of digital twin technologies in industrial use cases.

Dr. Karin Tischler received her Diplom in Mechanical Engineering from the Universität Karlsruhe (TH) in 2002 followed by a Doktorin der Ingenieurwissenschaften from KIT after serving as a Research Assistant in the Department for Measurement and Control Systems. Her foundational work was focussed on information fusion for autonomous driving. Since 2008, she has held various positions at Bosch Rexroth AG, including Project and Group Manager for Innovation and System Application in Mobile Controls. Responsible leader in the central development of systems & solutions she is a recognized expert in system optimization. As Innovation Cluster Owner she drives new business initiatives solving challenges regarding energy transition and digitalization.