JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Lucca Leserer, BSc

Angefertigt am
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuer
Assoz.-Prof. Mag. Dr.
Christoph Schütz

Mai 2026

Konzeption und prototypische Umsetzung eines Asset Management Portals für ein globales Maschinenbauunternehmen

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Kurzfassung

Produzierende Unternehmen stehen vor der Herausforderung, produktionsrelevante Assets aus historisch gewachsenen, heterogenen IT- und OT-Systemlandschaften interoperabel, skalierbar und semantisch konsistent abzubilden. Bestehende Integrationsansätze adressieren entweder die Modellierung der Verwaltungsschale oder einzelne Integrationsaspekte, ohne eine durchgängige Architektur von der Datenquelle bis zur Verwaltungsschale vorzuschlagen.

Die vorliegende Arbeit entwickelt eine zweigeteilte Systemarchitektur, die Datenintegration und semantische Modellierung konsequent trennt. Eine hexagonale Integrations- und Dateninfrastruktur bindet heterogene Quellsysteme über technologiespezifische Adapter an und überführt deren Daten mittels einer dreistufigen Medaillen-Pipeline (Bronze, Silver, Gold) in stabile, quellsystemunabhängige Datenstrukturen. Darauf aufbauend erzeugt ein konfigurationsbasierter Mapping-Service anhand deklarativer Regeln Submodelle, folgend der Definition der Industrial Digital Twin Association (IDTA), und persistiert diese über die Eclipse-BaSyx-Laufzeitumgebung als standardkonforme Verwaltungsschalen.

Die Architektur wird prototypisch umgesetzt und anhand eines industriellen Anwendungsfalls validiert, in dem Daten eines Fünf-Achs-Bearbeitungszentrums aus ERP-, TDM-, CAD/CAM- und OPC UA-Quellsystemen konsolidiert und in die drei IDTA-Submodelle Digital Nameplate, Technical Data und Process Parameters überführt werden. Die Arbeit zeigt dabei, dass eine standardkonforme Integration einer Asset Administration Shell über heterogene Quellsysteme hinweg mit ausschließlich quelloffenen Komponenten realisierbar ist.

Abstract

Manufacturing companies face the challenge of representing production-relevant assets from historically grown, heterogeneous IT and OT system landscapes in an interoperable, scalable, and semantically consistent manner. Existing integration approaches either address Asset Administration Shell modeling or individual integration aspects without proposing an end-to-end architecture from data source to Asset Administration Shell.

This thesis develops a two-layer system architecture that strictly separates data integration from semantic modeling. A hexagonal integration and data infrastructure connects heterogeneous source systems via technology-specific adapters and transforms their data through a three-tier medallion pipeline (Bronze, Silver, Gold) into stable, source-system-independent data structures. Building on this foundation, a configuration-based mapping service generates submodels, which follow the definition of the Industrial Digital Twin Assocation (IDTA), using declarative rules and persists them as standards-compliant Asset Administration Shells through the Eclipse BaSyx runtime environment.

The architecture is implemented as a prototype and validated using an industrial use case in which data from a five-axis machining center is consolidated from ERP, TDM, CAD/CAM, and OPC UA source systems and transferred to the three IDTA submodels: Digital Nameplate, Technical Data, and Process Parameters. This work demonstrates that a standards-compliant integration of an Asset Administration Shell across heterogeneous source systems can be achieved using exclusively open-source components.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Produzierende Unternehmen stehen im Zuge zunehmender Digitalisierung und Automatisierung vor der Aufgabe, Transparenz über den Zustand ihrer produktionsrelevanten Assets – Maschinen, Werkzeuge und Fertigungsressourcen – herzustellen. Standardisierte, maschinenlesbare Repräsentationen der Asset-Zustände gewinnen dabei an Bedeutung. Mit der *Asset Administration Shell* (AAS, dt. Verwaltungsschale) existiert ein vielversprechender Ansatz – standardisiert in IEC 63278-1, vorangetrieben durch die *Plattform Industrie 4.0* und die *Industrial Digital Twin Association* (IDTA) – zur einheitlichen, interoperablen Beschreibung von Asset-Zuständen über Systemgrenzen hinweg.

In der betrieblichen Realität stehen dem Anspruch nach einer integrierten Repräsentation von Asset-Zuständen jedoch historisch gewachsene, heterogene Systemlandschaften aus Information Technology (IT) und Operational Technology (OT) entgegen. Spezialisierte Systeme wie ein Enterprise Resource Planning (ERP) System, Manufacturing Execution System (MES), Product Lifecycle Management (PLM) System, Computer Aided Design (CAD) bzw. Computer Aided Manufacturung (CAM) System, Tool Data Management (TDM) System und Maschinensteuerungen operieren mit jeweils eigenen Datenmodellen, Schnittstellen und Integrationsparadigmen. Daraus resultieren fragmentierte, punktuelle Integrationslösungen, die weder skalierbar noch semantisch konsistent sind. Zwischen den heterogenen Quelldaten solcher Systemlandschaften und einer standardkonformen Repräsentation von Asset-Zuständen klafft eine architektonische Lücke, die in der Literatur bislang nur partiell adressiert wird [1]. Bestehende Arbeiten fokussieren entweder auf die AAS-Modellierung selbst oder auf einzelne Integrationsaspekte, ohne eine durchgängige Architektur von der Datenquelle bis zur Verwaltungsschale vorzuschlagen. Diese Lücke wird in der vorliegenden Arbeit adressiert und als Design-Problem aufgegriffen. Es fehlt an einer architektonischen Lösung, die skalierbare, interoperable und semantisch konsistente Asset-Repräsentationen in heterogenen Produktionsumgebungen ermöglicht. Bestehende Integrationsansätze erweisen sich entweder als systemspezifisch und damit schwer erweiterbar oder sie abstrahieren zu stark von der fachlichen Domäne und büßen Präzision ein. Daraus ergibt sich die zentrale Forschungsfrage dieser Arbeit:

> *Wie kann eine Systemarchitektur zur Datenintegration gestaltet werden, die eine interoperable, skalierbare und semantisch konsistente Repräsentation von Asset-Zuständen in heterogenen IT- und OT-Systemlandschaften ermöglicht?*

Ziel dieser Arbeit ist der Entwurf und die prototypische Umsetzung einer solchen Systemarchitektur. Der Entwurf folgt einem zweigeteilten Ansatz. Eine *Integrations- und Datenarchitektur* auf Basis eines hexagonalen Architekturmusters bindet heterogene Quellsysteme über eine dreistufige Medaillen-Pipeline aus Bronze-, Silver- und Gold-Schicht an, normalisiert und konsolidiert deren Daten einem stabilen Datenvertrag folgend. Darauf aufbauend stellt eine *AAS-Semantik- und Managementschicht* mithilfe von Eclipse BaSyx eine standardkonforme, IDTA-Template-basierte Repräsentation der Assets als Verwaltungsschalen bereit.

In dieser Arbeit wird die prototypische Umsetzung und Validierung der vorgeschlagenen Architektur anhand eines industriellen Anwendungsfalls durchgeführt. Dieser Anwendungsfall beinhaltet die Integration von Daten eines Fünf-Achs-Bearbeitungszentrums aus den Bereichen ERP, TDM, CAD/CAM und Open Platform Communications Unified Architecture (OPC UA) in IDTA-konforme AAS-Submodelle. Obwohl der Anwendungsfall von einem konkreten Unternehmen stammt, kann die Architektur grundsätzlich auf vergleichbare Unternehmen übertragen werden, da die verwendete Architektur auf einer quellsystemunabhängigen Integrationslogik, standardbasierten AAS-Modellierung und modularen Schichtenstruktur aufgebaut ist. Die getroffenen Architekturentscheidungen werden systematisch dokumentiert um die Nachvollziehbarkeit des Entwurfsprozesses zu gewährleisten und die Anpassung der Architektur für andere Unternehmen zu ermöglichen. Der Anspruch der Arbeit liegt dabei auf der Demonstration der grundsätzlichen Umsetzbarkeit der Architektur.

Diese Arbeit ist wie folgt aufgebaut. Kapitel 2 beschreibt den Stand der Technik. Kapitel 3 untersucht die bestehende Systemlandschaft und konkretisiert das Design-Problem, um daraus funktionale und nicht-funktionale Anforderungen sowie Architekturziele abzuleiten. Kapitel 4 konzipiert eine zweigeteilte Architektur und dokumentiert die getroffenen Architekturentscheidungen. Kapitel 5 beschreibt die prototypische Umsetzung der Architektur im konkreten Anwendungsfall mit dem Fünf-Achs-Bearbeitungszentrum. Kapitel 6 evaluiert den Prototyp auf Basis der definierten Anforderungen. Kapitel 7 reflektiert die Ergebnisse und diskutiert Einschränkungen. Kapitel 8 fasst die Erkenntnisse zusammen und bietet einen Ausblick auf potenzielle Erweiterungen.

Kapitel 2

Stand der Technik

Steigende Anforderungen an Skalierbarkeit, Echtzeitfähigkeit und Interoperabilität prägen die Verarbeitung und Integration von Daten in modernen, verteilten Softwaresystemen, insbesondere im industriellen Umfeld. [24] Aus den Anforderungen der praktischen Umsetzung dieser Arbeit ergibt sich daher die Notwendigkeit, sowohl grundlegende Integrations- und Verarbeitungsparadigmen als auch spezifische Industriestandards systematisch zu betrachten.

Dieses Kapitel führt zunächst in die Konzepte des Event Streamings und der Datenintegration ein und ordnet deren Relevanz für entkoppelte, ereignisbasierte Architekturen ein. Anschließend werden mit OPC UA und der Asset Administration Shell (AAS) zentrale Komponenten für die standardisierte Erfassung, semantische Strukturierung und interoperable Bereitstellung industrieller Asset-Daten dargestellt. Darauf aufbauend wird mit der hexagonalen Architektur eine methodische Grundlage behandelt, die eine robuste Strukturierung der Lösung unterstützt. Abschließend werden verwandte Arbeiten und bestehende Systemansätze eingeordnet, um den Stand der Technik im Kontext der vorliegenden Zielsetzung zu verorten.

2.1 Event Streaming und Datenintegration

Die zunehmende Vernetzung industrieller Systeme sowie der steigende Bedarf an Echtzeitdaten führen zu wachsenden Anforderungen an die Integration und Verarbeitung von Daten in verteilten Architekturen [15]. Klassische, synchron ausgelegte Kommunikationsmodelle stoßen dabei insbesondere hinsichtlich Skalierbarkeit, Flexibilität und Entkopplung an ihre Grenzen [13]. Event Streaming beschreibt ein Architektur- und Integrationsparadigma, bei dem Zustandsänderungen oder Ereignisse als fortlaufender Datenstrom erfasst, übertragen und verarbeitet werden. In klassischen Request-Response-Modellen sendet ein Aufrufer eine Anfrage an einen definierten Empfänger und wartet blockierend auf dessen Antwort. Sender und Empfänger sind dabei direkt aneinander gekoppelt [13]. Im Gegensatz dazu werden in ereignisbasierten Architekturen Zustandsänderungen asynchron publiziert und von interessierten Konsumenten unabhängig verarbeitet, ohne dass eine solche direkte Kopplung besteht. Diese lose Kopplung stellt eine zentrale Eigenschaft ereignisbasierter Systeme dar und ermöglicht eine flexible Erweiterung sowie den unabhängigen Betrieb einzelner Komponenten und die zeitlich entkoppelte Ereigniserzeugung und -verarbeitung. Ereignisse können persistiert und zu einem späteren Zeitpunkt erneut verarbeitet werden, wodurch sowohl Fehlertoleranz als auch Nachvollziehbarkeit erhöht werden. Insbesondere in datenintensiven Systemen erlaubt dieser Ansatz eine robuste Verarbeitung großer Datenmengen bei gleichzeitiger horizontaler Skalierung [25]. Als Referenztechnologie für Event Streaming hat sich Apache Kafka etabliert. Kafka wurde ursprünglich als verteiltes Commit-Log konzipiert und ermöglicht die persistente,

sequentielle Speicherung von Ereignissen in sogenannten Topics [25]. Produzenten veröffentlichen Ereignisse, während Konsumenten diese unabhängig voneinander lesen und verarbeiten können. Die Architektur von Kafka trennt dabei bewusst Datenpersistenz und Datenverarbeitung, was eine parallele Nutzung von Datenströmen sowie die Wiederverarbeitung historischer Daten erlaubt [25]. Aufgrund dieser Eigenschaften hat sich Kafka als eine der am weitesten verbreiteten Plattformen für skalierbare Event-Streaming-Systeme etabliert.

Während Event-Streaming-Plattformen primär den Transport und die Verarbeitung von Ereignissen adressieren, stellen industrielle Umgebungen zusätzliche Anforderungen an die Datenintegration. Produktionsanlagen, Steuerungssysteme und eingebettete Geräte sind häufig durch heterogene Kommunikationsprotokolle, proprietäre Schnittstellen und unterschiedliche Datenmodelle gekennzeichnet [29]. Für eine nachhaltige Integration ist es daher nicht ausreichend, Daten lediglich zu übertragen, sondern vielmehr müssen diese in ihrem fachlichen Kontext interpretierbar sein. OPC Unified Architecture (OPC UA) wurde entwickelt, um diesen Anforderungen gerecht zu werden. Als plattform- und herstellerunabhängiger Standard ermöglicht OPC UA die interoperable und sichere Kommunikation zwischen industriellen Systemen unterschiedlicher Ebenen [29]. Im Gegensatz zu klassischen Schnittstellenansätzen kombiniert OPC UA Kommunikationsmechanismen mit einem umfassenden Informationsmodell. Die grundlegende Funktionsweise von OPC UA ist in Abbildung 2.1 schematisch dargestellt. Die Abbildung verdeutlicht, dass OPC UA auf zwei zentralen Fundamenten aufbaut: der Kommunikationsinfrastruktur und der Daten- beziehungsweise Informationsmodellierung [29, Fig. 1.6, S. 30].

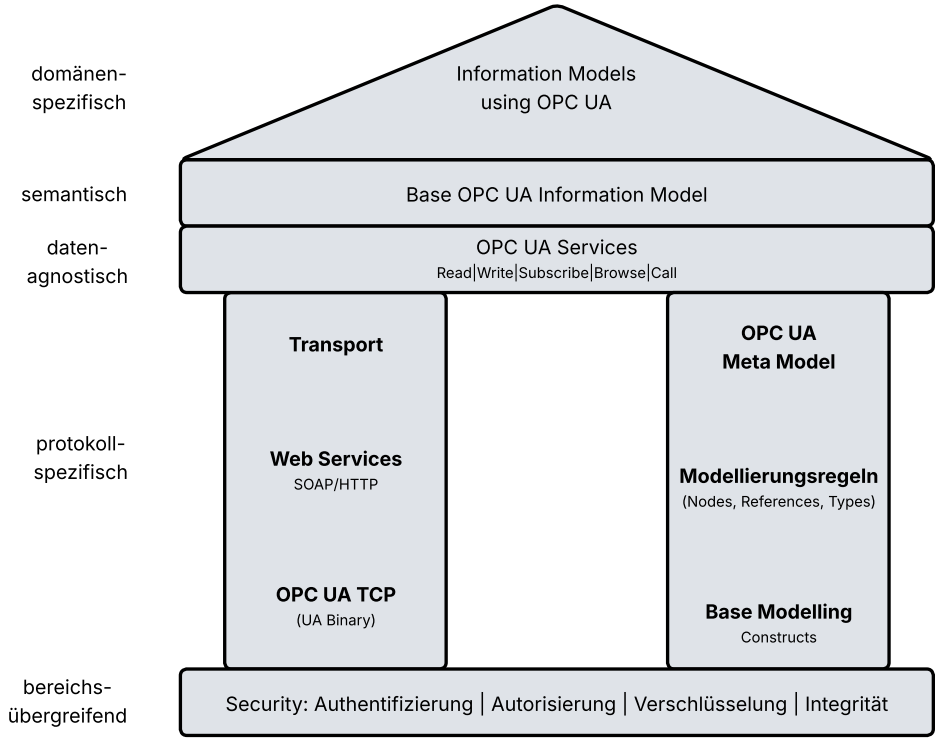

Abbildung 2.1: Grundlagen der OPC UA-Architektur (eigene Darstellung nach [29, S. 30])

Im unteren Bereich der Darstellung sind die Transportmechanismen verortet, welche die physische Übertragung von Nachrichten zwischen verteilten Systemen ermöglichen [29]. Diese Ebene ist bewusst von den darüberliegenden Architekturelementen getrennt und erlaubt unterschiedliche Protokollbindungen, ohne die logische Struktur der Daten zu beeinflussen. Direkt darüber ist die Sicherheitsinfrastruktur angesiedelt, welche grundlegende Mechanismen wie Authentifizierung, Autorisierung, Verschlüsselung und Integritätsprüfung bereitstellt und unabhängig vom konkreten Transport wirkt. Aufbauend auf diesen Ebenen definiert OPC UA eine Service-Schicht, über die Clients mit Servern interagieren. Die Services sind datenagnostisch ausgelegt und bilden die einheitliche Schnittstelle zum Zugriff auf Informationen, etwa zum Lesen, Schreiben oder Abonnieren von Datenstreams [29]. Die oberste Schicht der Architektur bildet die Informationsmodellierung, welche die semantische Beschreibung von Daten, Objekten und deren Beziehungen ermöglicht. Zentrales Element dieser Informationsmodellierung ist ein semantisch strukturierter Adressraum, in dem Daten als Knoten mit definierten Typen, Attributen und Relationen beschrieben werden [29]. Dadurch können nicht nur einfache Messwerte, sondern auch komplexe Strukturen, Zustände und Beziehungen zwischen Objekten abgebildet werden. Diese Form der Modellierung ermöglicht eine einheitliche Interpretation von Daten über System- und Herstellergrenzen hinweg und reduziert den Integrationsaufwand in heterogenen Umgebungen. Der Zugriff auf den Adressraum erfolgt dabei über standardisierte, datenagnostische Services, welche das Transmission-Control-Protocol (TCP) oder webbasierte Protokolle nutzen [29]. Sicherheitsmechanismen sind dabei integraler Bestandteil der Architektur und nicht nachträglich ergänzt [29]. Diese Eigenschaften machen OPC UA zu einem zentralen Baustein für die sichere Integration von Operational Technology (OT) und Information Technology (IT) Systemen.

Zusammenfassend adressieren Event-Streaming-Ansätze und OPC UA unterschiedliche, sich ergänzende Aspekte der Datenintegration. Event Streaming ermöglicht die skalierbare, asynchrone Verarbeitung großer Datenmengen, während OPC UA strukturierte und semantisch beschreibbare Daten aus industriellen Systemen bereitstellt. Die Kombination beider Ansätze schafft eine Grundlage für moderne datengetriebene Architekturen, in denen industrielle Daten kontextualisiert erfasst und anschließend effizient weiterverarbeitet werden können. Messaging-basierte Architekturen lösen dabei die zeitliche und räumliche Kopplung zwischen Systemen, adressieren jedoch nicht die semantische Heterogenität unterschiedlicher Datenmodelle. Für diesen Zweck beschreiben Hohpe und Woolf das *Canonical Data Model* als neutrales Zwischenformat, das heterogene Quell- und Zielstrukturen entkoppelt [14]. Diese Trennung von industrieller Datenerfassung und ereignisbasierter Datenverarbeitung bildet eine zentrale Grundlage für die im weiteren Verlauf dieser Arbeit betrachteten Architekturentscheidungen. Hohpe und Woolf bieten in [15] eine umfassende Behandlung der zugrundeliegenden Integrationsmuster.

Ein wiederkehrendes Prinzip datenintensiver Systeme ist die gestufte Veredelung von Rohdaten: Anstatt Quelldaten in einem einzelnen Schritt zu transformieren, durchlaufen sie mehrere, klar abgegrenzte Verarbeitungsstufen mit jeweils steigender Strukturierung und fachlicher Anreicherung [24]. Databricks beschreibt dieses Vorgehen als schrittweise Verbesserung von Struktur und Qualität der Daten beim Durchlaufen aufeinander aufbauender Verarbeitungsschichten [5]. In der industriellen Praxis hat sich für dieses Prinzip die Bezeichnung *Medaillen-Architektur* (engl. Medallion Architecture) etabliert, die Daten in drei aufeinander aufbauenden Schichten organisiert: Eine *Bronze*-Schicht nimmt Rohdaten unverändert und append-only auf, eine *Silver*-Schicht führt Bereinigung, Filterung und Normalisierung durch, und eine *Gold*-Schicht stellt kuratierte, geschäftslogisch angereicherte Daten für nachgelagerte Konsumenten bereit. Das Muster überträgt damit etablierte Prinzipien der gestuften Datenverarbeitung auf moderne, ereignisbasierte

Architekturen und bildet eine geeignete Strukturierungsgrundlage für die Integration heterogener Quelldaten in einem einheitlichen Datenvertrag. Kleppmann behandelt diese und verwandte Konzepte datenintensiver Architekturen eingehend in [24].

