Integrating IoT Infrastructures in Industrie 4.0 Scenarios with the Asset Administration Shell

Sven Erik Jeroschewski, Johannes Kristan, Milena Jäntgen, and Max Grzanna

Bosch.IO GmbH, Ullsteinstr. 128, 12109 Berlin, Germany
{firstname.lastname}@bosch.io

Abstract. The Asset Administration Shell (AAS) specifies digital twins to enable unified access to all data and services available for a physical asset to cope with heterogeneous and fragmented data sources. The setup of an AAS infrastructure requires the integration of all relevant devices and their data. As the devices often already communicate with an IoT backend, we present three approaches to integrate an IoT backend with an AAS infrastructure, share insights into an implementation project, and briefly discuss them.

Keywords: IoT · Asset Administration Shell · I40 · digital twin.

1 Introduction

The rising heterogeneity and complexity of their environments make it hard for manufacturers to adapt to changes, integrate new components, and prevent fast and sound decision-making, as relevant data may be theoretically available but practically not accessible where it is needed [8].

The Asset Administration Shell (AAS) [5, 6] is a building block in achieving interoperability in Industrie 4.0 scenarios by specifying interaction models, formats, and abstractions for the handling and access of information as digital twins. Various domains already adopt the AAS for scenarios like Digital Calibration Certificates (DCC) [4], or accessing semantically and syntactically aligned data sets for training, and re-using higher quality AI models [8].

To benefit from the AAS, manufacturers link existing systems, services, and devices with an AAS infrastructure, which may result in high configuration efforts for each device and possibly long down times. Many connected devices already communicate with an Internet of Things (IoT) backend [1, 7], which manages the device state, collects data, and routes messages. Often, it is thus easier to connect the AAS to an IoT backend and leave each device unchanged.

2 Integration Approaches

The AAS defines flows for data retrieval (Fig. 1) [6], which the presented integration approaches need to fit to. The flow starts by requesting an AAS ID from the AAS discovery interface based on a (local) specific asset ID or a global asset

ID. With the AAS ID, the application retrieves an endpoint for the AAS through the AAS Registry interface. The application then requests the Submodel (SM) ID from that AAS endpoint and uses this SM ID to get the SM endpoint from the SM Registry. From that SM endpoint, the user can request the SM Element (SME), which contains the required value.

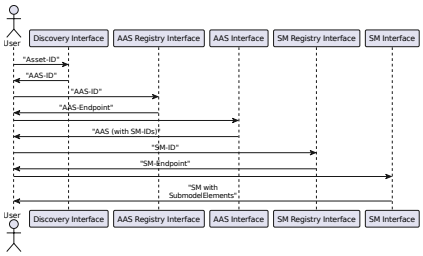

Fig. 1. Sequence of data flow through AAS infrastructure

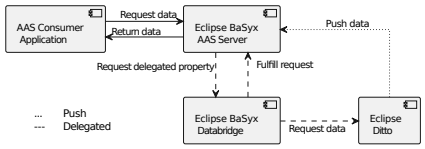

Fig. 2. Implemented integration approaches

This generic flow shows that an IoT backend integration essentially boils down to making device data available via SMs and their SMEs as an SM Interface Endpoint. We identify three approaches for this: The IoT backend may *push* latest updates to an AAS SM server or the AAS SM server *pulls* the current state from the IoT backend either via a *wrapper* or via a *bridge*.

Push: Whenever the IoT backend receives an update from a device, a backend component transforms and pushes the data in the AAS format [5] to an SM server. This approach allows for re-using a generic implementation of the SM Server with the drawback of duplicate data storage in the IoT backend and the SM server, leading to potential synchronization and data consistency issues.

Pull via Bridge: Some AAS SM servers support delegating requests for specific SME values to other endpoints like a data bridge, which then retrieves the actual data from the IoT backend and applies transformation logic. With this approach, one can use a generic AAS server implementation and enable mixed scenarios where only a few requests get delegated to one or multiple IoT backends while the SM server stores all other SMEs. However, it requires the AAS server component to provide such functionality.

Pull via Wrapper: It is also possible to add a custom wrapper that implements the SM interface for the client and fetches the required data from the IoT backend. This approach does not require data duplication but may impose implementation efforts concerning identifier mapping and coupling between the wrapper and the IoT backend.

As part of the project GEMIMEG-II [4], which works on DCCs and better data orchestration, we implemented the approaches *push* (dotted) and *pull* via a bridge (dashed) as depicted in Fig. 2. We used Eclipse Ditto [3] as an IoT backend and Eclipse BaSyx as AAS infrastructure [2].

