applied sciences

MDPI

Article

Integration Approaches for Digital Twins in Dataspaces

Carlos Schmidt [1], Friedrich Volz [1,*], Ljiljana Stojanovic [1] and Holger Kett [2]

[1] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB, 76131 Karlsruhe, Germany; carlos.schmidt@iosb.fraunhofer.de (C.S.); ljiljana.stojanovic@iosb.fraunhofer.de (L.S.)
[2] Fraunhofer Institute for Industrial Engineering IAO, 70569 Stuttgart, Germany; holger.kett@iao.fraunhofer.de
* Correspondence: friedrich.volz@iosb.fraunhofer.de

Abstract

Digital twins (DTs) based on the Asset Administration Shell (AAS) enable interoperable representations of industrial assets, while dataspaces like those defined by the International Data Spaces (IDS) facilitate sovereign, policy-governed data exchange across organizations. However, integrating AAS-based DTs into dataspaces remains challenging due to manual processes, synchronization issues, and varying implementation strategies. This paper addresses these gaps by first presenting a structured analysis of existing integration approaches and evaluating the user requirements to be met. We then introduce a flexible reference implementation extending the Eclipse Dataspace Connector (EDC), which automates AAS registration into dataspaces. Finally, we conduct a qualitative comparison with state-of-the-art prototypes from initiatives such as Catena-X and Factory-X and demonstrate the application of our integration approach in the MODAI project for sensor-driven manufacturing data sharing. Results indicate that the integration approach reduced manual effort and provides synchronization between digital twins and dataspace.

Keywords: dataspace; digital twin; Asset Administration Shell (AAS); Industry 4.0

Academic Editors: Jozef Svetlík, Rudolf Jánoš and Ján Semjon

Received: 2 October 2025
Revised: 23 October 2025
Accepted: 26 October 2025
Published: 30 October 2025

Citation: Schmidt, C.; Volz, F.; Stojanovic, L.; Kett, H. Integration Approaches for Digital Twins in Dataspaces. *Appl. Sci.* **2025**, *15*, 11623. https://doi.org/10.3390/app152111623

1. Introduction

Digital twins (DTs) are digital representations of physical assets, systems, or processes that support real-time monitoring, simulation, and analysis throughout their life cycle [1,2]. Acting as a bridge between the physical and digital worlds, DTs enable data-driven decision-making across a wide range of domains, including manufacturing. As their adoption increases, the need for standardized models becomes critical to ensure interoperability, scalability, and reusability across heterogeneous systems and stakeholders.

In the industrial domain, the Asset Administration Shell (AAS) provides a standardized framework for DTs, offering a common, machine-interpretable information model and well-defined interfaces. The Industrial Digital Twin Association (IDTA) is now responsible for the ongoing development and standardization of the AAS [3], enabling machines and software systems to "speak the same language" regardless of manufacturer, technology, or life cycle phase.

While the AAS specification ensures syntactic and semantic interoperability within a production site, it previously lacked guidance on how data contained in an AAS can be securely shared across company borders. This gap is now addressed in the recently published Part 4 of the AAS [4], which introduces mechanisms for secure data exchange through attribute-based access control (ABAC). Additionally, it introduces the concept of dataspaces as the context in which usage contracts between companies can be applied, thereby fostering trusted and controlled cross-company collaboration [4].

This vision of cross-organizational data sharing aligns closely with the work of the International Data Spaces Association (IDSA), which developed the International Data Spaces Reference Architecture Model (IDS-RAM) [5] to facilitate sovereign, policy-governed data exchange across organizational boundaries. The IDS architecture defines standardized building blocks (e.g., connectors, brokers, clearing houses) along with open-source reference implementations that support usage contracts, policy enforcement, and trust among participants.

Taken together, AAS and IDS offer complementary capabilities. The AAS enables plug-and-play interoperability on the shop floor—primarily within organizational boundaries. IDS-compliant dataspaces provide the governance, trust mechanisms, and contractual semantics required for data exchange between organizations. Recognizing this complementarity, several initiatives (e.g., Catena-X [6], Factory-X [7], CircularTwAIn [8]) have explored different patterns for integrating AAS with dataspaces. Several publications describe their integration of AAS into dataspaces [9–18]. However, these integration approaches vary significantly in terms of where the integration occurs, which extensions are introduced, and how much complexity is imposed on end users.

In [19], the authors identified nine articles regarding manufacturing networks with AAS and dataspaces based on IDS components. A common integration pattern involves mapping AAS submodels and metadata to IDS assets within an IDS connector. In this context, an IDS asset represents a data resource—such as an AAS submodel—along with its associated metadata and usage policies. The connector then publishes these assets as data offerings in a dataspace catalog, making them discoverable by other participants under specified usage policies.

In [20], the authors describe some limitations of this mapping approach. Namely, the unknown granularity for the AAS and the incompatibility between different dataspace implementations. Regarding the granularity, for each AAS instance, the specific parts intended for sharing (submodels, submodel-elements) must be registered in the catalog, with an associated usage policy that defines how the data can be accessed and under what conditions. Currently, this process is largely manual: data providers create IDS assets corresponding to selected AAS submodels or elements and explicitly define the associated metadata and contractual terms. In [21], the authors describe the manual creation of "resources in the supplier connector" for a manufacturing-as-a-service system. However, in their case, a resource is not created for a submodel or element but for a list of every AAS in the system, since their use case focuses on execution of manufacturing orders. The latter limitation of incompatibility has been mainly solved by the adoption of the Dataspace Protocol in IDS-based components [5].

Despite the high effort of manually describing the AAS and their contracts in the dataspace, this limitation is only occasionally addressed in publications (e.g., [12]). Especially in cases of high granularity on the element level, manual creation is a tedious workload. For example, the authors in [22] implemented an AAS Manager that creates the resources for each submodel to be shared.

Efforts to automate this process have led to several implementation strategies. One approach involves reading submodels from existing AAS services and dynamically generating IDS catalog entries. Another embeds AAS services directly within the connector, allowing for tighter integration and real-time synchronization. However, these strategies introduce trade-offs. In cases where companies already operate AAS services as part of their production infrastructure, automation may require re-hosting, redesign, or modification of existing systems—introducing additional complexity and increasing the overall burden of adoption.

In light of these challenges, this paper makes the following contributions: Firstly, a structured analysis of existing approaches for integrating AAS-based DTs into dataspaces, including an evaluation of how authorization, usage contracts, automation, etc., are addressed. Secondly, a flexible reference implementation built on the IDS open-source stack and the Eclipse Dataspace Connector (EDC) [23] is presented, which automates the integration of AAS instances into a dataspace with reduced manual effort. Finally, a qualitative comparison of our implementation against state-of-the-art prototypes (e.g., from Catena-X, Factory-X) is conducted.

The rest of this paper is organized as follows: Section 2 provides background on dataspaces and the AAS landscape for DTs. Section 3 describes the user requirements and architectural design for automated data integration into dataspaces. Section 4 presents our approach and implementation details. Section 5 compares our approach with existing solutions using the proposed comparison framework. Section 6 summarizes the findings and outlines future research directions, especially in the direction of "twin-native" connectors specifically designed for sharing DTs in dataspaces.

2. Background

2.1. Dataspaces

Europe's data strategy has shifted from isolated data markets to federated dataspaces, i.e., interoperable infrastructures in which participants keep full control over the data they share. The Data Space Support Centre [24] defines the Data Space Blueprint, a technology-agnostic reference architecture. This blueprint is intended to give every dataspace project a common vocabulary, a canonical set of building blocks, and a minimal interoperability profile, while leaving enough freedom to choose concrete products or open-source implementations. In our work, we adopt the blueprint and implement most of its components with the Eclipse Dataspace Components [25].

The blueprint distinguishes two categories of dataspace building blocks:

- Technical building blocks: These cover all core capabilities—from secure data transfer in a "minimal dataspace" to advanced services such as marketplaces. Most are domain-agnostic, but the block dealing with data models and formats must be tailored to each sector (e.g., Industry 4.0) [26].
- Business and organizational building blocks: These capture the organizational, business, and operational agreements needed to realize and run a decentralized, multi-stakeholder architecture.

Within this ecosystem, dataspace participants are typically divided into data providers, who offer controlled access to data governed by usage policies, and data consumers, who access this data after accepting or negotiating usage terms.

A key technical component supporting these interactions is the EDC, part of the Eclipse Dataspace Components. The EDC is an open-source, Apache-2.0 licensed reference connector implementation that enables secure and controlled data sharing between companies, serving as the primary interface for data exchange within dataspaces.

The EDC consists of two logical planes:

- Control plane: Asset and policy registration, catalog API, contract negotiation, identity and trust, and management API.
- Data plane: Actual data transfer via protocols (usually HTTP pull/push).