2.2 Asset Administration Shell (AAS)

Die zunehmende Digitalisierung im Kontext von Industrie 4.0 erfordert standardisierte Ansätze, um Informationen über physische und virtuelle Assets, also eindeutig identifizierbare materielle oder immaterielle Objekte wie Maschinen, Produkte, Software, Daten oder Dienstleistungen, konsistent und interoperabel bereitzustellen [16, 33].

Die Asset Administration Shell (AAS) stellt in diesem Zusammenhang das digitale Abbild eines Assets dar und fungiert als zentrales Element einer Industrie-4.0-Komponente. Sie ermöglicht es, sowohl statische Informationen wie Herstellerangaben oder technische Daten als auch dynamische Betriebs- und Zustandsdaten über den gesamten Lebenszyklus hinweg maschinenlesbar zu verwalten und auszutauschen [16, 33]. Die AAS fungiert als digitale Verwaltungsschicht und dient nicht nur der Speicherung von Daten, sondern auch deren strukturierter Bereitstellung in interoperablen Formaten. Infolgedessen können Assets aktiv in cyber-physischen Produktionssystemen eingebunden werden. Letztere umfassen Produktionsumgebungen, in denen physische Prozesse eng mit vernetzten digitalen Prozessen integriert sind. Dadurch werden Echtzeitüberwachung, Steuerung, Autonomie und Selbstoptimierung ermöglicht [27].

Die Plattform Industrie 4.0 unterscheidet in ihrer Spezifikation zwischen drei Typen der AAS, die den Funktionsumfang und den Grad der Autonomie abbilden [33]:

- **Typ 1 (passiv):** stellt ausschließlich Daten zur Verfügung und liefert Werte nur auf Anfrage. Diese Ausprägung dient primär der standardisierten Informationshaltung ohne eigenes Verhalten.
- **Typ 2 (reaktiv):** erweitert die Funktionalität, indem sie auf externe Anfragen oder Ereignisse reagieren kann, beispielsweise durch die Aktualisierung von Zustandsdaten oder die Bereitstellung von Benachrichtigungen.
- **Typ 3 (proaktiv):** initiiert eigenständig Aktionen und Entscheidungen, etwa durch proaktive Wartungshinweise, Vorschläge zur Prozessoptimierung oder das autonome Anpassen von Parametern. Er bildet damit die höchste Ausprägung ab und nähert sich autonomen Produktionssystemen an.

Diese Typisierung macht deutlich, dass die AAS nicht nur als statische Datenschnittstelle fungiert, sondern zunehmend ein aktiver Teilnehmer innerhalb vernetzter Produktionssysteme ist [33]. Ein schematischer Überblick über die Typen ist in Abbildung 2.2 dargestellt.

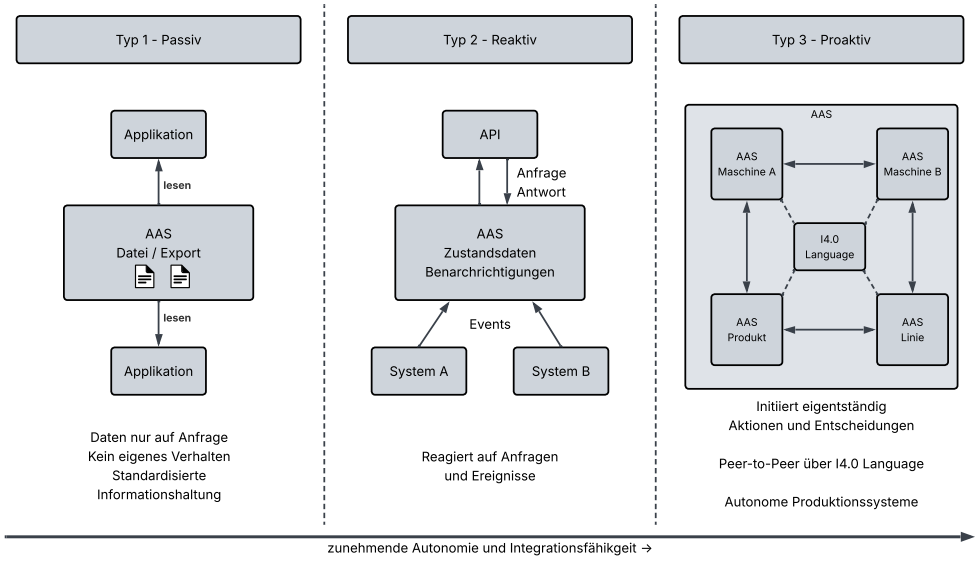

Abbildung 2.2: Unterscheidung der AAS-Typen 1–3 (eigene Darstellung nach [33])

Die Inhalte einer AAS sind modular in sogenannten Submodellen organisiert, die jeweils spezifische Asset-Aspekte beschreiben [33]. Ein zentrales Submodell ist das *Digitale Typenschild (Digital Nameplate)*, das standardisierte Informationen wie Hersteller, Seriennummer, Zertifikate und Version beinhaltet. Weitere Submodelle behandeln technische Eigenschaften (z. B. Leistungsdaten, Material) sowie Betriebs- und Zustandsdaten (z. B. Sensormessungen) und Wartung (z. B. Serviceintervalle, Anleitungen). Darüber hinaus nutzen Submodelle semantische Referenzen auf etablierte Standards wie eCl@ss oder IEC CDD, um Interoperabilität und Eindeutigkeit zu gewährleisten. Neue Bereiche wie Nachhaltigkeit und der digitale Produktpass gewinnen zudem zunehmend an Bedeutung [35, 38].

Die Industrial Digital Twin Association (IDTA) stellt eine wachsende Bibliothek vom Gremium freigegebener Submodell-Templates bereit, die mit dem AAS-Metamodell nach IEC 63278-1 konform und semantisch durch Referenzen auf internationale Klassifikationssysteme wie ECLASS verankert sind [21]. Im Rahmen dieser Arbeit werden drei dieser Templates berücksichtigt, die gemeinsam eine standardkonforme Abbildung von identifizierenden Stammdaten, technischen Geräteparametern und operativen Prozessparametern ermöglichen.

Das *Digital Nameplate for Industrial Equipment* [19] in Version 3.0.1 (Oktober 2025, IDTA 02006) standardisiert die digitale Repräsentation des Typenschilds eines Industriegeräts. Das Template umfasst identifizierende Merkmale wie Herstellername, Seriennummer und Baujahr sowie Kontaktinformationen und Versionsangaben von Hard- und Software. Es bildet die primäre Informationsschicht zur eindeutigen Identifikation eines Assets.

Eine generische Struktur zur Ablage von technischer Produktdaten, welche in die SubCollections *GeneralInformation*, *ProductClassifications* und *TechnicalPropertyAreas* gegliedert ist werden von dem Submodell Template *Generic Frame for Technical Data for Industrial Equipment in Manufacturing* [18] in Version 2.0 (März 2025, IDTA 02003) definiert. Das Template ergänzt das Digital Nameplate um gerätespezifische technische

Parameter und ermöglicht die strukturierte Ablage herstellerbezogener sowie konfigurationsbezogener Eigenschaften.

Das *Process Parameters — Part 1: Type* [20] in Version 1.0 (Juli 2025, IDTA 02031-1) strukturiert Prozessparameter auf Basis eines PPR-Modells (Product–Process–Resource). Es sieht obligatorische Felder für Prozessbezeichnung, Prozessbeschreibung und geplante Prozesszeit sowie Sub-Collections für Prozess-, Ressourcen- und Produktparameter vor. Das Template adressiert operative Betriebsdaten und eignet sich insbesondere für die Abbildung fertigungsbezogener Prozessinformationen.

Weitere IDTA-Templates wie *Contact Information* (IDTA 02002) oder *Hierarchical Structures* (IDTA 02011-1) werden im Rahmen dieser Arbeit nicht berücksichtigt, da die dafür erforderlichen Datenklassen durch die betrachteten Quellsysteme nicht vollständig abgedeckt werden.

Die AAS bietet durch ihre Struktur Grundlagen für Interoperabilität, Automatisierung (Plug-and-Produce), Predictive Maintenance sowie regulatorische Anwendungen wie den digitalen Produktpass [1]. So ermöglicht die AAS im Bereich der Produktionslinien-Konfiguration die automatische Ressourcenplanung und Rekonfiguration modularer Fertigungsumgebungen (Plug-and-Produce), während sie im Kontext von Predictive Maintenance strukturierte Asset-Daten bereitstellt, auf deren Grundlage vorausschauende Wartungsstrategien zur Verlängerung der Anlagenlebensdauer umgesetzt werden können [1]. Damit fungiert die AAS als Brücke zwischen technologischer Infrastruktur und ökologischen bzw. wirtschaftlichen Anforderungen der Industrie.

Die internationale Norm *IEC 63278-1* standardisiert Struktur, Metamodell und Funktionalität der AAS, während die Plattform Industrie 4.0 die Spezifikation kontinuierlich weiterentwickelt [16, 33]. Für die Umsetzung in der Praxis stehen verschiedene Open-Source-Lösungen zur Verfügung, die unterschiedliche Schwerpunkte verfolgen und damit verschiedene Anwendungsszenarien adressieren [1]. In der vorliegenden Arbeit werden drei Implementierungen betrachtet: Eclipse BaSyx, FA³ST und AASX-Server.

Eclipse BaSyx bietet eine vollständige Laufzeitumgebung für Verwaltungsschalen [10]. Dazu gehören das *AAS Environment*, die *AAS Registry*, die *Submodel Registry*, der *AAS Discovery Service* sowie die *AAS Web UI* als browserbasierte Verwaltungsoberfläche. BaSyx ist damit vor allem auf den industriellen Einsatz in komplexen Produktionsumgebungen ausgerichtet.

Im Detail umfasst die BaSyx-Laufzeitumgebung folgende spezialisierte Dienste, die gemeinsam eine modulare und skalierbare Infrastruktur für Verwaltungsschalen bilden.

- *AAS Environment*: bildet den zentralen Datenspeicher der Laufzeitumgebung und stellt zwei Representational State Transfer (REST) -Schnittstellen über den Port 8081 bereit. Über `/shells` werden Verwaltungsschalen verwaltet, über `/submodels` die zugehörigen Submodelle. Beide Schnittstellen unterstützen vollständige Create, Read, Update und Delete (CRUD) -Operationen und können neben einem In-Memory-Speicher auch mit MongoDB als persistentem Backend betrieben werden [9, 10].
- *AAS Registry*: dient als zentrales Verzeichnis für Verwaltungsschalen. Über die Schnittstelle `/shell-descriptors` werden AAS-Deskriptoren mit ihren Identifikatoren und Endpunkten registriert, sodass Clients eine AAS-Instanz systemweit auffinden und über Port 8082 ansprechen können. Als Persistenzbackend wird MongoDB unterstützt [10].
- *Submodel Registry*: stellt analog zur AAS Registry ein dediziertes Verzeichnis für Submodelle auf Port 8083 bereit. Über `/submodel-descriptors` können Submodell-

Deskriptoren registriert und abgefragt werden. Auch dieser Dienst unterstützt MongoDB als persistentes Backend und kann optional um Autorisierung oder Kafka-basiertes Event-Verarbeitung erweitert werden [7, 8].

- *AAS Discovery Service*: ermöglicht die Suche nach Verwaltungsschalen anhand von Asset IDs über die Schnittstelle `/lookup/shells`. Damit werden insbesondere komplexe Anwendungsfälle wie hierarchische Asset-Strukturen oder Stücklisten unterstützt. Zudem wird der Discovery Service über Port 8084 bereitgestellt. [6, 10].
- *AAS Web UI*: stellt eine browserbasierte Oberfläche bereit, über die Verwaltungsschalen, Submodelle und Concept Descriptions interaktiv angezeigt und verwaltet werden können. Die Oberfläche kommuniziert ausschließlich über die REST-Schnittstellen der übrigen BaSyx-Dienste und ist auf Port 3000 erreichbar [10].

Diese Komponenten sind als eigenständige Microservices ausgeführt, können in Docker-Umgebungen betrieben werden und sind über standardisierte REST-Schnittstellen interoperabel nutzbar [10]. Als gemeinsames Persistenz-Backend wird MongoDB eingesetzt, das von AAS Environment, AAS Registry und Submodel Registry genutzt wird. Damit schafft BaSyx eine flexible Grundlage für verteilte Industrie-4.0-Szenarien, bei denen Verwaltungsschalen und Submodelle dynamisch eingebunden, verwaltet und aufgefunden werden können. Abbildung 2.3 veranschaulicht die Komponentenstruktur der BaSyx-Laufzeitumgebung sowie die bereitgestellten Schnittstellen und deren Abhängigkeiten im Überblick.

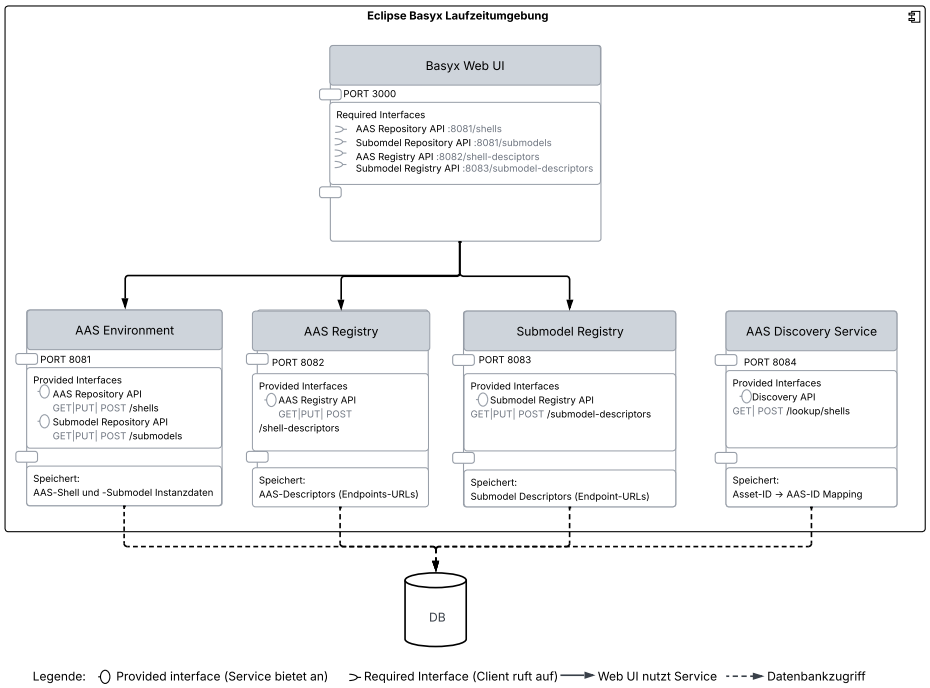

Abbildung 2.3: Komponentenarchitektur der Eclipse-BaSyx-Laufzeitumgebung (eigene Darstellung nach [10])

Das Fraunhofer-Framework *FA³ST* (Fraunhofer Advanced Asset Administration Shell Tools for Digital Twins) verfolgt im Gegensatz dazu einen leichtgewichtigen, modularen Ansatz [36]. Es basiert auf Microservices und ist eng an der aktuellen AAS-Spezifikation

ausgerichtet, was es insbesondere für Forschung, Prototyping und agile Entwicklungsprozesse geeignet macht.

Der *AASX-Server* schließlich stellt eine minimalistische Referenzlaufzeitumgebung dar, die primär auf das Laden und Bereitstellen von AASX-Dateien fokussiert ist [17]. Im Unterschied zu BaSyx oder FA³ST bietet er keine komplexen Framework-Funktionalitäten, sondern ermöglicht eine einfache Bereitstellung von Verwaltungsschalen in Test- und Anwendungsszenarien.

Für die prototypische Umsetzung in dieser Arbeit wird Eclipse BaSyx als Referenzimplementierung ausgewählt, da es die umfassendste Laufzeitumgebung sowie aktiv gepflegte Schnittstellen für die in dieser Arbeit relevanten Integrationsszenarien bereitstellt. FA³ST und der AASX-Server dienen als Vergleichsperspektive: FA³ST wird in Abschnitt 2.4 im Kontext verwandter Systemansätze eingeordnet, während der AASX-Server aufgrund seines minimalistischen Charakters als Referenzpunkt für einfache Bereitstellungsszenarien herangezogen wird.

2.3 Hexagonale Architektur

Die hexagonale Architektur — auch als Ports-and-Adapters-Architektur bekannt — wurde von Alistair Cockburn als konzeptionelle Antwort auf die strukturellen Schwächen klassischer Schichtenarchitekturen entwickelt [4]. Klassische Schichtmodelle trennen zwar technische Ebenen voneinander, erzeugen jedoch häufig eine implizite Abhängigkeit der Domänenlogik von konkreten technischen Implementierungen — etwa von Datenbankzugriffen, UI-Frameworks oder Netzwerkprotokollen. Diese enge Kopplung erschwert Testbarkeit, Austauschbarkeit und langfristige Wartbarkeit von Systemen [4]. Das hexagonale Architekturmuster adressiert diese Problematik, indem es die fachliche Kernlogik als eigenständige, nach außen vollständig isolierte Einheit modelliert.

Grundlegendes Ziel des Musters ist es, die Anwendungslogik von ihrer technischen Umgebung zu entkoppeln, sodass sie gleichwertig von unterschiedlichen Einstiegspunkten gesteuert und aufgerufen werden kann — sei es durch eine Benutzeroberfläche, einen automatisierten Test, ein Skript oder ein externes System [4]. Diese Symmetrie der Zugangswege ist ein zentrales Charakteristikum, das die hexagonale Architektur von hierarchisch geordneten Schichtenmodellen grundlegend unterscheidet.

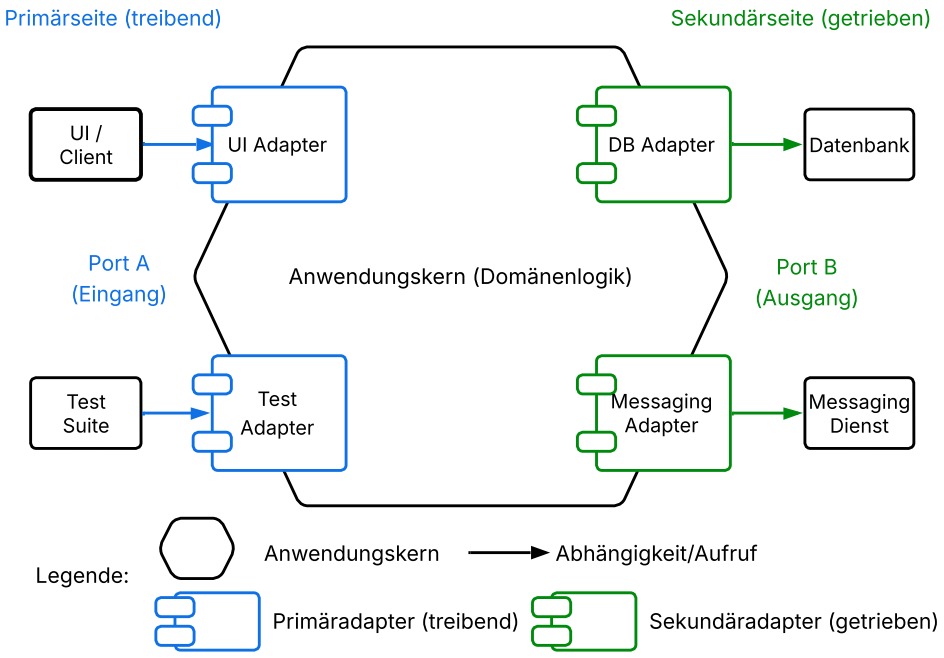

Abbildung 2.4: Schematische Darstellung der hexagonalen Architektur mit Input- und Output-Ports sowie technologiespezifischen Adaptern (nach [4])

Das Muster strukturiert ein System in drei konzeptionelle Bereiche: den Domänenkern, die Ports und die Adapter, wie in Abbildung 2.4 schematisch dargestellt. Der Domänenkern implementiert die fachliche Geschäftslogik und ist vollständig unabhängig von technischen Details. Er kennt weder die konkrete Herkunft eingehender Daten noch die technische Realisierung seiner Ausgaben. Ports definieren als abstrakte Schnittstellen die erlaubten Interaktionspunkte zwischen Domänenkern und Außenwelt. Auf der Eingangsseite nehmen Input-Ports Ereignisse oder Anfragen externer Akteure entgegen; auf der Ausgangsseite ermöglichen Output-Ports dem Kern die Kommunikation mit externen Ressourcen wie Datenbanken, Nachrichtensystemen oder nachgelagerten Diensten. Cockburn bezeichnet diese beiden Seiten konzeptionell als primäre Ports (treibend, von außen angestoßen) und sekundäre Ports (getrieben, vom Kern initiiert) [4]. Adapter schließlich realisieren diese Ports technologiespezifisch: Sie übersetzen zwischen dem abstrakten Port-Protokoll und dem konkreten Kommunikationsformat des jeweiligen externen Systems [4].