Eclipse Ditto is an IoT backend built of micro-services, which evolves around the concept of `Things` representing the state of a device. Each `Thing` has `Properties` grouped as `Features`, which may change over time (e.g. sensor values). One can express constant values, such as identifiers, as `Attributes`. Grouping of `Things` is possible by assigning them to a `Namespace`. Ditto comprises a Connectivity API for the integration with other systems, which allows to provide JavaScript code, which gets executed on events (e.g. changing a `Thing`).

Eclipse BaSyx is an open-source framework to realize an Industrie 4.0 middleware [2] based on the AAS Spec. [5, 6].

For *push*, we configured a Ditto instance through the connectivity API to forward changes of a device and its corresponding `Thing` to a BaSyx SM Server. This results in duplicated data storage in Ditto and the BaSyx SM Server.

For *pull*, the BaSyx SM server supports delegation and calls the endpoint of a bridge component for each request for a corresponding SME. Since Ditto has the option to return the value without additional payload, the main task of the bridge is to perform the authorization flow of Ditto.

The *pull* via wrapper was not realized for AAS as it would require high implementation efforts, as demonstrated by the integration of the Web of Things (WoT) by the Eclipse Ditto Project [3].

The mapping between concepts of Ditto and AAS is depicted in Tab. 1.

Table 1. Concept mapping from Eclipse Ditto to the AAS

Eclipse Ditto	**Asset Administration Shell**
Namespace	Asset Administration Shell
Thing	-
Features	Submodel
Property	Submodel Element
Attribute	Submodel Element

3 Discussion

Based on our observations, the pull approach with a wrapper is a good trade-off for scenarios with a high and medium frequency of sensing and actuation updates. But the development and operation of new software artifacts lead to higher engineering costs and operation efforts in comparison to the other approaches. The push approach is a good solution for scenarios with many data reads and few data updates but it lacks a good way of pushing actuation information to the IoT backend and introduces risks regarding data inconsistency. Compared to the other presented approaches, the pull approach with data bridge seems to have lower engineering cost and is easier to operate allowing to get started a bit faster, but it is not so well-suited when the frequency of data access rises.

As we draw our conclusions about the three approaches solely from our observations in one project, it is worthwhile to extend the analysis and run some even quantifiable evaluations based on further projects or in controlled environments. We also have not yet looked further into executing AAS operations. Once BaSyx supports authenticating during requests for delegated data, we may also try to retrieve the raw data from Eclipse Ditto without using a data bridge.

4 Summary

We presented the architectural approaches, push, pull with wrapper, and pull with data bridge, to integrate existing IoT backends with the AAS. Based on the experiences gained in the GEMIMEG-II project, we discussed the approaches without identifying a preferred option since each alternative has different advantages and drawbacks for the sensing and actuating frequency or the engineering and operation cost.

Acknowledgements The research has received funding from the the Federal Ministry for Economic Affairs and Climate Action of Germany under the funding code 01MT20001J. The responsibility for the content of this publication lies with the author(s).

References

1. Banijamali, A., Heisig, P., Kristan, J., Kuvaja, P., Oivo, M.: Software architecture design of cloud platforms in automotive domain: An online survey. In: 2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA). pp. 168–175 (2019)
2. BaSyx, E.: Eclipse basyx - industry 4.0 operating system. `https://www.eclipse.org/basyx/` (2023)
3. Ditto, E.: Eclipse ditto. `https://www.eclipse.org/ditto` (2023)
4. Hackel, S., Schönhals, S., Doering, L., Engel, T., Baumfalk, R.: The digital calibration certificate (dcc) for an end-to-end digital quality infrastructure for industry 4.0. Sci **5**(1) (2023). https://doi.org/10.3390/sci5010011, `https://www.mdpi.com/2413-4155/5/1/11`
5. IDTA: Spec. of the Asset Administration Shell - Part 1: Metamodel (April 2023), `https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/04/IDTA-01001-3-0_SpecificationAssetAdministrationShell_Part1_Metamodel.pdf`
6. IDTA: Spec. of the Asset Administration Shell - Part 2: Application Programming Interface (April 2023), `https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/04/IDTA-01002-3-0_SpecificationAssetAdministrationShell_Part2_API.pdf`
7. Kristan, J., Azzoni, P., Römer, L., Jeroschewski, S.E., Londero, E.: Evolving the ecosystem: Eclipse arrowhead integrates eclipse iot. In: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium. pp. 1–6 (2022)
8. Rauh, L., Reichardt, M., Schotten, H.D.: Ai asset management: a case study with the asset administration shell (aas). In: 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–8 (2022)