The control plane is where data or a reference to the data is registered as assets in the internal AssetStore. Contracts are attached to the assets to describe who may access the data and which usage policies need to be accepted by the data consumers. Assets and contracts are together offered as datasets in the provider's catalog. Data consumers also

have the option to make counteroffers based on the provider's contract. Figure 1 shows the interaction between data providers and data consumers with the different planes of the EDC.

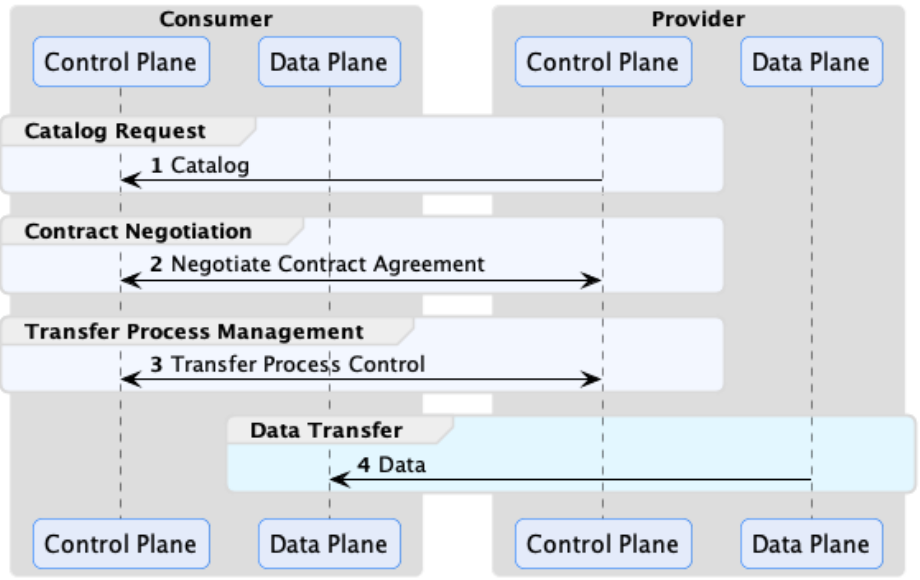

Figure 1. Control and data plane between consumer and provider with arrows indicating data flow, reproduced with permission from [23].

The consumer requests the catalog from the provider and selects an asset to negotiate on. In many cases the contract attached to the asset is accepted as-is by the consumer, but in every case a contract offer is sent. The transfer process is then initiated either by provider push or consumer pull so that the data transfer between the data planes starts.

2.2. Asset Administration Shell as Digital Twin

The IDTA assumed responsibility for further developing and maintaining the AAS specification, keeping it up to date with industry needs and international standards. It describes the AAS as a standardized DT of a physical asset that enables different manufacturing applications to work together. It clearly points to the asset it represents, contains digital submodels that represent many aspects of the asset, and lists the technical services offered by the asset [3].

Ideally, an AAS covers most information and functions related to an asset—its features, parameters, sensor data, and operating abilities. All these data are arranged in submodels, which describe one aspect of the asset, for example, the technical data or the carbon footprint. So-called submodel templates predefine the submodel and are managed by the IDTA to cover common aspects of the asset in a standardized way. Figure 2 shows the structure of the AAS.

Each submodel consists of submodel elements, which can be properties, operations, files, etc., of the asset. They can be grouped in SubmodelElementCollections and SubmodelElementLists. Different communication protocols can connect the AAS to the real-world asset to fill the properties and execute the operations.

The AAS standard is split into several parts. Part 1 explains the meta-model and how data is serialized, while Part 2 defines the API for communication with AAS instances during operation. Further parts are still being worked on. Part 4 was recently published and deals with security in the form of ABAC. It also mentions a dataspace with AAS where the AAS can "support Identity Providers of dataspaces or any other multilateral company use cases" [4].

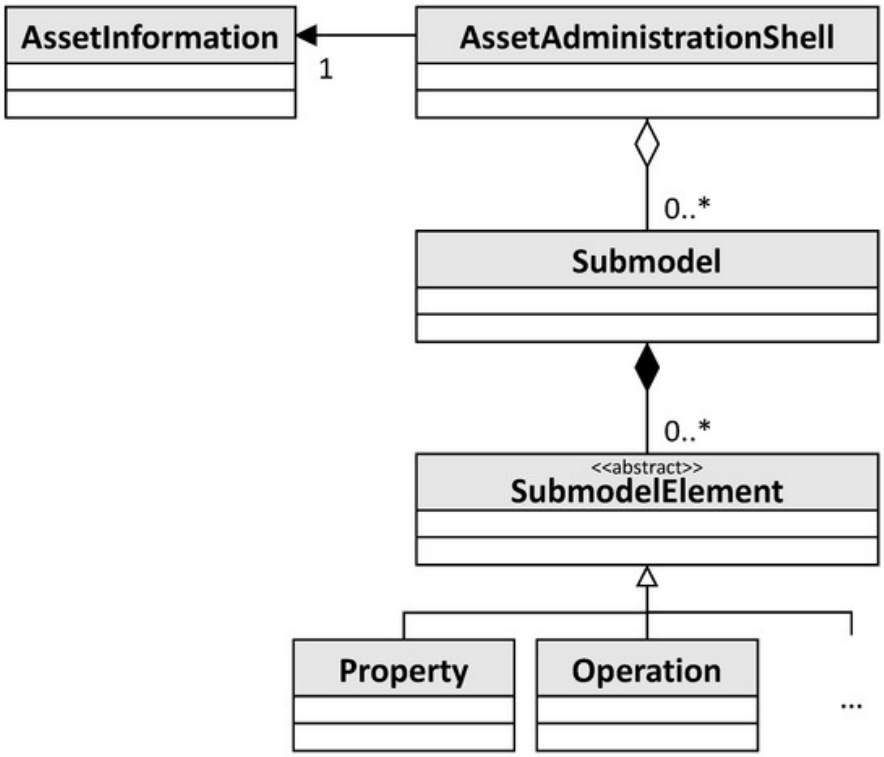

Figure 2. UML class diagram of simplified AAS meta-model, reproduced with permission from [27]. 0,*,1 are standard UML notation for min, max, multiplicity.

Part 2 of the AAS specifies different AAS APIs, such as the submodel repository API to create and delete submodels or the submodel registry API to create and access descriptors of submodels. In this context, a submodel registry is a software component that contains a list of descriptors that describe the submodels and where to find them. Analogously, there is an AAS registry that describes the AAS. So-called AAS services or AAS servers usually combine the AAS repository API with the AAS service API to manage and grant access to the AAS.

2.3. Why to Use Dataspaces with Digital Twins

While data sharing can be conducted solely with DTs, e.g., AAS communication with AAS API, most DT specifications do not deal with data-sovereign sharing to the same extent as the IDS. For example, the topic of data usage contracts is not addressed by the AAS. The AAS defines ABAC, an attribute-driven authorization model concerned with granting access, while Usage Control is a superset that governs the entire usage phase, extends the decision with obligations/conditions, and can continuously enforce or revoke permissions during and after use. Table 1 compares an AAS-only approach with a dataspace integration approach.

Table 1. Comparison between a data integration approach with only AAS or dataspace.

Aspect	AAS-Only Approach	Dataspace Approach
Identity	Entity-specific	Company-specific
Access control	ABAC only	Usage control policies and usage contracts
API	AAS API	Dataspace protocol
Datasets	AAS metamodel and data	any

In a dataspace, the identity of a provider or consumer is usually company-specific, while the AAS identifies the asset itself. The Dataspace Protocol is the API that all connectors utilize [5].

While dataspaces and DTs are two distinct concepts, they can be combined to enable interoperable and standardized data exchange within industrial ecosystems [28]. Several research projects have defined data integrations between DTs and dataspace, which will be discussed in Section 5.

2.4. Business Use Case of MODAI Project

Our use case consists of multiple press shops with several press machines and sensors, continuously monitoring elastic deformations that occur during each press stroke. The resulting force curves serve as digital fingerprints of the forming operation, enabling real-time detection of anomalies, wear, or manual interventions. This highly localized, high-frequency sensor data is preprocessed by embedded AI and enriched with contextual information close to the machine—creating a valuable data asset at the machine edge. Describing this data asset with AAS submodel templates allows easier use outside the press itself, for example, by the press manufacturer or the customers of the press shop. Figure 3 shows the part manufacturer setup in the use case.