Diese strukturelle Trennung hat weitreichende Konsequenzen für die Qualitätseigenschaften eines Systems. Da der Domänenkern ausschließlich über abstrakte Ports mit seiner Umgebung interagiert, lassen sich technologiespezifische Adapter jederzeit austauschen, ohne die fachliche Logik zu berühren. Ebenso kann der Kern vollständig in Isolation getestet werden, indem reale Adapter durch Testadapter (Stubs, Mocks) ersetzt werden [4]. Diese Eigenschaft ist insbesondere in komplexen Integrationsszenarien von hoher praktischer Bedeutung, da sie eine schrittweise Entwicklung und unabhängige Validierung einzelner Systembestandteile ermöglicht. Eine systematische Behandlung von Qualitätsattributen in Softwarearchitekturen bietet [2].

Die hexagonale Architektur gilt als konzeptioneller Vorläufer späterer Architekturansät-

ze wie der Onion-Architektur [32] und der Clean Architecture [30]. Allen diesen Mustern ist das gemeinsame Prinzip der Abhängigkeitsinversion gemein: Technische Details hängen von fachlichen Abstraktionen ab, nicht umgekehrt [30]. Die hexagonale Architektur formuliert dieses Prinzip erstmals in einer explizit symmetrischen Form, die keine privilegierte Eingangsseite kennt und damit besonders gut für Systeme geeignet ist, die über mehrere gleichwertige Integrationspunkte verfügen [4].

Für die vorliegende Arbeit ist das Muster aus zwei Gründen von besonderer Relevanz. Erstens operiert das entwickelte System an der Schnittstelle mehrerer heterogener Quellsysteme mit unterschiedlichen Protokollen (REST, OPC UA, Kafka), die als primäre Adapter modelliert werden können, ohne die interne Verarbeitungslogik zu beeinflussen. Zweitens erlaubt die strikte Trennung zwischen Integrationskern und technologiespezifischer Anbindung eine gezielte Lokalisierung von Änderungsverantwortung — ein zentrales Ziel der in dieser Arbeit konzipierten Architektur. Auf die konkrete Übertragung des Musters auf Systemintegrationsebene wird in Kapitel 4.5 eingegangen.

2.4 Verwandte Arbeiten und bestehende Systemansätze

Die vorliegende Arbeit bewegt sich an der Schnittstelle der beiden technischen Forschungsfelder, welche die standardisierte Repräsentation und Integration von Asset-Daten mittels Asset Administration Shell und die strukturierte Gestaltung von Integrationsarchitekturen unter Anwendung etablierter Architekturmuster behandeln. Der nachfolgende Abschnitt ordnet den entwickelten Systemansatz in den bestehenden Forschungsstand ein und beleuchtet, wo bestehende Arbeiten ansetzen, welche Aspekte sie nicht adressieren und welcher Beitrag im Rahmen dieser Arbeit neu geleistet wird.

Im Bereich der AAS-Architektur haben Schnicke u. a. einen grundlegenden Beitrag zur Strukturierung von Integrationsszenarien geleistet [34]. Die Autoren entwickeln wiederverwendbare Architekturschablonen, sogenannte AAS-Blueprints, welche die standardisierte Anbindung heterogener Quellsysteme an eine AAS-Infrastruktur beschreiben. Dabei identifizieren sie wiederkehrende Muster in der Schnittstellengestaltung und zeigen, dass sich typische Integrationsflüsse auf eine begrenzte Anzahl generischer Strukturen zurückführen lassen. Der Ansatz eignet sich insbesondere als konzeptionelle Grundlage für die Planung neuer AAS-Integrationen, adressiert jedoch nicht die technische Realisierung einer mehrstufigen Datenverarbeitungspipeline zwischen Quellsystem und AAS-Zielschicht. Fragen der Datenstabilisierung, der Schemaentkopplung oder der schrittweisen Konsolidierung über mehrere Verarbeitungsebenen bleiben dabei ausgeklammert. Die ereignisbasierte Verarbeitung wird im Kontext des Use-Cases UC.2 (hochfrequente Datenquellen via OPC UA-Subscription) skizziert, beschränkt sich jedoch auf eine anwendungsfallspezifische Architekturvorgabe. Eine systemunabhängig wiederverwendbare, generische Ereignispipeline wird nicht ausgearbeitet, was dazu führt, dass die Dimension damit nur bedingt adressiert wird, und somit nicht als allgemeingültiges Architekturmerkmal realisiert wird.

Eine technisch orientierte Perspektive nehmen Jacoby u. a. mit dem FA^{3}ST Service ein, einer reaktiven Open-Source-Referenzimplementierung des AAS-Standards [23]. Der Beitrag beschreibt eine offene, durch Schnittstellen modularisierte Architektur, die über eine *AssetConnection*-Schicht die Synchronisation des digitalen Zwillings mit physischen Assets über verschiedene Protokolle wie OPC UA, Hypertext Transfer Protocol (HTTP) und Message Queuing Telemetry Transport (MQTT) ermöglicht. Der Fokus liegt dabei konsequent auf der AAS-seitigen Infrastruktur.Allerdings bleiben die Fragen, wie Daten aus heterogenen Quellsystemen integriert, über welche Transformations-

stufen sie aufbereitet und in welcher Architektur die vorgelagerte Integrationsschicht gestaltet wird, explizit ausgeklammert. FA^{3}ST setzt damit die Existenz konsolidierter, AAS-kompatibler Eingangsdaten voraus, ohne deren Herleitung zu adressieren. Die *AssetConnection*-Schicht ermöglicht eine protokollspezifische Wertzuordnung zwischen Submodell-Elementen und Asset-Datenpunkten; sie stellt jedoch kein eigenständiges, konfigurierbares Transformationsartefakt dar, das heterogene Quelldaten systematisch auf IDTA-konforme Submodell-Strukturen abbildet. Die in der vorliegenden Arbeit erörterte Transformationsverantwortung ist in der Verbindungskonfiguration implizit enthalten und nicht als separates, austauschbares Artefakt externalisiert. Damit ist das Merkmal nur bedingt erfüllt.

Fischer u. a. adressieren eine strukturelle Lücke der AAS-Spezifikation: Da Verwaltungsschalen standardmäßig nur den aktuellen Datenzustand vorhalten, entwickeln die Autoren Architekturschablonen für die persistente Speicherung historischer Daten im AAS-Kontext [11]. Die vorgestellten Blueprints erweitern bestehende AAS-Infrastrukturen um eine *HistoryStorage*-Komponente und nutzen Eclipse BaSyx als Referenzimplementierung. Die Arbeit leistet damit einen Beitrag zur Datenhaltungsschicht des AAS-Ökosystems, adressiert jedoch ausschließlich die Speicherung nachgelagerter Daten. Fragen der Datenintegration aus heterogenen Quellsystemen, der schichtenweisen Vorverarbeitung sowie der semantischen Abbildung auf IDTA-konforme Submodelle liegen außerhalb des Untersuchungsrahmens. Die ereignisbasierte Verarbeitung ist insofern bedingt adressiert, als Blueprint 2 ein bestehendes Eventing-System der AAS-Infrastruktur als Speicherauslöser nutzt. Dieses dient jedoch ausschließlich der internen Benachrichtigung bei Submodell-Änderungen und sieht die Integration externer, heterogener Ereignisquellen als Datenlieferanten nicht vor.

Eine umfassende Einordnung des gesamten AAS-Ökosystems nehmen Alexopoulos u. a. in einer strukturierten Übersichtsarbeit vor [1]. Die Autoren analysieren Motivationslagen, typische Implementierungshürden und bestehende Systemlösungen und identifizieren dabei explizit offene Forschungsfragen. Unter anderem wird festgestellt, dass ereignisgesteuerte Anbindungskonzepte zunehmend als geeignete Architekturvariante diskutiert werden, in der Praxis jedoch selten mit einer vollständig strukturierten Datenpipeline verbunden sind. Diese Beobachtung deckt sich mit der Ausgangsproblematik der vorliegenden Arbeit und unterstreicht die Relevanz eines durchgängigen Architekturansatzes, der Datenintegration und Semantische Modellierung für die AAS gemeinsam adressiert. Da es sich um eine Übersichtsarbeit handelt, verbleiben sowohl die Heterogenitätsbehandlung als auch die ereignisbasierte Verarbeitung auf der Ebene einer Problemcharakterisierung. Dabei wird Heterogenität als zentrales Implementierungshindernis identifiziert, aber keine konkrete Lösungsarchitektur erarbeitet. Zudem werden zwar ereignisbasierte Konzepte als vielversprechende Richtung diskutiert, jedoch wird dort ebenfalls keine architektonische Ausprägung spezifiziert. Die beiden Dimensionen werden demnach adressiert, wobei dies lediglich auf der konzeptionellen Ebene erfolgt. Eine realisierte Systemlösung ist nicht gegeben.

Im Bereich ereignisgesteuerter AAS-Architekturen wird die Implementierung dezentraler, modularer Agenten mit ereignisbasierter Kommunikation im Kontext einer Smart Factory demonstriert [3]. Die Autoren implementieren mittels des BaSyx Python SDK eine Agent-Server-Architektur, die OPC-UA-basierte Ereigniskommunikation mit einer zentralen AAS-Repository-Anbindung verbindet. Dadurch wird eine dezentrale Steuerung von Produktionsmodulen ermöglicht. Die vorliegende Arbeit bildet somit den Prototyp einer Grundlage für modulare, ereignisbasierte AAS-Implementierungen. Der Fokus der vorliegenden Arbeit liegt nicht auf der architektonischen Strukturierung der Verarbeitungsstufen zwischen Ereigniserzeugung und AAS-Schreibvorgang, etwa in Form

gestufter Datentransformations- und Konsolidierungsschichten. Der Übergang von den ursprünglichen Rohdaten zu einem für die AAS-Semantik geeigneten Datenstand erfolgt nicht in eigenständigen Verarbeitungsschichten.

Guo u. a. vergleichen drei bestehende Konzepte zur Datenintegration in die Asset Administration Shell und bewerten diese anhand von zwölf Kriterien [12]. Die untersuchten Ansätze umfassen das BaSyx Python SDK, die BaSyx DataBridge sowie den deskriptiven Ansatz über Asset Interfaces Description (AID) und Asset Interfaces Mapping Configuration (AIMC) als standardisierte Submodelle. Die Autoren empfehlen einen synergistischen Ansatz, der die Stärken der drei Konzepte verbindet. Der Beitrag liefert damit eine strukturierte Bewertungsgrundlage auf Werkzeugebene. Eine übergreifende Integrationsarchitektur mit gestufter Datenverarbeitungspipeline zwischen heterogenen Quellsystemen und der AAS wird nicht konzipiert. Die semantische Modellierung ist insofern bedingt adressiert, als die AAS als Zielsystem vorausgesetzt wird. Die inhaltliche Befüllung IDTA-konformer Submodelle mit semantischer Annotation liegt jedoch außerhalb des Untersuchungsrahmens. Die Heterogenitätsbehandlung wird nicht explizit thematisiert, da die Analyse auf der Ebene der AAS-nahen Integrationsschnittstelle ansetzt und nicht auf der Ebene strukturell verschiedenartiger Quellsysteme wie ERP, TDM oder OPC UA. Eine ereignisbasierte Verarbeitungskette wird nicht beschrieben. Die architektonische Entkopplung wird nicht als Gestaltungsziel formuliert. Ein explizites Transformationsartefakt mit definierten Mapping-Regeln und Schnittstellenvertrag zur IDTA-konformen Submodell-Generierung ist nicht Gegenstand der Arbeit.

Kristan u. a. systematisieren die Integration von IoT-Backends in AAS-Infrastrukturen anhand des Katalogs der Enterprise Integration Patterns (EIP) [26]. EIP bezeichnen eine etablierte Sammlung von Entwurfsmustern für die nachrichtenbasierte Systemintegration. Auf dieser Grundlage leiten die Autoren vier Integrationsansätze ab, nämlich *Push with Adapter*, *Push with Bridge*, *Pull with Wrapper* und *Pull with Bridge*. Diese Ansätze werden hinsichtlich Engineering-Aufwand, Betriebskosten, Datenkonsistenz und Skalierbarkeit evaluiert. Die Autoren empfehlen *Pull with Bridge* als in den meisten Szenarien geeignetste Lösung. Die Arbeit liefert damit eine strukturierte Taxonomie für Integrationsstrategien im AAS-Kontext. Eine mehrstufige Datenverarbeitungspipeline zwischen Quellsystem und AAS-Zielschicht sowie die semantisch korrekte Erzeugung IDTA-konformer Submodelle liegen außerhalb des Untersuchungsrahmens. Die semantische Modellierung ist insofern bedingt adressiert, als die AAS als Zielsystem vorausgesetzt und deren Anbindung beschrieben wird. Die inhaltliche Befüllung von Submodellen gemäß IDTA-Templates oder deren semantische Annotation mittels `semanticId` wird jedoch nicht thematisiert. Die Heterogenitätsbehandlung liegt außerhalb des Untersuchungsrahmens, da sämtliche vier Integrationsansätze sich ausschließlich auf IoT-Backends als Quelldomäne beziehen. Eine strukturelle Heterogenität zwischen verschiedenartigen Quellsystemen wie ERP, TDM und OPC UA wird nicht adressiert. Die ereignisbasierte Verarbeitung ist bedingt realisiert. Die Push-basierten Ansätze (*Push with Adapter*, *Push with Bridge*) weisen einen ereignisähnlichen Charakter auf, während die Pull-basierten Varianten (*Pull with Wrapper*, *Pull with Bridge*) als Polling-Mechanismus konzipiert sind. Eine durchgängig ereignisgesteuerte Datenverarbeitungspipeline wird nicht beschrieben. Die architektonische Entkopplung ist bedingt realisiert. Die Bridge-Komponente trennt IoT-Backend und AAS-Infrastruktur auf Musterebene. Eine systemübergreifende Entkopplung mit persistiertem Datenvertrag, etwa durch eine Messaging-Plattform, wird nicht konzipiert. Das explizite Transformationsartefakt ist bedingt vorhanden. Der *Message Translator* wird als EIP-Entwurfsmuster diskutiert und seine Platzierung analysiert. Ein konkretes, konfigurierbares Artefakt mit definiertem Schnittstellenvertrag zur IDTA-konformen Submodell-Generierung wird jedoch nicht spezifiziert.

Die analysierten Beiträge adressieren jeweils relevante Teilaspekte des in dieser Arbeit verfolgten Ziels, lassen jedoch in ihrer Kombination eine Lücke erkennen. Tabelle 2.1 stellt die betrachteten Arbeiten strukturiert gegenüber und verdeutlicht, welche Merkmalsdimensionen jeweils abgedeckt werden.

Tabelle 2.1: Einordnung verwandter Arbeiten nach zentralen Merkmalsdimensionen

Systemansatz	**Semantische Modellierung**	**Heterogenitätsbehandlung**	**Ereignisbasierte Verarbeitung**	**Architektonische Entkopplung**	**Explizites Transformationsartefakt**
AAS-Blueprints [34]	✓	✓	bedingt	–	–
FA^{3}ST Service [23]	✓	–	–	–	bedingt
Hist. Datenspeicher AAS [11]	✓	–	bedingt	–	–
AAS-Übersichtsstudie [1]	✓	bedingt	bedingt	–	–
AAS-Agenten (dezentral) [3]	✓	–	✓	–	–
AAS-Integrationskonzepte [12]	bedingt	–	–	–	bedingt
IoT-Backends in der AAS [26]	bedingt	–	bedingt	bedingt	bedingt
Asset Management Portal (diese Arbeit)	✓	✓	✓	✓	✓

Wie die Gegenüberstellung zeigt, fehlt in der gesichteten Literatur eine Arbeit, die semantische Modellierung, Heterogenitätsbehandlung, ereignisbasierte Verarbeitung, architektonische Entkopplung und ein explizites Transformationsartefakt gemeinsam konzipiert und prototypisch umsetzt. Die vorliegende Arbeit leistet diesen Beitrag, indem sie eine durchgängige Systemarchitektur entwickelt, die von der heterogenen Datenpersistenz über die strukturierte Konsolidierung bis zur semantisch korrekten Erzeugung standardkonformer Verwaltungsschalen reicht. Der Mapping-Service nimmt dabei die Rolle des zentralen Architekturartefakts ein, das beide Schichten miteinander verbindet und die semantische Verantwortung explizit lokalisiert.

Kapitel 3

Problem- und Anforderungsanalyse

Dieses Kapitel beschreibt die konkrete Ausgangssituation des betrachteten Maschinenbauunternehmens und leitet daraus systematisch Anforderungen und Architekturziele ab. Zunächst wird die bestehende Systemlandschaft analysiert (Abschnitt 3.1). Darauf aufbauend wird das zugrunde liegende Design-Problem formuliert (Abschnitt 3.2), aus dem funktionale (Abschnitt 3.3) und nicht-funktionale Anforderungen (Abschnitt 3.4) abgeleitet werden. Abschließend werden übergeordnete Architekturziele definiert (Abschnitt 3.5), die als Leitlinien für die nachfolgende Architekturkonzeption dienen.

3.1 Analyse der Ist-Situation

Die bestehende Systemlandschaft des betrachteten Maschinenbauunternehmens ist durch eine hohe Heterogenität gekennzeichnet. Im Produktionsumfeld kommen verschiedene Systeme parallel zum Einsatz, die jeweils spezifische Aufgaben innerhalb des Produktlebenszyklus erfüllen. Dazu zählen unter anderem ein CAD/CAM-System, TDM-System, ERP-System, MES sowie die Maschinensteuerungen selbst. Diese Systeme verfügen jeweils über eigene Datenmodelle, Schnittstellen und Integrationsmechanismen. Eine durchgängige und konsistente Integration der Daten ist dadurch nur eingeschränkt möglich. Insbesondere die fehlende semantische Abstimmung zwischen den Systemen erschwert eine einheitliche Sicht auf produktionsrelevante Assets wie Maschinen oder Werkzeuge.

Die eingesetzten Schnittstellentechnologien verdeutlichen zusätzlich die Komplexität der bestehenden Architektur. Während einzelne Systeme moderne REST-basierte Schnittstellen bereitstellen, erfolgt der Datenaustausch in anderen Fällen weiterhin über File-Shares oder proprietäre Herstellerschnittstellen. Ergänzend existieren Excel-basierte Zwischenlösungen, die Integrationslücken überbrücken, jedoch gleichzeitig Redundanzen und Inkonsistenzen begünstigen. Für das ERP-System stehen beispielsweise Open-Data-Protocol-Schnittstellen sowie Remote Function Calls zur Verfügung, während Maschinen aktuell überwiegend über File-Shares angebunden sind und nur teilweise proprietäre APIs nutzen. Perspektivisch ist vorgesehen, diese Anbindungen durch standardisierte Mechanismen zu ersetzen, um eine strukturierte und interoperable Maschinenintegration zu ermöglichen.

Die Koexistenz unterschiedlicher Integrationsparadigmen, wie synchrone Abfragen, dateibasierter Austausch und punktuelle ereignisbasierte Kommunikation, führt zu erhöhtem Integrationsaufwand sowie zu eingeschränkter Transparenz über den aktuellen Zustand produktionsrelevanter Assets. Daten liegen verteilt in unterschiedlichen Systemen vor, unterscheiden sich hinsichtlich Struktur und Aktualität und sind nur mit erheblichem Transformationsaufwand konsolidierbar. Insbesondere dynamische Zustandsdaten

aus dem Produktionsumfeld stehen häufig nicht in einem konsistenten Zusammenhang mit den zugehörigen Stammdaten.

Ein zentrales Problem besteht somit nicht ausschließlich in der technischen Integration einzelner Systeme, sondern in der fehlenden übergreifenden Architektur zur konsistenten Repräsentation und Bereitstellung von Asset-Informationen. Die derzeitige Systemlandschaft begünstigt punktuelle, systemspezifische Integrationslösungen, ohne eine stabile, standardisierte und skalierbare Gesamtstruktur zur Verfügung zu stellen. Obwohl die dargestellte Situation konkret auf das betrachtete Unternehmen bezogen ist, entspricht sie strukturell einer typischen Ausgangslage produzierender Unternehmen mit einer über einen längeren Zeitraum entstandenen IT- und OT-Systemlandschaft. Die Kombination aus heterogenen Datenquellen, proprietären Schnittstellen, unterschiedlichen Änderungsfrequenzen und fehlender semantischer Harmonisierung stellt eine generische Integrationsherausforderung dar, die über den Einzelfall hinaus Relevanz besitzt. Aus dieser Konstellation ergibt sich die Notwendigkeit, das zugrunde liegende Design-Problem zu formulieren, das als Grundlage für die Ableitung konsistenter Architekturentscheidungen dient.

3.2 Formulierung des Design-Problems

In der industriellen Praxis existieren typischerweise mehrere spezialisierte Systeme, die unterschiedliche Phasen des Produktlebenszyklus unterstützen. Diese Systeme werden häufig unabhängig voneinander eingeführt und weiterentwickelt. Eine übergreifende Architektur zur konsistenten Repräsentation und Bereitstellung von Asset-Informationen ist dabei oftmals nicht vorhanden. Stattdessen entstehen inkrementelle Integrationslösungen, die einzelne Systeme miteinander koppeln, jedoch keine langfristig stabile und skalierbare Gesamtstruktur gewährleisten.