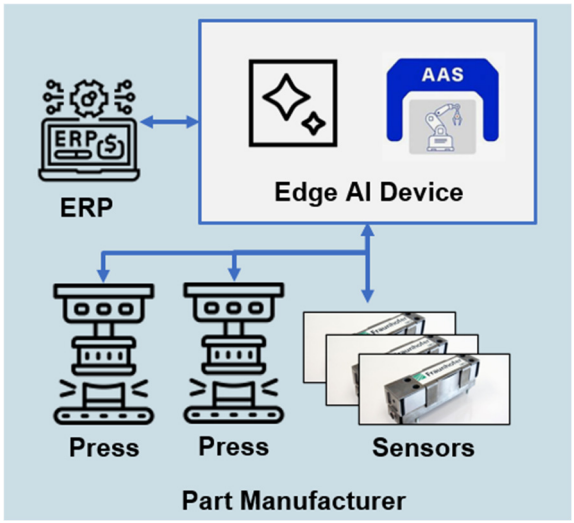

Figure 3. Part manufacturer in MODAI use case with arrows indicating data flow.

The potential of such sensor data extends well beyond the press itself. In quality management, deviations in press signatures can be linked to downstream part defects or machine faults or traced back to material inconsistencies, thereby enhancing root-cause analysis. Finally, procurement and supplier management gain access to live evidence about material and machine behavior during forming—opening the door to collaborative quality assurance with upstream partners.

Sharing this data across the supply chain is useful but not yet common since the way to share it is not harmonized, and data usage terms must be defined in actual contracts. By utilizing a dataspace, the press shop would be able to share this data with the customer in a harmonized way. However, if the press shop must actively manage the data to be shared and update metadata, onboarding friction and additional effort occur. The software landscape of the part manufacturers also heavily varies case by case with different software implementations to be utilized or already in use. Initial workforce orientation efforts with the DT and dataspace software, as well as contracts and policies, also limit broad adoption. The challenge that the press shops are faced with when trying to integrate an AAS—or any kind of data—into the dataspace is the need of expert DT and dataspace knowledge

in the field. Additionally, the frequency of the data makes it necessary to expand on the structure of the AAS, e.g., creating new time-series data segments. The press shops are a highly dynamic environment with frequent tool, configuration, and setup changes.

What enables this broad organizational impact is not just the sensor itself, but the structured exposure of its data through an interoperable framework. By wrapping the sensor, its metadata, and its AI-generated features within a standardized AAS, the press becomes semantically visible and machine-readable within IT systems across the factory.

It is also vital to prepare machines and sensors in the factory for a bidirectional data and control flow across the value chain, where decisions are made not only for one factory location.

When integrated into an industrial dataspace, these AAS-based digital twins can be securely discovered, accessed, and governed across department and company boundaries. This allows, for example, quality managers to retrieve anonymized press performance profiles from suppliers, or planners to orchestrate forming lines based on live status data.

In essence, AAS and dataspaces together act as the bridge that turns isolated sensor events into enterprise-wide intelligence. While the sensor provides the raw capability, it is the structured, sovereign, and policy-controlled integration of its data into dataspaces that truly unlocks its cross-functional value. This example illustrates how a seemingly narrow sensor application can catalyze transformation across organizational layers—when embedded in a federated, interoperable data architecture.

3. User Requirements and Architecture for AAS Dataspace Integration

3.1. Requirements

Table 2 shows an overview of functional and non-functional user requirements for a component ("system") that helps to automate the process of integrating digital twins into the dataspace.

The base requirement is to support providers to automatically share the AAS from a service or registry at the property level or submodel level (FR1). The property level means that data consumers can negotiate for access on a single property, while at the submodel level, consumers will negotiate for the full submodel.

Every time there is a structural change in the AAS, users would have to update or delete the EDC asset for it while creating a new one. This is why a synchronization between AAS structure and published dataspace metadata is required (FR2).

Some data providers prefer to only share a subset of the AAS [21] and grant access at the submodel level, which means that after successful negotiation on a certain submodel, all properties of this submodel are accessible (FR3).

Different projects utilize dataspace connectors like the EDC and modify it for their use case, for example, the Tractus-X EDC [29] or MX-Port [30]. Likewise, there are AAS implementations with a focus on different features [27]. A successful integration should thus support different connectors and AAS implementations (FR4).

Additionally, managing and attaching data contracts to the data results in high manual effort for the user. An integration component should assist the user in attaching contracts to certain elements while also allowing attachment of a pre-configured contract to all shared elements (FR5).

Use cases with high data throughput require a high amount of scaling in the cloud, which means the integration approach should not tightly couple any software components (FR6).

As the authors of [12] also point out, another important requirement is offering data consumers a discovery mechanism for a published AAS in the dataspace (FR7). In dataspaces, a catalog is typically used to offer any published data to all partners. A

limitation of this is the flat structure in which datasets are presented within the catalog. To account for structured entities like an AAS, an AAS-aligned discovery mechanism is needed.

Table 2. Requirements for an integration component between digital twins and dataspace.

#	Requirement Name	Description
FR1	Assisted AAS Registration	The system shall assist in the registration of existing AAS repositories/registries as datasets so that their metadata and endpoints are known to the dataspace. In case an AAS is available only as a file, the system shall instantiate the corresponding AAS repository.
FR2	Structure and Metadata Synchronization	The system shall reflect changes to the AAS structure and any published metadata in the dataspace. This includes, for example, new properties or changing a schema version.
FR3	Selective Sharing of AAS	The system shall provide an option to register only specific parts of an AAS. This includes sharing a single entity, an entity with all its child elements, or a child element by itself.
FR4	Support of Different Connectors and AAS Implementations	The system shall support different implementations of dataspace connectors and AAS repository implementations.
FR5	Assisted Contract Management	The system shall provide per-entity selection of access and usage policies for sharing the AAS in the dataspace as well as supply default policies for an assisted onboarding process.
FR6	Scaling Distributed in Cloud	The implementation of the system shall not interfere with the modularity of the overall dataspace connector, i.e., not require a direct connection to both the administrative and the data management parts of the connector such that both can be individually scaled.
FR7	AAS Discovery in Dataspace	The system shall provide an AAS-aligned discovery mechanism, showing the published AAS entities structured within the AAS environment.
FR8	Zero-Downtime Integration	Integration of AAS into the dataspace shall be available with running AAS services and dataspace connectors, requiring no reconfiguration to both.
FR9	AAS API Support	The system shall support not only secure data transfer of the AAS submodel data (or their elements) but also other AAS functionalities like invoking operations and monitoring events.
NFR1	Security	The system should be secure and not tamper with the security configured in the connector. Additionally, it should support integrating access-protected AAS.
NFR2	Standard Compliance	Extensions or modifications to either DTs or dataspace specifications should be minimal, as they may require modifying implementations when onboarding or transferring data.
NFR3	Synchronization Latency and Consistency	After an update is acknowledged by an AAS, any subsequent catalog request by a dataspace participant shall contain that update.

The next requirement is about the support of existing and running DT or dataspace services by the integration components (FR8). In brownfield use cases where dataspace or AAS infrastructure is already up and running, it is important to provide a solution where deployments are unaffected by the integration component.

Lastly, to offer the full feature set of an AAS over dataspaces, an integration solution must provide functionality beyond reading provider AAS data (FR9). This includes supporting manipulation of AAS data or invoking an AAS operation. Invoking an operation may, for example, require input variables by the data consumer, which must be transmitted, typically over the data plane.

A non-functional requirement is the security of the connector that should not be compromised by the integration components, which have access to the connector API (NFR1). Additionally, an AAS may be access-protected via authentication or access rules. Supporting such an AAS in a way where the access is only granted to authorized users can be beneficial.

Some integration approaches choose to extend parts of either DT or dataspace standards to simplify the onboarding process or reduce user workload. For example, the

EDC asset could be extended to store the AAS metadata. However, any modification to a standard might result in failures when utilizing reference implementations and reduces compatibility with tools following the specifications (NFR2).

Lastly, some manufacturing use cases require low latency when new data from the DT is published to the dataspace (NFR3), for example, for fault detection.

3.2. Comparison of Approaches for AAS–Dataspace Integration

Based on the availability of existing approaches [6–8,22,26], we identified three software architectures for a component that automates the integration of digital twins into the dataspace, which are conceptually shown in Figure 4.

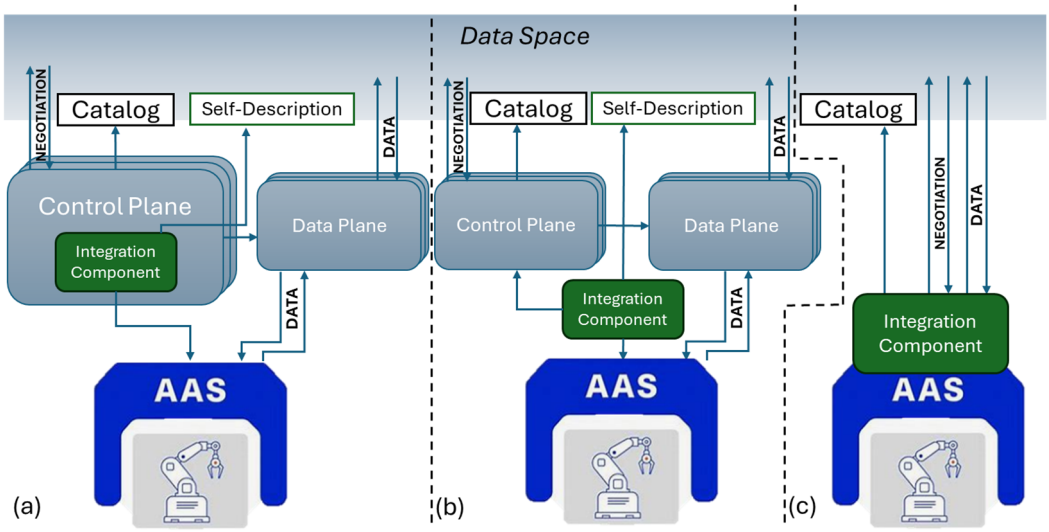

Figure 4. Three possible architectures for a component to integrate AAS into the dataspace separated by dotted lines: (**a**) implement an extension of the EDC control plane; (**b**) the integration component is independent of AAS and dataspace components; (**c**) integrate the dataspace connector into the AAS repository.

EDC Extension. In the first approach, the integration functionality is incorporated into the EDC by leveraging its extensibility, specifically creating a connector extension that contains integration logic. This allows for direct access to connector components, even if they are not accessible via an outward-facing API, as well as simplifying the installation and deployment process and thereby improving usability. However, this tight integration of the extension into the connector comes with the limitation of existing connector deployments being difficult to extend the integration component, because an extension needs to be present during the connector's startup process (FR8). Also, compatibility of the extension is restricted to the connector it was implemented for, possibly but not necessarily including its derivatives like the Tractus-X EDC (FR4).

Standalone Component. The second alternative is to keep the integration component independent from the connector's control plane, handling registration and synchronization of AAS only with the connector's outward-facing APIs like the EDC management API [23]. This approach allows for easier integration into existing systems (FR8), as no build configurations must be changed. Also, a standalone integration component is compatible with any dataspace connector implementations exposing the same API, with the potential to extend compatibility to connectors with different APIs (FR4). Another benefit of deploying the integration component separately is enhanced robustness to connector code changes, as it remains unaffected by modifications to the connector's internal components, which may change more frequently than its API. However, the independence from the connector results in greater overhead in terms of configuration, deployment, and administration by

operating the extension separately (FR6). Also, in comparison to the EDC extension approach, configuring external connector endpoints to provide the complete structure of the shared AAS as a discovery mechanism is not possible. So, a data consumer is limited to the dataspace catalog or needs the API endpoint of the integration component for AAS-aligned discovery (FR7). The authors of [22] have implemented the standalone approach to connect an AAS server to the EDC. Although their component, the "AAS Manager", can register submodels to the EDC, it cannot synchronize changes and does not register submodel elements as separate entities.

AAS Repository Extension. In the third approach, the necessary logic to publish AAS data in the dataspace is injected into the AAS repository in the form of a software library, as depicted in Figure 4. This approach was introduced in the Factory-X project [7] as the "Dataspace Protocol Lib" [31] and implements cataloging, negotiating, and data transfer via the Dataspace Protocol, while access control must be implemented specific to each AAS repository. Combining the AAS repository with the dataspace component results on the one hand in an easier deployment and maintenance, but on the other hand creates a monolithic software component with no possibility of scaling as well as no separation of concern. The component is agnostic to the type of data it can provide in a dataspace; therefore, the AAS repository API must be connected manually in the form of additional code. This also means that to integrate the component into the repository, they need to share the same programming language—in the case of the Dataspace Protocol Lib, this would be Java—restricting the component's overall compatibility to possible AAS repositories.

Table 3 summarizes the comparison of different integration approaches regarding the above requirements.

Table 3. Requirements met by the three alternatives to integrating AAS into dataspaces based on current implementations. [a]: Possible with additional implementation effort per AAS repository.

	EDC Extension	Standalone	AAS Repository Extension
		Requirement Fulfillment	
FR1	Yes	Yes	No [a]
FR2	Yes	No [a]	Yes
FR3	Partial	Partial	Yes
FR4	Yes	Yes	No [a]
FR5	Partial	Partial	No
FR6	Yes	Yes	No
FR7	Yes	No	No
FR8	No	Yes	No
FR9	Partial	Partial	No [a]
NFR1	Yes	Yes	Yes
NFR2	Yes	Yes	Yes
NFR3	No	No	Yes

4. Our AAS Integration Approach

Based on the analysis given in the previous chapter, in this chapter we provide detailed information on how we have realized the first approach and how to realize the second approach to meet further requirements. The third approach requires tight integration into any AAS repository implementation with additional source code. As such, it lacks interoperability with respect to AAS repository and dataspace connector implementations. Therefore, we did not consider this approach for the development of our integration component. In Section 4.1, we present the existing solution for integrating an AAS into

dataspaces, analyze its architecture, and discuss which requirements from Section 3 it currently fulfills. In Section 4.2, we propose a set of changes to the architecture to better accommodate the stated requirements, which leads to the second approach for integration. In Section 4.3, we present our approach in the MODAI use case.

4.1. Current Architecture of EDC Extension

The EDC extension for AAS [32] is a suite of software components assisting in the registration and use of Asset Administration Shells (AASs) in dataspaces via the EDC. Figure 5 shows the current architecture of the EDC extension for AAS, consisting of three control plane extensions as well as one data-plane extension:

- The AAS extension registers and synchronizes AAS elements at the EDC.
- The client extension provides functionality for assisted contract negotiation and data transfer.
- The public management API extension provides a public endpoint for the extension's discovery mechanism, the AAS self-description.
- The AAS data plane is an extension of the EDC's default HTTP data plane and offers additional communication possibilities with AAS repositories for a data consumer, such as invoking AAS operations.

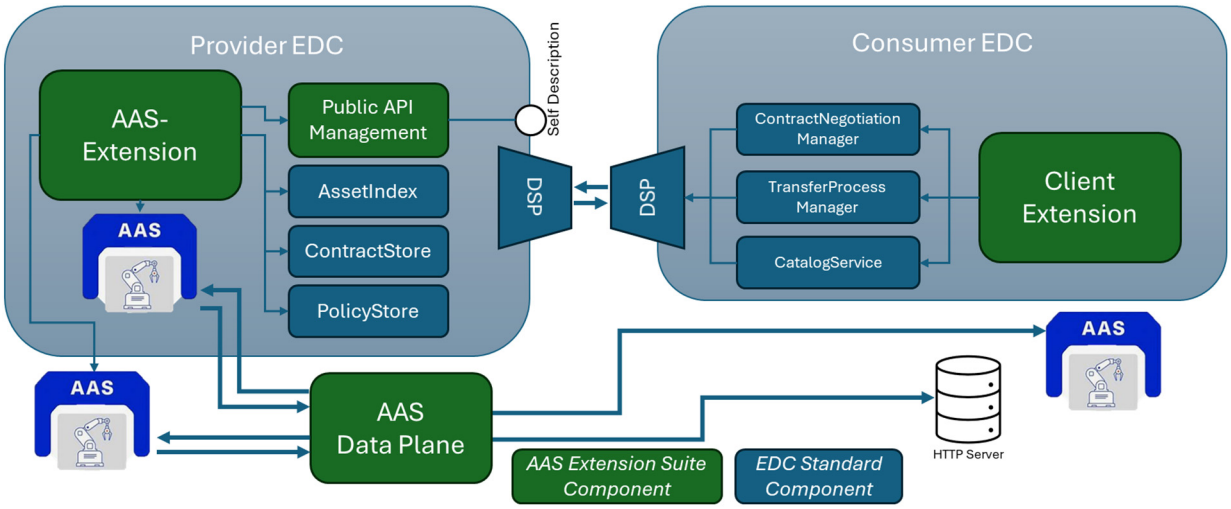

Figure 5. Current state of the EDC extension suite for AAS (green) within the context of the EDC (blue) with arrows indicating interaction flows.

The AAS extension can register any referable entity of an AAS environment into the dataspace by automatically registering them as EDC assets. However, only a reference to an AAS element along with its metadata is stored inside the EDC, not the actual data (e.g., the value of an AAS property). Along with each AAS element, the extension also registers a contract containing access policies making the AAS elements accessible through the dataspace as well as contract (or usage) policies defining the usage rules of the data. A default contract can be specified, and modification of each contract is possible during runtime. Another key feature of the extension is the continuous synchronization of the EDC's AssetStore with the AAS repository. This means that whenever an AAS element is added, removed, or altered, the extension automatically updates the EDC's internal state to reflect these changes.