Mit zunehmender Digitalisierung steigen zugleich die Anforderungen an Transparenz, Nachvollziehbarkeit und Interoperabilität. Produktionsrelevante Assets wie Maschinen, Werkzeuge oder Fertigungsaufträge sollen systemübergreifend eindeutig identifizierbar sein und in ihrem aktuellen Zustand konsistent interpretiert werden können. Dies erfordert nicht nur eine technische Anbindung der beteiligten Systeme, sondern eine architektonische Lösung, die Datenintegration, Entkopplung, Standardisierung und semantische Konsistenz gleichermaßen berücksichtigt. Der Begriff der Architektur wird in dieser Arbeit gemäß ISO/IEC/IEEE 42010 verstanden als *"fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution"* [22].Vor diesem Hintergrund lässt sich das zugrundeliegende Design-Problem wie folgt definieren.

Diese Arbeit folgt der Design-Science-Research-Methodologie nach Wieringa [37]. Der Ausgangspunkt der vorliegenden Arbeit ist folgendes Design-Problem, das einerseits praktisch motiviert ist durch die konkrete Systemsituation in industriellen Fertigungsumgebungen, andererseits aber auch wissenschaftlich relevant ist, da vergleichbare Integrationsarchitekturen in der Literatur bislang nur partiell adressiert werden.

> **Design-Problem**. In industriellen Produktionsumgebungen mit historisch gewachsenen, heterogenen IT- und OT-Systemlandschaften liegen Asset-Zustandsdaten entweder in systemspezifischen, miteinander inkompatiblen Strukturen vor oder werden in zu stark abstrahierter Form dargestellt und verlieren dadurch fachliche Präzision. Dadurch wird die systemübergreifende

Abfrage von Asset-Zustandsdaten für Monitoring- und Steuerungszwecke erheblich erschwert. Ziel ist daher der Entwurf einer Systemarchitektur zur Datenintegration, die eine interoperable, skalierbare und semantisch konsistente Repräsentation von Asset-Zuständen ermöglicht um damit eine effiziente und konsistente Abfrage von Asset-Zustandsdaten zu unterstützen.

Um das vorliegende Design-Problem zu adressieren wird eine Systemarchitektur als Artefakt entwickelt und deren Funktionalität prototypisch demonstriert und evaluiert. Die nachfolgenden funktionalen und nicht-funktionalen Anforderungen konkretisieren die notwendigen Systemfähigkeiten und Qualitätsmerkmale, die zur Lösung des formulierten Design-Poblems erforderlich sind.

3.3 Funktionale Anforderungen

Aus dem in Abschnitt 3.2 formulierten Design-Problem ergeben sich konkrete funktionale Anforderungen an die zu entwickelnde Systemarchitektur sowie an den darauf aufbauenden Prototypen. Diese Anforderungen beschreiben notwendige Systemfähigkeiten, die zur Adressierung des identifizierten Design-Problems erforderlich sind. Der Schwerpunkt liegt dabei nicht auf einer vollständigen Integration sämtlicher bestehender Unternehmenssysteme, sondern auf der prototypischen Realisierung zentraler architektonischer Mechanismen. Ziel ist es, die grundsätzliche Eignung der Architektur zur Lösung des abstrahierten Integrationsproblems nachzuweisen.

Die funktionalen Anforderungen werden in Tabelle 3.1 zusammengefasst. Sie adressieren insbesondere die standardisierte Anbindung heterogener Datenquellen, die konsistente Repräsentation von Asset-Informationen sowie die ereignisbasierte Verarbeitung und Weitergabe von Zustandsänderungen.

Tabelle 3.1: Funktionale Anforderungen

ID	Beschreibung
FA1	Prototypische Anbindung mindestens einer Maschine über einen standardisierten Kommunikationsmechanismus (z. B. OPC UA) zur Demonstration einer interoperablen Maschinenintegration.
FA2	Konsolidierte Erfassung und Harmonisierung von produktionsrelevanten Asset-Daten aus unterschiedlichen Quellsystemen in einer konsistenten, domänennahen Repräsentation.
FA3	Erkennung, Verarbeitung und Weiterleitung relevanter Datenänderungen zur Entkopplung von Datenquellen und Datenkonsumenten. Datenänderungen sollen durch die Quellsysteme signalisiert werden können.
FA4	Bereitstellung einer standardisierten, semantisch strukturierten Repräsentation von Assets auf Basis definierter Informationsmodelle (z. B. IDTA-Submodell-Templates).
FA5	Das System muss sowohl semantisch stabile Asset-Zustandsdaten als auch hochfrequente Telemetriedaten verarbeiten und bereitstellen können. Die Aktualisierungsfrequenz einzelner Datenklassen muss unabhängig steuerbar sein.

Die formulierten funktionalen Anforderungen beschreiben damit keine technologiespezifischen Implementierungsdetails, sondern notwendige Systemfähigkeiten, die aus dem abstrahierten Integrationsproblem resultieren. Die konkrete technologische Ausprägung dieser Anforderungen erfolgt im Rahmen der Architekturkonzeption und wird durch die dokumentierten Architekturentscheidungen nachvollziehbar begründet.

3.4 Nicht-funktionale Anforderungen

Neben den funktionalen Systemfähigkeiten ergeben sich aus dem formulierten Design-Problem sowie aus den organisatorischen und technischen Rahmenbedingungen zentrale nicht-funktionale Anforderungen. Diese betreffen qualitative Eigenschaften der Architektur und bestimmen maßgeblich deren langfristige Eignung im industriellen Einsatz.

Während die funktionalen Anforderungen beschreiben, *was* das System leisten muss, konkretisieren die nicht-funktionalen Anforderungen, *wie* diese Leistungen architektonisch realisiert werden sollen. Sie bilden damit wesentliche Bewertungskriterien für die im weiteren Verlauf getroffenen Architekturentscheidungen. Die nicht-funktionalen Anforderungen sind in Tabelle 3.2 strukturiert dargestellt.

Tabelle 3.2: Nicht-funktionale Anforderungen

ID	Kategorie	Beschreibung
NFA1	Skalierbarkeit	Die Architektur muss in der Lage sein, steigende Datenmengen – insbesondere eine wachsende Anzahl von Assets, Informationsmodellen und Ereignissen – ohne grundlegende strukturelle Änderungen zu verarbeiten.
NFA2	Performanz und Aktualität	Die Architektur soll so ausgelegt sein, dass eine spätere Erweiterung hin zu echtzeitnaher Verarbeitung möglich ist, ohne dass zentrale Komponenten grundlegend neu gestaltet werden müssen.
NFA3	Interoperabilität	Es sind offene und standardisierte Schnittstellen zu verwenden, um die Integration heterogener Systeme zu erleichtern und zukünftige Erweiterungen zu unterstützen.
NFA4	Wartbarkeit und Modularität	Die Architektur ist modular zu strukturieren, sodass einzelne Komponenten unabhängig weiterentwickelt, ausgetauscht oder erweitert werden können, ohne die Gesamtstruktur zu destabilisieren.
NFA5	Erweiterbarkeit	Die Architektur muss es ermöglichen, zusätzliche Quellsysteme, neue Datenkonsumenten sowie weitere Asset-Klassen anzubinden, ohne bestehende Komponenten oder Verarbeitungslogiken grundlegend ändern zu müssen.

Die nicht-funktionalen Anforderungen beziehen sich somit primär auf die Qualitätsmerkmale Skalierbarkeit, Erweiterbarkeit, Interoperabilität und Wartbarkeit. Gleichzeitig wird berücksichtigt, dass Aspekte wie Performanz und Latenz vor allem im Hinblick auf eine spätere Erweiterung der Lösung relevant sind. Gemeinsam mit den funktionalen Anforderungen bilden diese Qualitätsziele den Referenzrahmen für die nachfolgend entwickelte Architekturkonzeption und die dokumentierten Architekturentscheidungen.

3.5 Abgeleitete Architekturziele

Aus dem in Abschnitt 3.2 formulierten Design-Problem sowie den daraus abgeleiteten funktionalen und nicht-funktionalen Anforderungen ergeben sich übergeordnete Architekturziele. Diese Ziele konkretisieren die angestrebten strukturellen Eigenschaften der Systemarchitektur und dienen als Leitlinien für die im folgenden Kapitel getroffenen Architekturentscheidungen.

Während die Anforderungen notwendige Systemfähigkeiten und Qualitätsmerkmale beschreiben, formulieren die Architekturziele Gestaltungsprinzipien, welche die langfristige Stabilität, Erweiterbarkeit und Kohärenz der Lösung sicherstellen sollen. Die abgeleiteten Architekturziele sind in Tabelle 3.3 zusammengefasst.

Tabelle 3.3: Abgeleitete Architekturziele

ID	**Architekturziel**
AZ1	Etablierung einer konsistenten und zentral auswertbaren Repräsentation von Maschinen- und Asset-Zuständen zur Erhöhung der Transparenz über produktionsrelevante Informationen.
AZ2	Reduktion der Schnittstellenheterogenität durch den Einsatz standardisierter Integrations- und Kommunikationsmechanismen.
AZ3	Entkopplung von Datenquellen und Datenkonsumenten durch den Einsatz ereignisbasierter Architekturmuster zur Unterstützung von Skalierbarkeit, Erweiterbarkeit und Robustheit.
AZ4	Klare Trennung von Datenintegration und semantischer Modellierung zur Sicherstellung von Modularität und Wiederverwendbarkeit der Integrationslogik.
AZ5	Sicherstellung von Erweiterbarkeit und Evolvierbarkeit der Architektur als Grundlage für eine schrittweise Skalierung.

Die formulierten Architekturziele verdeutlichen, dass die angestrebte Lösung nicht lediglich auf die Integration einzelner Systeme abzielt, sondern auf die Gestaltung einer strukturell tragfähigen Integrationsarchitektur. Transparenz über Asset-Zustände wird dabei als primäres fachliches Ziel verstanden, während Standardisierung, Entkopplung und Modularität als notwendige architektonische Enabler fungieren.

Die Architekturkonzeption in Kapitel 4 operationalisiert diese Ziele durch konkrete Strukturentscheidungen, deren Begründung und Bewertung jeweils in Form eines Architecture Decision Record (ADR) dokumentiert wird.

Kapitel 4

Architekturkonzeption

Aufbauend auf dem in Kapitel 3 formulierten Design-Problem, den daraus abgeleiteten funktionalen und nicht-funktionalen Anforderungen sowie den definierten Architekturzielen beschreibt dieses Kapitel die Konzeption der Systemarchitektur für ein Asset Management Portal.

Das Asset Management Portal wird dabei nicht ausschließlich als grafische Benutzeroberfläche verstanden, sondern als ganzheitliches System zur konsistenten Erfassung, Harmonisierung, semantischen Modellierung und standardisierten Bereitstellung von Asset-Informationen. Die Benutzeroberfläche stellt lediglich eine Zugriffsschicht auf eine darunterliegende Integrations- und Dateninfrastruktur dar, welche die eigentliche Grundlage für Transparenz, Interoperabilität und Skalierbarkeit bildet.

Ziel der Architekturkonzeption ist es, eine strukturell tragfähige und erweiterbare Lösung zu entwickeln, die die identifizierte Problemklasse heterogener IT- und OT-Landschaften adressiert und zugleich als belastbare Grundlage für eine spätere produktive Nutzung in einem global agierenden Maschinenbauunternehmen geeignet ist. Die prototypische Umsetzung (siehe Kapitel 5) dient dabei der Validierung der gewählten Strukturentscheidungen und der praktischen Demonstration der konzeptionellen Eignung.

Die entwickelte Gesamtarchitektur des Systems folgt einem zweigeteilten Ansatz. Sie trennt bewusst zwischen einer Integrations- und Dateninfrastruktur zur konsistenten Erfassung und Verarbeitung heterogener Quellsysteme und einer darauf aufbauenden semantischen Verwaltungsschicht zur standardisierten Repräsentation von Assets. Diese Struktur operationalisiert insbesondere die in Kapitel 3 formulierten Architekturziele der Entkopplung, Modularität und Interoperabilität.

Die nachfolgenden Abschnitte beschreiben zunächst die Gesamtarchitektur und deren Systemkontext, anschließend die Vorteile der gewählten Struktur, die Ausgestaltung der Architektur der Integrations- und Dateninfrastruktur sowie die Architektur der Asset Administration Shell. Abschließend werden die zentralen Architekturentscheidungen in Form von Architecture Decision Records (ADRs) dokumentiert.

4.1 Gesamtarchitektur und Systemkontext

Die im Rahmen dieser Arbeit entwickelte Gesamtarchitektur zielt darauf ab, eine skalierbare, erweiterbare und semantisch konsistente Integration von Asset-Daten zu ermöglichen. Hierzu wird eine zweigeteilte Architektur gewählt, die eine klare Trennung zwischen der ereignisbasierten Datenintegration und der AAS-basierten Semantik- und Verwaltungsschicht vorsieht. Diese Trennung stellt sicher, dass unterschiedliche Anforderungen an Datenaufnahme, Verarbeitung und semantischer Modellierung unabhängig voneinander adressiert werden können.

Abbildung 4.1 zeigt den ersten Teil der Gesamtarchitektur, welcher als Integrations- und Datenarchitektur auf Basis eines hexagonalen Architekturansatzes ausgeführt ist. In dieser Schicht werden Daten aus heterogenen Quellsystemen wie Maschinensteuerungen, ERP-, MES-, CAD/CAM- oder PLM-Systemen angebunden. Der Zugriff erfolgt über technologieabhängige Adapter, beispielsweise REST-Schnittstellen, OPC UA oder Ereignis-Streaming über Apache Kafka (im Folgenden: Kafka). Diese Adapter sind über klar definierte Input Ports an den Integrationskern angebunden und ermöglichen eine entkoppelte Aufnahme der eingehenden Daten.

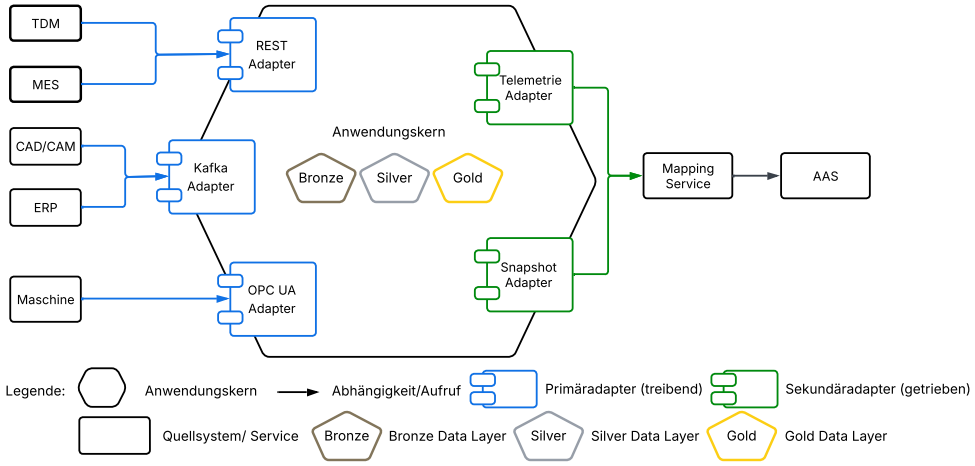

Abbildung 4.1: Hexagonale Daten- und Integrationsarchitektur mit Bronze-, Silver- und Gold-Schicht sowie nachgelagertem Mapping-Service (eigene Darstellung)

Innerhalb des Integrationskerns erfolgt eine domänennahe Verarbeitung der eingehenden Informationen in drei aufeinanderfolgenden Schichten: Die *Bronze-Schicht* persistiert die Rohdaten verlustfrei in ihrer Ursprungsform, die *Silver-Schicht* überführt diese in ein normalisiertes, domänennahes Schema, und die *Gold-Schicht* stellt eine bereinigte, für die Weitergabe optimierte Repräsentation bereit. Die zentrale Verarbeitungslogik ist dabei bewusst AAS-agnostisch ausgelegt und übernimmt keine semantische Interpretation im Sinne der Asset Administration Shell. Stattdessen werden die Daten über klar definierte Output Ports bereitgestellt, wobei logisch zwischen semantisch stabilen Asset-Snapshot-Daten und hochfrequenten Telemetriedaten unterschieden wird. Beide Datenklassen werden über separate Export-Schnittstellen aus dem Integrationskern herausgeführt und als ereignisbasierte Nachrichten an den nachgelagerten Mapping-Service übergeben, der außerhalb des hexagonalen Kerns als eigenständige Komponente operiert.

Abbildung 4.2 zeigt das UML-Komponentendiagramm der AAS-Semantik- und Verwaltungsschicht. Es umfasst zwei klar voneinander abgegrenzte Laufzeitumgebungen: die *Mapping-Service-Laufzeitumgebung* als semantische Verarbeitungsschicht sowie die *Eclipse-BaSyx-Laufzeitumgebung* als standardkonforme Infrastruktur zur Verwaltung und Persistenz der erzeugten Verwaltungsschalen. Ein Message Broker fungiert als Übergabepunkt zwischen vorgelagerter Daten- und Integrationsinfrastruktur und dieser Semantikschicht und gewährleistet eine lose zeitliche und technische Kopplung beider Architekturteile.

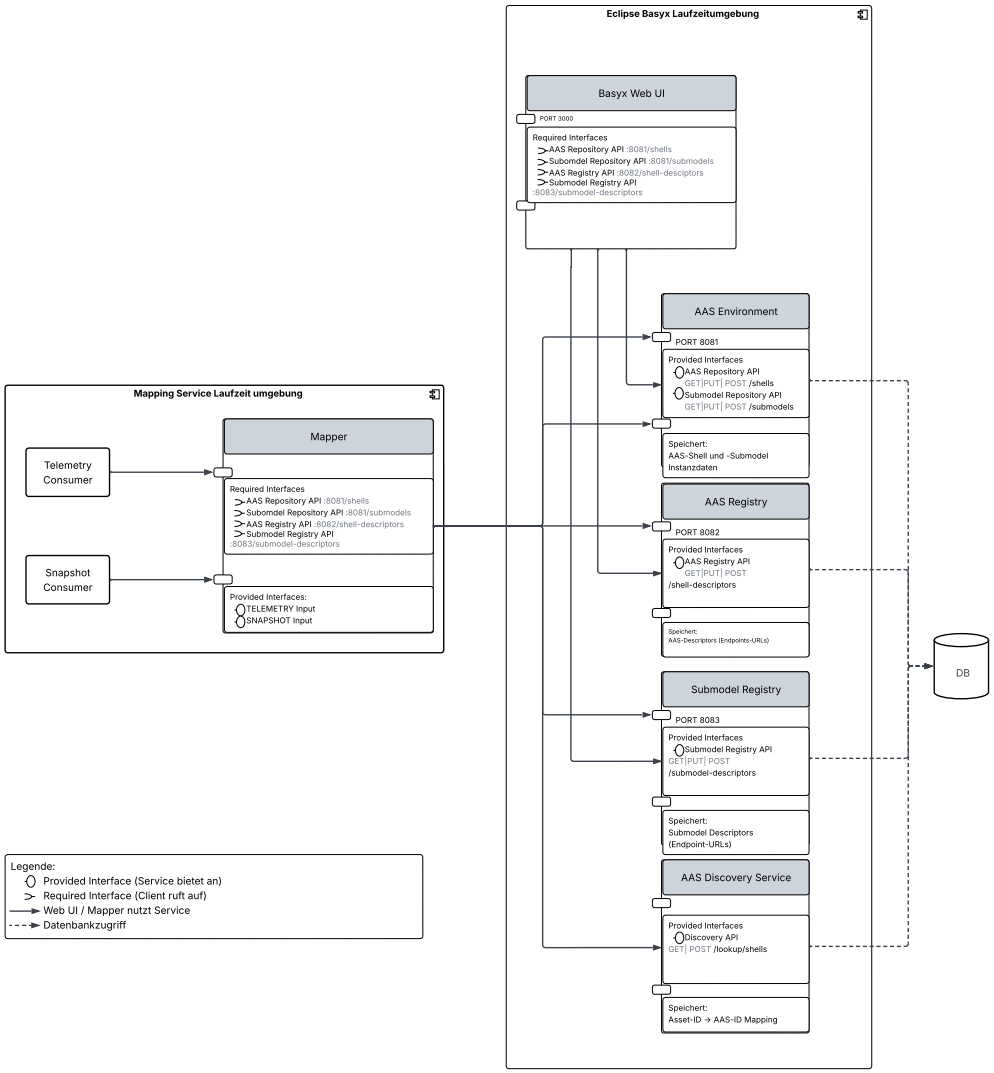

Abbildung 4.2: UML-Komponentendiagramm der AAS-Semantikschicht mit Mapping-Service- und Eclipse-BaSyx-Laufzeitumgebung (eigene Darstellung)

Die *Mapping-Service-Laufzeitumgebung* besteht aus zwei Consumer-Komponenten sowie dem *Mapper* als zentraler Transformationskomponente. Der *Telemetry Consumer* und der *Snapshot Consumer* konsumieren eingehende Ereignisse vom vorgelagerten Kafka-Broker und kommunizieren mit dem Mapper ausschließlich über die intern bereitgestellte Schnittstelle *Mapper*. Diese explizite Schnittstellendefinition innerhalb des Pakets stellt sicher, dass die Consumer-Komponenten von der konkreten Implementierung des Mappers entkoppelt sind und die Transformationslogik unabhängig ausgetauscht oder weiterentwickelt werden kann.

Der Mapping-Service bildet den semantischen Kern der AAS-Schicht. Er übernimmt die regelbasierte Transformation eingehender Asset-Snapshot- und Telemetrieereignisse in AAS-konforme Strukturen und kommuniziert hierfür direkt mit der Eclipse-BaSyx-Laufzeitumgebung. Die Abhängigkeiten des Mapping-Service zu den BaSyx-Komponenten sind als *«use»*-Beziehungen zu den jeweiligen bereitgestellten Schnittstellen modelliert.

Die *Eclipse-BaSyx-Laufzeitumgebung* stellt vier spezialisierte Dienste bereit. Die *AAS Environment*-Komponente (Port 8081) exponiert die Schnittstellen */shells* und */submodels*, über die der Mapping-Service Verwaltungsschalen und Submodelle anlegt und aktualisiert. Die *AAS Registry* (Port 8082) stellt */shell-descriptors* bereit und ermöglicht die Registrierung neu erstellter Verwaltungsschalen. Die *Submodel Registry* (Port 8083) exponiert */submodel-descriptors* für die Registrierung von Submodellen. Der *AAS Discovery Service* (Port 8084) stellt über */lookup//shells* eine Asset-basierte Suche nach registrierten Verwaltungsschalen bereit. Die *BaSyx Web UI* (Port 3000) bildet die grafische Zugriffsschicht auf die verwalteten Artefakte und ist ausschließlich für administrative und explorative Zugriffe vorgesehen.

Die persistente Datenhaltung aller BaSyx-Dienste erfolgt über eine gemeinsam genutzte MongoDB-Instanz. *AAS Environment*, *AAS Registry* und *Submodel Registry* greifen gemeinsam auf die Datenbank zu. Diese zentrale Persistenzschicht entkoppelt die Laufzeitlogik der einzelnen Dienste von der konkreten Datenhaltung und erlaubt eine unabhängige Skalierung beider Ebenen.

Durch die explizite Modellierung beider Laufzeitumgebungen als abgeschlossene Pakete wird eine klare Verantwortlichkeitstrennung zwischen semantischer Transformation und standardkonformer Verwaltung erreicht. Änderungen an Mapping-Regeln, Submodell-Templates oder Consumer-Logik bleiben auf die Mapping-Service-Laufzeitumgebung beschränkt und erfordern keine Anpassungen an der BaSyx-Infrastruktur. Dies gilt ebenfalls für Änderungen an der Basyx-infrastruktur, welche dann wiederum keine Änderungen an der Mapping-Service-Laufzeitumgebung bedingen. Die Architektur ist damit so ausgelegt, dass neue Asset-Typen, zusätzliche Submodelle oder erweiterte Registrierungslogiken durch Anpassung des Mappers integriert werden können, ohne grundlegende strukturelle Änderungen am Gesamtsystem vorzunehmen.

4.2 Vorteile der zweigeteilten Architektur

Die Entscheidung für eine zweigeteilte Architektur basiert auf der bewussten Trennung unterschiedlicher Verantwortlichkeiten innerhalb des Systems. Dabei werden die Aufgaben der Datenintegration und -Bereitstellung klar von der semantischen Modellierung und Verwaltung der Asset-Daten getrennt. Ziel dieses Ansatzes ist es, eine modulare, skalierbare und wartbare Architektur zu schaffen, die über den konkreten Anwendungsfall der Asset Administration Shell hinaus wiederverwendbar ist.

Die vorgelagerte Daten- und Integrationsinfrastruktur übernimmt die Aufgabe, Asset-Daten aus heterogenen Quellsystemen zu erfassen, zu konsolidieren und ereignisbasiert bereitzustellen. Diese Schicht ist bewusst generisch gehalten und stellt die erfassten Informationen in einer domänennahen Form zur Verfügung, ohne eine AAS-spezifische Semantik zu erzwingen. Dadurch kann dieselbe Daten- und Integrationsinfrastruktur nicht nur für die AAS-Anbindung, sondern auch für weitere Anwendungsfälle wie Analysen, Monitoring oder Reporting genutzt werden. Eine direkte Kopplung der Datenquellen an die AAS würde hingegen zu spezialisierten Einweg-Schnittstellen führen, die jeweils nur einen einzelnen Konsumenten bedienen und die Wiederverwendbarkeit der Integrationslogik erheblich einschränken.

Ein weiterer wesentlicher Aspekt ist die klare Trennung der fachlichen Verantwortlichkeiten. Während die Daten-und Integrationsinfrastruktur für den Umgang mit Schnittstellenheterogenität, Ereignisverarbeitung und Datenverteilung zuständig ist, konzentriert sich die AAS-Architektur ausschließlich auf die semantische Abbildung der Asset-Daten

gemäß den Vorgaben der Asset Administration Shell. Diese Trennung der Zuständigkeiten reduziert die Komplexität der einzelnen Architekturbestandteile und verhindert eine Vermischung technischer Integrationslogik mit fachlicher Modellierungslogik. Änderungen an der AAS-Semantik oder an Submodell-Templates können somit unabhängig von der Datenintegration vorgenommen werden und umgekehrt.

Die zweigeteilte Architektur trägt zudem zur verbesserten Nachvollziehbarkeit und Fehlersuchbarkeit des Systems bei. Durch die ereignisbasierte Bereitstellung der Daten lassen sich Datenflüsse auf Asset- und Topic-Ebene eindeutig verfolgen. Fehler oder Inkonsistenzen können klar der jeweiligen Architekturschicht zugeordnet werden, etwa der Datenquelle, der Integrationslogik oder der semantischen Transformation in der AAS-Schicht. Ohne diese Trennung wäre eine isolierte Analyse einzelner Verarbeitungsschritte deutlich erschwert, da Fehler erst am Ende der Verarbeitungskette sichtbar würden.

Darüber hinaus unterstützt die Architektur eine konsistente und zentrale Standardisierung der Asset-Daten. Die AAS-Schicht fungiert als alleiniger Ort der semantischen Modellierung und stellt sicher, dass Submodell-Templates einheitlich angewendet, versioniert und weiterentwickelt werden. Würde die semantische Abbildung bereits in den vorgelagerten Datenquellen oder Integrationsadaptern erfolgen, bestünde die Gefahr divergierender Interpretationen und inkonsistenter AAS-Strukturen. Die Zentralisierung der Semantik erleichtert somit die Durchsetzung von Standards und erhöht die Qualität der bereitgestellten Verwaltungsschalen.

Schließlich bietet die zweigeteilte Architektur Vorteile im Hinblick auf Skalierbarkeit und Erweiterbarkeit. Die Dateninfrastruktur kann unabhängig von der AAS-Schicht mit steigender Anzahl von Assets oder wachsendem Datenvolumen skaliert werden. Gleichzeitig ermöglicht die AAS-Architektur eine feingranulare Skalierung auf Asset-Ebene, beispielsweise durch den Einsatz asset-spezifischer Consumer. Neue Konsumenten oder zusätzliche Verarbeitungslogiken können an die bestehenden Datenströme angebunden werden, ohne bestehende Schnittstellen anzupassen oder neue Punkt-zu-Punkt-Verbindungen einführen zu müssen.

Insgesamt ermöglicht die zweigeteilte Architektur eine klare Strukturierung des Systems, fördert die Wiederverwendbarkeit zentraler Integrationskomponenten und schafft eine belastbare Grundlage für zukünftige Erweiterungen. Sie stellt damit eine geeignete Grundlage für die prototypische Umsetzung sowie für einen späteren Ausbau und produktiven Einsatz dar.

4.3 Daten- und Integrationsarchitektur und Ereignis-Streaming-Konzept

Die nachfolgend beschriebene Daten- und Integrationsarchitektur konkretisiert die in Abschnitt 4.2 eingeführte Trennung zwischen Datenintegration und semantischer Modellierung. Im Fokus steht die technische Ausgestaltung der Integrationsschicht, welche die Aufnahme, Verarbeitung und Bereitstellung von Asset-Daten organisiert. Dabei werden insbesondere die strukturellen Komponenten, der interne Datenfluss sowie das zugrunde liegende Ereignis-Streaming-Konzept beschrieben.

Abbildung 4.1 zeigt die Gesamtstruktur der Daten- und Integrationsarchitektur auf Basis eines hexagonalen Architekturansatzes. Unterschiedliche Quellsysteme, darunter Maschinensteuerungen sowie betriebliche IT-Systeme wie ERP, MES, oder CAD/CAM werden

über technologieabhängige Adapter angebunden. Die Integration erfolgt über klar definierte Input Ports, welche eine entkoppelte Aufnahme der eingehenden Daten ermöglichen. Die zentrale Business-Logik ist technologie- und zielsystemunabhängig ausgeführt und fungiert als Integrationskern zwischen den angebundenen Schnittstellen und den nachgelagerten Exporten.

Die Verarbeitung innerhalb des Integrationskerns erfolgt auf Basis eines domänennahen, konsistenten Datenmodells, welches als kanonische Repräsentation der integrierten Asset-Daten dient (vgl. Abschnitt 2.1). Dieses Modell abstrahiert von quellsystemspezifischen Datenstrukturen und bildet die Grundlage für eine einheitliche Weiterverarbeitung über alle angebundenen Systeme hinweg. Durch die lose Kopplung der Komponenten über einen Message Broker können Produzenten und Konsumenten unabhängig voneinander entwickelt, betrieben und skaliert werden.

Die interne Datenverarbeitung innerhalb der Datenarchitektur orientiert sich an einer Medaillen-Architektur, bestehend aus Bronze-, Silver- und Gold-Schichten. Diese wird im vorliegenden Kontext nicht primär als technisches Data-Engineering-Muster verstanden, sondern als bewusstes Schichtenmodell zur strukturellen Trennung unterschiedlicher Stabilitäts- und Verantwortungsbereiche. Durch die explizite Differenzierung der Schicht werden Reifegrade, Transformationsverantwortung und Schema-Stabilität architektonisch voneinander abgegrenzt. Die Schichtung dient somit nicht nur der schrittweisen Erhöhung der Datenqualität, sondern der gezielten Beherrschung von Heterogenität, Änderungsdynamik und wachsendem Datenvolumen. Die Medaillen-Architektur ist ausschließlich Bestandteil der Daten- und Integrationsarchitektur und endet vor der Übergabe der Daten an die AAS. Tabelle 4.1 stellt die drei Schichten anhand zentraler Kriterien gegenüber.

Tabelle 4.1: Charakteristika der Schichten der Medaillen-Architektur

Kriterium	**Bronze**	**Silver**	**Gold**
Primäre Zielsetzung	Verlustfreie Rohdatenpersistenz	Strukturierung und Harmonisierung	Stabiler Datenvertrag für Export
Datencharakter	Quellsystem-nah, unverändert	Domänennah, normalisiert	Bereinigt, konsolidiert
Grad der Transformation	Keine fachliche Transformation	Strukturelle Transformation	Selektion und Optimierung
Kopplung an Quellsysteme	Hoch	Reduziert	Gering
Stabilität des Schemas	Gering	Mittel	Hoch
Verantwortung im System	Ingestion	Transformationslogik	Ereignis-Erzeugung
Relevanz für externe Systeme	Keine direkte Nutzung	Keine interne Verarbeitung	Basis für Ereignis-Streaming
Änderungsfrequenz	Hoch	Mittel	Gering
Rolle im Gesamtsystem	Technischer Puffer	Domänenmodell	Übergabeschicht zur AAS

In der Bronze-Schicht werden die Rohdaten der jeweiligen Quellsysteme unverändert persistiert. Ziel dieser Schicht ist die verlustfreie Erfassung aller eingehenden Informatio-

nen, unabhängig von deren Struktur oder Qualität. Für jede angebundene Schnittstelle existieren dedizierte Bronze-Tabellen, welche die empfangenen Datensätze in ihrer ursprünglichen Form abbilden. Architektonisch fungiert diese Schicht als Isolationsschicht gegenüber Quellsystemvariabilität. Änderungen an Schnittstellen oder Datenstrukturen wirken sich zunächst ausschließlich auf die Bronze-Ebene aus und gefährden nicht unmittelbar die Konsistenz nachgelagerter Verarbeitungsschichten.

Auf Basis der Rohdaten erfolgt in der Silver-Schicht eine strukturierende Weiterverarbeitung. In diesem Schritt werden die Daten in ein konsistentes, domänennahes Schema überführt und bei Bedarf weiter aufgeteilt, um komplexe Rohstrukturen in logisch zusammenhängende Tabellen zu zerlegen. Die Silver-Schicht übernimmt damit die Rolle eines kanonischen Domänenmodells im Sinne von Abschnitt 2.1. Er reduziert quellsystemspezifische Divergenzen und schafft eine stabile, fachlich orientierte Repräsentation, welche als Referenzmodell für nachgelagerte Prozesse dient. Die Befüllung der Silver-Schicht erfolgt über automatisierte ETL-Pipelines, welche die jeweils vorgelagerte Bronze-Ebene konsumieren.

Die Gold-Schicht stellt die bereinigte und für die Weitergabe optimierte Repräsentation der Asset-Daten dar. In dieser Schicht werden ausschließlich die für nachgelagerte Systeme relevanten Informationen vorgehalten. Redundante oder nicht benötigte Attribute werden entfernt, und die Daten werden in einer Form bereitgestellt, die als stabiler Datenvertrag für die Ereignis-Erzeugung dient. Die Gold-Schicht definiert damit eine explizite Vertragszone zwischen interner Datenaufbereitung und externer Ereignisbereitstellung. Schema-Stabilität und deterministische Reproduzierbarkeit besitzen hier Priorität gegenüber maximaler Detailtiefe. Diese bewusste Reduktion verhindert eine unkontrollierte Propagation interner Strukturänderungen in nachgelagerte Systeme.

Die Verarbeitung der Daten über die einzelnen Ebenen hinweg ist darauf ausgelegt, konsistente und deterministische Ergebnisse zu liefern. Hierzu werden Idempotenzstrategien eingesetzt, insbesondere bei der Verarbeitung ereignisbasierter Datenströme. Wiederholte oder verspätet eintreffende Ereignisse führen somit nicht zu inkonsistenten Zuständen, sondern werden über Upsert-Operationen und Konsistenzprüfungen deterministisch verarbeitet. Ergänzend werden Versionsinformationen mitgeführt, um Änderungen an Asset-Daten nachvollziehbar zu halten.

Die Bereitstellung der Daten an nachgelagerte Systeme erfolgt über ein Ereignis-Streaming-Konzept auf Basis von Kafka. Das Ereignis-Streaming fungiert hierbei nicht lediglich als Transportmechanismus, sondern als strukturelles Entkopplungselement. Durch die asynchrone Publikation stabiler Gold-Schicht-Zustände wird eine zeitliche und technische Trennung zwischen Produzenten und Konsumenten erreicht. Die Daten- und Integrationsarchitektur bleibt somit unabhängig von konkreten Zielsystemen.

Änderungen an den in der Gold-Schicht vorliegenden Datensätzen werden in Form von Ereignissen publiziert und über dedizierte Topics bereitgestellt. Dabei wird zwischen den zwei Datenklassen Asset-Snapshot-Daten und Telemetriedaten unterschieden. Diese Trennung ist in der Architektur explizit durch separate Output Ports realisiert. Asset-Snapshot-Ereignisse enthalten eine vollständige, semantisch stabile Repräsentation eines Assets und sind für die Weiterverarbeitung in der AAS-Schicht vorgesehen. Telemetriedaten hingegen beschreiben hochfrequente Zustands- oder Messwerte und werden getrennt exportiert, um eine gezielte Drosselung oder Aggregation zu ermöglichen.

Die Anbindung der unterschiedlichen Quellsysteme erfolgt abhängig von deren technischen Möglichkeiten und Änderungsfrequenzen. Für Systeme mit vergleichsweise seltenen Datenänderungen werden REST-Schnittstellen eingesetzt. Hierbei kommt ein dedizierter Worker-Prozess zum Einsatz, der die Datenbanken der Quellsysteme periodisch abfragt

und erkannte Änderungen über eine REST-Schnittstelle an die Daten- und Integrationsinfrastruktur übermittelt. Die Ausführung dieses Polling-Mechanismus wird durch einen Hangfire-Server gesteuert, der Hintergrundprozesse zeitgesteuert auslöst. Die über REST empfangenen Datensätze werden unmittelbar in der Bronze-Schicht persistiert.

Für Systeme, die ereignisbasierte Schnittstellen unterstützen, erfolgt die Integration über Kafka. Änderungen an relevanten Datensätzen werden durch Producer unmittelbar an den Broker publiziert und anschließend von dedizierten Consumer-Komponenten konsumiert, welche die Ereignisse in die entsprechenden Bronze-Tabellen überführen. Dieser Ansatz ermöglicht eine nahezu verzögerungsfreie Weitergabe von Änderungen und reduziert den Aufwand für periodisches Polling.

Die Anbindung von Maschinensteuerungen ist konzeptionell über OPC UA vorgesehen. Ein OPC UA-Client abonniert relevante Knoten der Maschinensteuerung über Subscriptions mit konfigurierbaren Abtastraten. Die empfangenen Rohdaten werden, analog zu den übrigen Schnittstellen, zunächst in der Bronze-Schicht persistiert und anschließend über die bestehenden ETL-Pipelines weiterverarbeitet. Eine direkte Weitergabe der OPC UA-Daten an die AAS erfolgt nicht, um eine Überlastung der semantischen Schicht durch hochfrequente Telemetriedaten zu vermeiden.

In ihrer Gesamtheit operationalisiert die Medaillen-Architektur das zentrale Entkopplungsziel der Gesamtarchitektur. Variabilität wird in frühen Schichten absorbiert, Stabilität wird in späten Schichten erzielt. Die Daten- und Integrationsarchitektur bildet damit eine kontrollierte Transformationskette, deren Ergebnis als technisch stabiler und semantisch verwertbarer Zustand an die AAS-Schicht übergeben wird.

4.4 Architektur der Asset Administration Shell

Die AAS-Architektur bildet die semantische Zielarchitektur der in der Daten- und Integrationsinfrastruktur aufbereiteten Asset-Daten. Sie ist strikt von der vorgelagerten Datenintegration getrennt und übernimmt ausschließlich die Aufgabe, die bereitgestellten Asset-Informationen in Form standardkonformer Asset Administration Shells zu verwalten und bereitzustellen. Abbildung 4.2 zeigt den logischen Aufbau der AAS-Architektur sowie die Interaktion der beteiligten Komponenten.

Die AAS, welche nach der AAS-Architektur aufgebaut ist, konsumiert ausschließlich Ereignisse aus der Gold-Schicht der Datenarchitektur, welche über den Kafka-Broker bereitgestellt werden. Diese Ereignisse enthalten bereinigte und konsolidierte Asset-Snapshot-Daten sowie – optional und gedrosselt – ausgewählte Telemetrieinformationen. Die AAS fungiert damit nicht als direktes Integrationsziel für Roh- oder Zwischendaten, sondern als semantische Repräsentation eines stabilen Datenzustands.

Für jedes Asset ist ein dedizierter AAS-Consumer vorgesehen, der die dem Asset zugeordneten Topics konsumiert. Diese Consumer sind logisch voneinander entkoppelt und können unabhängig skaliert oder neu gestartet werden. Die Zuweisung eines Consumers pro Asset ermöglicht eine klare fachliche Abgrenzung sowie eine isolierte Fehlerbehandlung auf Asset-Ebene. Die Consumer übernehmen ausschließlich die Aufgabe, eingehende Ereignisse zu empfangen und an den nachgelagerten Mapping-Service weiterzuleiten.

Der Mapping-Service bildet den fachlichen Kern der AAS-Architektur. Er ist für die Transformation der empfangenen Asset-Snapshot-Ereignisse in AAS-konforme Strukturen verantwortlich. Die semantische Abbildung erfolgt anhand vordefinierter Mapping-Regeln und Templates, wodurch sichergestellt wird, dass Submodell-Templates konsis-

tent angewendet werden. Änderungen an der semantischen Struktur oder an den Submodellen können zentral vorgenommen werden, ohne Anpassungen an den vorgelagerten Consumer-Komponenten oder an der Daten- und Integrationsinfrastruktur zu erfordern.

Der Mapping-Service ist dabei nicht lediglich als technischer Adapter zu verstehen, sondern als zentrale semantische Transformationskomponente der AAS-Architektur. Während die vorgelagerten Consumer ausschließlich den Empfang und die Weiterleitung der Asset-Snapshot-Ereignisse übernehmen und die nachgelagerte BaSyx-Infrastruktur für die standardkonforme Verwaltung und Persistenz der erzeugten Artefakte zuständig ist, liegt die eigentliche Transformationslogik im Mapping-Service. Er bildet damit die entscheidende Vermittlungsschicht zwischen dem konsolidierten, jedoch noch AAS-agnostischen Datenzustand aus der Daten- und Integrationsinfrastruktur und der konkreten Repräsentation als Asset Administration Shell.