From the data consumer perspective, the client extension allows for automatic contract negotiation and data transfer using the Dataspace Protocol. For this, a data consumer needs to register a set of policies, which, when offered alongside a trusted data provider's asset,

will be automatically accepted to then obtain the asset data. With the client extension, AAS data from a provider can also be received directly at the consumer EDC to immediately view the transferred data.

The AAS data plane serves as a specialized communication channel to AAS repositories and is based on the EDC's HTTP data plane. On the one hand, it provides an abstraction to register AAS elements by supporting AAS references to them instead of URL paths. On the other hand, it provides data consumers the ability to invoke AAS operations on provider AASs. This is carried out by forwarding the necessary input variables from a consumer's data transfer request to the AAS repository of the provider. An example data transfer request containing operation variables could look like this (Listing 1):

Listing 1. Example TransferRequest of a data consumer containing operation variables.

```
{
"@type": "TransferRequest",
"protocol": "dataspace-protocol-http",
"counterPartyAddress": "https://provider-connector",
"contractId": "agreement-id",
"transferType": "HttpData-PUSH",
"dataDestination": {
"@type": "DataAddress",
"type": "HttpData",
"properties": {
"baseUrl": "https://my-http-receiver/receiveEndpoint",
"method": "PATCH",
"operation": {
"inputArguments": [
{
"value": 4,
"idShort": "in"
}
],
"inoutputArguments": {
"value": "original value",
"idShort": "note"
}
}
}
}
}
```

Lastly, with the public API management extension, the AAS discovery mechanism through self-descriptions is published, offering a structured view of the AAS in the dataspace. Additionally, temporary and authenticated endpoints can be created as a data transfer sink for EDC-to-EDC data transfers.

Having outlined the functionality of each extension making up the AAS extension suite, we now evaluate to what degree they satisfy the requirements established in Section 3.

FR1 is satisfied: Users can publish any number of AAS repositories or registries to the dataspace.

FR2 concerns the synchronization of structural and metadata changes from an AAS repository to the dataspace. This requirement is also satisfied, as the AAS extension periodically checks the AAS repository and compares it to the currently published assets.

FR3, which addresses selective sharing of an AAS, is only partly met: While the extension allows users to choose to only share the AAS submodels or all AAS elements, a user cannot select specific AAS entities to share.

FR4, which demands compatibility with different connectors and AAS implementations, is fully met: The AAS extension supports both the mainline EDC as well as the Tractus-X EDC. It is also compatible with all AAS repositories and registries that imple-

ment the AAS specification Part 2, including FA^{3}ST Service and Registry as well as their BaSyx counterparts.

FR5, which concerns assisted contract management, is partly fulfilled: While default policies are provided to support onboarding, the extension does not support per-entity selection of access and usage policies.

FR6 is fulfilled: The extension preserves the modularity of the EDC, allowing independent scaling of the control and data planes. Additionally, new AAS repositories can be created and registered with the extension.

FR7 is also satisfied: The AAS extension exposes a self-description endpoint showing all registered AAS environments' structures.

FR8, however, is not fulfilled: Since the extension must be deployed within the control plane, any running control plane needs to be redeployed when adopting the extension.

FR9 is partially fulfilled: Reading AAS data and invoking operations are supported, whereas data manipulation and event reading/subscription are not supported (note that the latter is not yet defined in AAS Specification Part 2).

NFR1 is related to security and is fulfilled: The extension leverages the same security mechanisms as the EDC control plane.

NFR2 is about compliance and met: Neither the AAS nor EDC specifications are modified or extended. AAS metadata is represented using EDC asset properties.

NFR3 is not fulfilled: The synchronization interval is configurable both at deployment time and at runtime, with a minimum interval of one synchronization per second. However, if a catalog request by a dataspace participant is received before a synchronization run completes, an inconsistent state of the AAS could potentially be sent in response. Thus, there is no guarantee that the catalog response always reflects the latest AAS state, due to the nature of periodic polling.

4.2. Redesign of the Current Architecture

To meet requirement FR8 (Zero-Downtime Integration) and make our integration component compatible with more brownfield use cases, the AAS extension needs to be deployable next to running EDC control plane and data plane instances. Consequently, all internal connections of the AAS extension to either of them must be substituted by remote connections. There are multiple ways of realizing this change, including extracting the AAS extension's functionality to a standalone runtime, for example, with the Spring Boot framework. The most straightforward and least invasive solution, though, is to create an EDC extension handling the communication with the remote control plane, as shown in Figure 6. This ensures continuing compliance with the EDC ecosystem and the ability to use EDC components such as the base runtime, extension handling, or the EDC's HTTP client implementation, while allowing for keeping much of the AAS extension in the current state. In fact, only borderline cases where the communication to the control plane fails would have to be handled. This would also mean that the AAS extension would still be usable in the same way as before, by deploying it as part of the EDC control plane.

The new extension introduces two primary capabilities: First, the control plane interfaces that are used by the AAS extension are implemented to allow for seamless transition between local and remote control plane communication, specifically the control plane's persistence layer components pertaining to EDC assets, policies, and contracts. Second, because communication with the control plane will take place over its management API via HTTP, all data is transferred in JSON format. This requires the serialization and deserialization of data objects, which is supported by EDC transformer libraries.

Additionally, to remove data plane dependencies, a new AAS library will consolidate common elements and eliminate dependency points (e.g., AASDataAddress).

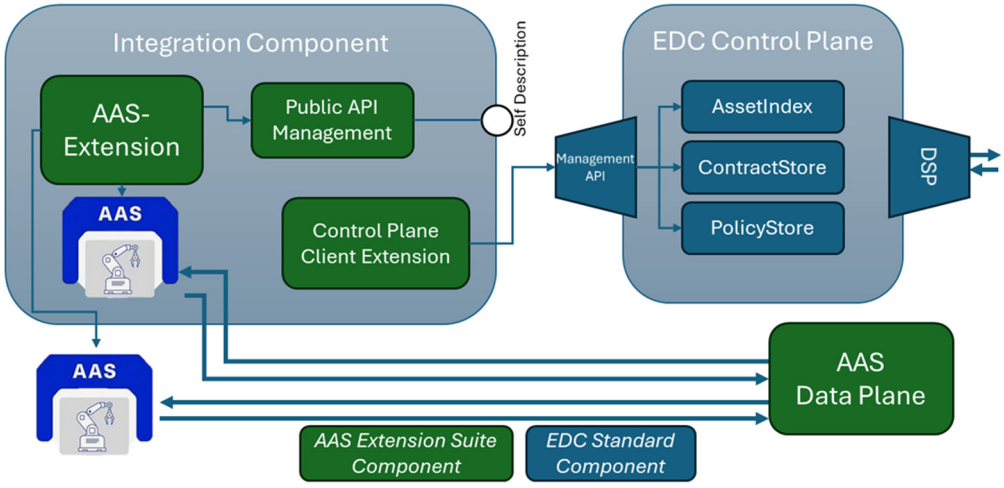

Figure 6. Revised architecture meeting requirement FR8: The integration component (marked green) is extracted from the EDC control plane (marked blue), and connections to the entity stores of the control plane are handled by a new component. Arrows indicate interaction flows between the components.

4.3. Integrating Factory Data in MODAI Project

In this section, we apply our approach in a press shop exchanging data with the customer who receives the finished parts. The data preprocessing of the sensors, press machine, and relevant MES/ERP systems takes place on an edge device. For the AAS service on the edge device, the Eclipse FA^{3}ST Service is set up via a Docker image. We utilized the current extension implementation, which is not a standalone integration approach. Figure 7 shows the implemented integration approach.