Die Aufgabe des Mapping-Service besteht darin, eingehende Asset-Snapshot-Ereignisse regelbasiert in AAS-konforme Strukturen zu überführen. Hierzu werden die im Ereignis enthaltenen Attribute auf die Zielstruktur der Verwaltungsschale, der zugehörigen Submodelle sowie der darin enthaltenen Submodell-Elemente abgebildet. Die Zuordnung erfolgt auf Basis definierter Mapping-Regeln und Templates, welche die semantische Interpretation der angelieferten Daten explizit beschreiben. Auf diese Weise wird sichergestellt, dass gleichartige Assets konsistent modelliert werden und dieselben fachlichen Informationen stets in derselben Zielstruktur repräsentiert sind.

Architektonisch folgt daraus eine bewusste Trennung zwischen Datenintegration und semantischer Modellierung. Die vorgelagerte Daten- und Integrationsinfrastruktur liefert einen stabilen, konsolidierten und fachlich verwertbaren Asset-Zustand, trifft jedoch keine Aussage darüber, wie dieser im Sinne der Asset Administration Shell semantisch zu interpretieren ist. Diese Verantwortung wird gezielt im Mapping-Service gebündelt. Dadurch können Änderungen an Submodell-Strukturen, Attributzuordnungen oder Benennungskonventionen zentral vorgenommen werden, ohne die vorgelagerten Schnittstellen, Consumer oder Verarbeitungsstufen der Daten- und Integrationsinfrastruktur anzupassen.

Zugleich schafft diese Zentralisierung die Grundlage für Wiederverwendbarkeit und Erweiterbarkeit. Neue Asset-Typen, zusätzliche Submodelle oder erweiterte semantische Zuordnungen können durch Anpassung der Mapping-Regeln und Templates in die bestehende Architektur integriert werden, ohne dass grundlegende Änderungen an der Daten- und Integrationsinfrastruktur erforderlich sind. Der Mapping-Service ist damit als eigenständiger architektonischer Kern der semantischen Verarbeitung zu verstehen und stellt eine wesentliche Eigenleistung der entwickelten Systemlösung dar.

Die erzeugten Asset Administration Shells und Submodelle werden über die BaSyx-API in die vorgesehenen Repositories überführt. Dabei werden AAS-Instanzen und Submodelle getrennt verwaltet. Die persistente Speicherung erfolgt über eine angebundene MongoDB, welche die dauerhafte Ablage sowie die Wiederverwendbarkeit der erzeugten Artefakte ermöglicht. Über die BaSyx UI können die registrierten Verwaltungsschalen visualisiert, durchsucht und administrativ verwaltet werden.

Die prototypische Umsetzung der AAS-Architektur erfolgt containerbasiert und wird über Docker orchestriert. Die einzelnen Architekturbestandteile, darunter die AAS-Consumer, der Mapping-Service, die BaSyx- Komponenten sowie die MongoDB, werden als eigenständige Container betrieben. Docker dient hierbei nicht nur der technischen Bereitstellung, sondern spiegelt zugleich die modulare Struktur der Architektur wider. Die klare Trennung der Services auf Container-Ebene unterstützt die Entkopplung der

Komponenten und erleichtert deren unabhängige Skalierung, Konfiguration und Wartung im Rahmen des Prototyps.

Die Kommunikation zwischen den Containern erfolgt über definierte Netzwerkverbindungen und standardisierte Schnittstellen. Abhängigkeiten zwischen den Services, beispielsweise zwischen Mapping-Service und BaSyx-Repositories, werden explizit über die Orchestrierung beschrieben, ohne eine enge Kopplung der Implementierungen zu erzwingen. Dadurch bleibt die Architektur auch in der konkreten Umsetzung flexibel und erweiterbar.

Ein zentrales Gestaltungsprinzip der AAS-Architektur ist die bewusste Entkopplung von Datenfrequenz und semantischer Aktualisierung. Hochfrequente Telemetriedaten werden nicht ungefiltert in die AAS übernommen, sondern, sofern relevant, in gedrosselter oder aggregierter Form verarbeitet. Diese aggregierte Form ist insofern erforderlich, als dass je nach Inhalt des Ziel-AAS-Submodells eine Aggregation mit Daten aus anderen Quellsystemen notwendig sein kann. Die AAS behält damit ihre Rolle als semantisch stabiler Informationsanker und wird nicht durch operative Echtzeitdaten überlastet.

Durch die Kombination aus ereignisbasierter Anbindung, zentraler Mapping-Logik und containerbasierter Umsetzung entsteht eine modulare und skalierbare AAS-Architektur. Sie ermöglicht eine konsistente semantische Repräsentation der Asset-Daten und bildet eine belastbare Grundlage für die prototypische Umsetzung sowie für einen späteren produktiven Ausbau.

4.5 Architektonische Gestaltungsprinzipien

Die in den vorherigen Abschnitten dargestellte Architektur basiert nicht ausschließlich auf technologischen Entscheidungen, sondern folgt einer Reihe übergeordneter Gestaltungsprinzipien, die sich aus den in Kapitel 3 formulierten Anforderungen und Architekturzielen ableiten. Diese Prinzipien strukturieren die Systemkonzeption und stellen sicher, dass die gewählte Lösung nicht nur funktional geeignet, sondern auch langfristig erweiterbar und wartbar ist.

Ein zentrales Leitmotiv der Architektur ist die konsequente Entkopplung von Systembestandteilen. Diese manifestiert sich sowohl in der Trennung zwischen Datenintegration und semantischer Modellierung als auch in der Nutzung ereignisbasierter Kommunikation über einen Message Broker. Durch diese Struktur werden Produzenten und Konsumenten zeitlich sowie technisch voneinander getrennt. Die hexagonale Ausprägung der Daten- und Integrationsarchitektur unterstützt diese Entkopplung zusätzlich, indem sie technologieabhängige Adapter klar vom Integrationskern trennt.

An dieser Stelle ist eine bewusste konzeptionelle Abgrenzung erforderlich: Das hexagonale Architekturmuster — ursprünglich von Cockburn als Ports-and-Adapters-Muster auf Anwendungsebene beschrieben — wird in dieser Arbeit nicht als Implementierungsmuster einzelner Softwarekomponenten, sondern als strukturierendes Prinzip auf Systemintegrationsebene angewendet. Die Rolle der Adapter übernehmen hier keine Klassen oder Module innerhalb einer Codebasis, sondern eigenständige Systemkomponenten: OPC UA-Clients, REST-Worker und Kafka-Producer bilden technologiespezifische Input Ports, während Kafka-Topics als Output-Adapter den Übergang in nachgelagerte Systeme definieren. Der Integrationskern entspricht der zentralen Daten- und Integrationsarchitektur mit ihrer Medaillen-Schichtung, die als technologie- und zielsystemunabhängige Verarbeitungslogik fungiert. Diese Übertragung des Musters auf die Systemebene ist eine in der Praxis etablierte Vorgehensweise und entspricht der von AWS beschriebenen Anwendung

hexagonaler Strukturprinzipien in serviceorientierten Integrationsarchitekturen [4]. Die bewusste Wahl dieser Abstraktionsebene ermöglicht es, das Kernprinzip der Entkopplung von Geschäftslogik und technologiespezifischer Anbindung auch in einer verteilten, heterogenen Systemlandschaft konsequent umzusetzen.

Darüber hinaus folgt die Architektur dem Prinzip der klaren Verantwortlichkeitszuweisung. Die Daten- und Integrationsinfrastruktur übernimmt ausschließlich die Aufgabe der Erfassung, Konsolidierung und strukturellen Aufbereitung von Asset-Daten, ohne eine semantische Interpretation im Sinne der Asset Administration Shell vorzunehmen. Die semantische Abbildung erfolgt erst in der nachgelagerten AAS. Diese Trennung reduziert die Komplexität der jeweiligen Schichten und verhindert eine Vermischung technischer Integrationslogik mit fachlicher Modellierungslogik.

Ein weiteres leitendes Prinzip ist die gezielte Lastentkopplung und Skalierbarkeit. Durch die Differenzierung zwischen semantisch stabilen Asset-Snapshot-Daten und hochfrequenten Telemetriedaten wird eine getrennte Behandlung unterschiedlicher Datenklassen ermöglicht. Hochfrequente Zustandsinformationen werden nicht ungefiltert in die semantische Schicht übernommen, wodurch eine Überlastung der AAS-Infrastruktur vermieden wird. Gleichzeitig erlaubt die ereignisbasierte Struktur eine unabhängige Skalierung von Produzenten und Konsumenten.

Die interne Datenverarbeitung orientiert sich zudem an dem Prinzip der deterministischen und konsistenten Weiterverarbeitung. Durch die Verwendung einer Medaillen-Architektur mit Bronze-, Silver- und Gold-Schicht wird eine schrittweise Qualitätssteigerung der Daten erreicht. Die Bronze-Schicht persistiert die Rohdaten der Quellsysteme unverändert, wohingegen die Silver-Schicht die Bereinigung, Normalisierung und Validierung übernimmt. Die Gold-Schicht fungiert als stabiler Datenvertrag für die ereignisbasierte Bereitstellung und bildet damit die konsolidierte Grundlage für nachgelagerte Systeme. Idempotente Verarbeitungsstrategien und Upsert-Mechanismen stellen sicher, dass wiederholte oder verspätet eintreffende Ereignisse nicht zu inkonsistenten Zuständen führen.

Schließlich ist die Architektur auf Erweiterbarkeit und Wiederverwendbarkeit ausgerichtet. Die Integrationslogik ist bewusst AAS-agnostisch gehalten, sodass sie auch für weitere Anwendungsfälle genutzt werden kann. Neue Quellsysteme, zusätzliche Konsumenten oder erweiterte Submodelle können in die bestehende Struktur integriert werden, ohne grundlegende architektonische Änderungen vornehmen zu müssen.

4.6 Dokumentation zentraler Architekturentscheidungen

Softwarearchitekturen entstehen als Ergebnis einer Vielzahl bewusster Entwurfsentscheidungen, die langfristige Auswirkungen auf Wartbarkeit, Erweiterbarkeit und Skalierbarkeit eines Systems haben. In der Praxis werden solche Entscheidungen häufig nur implizit getroffen oder nicht systematisch dokumentiert, was zu Wissensverlust und eingeschränkter Nachvollziehbarkeit führen kann. ADRs adressieren diese Problematik, indem sie einzelne, wesentliche Architekturentscheidungen strukturiert festhalten. Je nach verwendetem Standard umfassen ADRs unterschiedliche Inhalte wie den Entscheidungskontext, die betrachteten Alternativen, die getroffene Entscheidung sowie die daraus resultierenden Konsequenzen [31]. Im Rahmen dieser Arbeit wird das MADR-Format (Markdown Architecture Decision Record) eingesetzt. Das MADR-Format erweitert Nygards ursprüngliches Format, das sich auf die Grundstruktur aus Kontext, Entscheidung und Konse-

quenzen beschränkt, um explizite Abschnitte für Entscheidungstreiber sowie betrachtete Alternativen mit deren Vor- und Nachteilen. Da das MADR-Format die systematische Gegenüberstellung von Alternativen strukturell vorschreibt, wird die Entscheidungslogik transparent und intersubjektiv nachvollziehbar gemacht, was ein zentrales Kriterium wissenschaftlicher Dokumentation darstellt [28].

Im Verlauf der Konzeption wurden mehrere Entscheidungen identifiziert, die maßgeblichen Einfluss auf die Systemstruktur besitzen. Hierzu zählen insbesondere die Wahl einer zweigeteilten Gesamtarchitektur, die Ausgestaltung der Medaillen-Architektur zur schrittweisen Datenverarbeitung, das ereignisbasierte Integrationskonzept auf Basis von Kafka, die Trennung von Asset-Snapshot- und Telemetriedaten in separate Topics sowie die Auswahl des BaSyx-Stacks für die Umsetzung der Asset Administration Shell.

Tabelle 4.2 stellt die acht ADRs den funktionalen und nicht-funktionalen Anforderungen gegenüber, die sie primär adressieren. Die Zuordnung macht die Rückverfolgbarkeit von Anforderungen zu Architekturentscheidungen explizit und zeigt, dass jede Anforderung durch mindestens eine dokumentierte Entscheidung abgedeckt ist.

Tabelle 4.2: Architekturentscheidungen und adressierte Anforderungen

ADR	**Entscheidung**	**Adressierte Anf.**
ADR-001	Zweigeteilte Architektur – Daten- und Integrationsarchitektur und AAS-Architektur	FA2, NFA4, NFA5
ADR-002	Integrationsstrategie nach Änderungscharakteristik	FA2, FA3, NFA2, NFA5
ADR-003	Auswahl der Event-Streaming-Plattform	FA3, NFA1, NFA3
ADR-004	Topic-Zuschnitt und Trennung von Ereignis- und Zustandsdaten	FA3, FA5
ADR-005	Daten- und Integrationsarchitektur nach dem Bronze/Silver/Gold-Prinzip	FA2, FA5, NFA1, NFA2
ADR-006	Maschinenanbindung über OPC UA Subscriptions	FA1, NFA3
ADR-007	Auswahl des AAS-Stacks	FA4, NFA3
ADR-008	Synchronisations- und Mapping-Strategie zwischen Gold-Layer und AAS	FA4, NFA4, NFA5

Die vollständigen ADR-Dokumente sind im Anhang dieser Arbeit enthalten. Im Rahmen dieses Kapitels werden sie nicht im Detail wiedergegeben, sondern als strukturierte Entscheidungsgrundlage referenziert. Sie bilden die formale Dokumentation der architektonischen Festlegungen und ergänzen die zuvor beschriebene Architekturkonzeption um eine explizite Entscheidungslogik.

Kapitel 5

Prototypische Umsetzung

Aufbauend auf der in Kapitel 4 dargestellten Architekturkonzeption behandelt dieses Kapitel die konkrete prototypische Umsetzung der entwickelten Systemstruktur. Im Vordergrund steht dabei die praktische Realisierung der in den Architekturentscheidungen (ADR-001 bis ADR-008) festgehaltenen Zielarchitektur, um deren technische Umsetzbarkeit und strukturelle Tragfähigkeit unter realistischen technischen Rahmenbedingungen zu validieren. Die prototypische Umsetzung verfolgt dabei nicht das Ziel einer vollständig produktionsreifen Ausgestaltung, sondern dient der funktionsfähigen Realisierung der zentralen Architekturkomponenten sowie deren Zusammenspiel im Gesamtsystem. Besonderes Augenmerk gilt der Integration heterogener Quellsysteme, der schrittweisen Datenverarbeitung innerhalb der Medaillen-Architektur sowie der ereignisbasierten Anbindung der Asset Administration Shell.

5.1 Technische Gesamtarchitektur des Prototyps

Die implementierte Systemlandschaft folgt der in ADR-001 dokumentierten Entscheidung für eine zweigeteilte Gesamtarchitektur. Die klare Trennung zwischen Integrations- und Datenarchitektur einerseits und AAS-basierter Semantikschicht andererseits wird auch auf technischer Ebene vollständig beibehalten. Beide Systembereiche sind ausschließlich über einen Apache Kafka Cluster, im Folgenden als Kafka Cluster bezeichnet, gekoppelt, der als zentrales Kommunikations- und Synchronisationsmedium fungiert.

Abbildung 5.1 stellt den operativen End-to-End-Datenfluss des Prototyps als UML-Sequenzdiagramm dar. Die Darstellung zeigt die zeitlich geordnete Abfolge aller Interaktionen vom Quellsystem bis zur semantischen Repräsentation in der AAS-Architektur und verdeutlicht, wie Zustandsänderungen schrittweise aufgenommen, verarbeitet und persistiert werden.

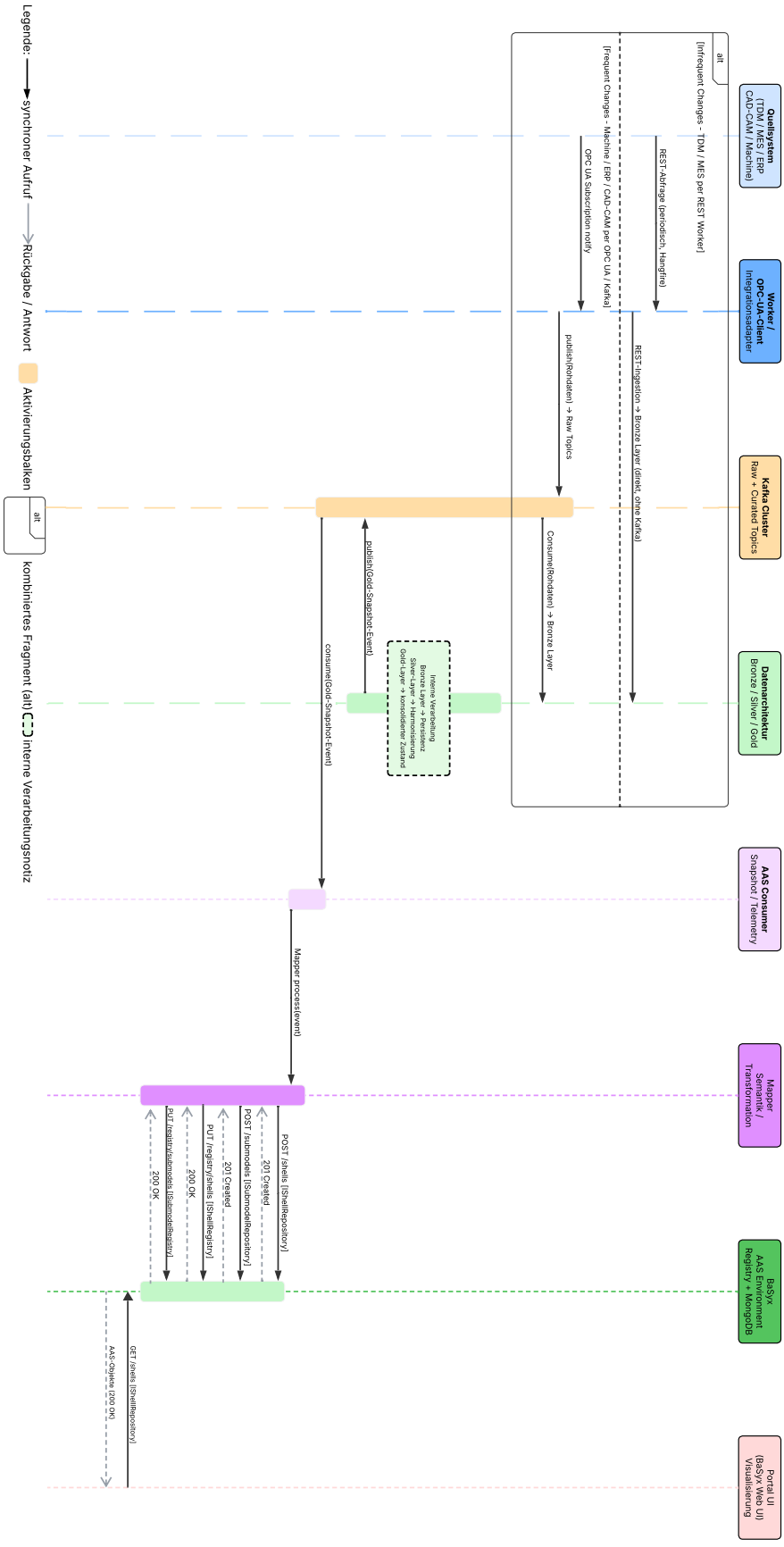

Abbildung 5.1: UML-Sequenzdiagramm: End-to-End-Datenfluss des prototypischen Asset Management Portals

Die Anbindung der Quellsysteme erfolgt gemäß der in ADR-002 festgelegten Integrationsstrategie differenziert nach Änderungsfrequenz und technischer Schnittstellenfähigkeit. Niedrigfrequente Systeme wie TDM und MES werden über einen zeitgesteuerten Worker-Prozess integriert, der Änderungen periodisch identifiziert und die vollständigen Datensätze über eine REST-Schnittstelle direkt an die Bronze-Schicht der Daten- und Integrationsinfrastruktur übermittelt. Dies geschieht dabei ohne Zwischenschaltung des Kafka-Clusters. Systeme mit nativer Ereignisfähigkeit wie ERP- und CAD/CAM-Systeme publizieren Änderungen hingegen als Kafka-Ereignisse in dedizierte Topics des Clusters, welche fortan als *Raw-Topics* bezeichnet werden. Maschinenzustände werden über einen OPC UA-Client erfasst, in ein einheitliches JSON-Ereignisformat projiziert und ebenfalls über Kafka publiziert, was der in ADR-006 dokumentierten Entscheidung zur Maschinenanbindung entspricht.

Unabhängig vom jeweiligen Integrationsmechanismus werden sämtliche eingehenden Daten zunächst in der Bronze-Schicht persistiert. Diese append-only Speicherung entspricht der in ADR-005 dokumentierten Entscheidung für eine Medaillen-Architektur zur schrittweisen Zustandskonsolidierung. Die Bronze-Schicht bildet dabei die erste Persistenzgrenze innerhalb des Systems. In der Silver-Schicht erfolgt die strukturelle Harmonisierung, während die Gold-Schicht den konsolidierten fachlichen Referenzzustand eines Assets repräsentiert.