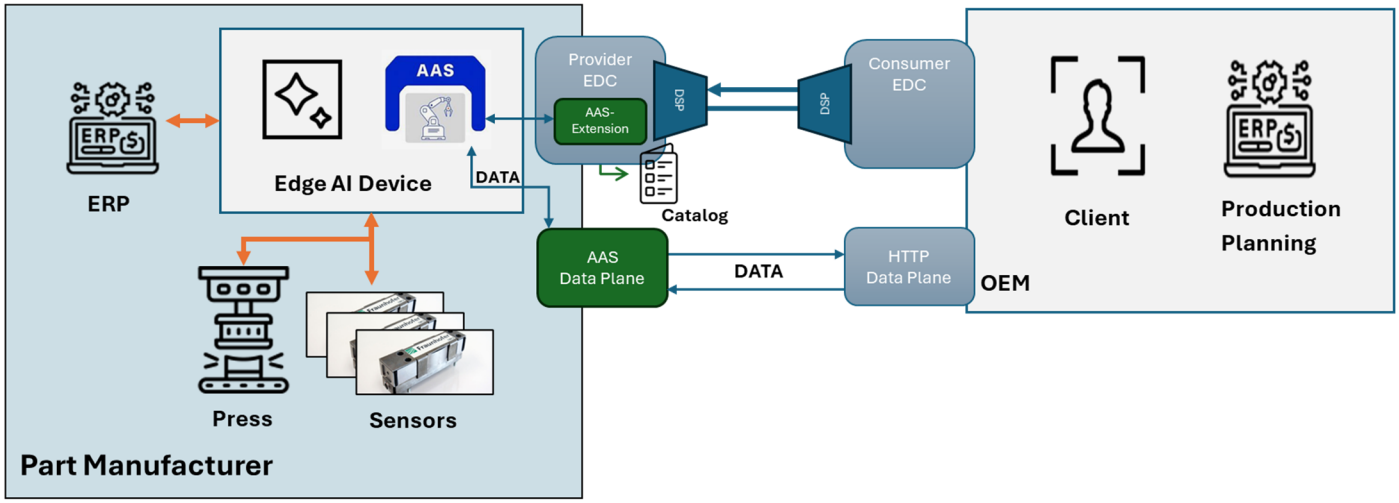

Figure 7. Press machine data integration with AAS and EDC extensions for AAS. Arrows indicate data and control flow.

The Eclipse FA^{3}ST Service contains several submodels for which EDC assets are created by the extension and kept synchronized. The onboarding takes place by sending the URL of the Eclipse FA^{3}ST Service to the extension. FR1 and FR2 are fulfilled by helping the MODAI part manufacturers with initial AAS onboarding and reducing management effort when new submodels or properties arrive in the AAS. Currently, all submodels inside the Eclipse FA^{3}ST Service are automatically chosen to be shared; therefore, FR3 is not fulfilled, but the implementation could be changed to allow for selective sharing, e.g., share only submodels "Time-Series" and "OperationalData". The extension implementation supports

EDC variants such as the Tractus-X EDC (FR4) and attaches the pre-configured default contract to each submodel (FR5). We opted to deploy the mainline EDC in MODAI since a registry-based discovery is not utilized. However, the regular EDC API must be used by the part manufacturer if he wishes to change the contract. This can be improved by a graphical interface for contract management that configures each asset. The synchronization delay is also a configuration parameter that specifies the time interval to wait before the Eclipse FA^{3}ST Service is checked for new changes. A low value increases network traffic but keeps the dataspace metadata neatly synchronized with the AAS. For a production forecast based on quality assessment of the produced parts, setting synchronization delay to sixty seconds is more than sufficient and does not overload the network.

Regarding FR8, the extension implementation already supports running AAS services such as the Eclipse FA^{3}ST Service on the edge device, but the EDC connector must be started in conjunction with the extension. This is a key drawback of the extension approach, because some of the part manufacturers deploy and utilize the connector standalone. While our extension approach also supports FR9 by extending the HTTP data plane with specific AAS calls, the AAS data plane must also be started with the connector. The advantage of the AAS data plane is that the OEM has more AAS interactions available compared to just pulling data via the HTTP data plane.

NFR1 security is partially fulfilled since the extension implementation requires the same API key as any other EDC API call. However, we currently only support AAS services/registries protected by basic authentication and not by the newly released Part 4 ABAC rules. The extension approach does not make any changes to the AAS or dataspace specifications. However, we added a new API endpoint in the connector to make discovery for the OEM easier so that it does not need to browse the catalog, which is presented as a flat list of EDC assets. This is especially useful if the part manufacturers have many different AAS services running and adds some privacy because the available AAS are only shared after negotiation for DTR access. However, all of these discovery mechanisms are project-specific and currently not part of the DT or dataspace specifications.

Because the use case is an ongoing research project, we can only make qualitative statements regarding the evaluation so far. Before the project, the press shop testbed was not equipped with any sensors, edge devices, or DTs. Since there are no running connectors or DT services, our extension approach is suitable. However, the scenario might be different in other press shops. There are also no DT or dataspace experts present for prolonged durations in the press shop. The systems are deployed initially and then maintained at intervals. If the DT structure grows because of new batches or new pressed product types, the dataspace catalog would not include these changes. A workaround would be to share the complete AAS as one entity, but the customer would have to request the complete AAS with each data transfer request. Within the AAS, we opted to structure the data in batches, implying that new elements are created for a batch. The customer can then request specific batch data by negotiating on the specific batch dataspace entry. To reflect new batches in the dataspace, an integration component with synchronization is indispensable, and a lack thereof would mean that the press shop and customer would not consider this kind of dataspace data sharing.

5. Comparison with State-of-the-Art

In this chapter we will elaborate on the integration approaches applied in different dataspace projects that utilize AAS. After that, we compare it with our approach and discuss whether switching to another integration component would fulfill the requirements better.

In other related dataspace projects, the AAS is also used for providing data to the dataspace by mapping AAS metadata to IDS assets. The main differences are in the

IDS connectors and AAS implementations used, their integration patterns, and whether integration components are used or not. This section compares our EDC extension for AAS against state-of-the-art projects and evaluates how well each approach meets the user requirements defined in Section 3.

The prevailing pattern is to map the AAS entities (typically submodels and their descriptors) to dataspace assets and contracts in the connector. The concrete choices of AAS stack (e.g., registry/service), connector, and discovery mechanism vary, and these differences determine how much of the integration effort is manual, how reliably the mapping stays in sync over time, and how approachable the setup is for non-experts.

An AAS registry can contain descriptors of many AAS and their submodels. Submodel descriptors carry endpoint information pointing to the concrete service offering the submodel. Our extension implementation supports both AAS registries and services. It reads AAS descriptors from either an AAS service or an AAS registry and then maintains the corresponding EDC assets, data addresses, and contract definitions.

Catena-X uses a Digital Twin Registry (DTR) at the provider side to make the AAS and their submodels discoverable. It is based on the AAS registry specifications, but their submodel descriptors reference EDC endpoints instead of AAS endpoints. This DTR is used to discover available AAS and submodels. An EDC asset is created for this DTR so that consumers need to negotiate on DTR access first. In this approach, providers still need to create and maintain EDC assets that correspond to the AAS and submodels described in the DTR. The connector used is the Tractus-X EDC, a modified variant based on the mainline EDC. We are not aware of an integration component provided by Catena-X to automate the AAS integration. In short, the providers are required to manually create the EDC assets, and therefore the requirements for an integration component do not apply. Their DTR approach could be supplemented by any integration approach specified in Section 3.2. While Catena-X adds EDC endpoints to submodel descriptors in the DTR, the Tractus-X EDC still needs EDC asset and contract definitions for the AAS and submodels. Our extension implementation could automate this end-to-end mapping from AAS/DTR descriptors since support for the Tractus-X EDC was added. This would be useful for providers that operate outside or alongside Catena-X components or that need a uniform process across multiple ecosystems.

Factory-X is a project based on the Catena-X results but applies the results in manufacturing use cases. They are both part of the Manufacturing-X initiative. It provides hardened connector deployments, often referred to as MX-Ports, in different architecture configurations such as Hercules, Leo, and Orion [33]. AAS plays a central role in the Hercules and Leo approaches of Factory-X, but only the Hercules configuration utilizes the EDC and the DTR. The MX-Port is currently based on the Tractus-X EDC developed in Catena-X. Like in Catena-X, our extension approach can be adopted for the MX-Port Hercules configuration to create dataspace assets for AAS servers and registries. We recently added support for the Tractus-X EDC variant in the extension implementation, which the current MX-Port is based on. However, with time, the implementation might undergo critical changes requiring specific changes in the extension implementation.

In Factory-X, the third integration approach mentioned in Section 3.2 has also been developed. For this, a Dataspace Protocol library is available here [31] that was integrated into the Eclipse BaSyx Server, an AAS service implementation, as a proof of concept. Limitations are that the current library is implemented for Eclipse BaSyx [34] (FR4). We are currently evaluating which steps are necessary to integrate the DSP library into the Eclipse FA^{3}ST Service [35]. Our current extension implementation is an abstraction of common AAS 3.0.1 APIs and is therefore designed to accommodate any AAS implementation. The DSP library integration approach currently dynamically creates the data assets for the

catalog when a catalog request is sent to it. It is implementation-specific as to which elements are shared and which contracts are attached (FR2, FR3). Currently, data assets are created for the AAS elements to which the consumer has access, defined in the BaSyx access control policy. The approach is limited to assisting the user in creating data assets in the catalog for all the AAS and submodels inside the Eclipse BaSyx Server equipped with the DSP library.

To scale the DSP library approach in the cloud, the AAS services must be scaled, effectively providing their own control and data plane. However, it currently does not yet support multiple control and data planes. Regarding the AAS Discovery (FR7), the library implementation approach does not offer any additional mechanisms, and it does not support integrating with running connectors or running AAS services/registries (FR8). The DSP library provides a common HTTP data plane as found in the EDC (FR9). Regarding security, the mapping between AAS security and the dataspace catalog is not implemented, as is the case in all other integration approaches. However, Eclipse BaSyx Server provides its own access control implementation currently not based on AAS Part 4. Except for additional discovery mechanisms, all integration approaches are fully compliant with the AAS and dataspace specifications. Regarding the synchronization latency and consistency (NFR2), this approach offers the least synchronization delay by coupling the catalog and the AAS in one component. Compared to the other integration approaches, high-frequency synchronization does not cause high network load but requires some processing and memory bandwidth. The current library implementation synchronizes on user catalog requests but could be modified to synchronize at given time intervals.

In [36], the authors describe the System of Records integration into the AAS dataspace, but the work focuses on data integration into the AAS, for example, data from ERP systems. In [22], the authors implement a standalone approach called AAS Manager. This tool is located between the AAS servers and the EDC. Their current implementation fulfills many requirements of a standalone approach, like FR1, FR3, FR4, FR5, FR6, and FR8. We could not identify a synchronization mechanism (FR2), so the user must repeat the request to the AAS Manager to update the resources. The authors introduce a novel approach to manage the contracts in the AAS, an ODRL submodel template. In this submodel, the policies are described, and the AAS Manager maps them into EDC contracts later. A variant of this was also introduced in [10], where the authors add the concept of usage policies inside a shared AAS, complementing the AAS access control [4]. A data consumer then must evaluate those policies when it shares the AAS with yet another party. In [26], the authors propose an EDC extension publishing AAS elements. Their focus lies on the selective sharing (FR3) and per-element usage policies (FR5), while also fulfilling requirements FR1 and NFR3. Drawbacks of their approach are the lack of synchronization (FR2) and the need to modify each AAS element regarding if it should be published and which policies apply by adding "HasExtension" attributes.

In summary, the tooling for the integration components for AAS into a dataspace is still at an early stage, with many developments and improvements yet to come. The focus of current dataspace projects is the specification and development of connector reference implementations while fulfilling specific use case requirements outside onboarding and integration tooling.

6. Conclusions and Future Work

In this paper, we discussed the integration of digital twins in dataspaces, more specifically the automated integration of the AAS into the IDS with an integration software component. The recurrent pattern to share AAS is the creation of IDS assets pointing to AAS data in the dataspace catalog.

The dataspace projects have converged on reusing AAS as the semantic anchor for dataspace publication, for example, in Catena-X and Factory-X. While their focus is currently not on simplified onboarding and integration tooling, some approaches have been implemented, namely the DSP-Lib and our EDC extension for AAS. We followed the extension approach but decided to switch to a standalone approach because the requirements are better covered, especially in supporting already running connectors.

The main pain points of dataspace integration are repetitive asset modeling inside connectors and the steep learning curve for non-experts. These are all addressed by integration tools following the three integration approaches discussed in Section 3.2. Our EDC extension for AAS is one example that closes these gaps by automatically deriving EDC assets and attaching contracts to AAS metadata and descriptors. As such, the connectors are extended to "twin-native" connectors, which specialize in handling DTs like the AAS.

We contributed (i) a structured requirements framework for automated AAS–dataspace integration, (ii) an analysis of three architectural approaches (connector extension, standalone component, repository extension), (iii) a description of our current implementation of an EDC extension with an AAS-aware data plane, (iv) a redesign to enable zero-downtime integration against running connectors, and (v) a comparative perspective on Catena-X and Factory-X practices and our application in the MODAI use case.

Both the connector extension and standalone approaches are viable, but they optimize for different constraints. The extension provides deep integration and feature completeness (e.g., specialized AAS operations via a custom data plane) and strong coupling with the connector. The standalone approach is superior for brownfield deployments, avoiding connector redeployment and supporting the EDC connector landscape. The repository extension approach (as seen with the Dataspace Protocol Lib) simplifies deployment in homogeneous stacks by creating monolithic coupling with the AAS services.

Requirements coverage with our current EDC extension is strong for automated onboarding (FR1), synchronization (FR2, FR7), scaling compatibility (FR6), multi-connector/AAS support (FR4), and AAS-specific data operations (FR9). It partially fulfills requirements for selective sharing (FR3) and assisted contract management (FR5), but does not yet meet requirements for zero-downtime of running connectors (FR8). Security and standard compliance are partially preserved (NFR1, NFR2), with support for AAS ABAC being planned for the future.

An AAS-aware data plane enables operation invocation with parameters, paving the way for richer interactions between business partners. Discovery remains a critical issue because flat connector catalogs hinder usability. It is therefore often complemented with AAS registries (e.g., DTR) or publishing structured self-descriptions.

Our results in the MODAI use case indicate that integration automation in connectors significantly lowers onboarding friction, reduces manual asset modeling, and brings contract management closer to operational and supply-chain data. However, we did not evaluate the benefits in a quantitative manner. The comparison was focused on whether the requirements of Section 3.1 were met. Moving from a connector extension to a standalone approach appears to be the most promising path for broad adoption, especially in brownfield contexts.

Regarding future work, we are planning to adapt our implementation to be a more modular tool so that users with a different requirement focus have more options in integrating DTs. Some users might not require a brownfield integration and want to focus on AAS-specific data transfers. For example, the exchange of AAS machine events is a research focus in the Factory-X project. We are also active in the development of frontend user interfaces to make interaction with the integration tool easier. A particular point of interest is the ongoing implementation of AAS Part 4 Security, which allows users to protect

the AAS from unauthorized access. However, there is no clear concept of the interaction between the data usage contracts in the dataspace and the access control rules of the AAS, or how data usage contracts should be generated when integrating AAS into the dataspace. The authors in [22], for example, chose to create an ODRL submodel template. It might be feasible to convert access control rules directly into contracts in an integration tool, but whether this requirement is needed will be decided in the manufacturing use cases. The authors of [20] also described limitations regarding the current monitoring and enforcement of these data policies. They also mention that "solutions for cross-company access control are also required to regulate access to sensitive data within companies". This directly relates to the access control described in AAS Part 4 Security.

Realizing this vision will require further joint work across communities (IDTA, IDSA, DSSC, Catena-X/Factory-X), harmonizing the integration and discovery of digital twins into dataspaces. With these steps, industrial ecosystems can unlock sovereign, scalable, and semantically rich data sharing where digital twins based on industrial standards provide key data in dataspaces.

Author Contributions: Conceptualization, C.S., F.V., and L.S.; methodology, C.S., F.V., and L.S.; software, C.S. and F.V.; writing—original draft preparation, C.S., F.V., L.S., and H.K.; writing—review and editing, L.S., F.V., and C.S.; supervision, L.S.; project administration, L.S.; funding acquisition, L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was financially supported by the Ministry of Trade, Industry, and Energy (MOTIE), Korea, under the "Global Industrial Technology Cooperation Center (GITCC) program" supervised by the Korea Institute for Advancement of Technology (KIAT). (Task No. P0028468). This research was financially supported by SM4RTENANCE—Digital Europe Grant Agreement N.101123490.

Data Availability Statement: The original data presented in this study are openly available in the GitHub repository at [32].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Grieves, M.; Vickers, J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*; Springer: Cham, Switzerland, 2017. [CrossRef]
2. Industrial Internet Consortium. White Paper: Digital Twins for Industrial Applications. Available online: https://www.iiconsortium.org/pdf/IIC_Digital_Twins_Industrial_Apps_White_Paper_2020-02-18.pdf (accessed on 13 July 2025).
3. Industrial Digital Twin Association e.V. Available online: https://industrialdigitaltwin.org (accessed on 6 July 2025).
4. Industrial Digital Twin Association e.V. Asset Administration Shell Specification: Part 4: Security (IDTA-01004), Version 3.0.1. July 2025. Available online: https://industrialdigitaltwin.io/aas-specifications/IDTA-01004/v3.0.1/index.html (accessed on 26 September 2025). [CrossRef]
5. Otto, B.; Steinbuß, S.; Teuscher, A.; Bader, S. IDS Reference Architecture Model 4.0. Available online: https://docs.internationaldataspaces.org/ids-knowledgebase/ids-ram-4 (accessed on 6 September 2025).
6. Catena-X Automotive Network e.V. Catena-X. Available online: https://catena-x.net/en/ (accessed on 26 May 2025).
7. Factory-X. The Digital Ecosystem. Available online: https://factory-x.org/ (accessed on 26 May 2025).
8. CircularTwAIn. AI Platform for Integrated Sustainable and Circular Manufacturing. Available online: https://www.circular-twain-project.eu/ (accessed on 26 May 2025).
9. Manoury, M.; Riedelsheimer, T.; Hellmeier, M.; Meyer, T. Supporting Changes in Digital Ownership and Data Sovereignty Across the Automotive Value Chain with Catena-X. *Procedia Comput. Sci.* **2025**, *253*, 374–383. [CrossRef]
10. Hang, J.H.; Charles, D.S.; Gan, Z.H.; Gan, S.K.; Lim, Y.M.; Lee, W.P.; Wong, T.L.; Goh, C.P. Constructing a Real-Time Value-Chain Integration Architecture for Mass Individualized Juice Production. *Information* **2022**, *13*, 56. [CrossRef]
11. Bader, S.R.; Maleshkova, M. Towards Integrated Data Control for Digital Twins in Industry 4.0. 2020. Available online: https://publica.fraunhofer.de/handle/publica/408723 (accessed on 26 September 2025).