Die Überführung der Gold-Zustände in ereignisbasierte Nachrichten erfolgt über einen dedizierten Streaming-Prozess. Die daraus resultierenden *Gold-Topics* entsprechen der in ADR-004 definierten Topic-Design-Strategie, welche eine Trennung zwischen Rohdaten- und konsolidierten Export-Topics vorsieht. Die AAS-Architektur konsumiert ausschließlich diese konsolidierten Topics und greift nicht direkt auf Zwischenstände der Datenverarbeitung zu.

Die semantische Verarbeitung innerhalb der AAS-Schicht erfolgt gemäß der in ADR-007 dokumentierten Entscheidung zur Nutzung des BaSyx-Stacks. Ein dedizierter *AAS Consumer* konsumiert die Gold-Snapshot-Ereignisse aus den Gold-Topics des Kafka-Clusters und leitet diese über die *IMapper*-Schnittstelle an den Mapper weiter. Der Mapper transformiert die angelieferten Asset-Snapshot-Daten in standardkonforme Asset Administration Shells und persistiert diese über vier definierte BaSyx-Schnittstellen.Dabei werden Verwaltungsschalen und Submodelle über *IShellRepository* und *ISubmodelRepository* des *AAS Environment*, welches über Port 8081 bereitgestellt wird, angelegt. Anschließend werden sie in der *AAS Registry* über *IShellRegistry* (Port 8082) und in der *Submodel Registry* über *ISubmodelRegistry* (Port 8083) registriert. Dies entspricht der in ADR-008 festgelegten Mapping-Strategie und stellt sicher, dass die semantische Modellierung vollständig in der AAS-Schicht zentralisiert bleibt, während die Integrationsarchitektur ausschließlich konsolidierte Zustände bereitstellt.

Die gesamte Systemlandschaft ist containerbasiert implementiert und wird über Docker orchestriert. Diese Laufzeitstruktur operationalisiert die in Kapitel 4 formulierten Prinzipien der Modularität und Entkopplung auch auf technischer Ebene. Die technische Gesamtarchitektur des Prototyps zeigt damit, dass die konzipierten Strukturentscheidungen nicht nur theoretisch tragfähig, sondern auch konsistent implementierbar sind.

5.2 Datenintegration

Dieser Teil realisiert die in ADR-002 definierte ereignisorientierte Entkopplung der Datenquellen sowie die in ADR-006 spezifizierte Maschinenanbindung über OPC UA. Dabei

folgt die konkrete Topic-Struktur der in ADR-004 definierten Struktur und der Umgang mit Löschungen von Datensätzen der in ADR-005 dokumentierten Strategie.

Die in Kapitel 4 dargestellte Integrationsarchitektur sieht für die Anbindung heterogener Quellsysteme unterschiedliche Kommunikations- und Erfassungsmechanismen vor. Diese Auswahl erfolgt abhängig von der technischen Schnittstellenfähigkeit der Systeme sowie insbesondere von deren Änderungsfrequenz. Für Systeme, deren Daten sich nur selten verändern, wird im Prototyp ein polling-basierter Ansatz verwendet, bei dem Änderungen periodisch identifiziert und als Batch an die Integrationsarchitektur übergeben werden. Systeme mit ereignisbasierter Schnittstelle werden hingegen direkt über Kafka integriert. Die nachfolgenden Abschnitte beschreiben die prototypische Ausgestaltung dieser Ansätze.

5.2.1 Polling-basierte Integration über den Worker

Die polling-basierte Integration niedrigfrequenter Quellsysteme operationalisiert die in ADR-002 dokumentierte Integrationsstrategie, nach der der Kommunikationsmechanismus anhand zweier Kriterien gewählt wurde. Diese beinhalten einerseits die Änderungsfrequenz der Daten sowie andererseits die technische Fähigkeit des Quellsystems zur ereignisbasierten Publikation. Für Systeme, deren Daten sich nur selten ändern und die keine native Ereignis-Schnittstelle bereitstellen, wird im Prototyp ein zeitgesteuerter Worker-Prozess eingesetzt.

Abbildung 5.2 zeigt den vollständigen Ablauf dieses Integrationsmechanismus. Der Prozess wird durch einen periodischen Hangfire-Job ausgelöst und durchläuft anschließend mehrere klar definierte Verarbeitungsphasen. Ziel ist die kontrollierte und deterministische Übertragung aller seit dem letzten Lauf geänderten Datensätze.

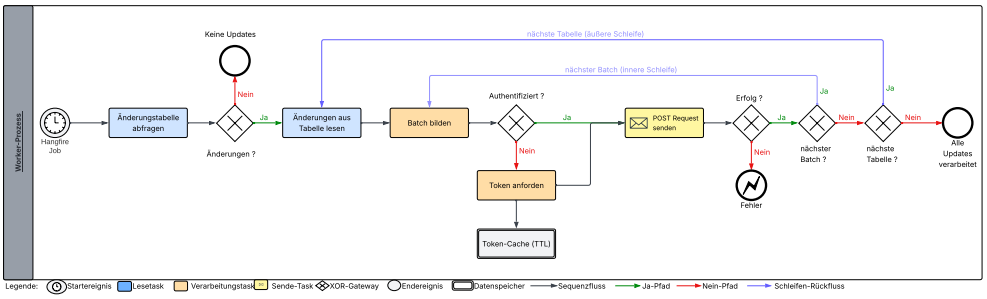

Abbildung 5.2: Ablaufmodell des polling-basierten Worker-Prozesses

Zu Beginn erfolgt die Abfrage einer zentralen Änderungs- beziehungsweise Steuerungstabelle. Die Identifikation neuer oder modifizierter Datensätze erfolgt ausschließlich basierend auf Zeitstempeln. Ein separates Change-Tracking oder triggerbasiertes Verfahren im Quellsystem wird bewusst nicht eingesetzt, wodurch die Integrationslogik vollständig außerhalb des operativen Systems verbleibt. Liegt kein Änderungszeitpunkt vor, der über den zuletzt verarbeiteten Zeitpunkt hinausgeht, wird der Integrationslauf ohne weitere Verarbeitung beendet. Diese frühe Abbruchbedingung reduziert unnötige Datenbankoperationen und stellt sicher, dass der Prozess auch bei häufiger zeitlicher Auslösung ressourcenschonend bleibt.

Werden Änderungen erkannt, verarbeitet der Worker sämtliche relevanten Tabellen innerhalb eines konsistenten Gesamtlaufs. Die Implementierung als Gesamtjob, und nicht als mehrere konkurrierende Teilprozesse, stellt sicher, dass die Reihenfolge der Verarbeitung reproduzierbar bleibt und keine konkurrierenden Datenbankzugriffe entstehen. Die identifizierten Datensätze werden in logische Batches überführt.

Im Prototyp wird bewusst nicht lediglich ein Delta einzelner Feldänderungen übertragen, sondern stets der vollständige fachliche Datensatz. Dadurch wird auf ein differenzielles Ereignismodell verzichtet und die Komplexität der nachgelagerten Zustandskonsolidierung reduziert, da jede Übertragung eine vollständige Repräsentation des aktuellen Quellsystemzustands enthält. Der Worker übernimmt somit ausschließlich die technische Änderungsdetektion und Übermittlung, nicht jedoch die fachliche Zustandsbildung.

Wie in Abbilding 5.2 visualisiert, wird vor dem Versand eines Batches der Authentifizierungszustand geprüft. Ist kein gültiges Token vorhanden, wird ein Authentifizierungsendpunkt aufgerufen und das erhaltene Token bis zu dessen Ablauf zwischengespeichert. Diese Token-Caching-Mechanik reduziert redundante Authentifizierungsaufrufe und stabilisiert den Integrationsprozess innerhalb eines Job-Durchlaufs. Dies ist insofern notwendig, da somit ebenfalls zukünftig das Laden der Daten auf mehere verschiedene Worker aufgeteilt werden kann.

Die Übertragung an die Integrationsarchitektur erfolgt über eine REST-Schnittstelle. Nach jedem Batch-Versand wird der Rückgabestatus geprüft. Bei erfolgreicher Übertragung wird der nächste Batch verarbeitet. Im Fehlerfall wird der Prozess kontrolliert beendet. Durch diese sequentielle Abarbeitung entstehen klar abgrenzbare Verarbeitungsschritte, wodurch Fehlerzustände eindeutig lokalisierbar bleiben.

Auf Integrationsseite werden die über REST empfangenen Datensätze unmittelbar in der Bronze-Schicht persistiert. Diese Persistierung folgt der in ADR-005 dokumentierten Entscheidung für eine Medaillen-Architektur mit append-only Rohdatenspeicherung. Dabei wird bewusst nicht der aktuelle fachliche Endzustand überschrieben, sondern ein ereignisorientiertes Protokoll aller eingehenden Änderungen geführt. Die Bronze-Schicht fungiert somit ausschließlich als technisches Ereignisprotokoll, während die fachliche Zustandsbildung explizit in die Silver-Schicht verlagert wird. Tabelle 5.1 zeigt die im Prototyp verwendete Struktur der Bronze-Persistierung für den Worker-Ingest.

Tabelle 5.1: Schema der Bronze-Persistierung für die Worker-basierte Integration

Attribut	**Beschreibung**
`event_key`	Eindeutiger Schlüssel des Ereignisses
`operation`	Art der Änderung (Insert, Update oder Delete)
`change_id`	Änderungs-ID aus dem Quellsystemkontext
`change_ts`	Änderungszeitpunkt (Timestamp-basierte Detektion)
`ingestion_ts`	Zeitpunkt der Ingestion in die Daten- und Integrationsinfrastruktur
`payload_json`	Vollständiger Datensatz als JSON-Payload
`ingestion_date`	Partitionierungsattribut (Ingestion-Datum)

Die Struktur trennt technische Änderungsmetadaten von der fachlichen Nutzlast. Während *change_ts* und *change_id* die zeitliche und logische Einordnung im Quellsystem ermöglichen, dokumentieren *ingestion_ts* und *ingestion_date* den Zeitpunkt der Übernah-

me in die Integrationsarchitektur. Der vollständige Datensatz wird im Feld *payload_json* als JSON gespeichert, wodurch eine verlustfreie Rekonstruktion des übertragenen Systemzustands gewährleistet bleibt.

Durch die append-only Speicherung werden mehrfach übertragene oder verspätet eintreffende Datensätze nicht überschrieben, sondern als separate Ereignisse persistiert. Die Idempotenz wird erst im Übergang zur Silver-Schicht hergestellt, indem dort anhand fachlicher Schlüssel und Änderungsmetadaten deterministisch entschieden wird, welcher Datensatz den gültigen Referenzzustand repräsentiert. Das resultierende Konsistenzmodell entspricht einer verzögerten, jedoch deterministischen Zustandsbildung, bei der technische Ereigniserfassung und fachliche Referenzbildung zeitlich entkoppelt sind.

Der Worker-Ansatz implementiert damit ein zweistufiges Integrationsmodell, indem zunächst eine technische Änderungsdetektion auf Basis von Zeitstempeln erfolgt und daran anschließend eine Snapshot-basierte Übertragung vollständiger Datensatzrepräsentationen. Die fachliche Zustandsbildung wird bewusst in die nachgelagerte Medaillen-Architektur verlagert und zentralisiert.

Insgesamt stellt der polling-basierte Worker-Prozess eine robuste Integrationslösung für niedrigfrequente Systeme dar. Er fügt sich konsistent in das ereignisbasierte Gesamtmodell der Architektur ein, ohne im Quellsystem selbst eine kontinuierliche Streaming-Infrastruktur vorauszusetzen.

5.2.2 Ereignisbasierte Integration über Kafka

Diese Teilkomponente setzt ADR-002 (Ereignisorientierung) um, indem Zustands- und Änderungsinformationen als Ereignisse in Kafka publiziert und konsumiert werden. Die Topic-Schnittstellen sind gemäß ADR-004 pro Asset strukturiert, und Delete-Ereignisse werden entsprechend ADR-005 als explizite Nachrichten verarbeitet.

Für Quellsysteme mit nativer Ereignisfähigkeit wird im Prototyp eine direkte Integration über einen zentralen Kafka-Cluster realisiert. Diese Ausgestaltung operationalisiert die in ADR-003 dokumentierte Entscheidung zur Nutzung einer Event-Streaming- Plattform als zentrales Integrationsmedium.

Der Raw-Integrationspfad für Business-Objekte aus dem ERP- und CAD-System basiert auf nativer Ereignisfähigkeit der jeweiligen Quellsysteme. Die Änderungsdetektion erfolgt hierbei innerhalb des jeweiligen Quellsystems. Änderungen an einem Business-Objekt führen zur Ausführung einer systemspezifischen Anpassungslogik, welche die Ereignisprojektion übernimmt und unmittelbar ein Kafka-Ereignis erzeugt. Jede Änderung wird als vollständige Snapshot-Repräsentation des jeweiligen Business-Objekts modelliert. Es wird bewusst auf ein differenzielles Änderungsmodell verzichtet. Dadurch ist jedes publizierte Ereignis in sich vollständig interpretierbar und unabhängig von der Kenntnis vorheriger Ereignisse. Die fachliche Zustandsbildung erfolgt nicht im Quellsystem und auch nicht im Broker, sondern in den nachgelagerten Verarbeitungsschichten der Medaillen- Architektur. Löschoperationen werden als explizite Delete-Ereignisse publiziert und nicht implizit aus dem Ausbleiben von Updates abgeleitet um die spätere Verarbeitung in den Silver-Tabellen zu beschleunigen.

Die erzeugten Ereignisse bestehen aus einem standardisierten Header sowie einer objektspezifischen Payload im JSON-Format. Der Header enthält systemübergreifend konsistente Metainformationen, darunter Ereignistyp, Universally Unique Identifier (UUID), sendendes Quellsystem, Schema-Version sowie einen Producer-Zeitstempel. Diese Metadaten ermöglichen eine eindeutige Identifikation jedes Ereignisses, eine versionsbezogene

Einordnung der Payload-Struktur sowie eine zeitliche Differenzierung zwischen Ereigniserzeugung und späterer Ingestion in die Integrationsarchitektur.

Die Publikation erfolgt in ein dediziertes Raw-Topic pro Business-Objekt-Typ. Diese Struktur entspricht der in ADR-004 festgelegten Topic-Design-Strategie, nach der eine klare Trennung fachlicher Objekttypen auf Topic-Ebene vorgesehen ist. Eine Vermischung unterschiedlicher Objekte innerhalb eines Topics wird bewusst vermieden, um eine eindeutige Verantwortlichkeitszuordnung und eine unabhängige Skalierung zu ermöglichen. Die Topic-Benennung folgt dem Schema *servicenumber.system-businessobject-raw.json*, wodurch die fachliche Herkunft und der Verarbeitungsstatus (Raw) unmittelbar ersichtlich sind.

Jedes Topic ist eigenständig partitioniert. Die Partitionierung ermöglicht eine parallele Verarbeitung durch Consumer-Gruppen, wobei innerhalb einer Partition die Reihenfolge der Ereignisse garantiert bleibt. Eine globale Reihenfolge über mehrere Partitionen hinweg wird nicht erzwungen und ist für die fachliche Konsolidierung nicht erforderlich, da die Zustandsbildung pro Business-Objekt erfolgt. Die Partitionierungslogik stellt somit einen Kompromiss zwischen Skalierbarkeit und deterministischer Verarbeitung dar.

Der Kafka-Broker fungiert als entkoppelnde Zwischeninstanz zwischen Produzenten und Konsumenten. Das Quellsystem ist ausschließlich für die fachlich korrekte Projektion des Ereignisses verantwortlich. Die Integrationsarchitektur übernimmt keine Validierung oder Interpretation des Business-Objekts, sondern stellt eine skalierbare und persistente Verteilung sicher. Diese klare Verantwortungstrennung verhindert eine Vermischung von fachlicher Modellierung und technischer Integrationslogik.

Die Raw-Ereignisse werden durch dedizierte Consumer konsumiert und unverändert in der Bronze-Schicht persistiert. Analog zum Worker-Ansatz fungiert die Bronze-Schicht als technisches Ereignisprotokoll. Eine unmittelbare Zustandskonsolidierung findet nicht statt. Die Konsolidierung und Referenzzustandsbildung werden erst in der Silver- beziehungsweise Gold-Schicht durchgeführt. Tabelle 5.2 dokumentiert die zugehörige Tabellenstruktur, die das Worker-Schema um transportbezogene Kafka-Metadaten erweitert.

Tabelle 5.2: Schema der Bronze-Persistierung für Kafka-basierte Raw-Ereignisse

Attribut	**Beschreibung**
`event_key`	Eindeutige Ereignis-ID (UUID)
`event_type`	Typ des Business-Objekts bzw. Ereignisses
`source_system`	Herkunftssystem des Ereignisses
`schema_version`	Version der Payload-Struktur
`producer_ts`	Erzeugungszeitpunkt im Quellsystem
`topic`	Kafka-Topic
`partition`	Kafka-Partition
`offset`	Kafka-Offset
`ingestion_ts`	Zeitpunkt der Persistierung in Bronze
`payload_json`	Vollständige Snapshot-Payload

Die Persistierung der Kafka-basierten Raw-Ereignisse in der Bronze-Schicht erfolgt in einer dedizierten Delta-Tabelle pro Business-Objekt. Die Tabellenstruktur erweitert das

in Tabelle 5.1 dargestellte Worker-Schema um transportbezogene Metadaten aus dem Kafka-Kontext.

Das resultierende Integrationsmodell kombiniert eine quellsystemseitige Ereigniserzeugung mit einer zeitlich entkoppelten, jedoch deterministischen Zustandsbildung im Data-Lake-Kontext. Durch vollständige Snapshot-Ereignisse, explizite Delete-Operationen und klar abgegrenzte Business-Objekt-Topics entsteht ein robustes, nachvollziehbares und skalierbares Ereignismodell, das sich konsistent in die Gesamtarchitektur einfügt.

5.2.3 Maschinenintegration

Die Maschinenintegration folgt ADR-006, indem Maschinenzustände über definierte OPC UA Subscriptions erfasst und über einen dedizierten OT/IT-Übergabepunkt in den ereignisbasierten Integrations-Fluss überführt werden.

Im Unterschied zu den in Abschnitt 5.2.1 und Abschnitt 5.2.2 beschriebenen Quellsystemen entstehen Maschineninformationen im Prototyp nicht primär als fachliche Business-Objekte, sondern als zustands- und prozessnahe Änderungsereignisse aus der Steuerungshierarchie. Ziel der Umsetzung ist daher nicht die fachliche Konsolidierung oder semantische Einordnung in dieser Stufe, sondern die kontrollierte und reproduzierbare Abbildung der OPC UA Zustandsänderungen in ein einheitliches Ereignisformat, das konsistent in Kafka (ADR-003) publiziert und anschließend wie alle übrigen Ereignisse in der Bronze-Schicht protokolliert werden kann (ADR-005).

Der in ADR-006 definierte OT/IT-Übergabepunkt wird im Prototyp durch Orchestra umgesetzt. Damit liegt die Verantwortung für die OPC UA-spezifische Kommunikation, die Abbildung der ValueSets sowie die technische Publikation in Kafka vollständig innerhalb einer Integrationskomponente. Nachgelagerte Verarbeitungsschritte interagieren ausschließlich mit Kafka und greifen nicht direkt auf OPC UA zu. Der Integrationspfad bleibt damit konsistent zur in ADR-002 festgelegten Strategie, dass die ereignisbasierte Erfassung, die technische Entkopplung über Kafka und die Verlagerung der fachlichen Zustandsbildung in die nachgelagerte Medaillen-Architektur verschoben werden.

Abbildung 5.3 zeigt die prototypische Umsetzung dieses Integrationspfads als Business Process Model and Notation (BPMN)-Prozessdiagramm. Die Darstellung gliedert sich in zwei Pools, die die in ADR-006 definierte OT/IT-Netztrennung abbilden. Dabei löst im OT-Bereich eine Zustandsänderung der Computerized Numerical Control (CNC)-Steuerung die Publikation über den OPC UA Server aus, der die Daten per Subscription bereitstellt. Der Nachrichtenfluss zwischen den Pools repräsentiert den definierten OT/IT-Übergabepunkt. Im IT-Bereich empfängt Orchestra die Subscription, reichert die empfangenen ValueSets mit statischem Maschinenkontext an, überführt sie über ein definiertes Mapping in die JSON-Zielstruktur und publiziert die serialisierten Ereignisse einschließlich Trace-ID und Kafka-Header in ein dediziertes Kafka-Topic. Durch diese Kapselung bleiben die Protokoll- und Modellierungsdetails von OPC UA auf den Übergabepunkt begrenzt, während Kafka als einheitliches Transport- und Entkopplungsmedium fungiert (ADR-003). Die Topic-Abgrenzung folgt ADR-004, indem Maschinenereignisse als eigenständige Datenklasse geführt und nicht mit anderen Business-Objekten vermischt werden.