12. Ajdinović, S.; Strljic, M.; Lechler, A.; Riedel, O. Interoperable Digital Product Passports: An Event-Based Approach to Aggregate Production Data to Improve Sustainability and Transparency in the Manufacturing Industry. In Proceedings of the 2024 IEEE/SICE International Symposium on System Integration (SII), Ha Long, Vietnam, 8–11 January 2024; IEEE: New York, NY, USA, 2024; pp. 729–734. [CrossRef]
13. Jungbluth, S.; Witton, A.; Hermann, J.; Ruskowski, M. Architecture for Shared Production Leveraging Asset Administration Shell and Gaia-X. In Proceedings of the 2023 IEEE 21st International Conference on Industrial Informatics (INDIN), Lemgo, Germany, 17–20 July 2023; IEEE: New York, NY, USA, 2023; pp. 1–8. [CrossRef]
14. Alexopoulos, K.; Weber, M.; Trautner, T.; Manns, M.; Nikolakis, N.; Weigold, M.; Engel, B. An industrial data-spaces framework for resilient manufacturing value chains. *Procedia CIRP* **2023**, *116*, 299–304. [CrossRef]
15. Silva, H.; Moreno, T.; Almeida, A.; Soares, A.L.; Azevedo, A. A Digital Twin Platform-Based Approach to Product Lifecycle Management: Towards a Transformer 4.0. In *Innovations in Industrial Engineering II. ICIENG 2022*; Machado, J., Soares, F., Trojanowska, J., Ivanov, V., Antosz, K., Ren, Y., Manupati, V.K., Alejandro Pereira, A., Eds.; Lecture Notes in Mechanical Engineering; Springer: Cham, Switzerland, 2023. [CrossRef]
16. Iñigo, M.; Legaristi, J.; Larrinaga, F.; Zugasti, E.; Cuenca, J.; Kremer, B.; Estepa, D.; Ayuso, M.; Montejo, E. Towards an Advanced Artificial Intelligence Architecture Through Asset Administration Shell and Industrial Data Spaces. In *Advances in Artificial Intelligence in Manufacturing*; Wagner, A., Alexopoulos, K., Makris, S., Eds.; ESAIM 2023. Lecture Notes in Mechanical Engineering; Springer: Cham, Switzerland, 2024. [CrossRef]
17. Moreno, T.; Almeida, A.; Toscano, C.; Ferreira, F.; Azevedo, A. Scalable Digital Twins for industry 4.0 digital services: A dataspaces approach. *Prod. Manuf. Res.* **2023**, *11*, 2173680. [CrossRef]
18. Yallıç, F.; Albayrak, Ö.; Ünal, P. Asset Administration Shell Generation and Usage for Digital Twins: A Case Study for Non-destructive Testing. In *IN4PL 2022, Proceedings of the 3rd International Conference on Innovative Intelligent Industrial Production and Logistics—Volume 1: ETCIIM, Valletta, Malta, 24–26 October 2022*; SciTePress: Setúbal Municipality, Portugal, 2022; pp. 299–306. [CrossRef]
19. Zink, F.; Fink, L.; Wallner, B.; Bleicher, F.; Trautner, T. Advancing the Collaboration in Manufacturing Networks A Systematic Literature Review on Implementations of Asset Administration Shells and Data Spaces. *Procedia CIRP* **2025**, *136*, 546–551, ISSN 2212-8271. [CrossRef]
20. Neubauer, M.; Steinle, L.; Reiff, C.; Ajdinović, S.; Klingel, L.; Lechler, A.; Verl, A. Architecture for manufacturing-X: Bringing asset administration shell, eclipse dataspace connector and OPC UA together. *Manuf. Lett.* **2023**, *37*, 1–6, ISSN 2213-8463. [CrossRef]
21. Inigo, M.A.; Legaristi, J.; Larrinaga, F.; Perez, A.; Cuenca, J.; Kremer, B.; Montejo, E.; Porto, A. Towards Standardized Manufacturing as a Service through Asset Administration Shell and International Data Spaces Connectors. In Proceedings of the IECON 2022—48th Annual Conference of the IEEE Industrial Electronics Society, Brussels, Belgium, 9 December 2022; IEEE: New York, NY, USA, 2022; pp. 1–6. [CrossRef]
22. Sidibe, G.D.S.; Dhouib, S. An Approach for Sovereign Data Exchange of AAS Digital Twins Through the International Data Space Network. In Proceedings of the 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 10–13 September 2024; IEEE: New York, NY, USA, 2024; pp. 1–4. [CrossRef]
23. Eclipse Foundation. Eclipse Dataspace Connector. Available online: https://eclipse-edc.github.io/ (accessed on 26 May 2025).
24. Data Spaces Support Centre. Data Spaces Blueprint. Available online: https://dssc.eu/space/BVE2/1071251457/ (accessed on 6 September 2025).
25. Eclipse Foundation. Eclipse Dataspace Components. Available online: https://projects.eclipse.org/projects/technology.edc (accessed on 26 May 2025).
26. Volz, F.; Sutschet, G.; Stojanovic, L.; Usländer, T. On the Role of Digital Twins in Data Spaces. *Sensors* **2023**, *23*, 7601. [CrossRef] [PubMed]
27. Jacoby, M.; Baumann, M.; Bischoff, T.; Mees, H.; Müller, J.; Stojanovic, L.; Volz, F. Open-Source Implementations of the Reactive Asset Administration Shell: A Survey. *Sensors* **2023**, *23*, 5229. [CrossRef] [PubMed]
28. Schöppenthau, F.; Patzer, F.; Schnebel, B.; Watson, K.; Baryschnikov, N.; Obst, B.; Chauhan, Y.; Kaever, D.; Usländer, T.; Kulkarni, P. Building a Digital Manufacturing as a Service Ecosystem for Catena-X. *Sensors* **2023**, *23*, 7396. [CrossRef] [PubMed]
29. Eclipse Tractus-X. Tractus-X EDC. Available online: https://github.com/eclipse-tractusx/tractusx-edc (accessed on 19 September 2025).
30. Factory-X. MX-Port Introduction. Available online: https://factory-x.org/factory-x-introduces-the-mx-port-concept/ (accessed on 19 September 2025).
31. Factory-X. Dataspace Protocol Lib. Available online: https://github.com/factory-x-contributions/dataspace-protocol-lib/ (accessed on 26 May 2025).
32. Fraunhofer IOSB. EDC Extension for AAS. Available online: https://github.com/FraunhoferIOSB/EDC-Extension-for-AAS (accessed on 26 May 2025).

33. Friedrich, C.; Vogt, S.; Rudolph, F.; Patolla, P.; Grützmann, J.M.; Hohmeier, O.; Richter, M.; Wenzel, K.; Reichelt, D.; Ihlenfeldt, S. Enabling Federated Learning Services Using OPC UA, Linked Data and GAIA-X in Cognitive Production. *J. Mach. Eng.* **2024**, *24*, 18–33. [CrossRef]
34. Eclipse Foundation. Eclipse BaSyx. Available online: https://eclipse.dev/basyx/ (accessed on 26 September 2025).
35. Eclipse Foundation. Eclipse FA^{3}ST. Available online: https://projects.eclipse.org/projects/dt.fa3st (accessed on 26 September 2025).
36. Danish, M.G.A.; Schnicke, F. Integrating Systems of Record (SOR) into the Asset Administration Shell (AAS) Dataspace. In Proceedings of the 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 10–13 September 2024; IEEE: New York, NY, USA, 2024; pp. 1–8. [CrossRef]