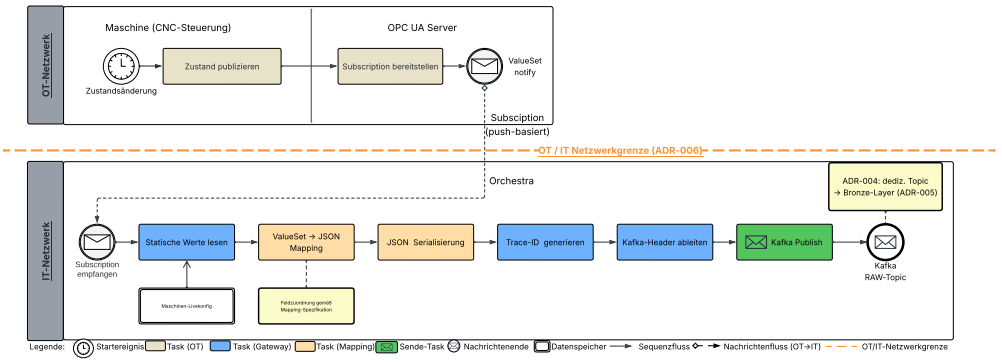

Abbildung 5.3: BPMN-Prozessdiagramm: Integrationspfad von OPC UA Zustandsänderungen zum dedizierten Kafka-Topic

Während das BPMN-Prozessdiagramm den Integrationspfad vom OPC UA Zustandsereignis bis zur Kafka-Publikation über beide Netzwerkbereiche hinweg zeigt, spezifiziert Tabelle 5.3 die normalisierte Zielstruktur der einzelnen Felder im Detail.

Die Projektion der OPC UA ValueSets in die JSON-Zielstruktur wird im Prototyp über ein fest definiertes Mapping realisiert. Um die Abbildung unabhängig von einer Tool- oder User Interface (UI)-spezifischen Darstellung nachvollziehbar zu dokumentieren, wird das Mapping in Tabelle 5.3 als technische Spezifikation aufgeführt. Die Tabelle beschreibt, welche ValueSets in welche Zielfelder überführt werden und welche Transformationsregeln dabei angewendet werden. Das Mapping bleibt bewusst technisch und beschränkt sich auf die Normalisierung und Vereinheitlichung der Struktur. Eine domänenspezifische Interpretation im Sinne der AAS erfolgt erst nachgelagert, nachdem die Ereignisse in der Bronze-Schicht protokolliert und in der Silver- beziehungsweise Gold-Schicht deterministisch konsolidiert wurden (ADR-005).

Tabelle 5.3: Mapping-Spezifikation: OPC UA ValueSets auf JSON-Zielstruktur

OPC UA ValueSet	**Quellkontext**	**Zielfeld (JSON)**	**Abbildung**
`opMode`	Zustandswert inkl. Node-Identifikation	`machine_state.op_mode`	Übernahme, Normalisierung auf Statuswerte
`actToolIdent`	Aktive Werkzeugkennung	`process.active_tool_id`	Übernahme, leer → `null`
`workPandProgName`	Arbeitsplan- und Programmkontext	`process.program_name`	Projektionale Ableitung
`sourcetimestamp`	Quellseitiger Zeitbezug	`meta.source_ts`	Übernahme als Ordnungsmerkmal
`servertimestamp`	Serverseitiger Zeitbezug	`meta.server_ts`	Übernahme zur Diagnose
Maschinen-/Linienkontext	Statische Konfiguration je Subscription	`meta.asset_id`	Statische Zuordnung
Trace-Kontext	Integrationskontext der Nachricht	`meta.trace_id`	Pro Nachricht generiert

Nach der Publikation werden die Maschinenereignisse analog zu den übrigen Kafka-basierten Raw-Ereignissen konsumiert und unverändert in der Bronze-Schicht persistiert. Dadurch entsteht auch für Maschinenzustände ein vollständiges, append-only Ereignisprotokoll, während die Bildung eines konsolidierten Referenzzustands erst in der Silver-Schicht erfolgt. Die Maschinenintegration implementiert damit denselben Entkopplungs- und Konsistenzansatz wie die übrigen Integrations-Pfade, indem einerseits die technische Ereigniserfassung und Verteilung über Kafka und andererseits deterministische Zustandsbildung in der Medaillen-Architektur umgesetzt werden.

5.3 Integrationskern: Medaillen-Architektur

Der Integrationskern implementiert die in ADR-003 festgelegten Prinzipien zur Persistierung und Nachvollziehbarkeit, indem eingehende Ereignisse zunächst vollständig protokolliert und anschließend deterministisch zu konsistenten Zuständen konsolidiert werden.

Die in ADR-005 dokumentierte Entscheidung für eine Medaillen-Architektur bildet das zentrale Verarbeitungskonzept der prototypischen Umsetzung. Während in Abschnitt 5.2.1 und Abschnitt 5.2.2 die Aufnahme heterogener Zustandsänderungen über REST beziehungsweise Kafka beschrieben wurde, adressiert der vorliegende Abschnitt die nachgelagerte Konsolidierung dieser Ereignisse innerhalb der Daten- und Integrationsinfrastruktur. Der Fokus liegt damit nicht auf der Erzeugung oder dem Transport von Ereignissen, sondern auf der kontrollierten und reproduzierbaren Zustandsbildung im Integrationskern.

Die Umsetzung erfolgt in Databricks auf Basis von Delta Lake und Apache Spark. Sämtliche Persistenzschichten sind als Delta Tables implementiert und werden inkrementell befüllt. Die Architektur folgt einem klaren Trennungsprinzip zwischen technischer Ereigniserfassung und fachlicher Zustandsrepräsentation. Die Bronze-Schicht fungiert als

unverändertes, append-only Ereignisprotokoll, während Silver und Gold ausschließlich konsolidierte Zustände repräsentieren. Durch diese Separation wird sichergestellt, dass sowohl die vollständige Ereignishistorie als auch ein jederzeit reproduzierbarer Referenzzustand vorliegen, ohne beide Sichtweisen innerhalb derselben Persistenzschicht zu vermischen.

Die drei Verarbeitungsebenen, Bronze Silver und Gold, übernehmen dabei klar abgegrenzte Verantwortlichkeiten. Bronze speichert eingehende Zustandsänderungen unverändert und ohne Konsolidierung. Silver überführt dieses Ereignisprotokoll in einen deterministischen, business-key-basierten Referenzzustand unter Berücksichtigung zeitlicher Ordnungsmerkmale. Gold transformiert diesen Zustand schließlich in einen konsumentenorientierten Datenvertrag, der als Grundlage für die nachgelagerte ereignisbasierte Bereitstellung dient.

Tabelle 5.4 fasst die systematische Abgrenzung der drei Schichten hinsichtlich Zielsetzung, Persistenztyp, Konsolidierungslogik und Exportrelevanz zusammen.

Tabelle 5.4: Gegenüberstellung der Medaillen-Schichten im Prototyp

Kriterium	**Bronze**	**Silver**	**Gold**
Ziel	Technisches Ereignisprotokoll	Deterministische Zustandsbildung	Kuratierter Referenzzustand
Tabellentyp	Append-only Delta Table	Delta Table, inkrementell (MERGE)	Delta Table, exportorientiert
Datenmodell	Quellnah, JSON-basiert	Typisiert, harmonisiert	Konsumentenorientiert
Konsolidierung	Keine	Business-Key-basiert, zeitlich geordnet	Projektionale Verdichtung
Delete-Handling	Ereignis wird gespeichert	Physische Löschung	Physische Löschung
Historienhaltung	Vollständig	Nur aktueller Zustand	Nur aktueller Zustand
Idempotenz	Nein	Ja (MERGE-basiert)	Ja
Exportrelevanz	Nein	Nein	Ja (CDC → Kafka)

Die Tabelle verdeutlicht, dass jede Schicht eine klar definierte semantische Rolle übernimmt. Während Bronze ausschließlich die technische Persistenz von Ereignissen sicherstellt, entsteht der fachliche Referenzzustand erst in der Silver-Schicht. Die Gold-Schicht bildet schließlich den stabilen Datenvertrag für nachgelagerte Systeme und markiert damit den Übergang von interner Zustandsbildung zur externen Bereitstellung. Diese klar abgegrenzte Verantwortungsstruktur ist wesentlich für die Reproduzierbarkeit, Wartbarkeit und Skalierbarkeit der gesamten Daten- und Integrationsinfrastruktur.

Abbildung 5.4 veranschaulicht den vollständigen ereignisgesteuerten Datenfluss als Unified Modeling Language (UML)-Sequenzdiagramm. Die Darstellung zeigt die zeitlich geordnete Interaktionsfolge der beteiligten Verarbeitungskomponenten. Ausgehend vom kontinuierlichen Kafka-Konsumzyklus des *Bronze Streaming Jobs* (dargestellt als *loop*-Fragment), über den tabellentriggergesteuerten Start des *Silver/Gold Batch Jobs* mit sequenziell abhängigen Silver- und Gold-Tasks, bis zur abschließenden Publikation der konsolidierten Gold-Snapshots in die Gold Topics des Kafka-Clusters durch den *Publishing Job* über den *Delta Change Data Feed* (CDF).

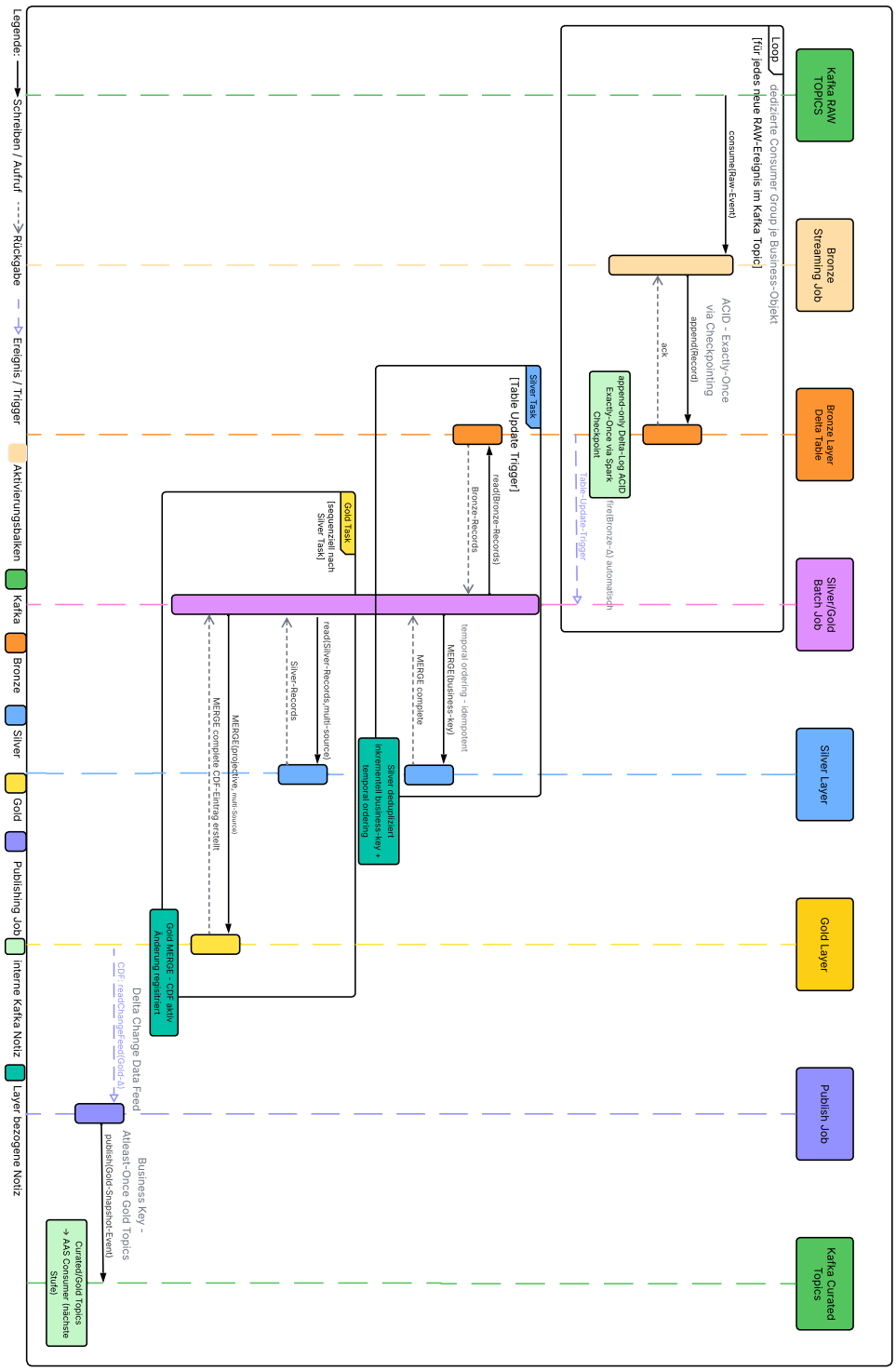

Abbildung 5.4: UML-Sequenzdiagramm: Ereignisgesteuerter Datenfluss der Medaillen-Architektur vom Bronze Streaming Job bis zur Kafka-Publikation (eigene Darstellung)

5.3.1 Bronze-Schicht als Ereignisprotokoll

Die Bronze-Schicht realisiert die in Abschnitt 5.3 beschriebene technische Persistenzebene und stellt die erste stabile Speichergrenze innerhalb der Daten- und Integrationsinfrastruktur dar. Ziel dieser Schicht ist nicht die Abbildung eines aktuellen Fachzustands, sondern die vollständige und unveränderte Erfassung sämtlicher eingehender Zustandsänderungen. Diese strikte Trennung zwischen Ereigniserfassung und Zustandsbildung ist eine zentrale Entwurfsentscheidung der Architektur (ADR-005) und bildet die Grundlage für Auditierbarkeit, Reproduzierbarkeit und Fehlertoleranz der gesamten Verarbeitungspipeline.

Die technische Umsetzung der Bronze-Ingestion erfolgt über einen zentralen Databricks-Streaming-Job, der mehrere Bronze-Tasks bündelt. Pro Business-Objekt wird ein dedizierter Consumer betrieben, inklusive eigener Consumer-Group-ID und separatem Checkpoint-Verzeichnis. Dieses Design stellt sicher, dass die Konsumposition deterministisch fortgeschrieben wird und ein Wiederanlauf ohne Datenverlust oder Doppelverarbeitung möglich ist. Die Checkpoint-basierte Offsetverwaltung von Spark Structured Streaming gewährleistet dabei Exactly-Once-Semantik auf Ausgabeebene. Dabei wird jedes Kafka-Ereignis genau einmal in der Bronze-Tabelle persistiert, selbst bei Jobunterbrechungen oder vorübergehenden Netzwerkfehlern.

Die konkrete Konfiguration des Databricks-Jobs verdeutlicht diese Entwurfsprinzipien auf Implementierungsebene. Der Bronze-Job ist als kontinuierlicher Streaming-Job mit automatischer Wiederholung im Fehlerfall konfiguriert. Pro Business-Objekt existiert ein eigenständiger Task, der über parametrisierte Eingaben gesteuert wird. Dabei werden der Kafka-Bootstrap-Server, der Topic-Name, die Consumer-Group-ID sowie der Pfad zur Ziel-Delta-Table und zum Checkpoint-Verzeichnis als Job-Parameter übergeben. Credentials für die Kafka-Authentifizierung werden nicht direkt in der Task hinterlegt, sondern ausschließlich über Databricks Secret Scopes referenziert. Diese Parametrisierung ermöglicht die Wiederverwendung derselben Task-Logik über mehrere Business-Objekte hinweg und erleichtert sowohl die Erweiterung um neue Quellen als auch die zentrale Wartung der Ingestion-Logik. Alle Bronze-Tasks teilen sich einen gemeinsamen, autoskalierenden Cluster (1–8 Worker), der eine ressourceneffiziente parallele Ausführung der Ingestion-Streams sicherstellt. Die Cluster-Konfiguration nutzt dabei den *USER_ISOLATION*-Modus, der eine isolierte Ausführung paralleler Tasks innerhalb desselben Clusters gewährleistet.

Die Persistierung erfolgt append-only in Delta Tables. Eine Zustandskonsolidierung findet bewusst nicht statt. Mehrfach eintreffende oder verspätete Ereignisse werden nicht überschrieben, sondern vollständig gespeichert. Löschoperationen werden als explizite Ereignisse protokolliert und ebenfalls unverändert persistiert. Die Wahl von Delta Lake als Speicherformat stellt sicher, dass auch die Ingestion selbst ACID-konform abläuft. Dabei verhindern atomare Schreibtransaktionen, dass halbfertige Zustände für nachgelagerte Verarbeitungsschritte sichtbar werden. Zudem ermöglicht das Delta-Transaktionslog sogenannte *Point-in-Time-Abfragen*, also Abfragen, welche Daten zu welchem vergangen Zeitpunkt zur Verfügung standen. Diese erleichtern die Fehleranalyse sowie die gezielte Wiederverarbeitung historischer Ereignisse.

Die Bronze-Schicht repräsentiert damit die technische Wahrheit der eingehenden Datenströme und bildet die Grundlage für eine nachgelagerte, deterministische Zustandsbildung in der Silver-Schicht. Da sämtliche Rohdaten dauerhaft und vollständig vorliegen, kann jede nachgelagerte Verarbeitungsstufe jederzeit vollständig neu aus Bronze abgeleitet werden. Dies ist insbesondere relevant, wenn sich Transformationslogiken ändern oder Fehler in nachgelagerten Schichten behoben werden müssen. Bronze dient in diesem Sinne

als unveränderlicher Ausgangspunkt für beliebige Reprocessing-Szenarien und entkoppelt die Qualität des Gesamtsystems von der Fehlerfreiheit einzelner Verarbeitungsschritte.

5.3.2 Silver-Schicht als deterministische Zustandskonsolidierung

Die Silver-Schicht realisiert die fachliche Zustandsbildung innerhalb des Integrationskerns und transformiert das in der Bronze-Schicht persistierte Ereignisprotokoll in einen konsolidierten Referenzzustand pro Business-Objekt. Während Bronze ausschließlich technische Ereignisse speichert, repräsentiert Silver erstmals einen domänennah strukturierten, aktuellen Fachzustand. Die Silver-Schicht übernimmt die Verantwortung für semantische Korrektheit und strukturelle Konsistenz.

Die Verarbeitung erfolgt inkrementell und wird je Business-Objekt isoliert betrieben. Für jedes Business-Objekt existiert ein eigener Silver/Gold-Job. Diese Entkopplung erhöht die Wartbarkeit und Skalierbarkeit der Verarbeitungspipelines und verhindert, dass Änderungen oder Fehler in einem Objektkontext unmittelbare Auswirkungen auf andere Business-Objekte haben. Die Isolierung ermöglicht darüber hinaus eine unabhängige Versionierung und das unabhängige Deployment einzelner Business-Objekt-Pipelines, was die evolutionäre Weiterentwicklung des Systems erheblich erleichtert.

Die Silver-Verarbeitung umfasst zunächst die strukturelle Normalisierung der eingehenden Ereignisse. Die in Bronze gespeicherte JSON-Payload wird in ein explizit typisiertes Schema überführt und quellsystemspezifische Strukturen werden harmonisiert. Dieses Schema-Mapping ist ein bewusst kontrollierter Übergang.Die Silver-Schicht legt ein stabiles, domänennahes Schema fest, das als Grundlage für die Referenzbildung dient. Sofern der fachliche Referenzzustand die Kombination mehrerer Quellen erfordert, erfolgt eine kontrollierte Zusammenführung in dieser Schicht.

Der Kernmechanismus der Zustandsbildung basiert auf einem business-key-orientierten Konsistenzmodell mit zeitlicher Ordnungslogik. Für jedes Business-Objekt wird anhand eines fachlichen Schlüssels genau ein Referenzzustand geführt. Tritt für denselben Schlüssel eine neue Zustandsänderung ein, wird deterministisch entschieden, ob diese den bestehenden Zustand ersetzen darf. Maßgeblich ist dabei ein fachlich relevantes Ordnungsmerkmal, beispielsweise *change_ts* aus dem Worker-Kontext oder *producer_ts* aus Kafka-Headern. Bei konkurrierenden Ereignissen wird stets der Datensatz mit dem höchsten Ordnungswert als gültiger Referenzzustand übernommen. Durch die konsequente Verwendung von Ereigniszeit anstelle von Verarbeitungszeit als Ordnungskriterium bleibt das Zustandsmodell korrekt, auch wenn Ereignisse mit Verzögerung eintreffen oder in abweichender Reihenfolge verarbeitet werden.

Durch diese Kombination aus Business Key und zeitlicher Ordnung entsteht ein deterministisches Zustandsmodell, das unabhängig von der Ankunftsreihenfolge der Ereignisse konsistente Ergebnisse liefert. Verspätete Zustandsänderungen oder Mehrfachzustellungen führen nicht zu inkonsistenten Tabellenständen, da die Referenzbildung ausschließlich auf dem definierten Ordnungsprinzip basiert. Die technische Umsetzung erfolgt mittels Delta-MERGE- Operationen, die eine idempotente Aktualisierung der Referenztabellen gewährleisten. Die MERGE-Semantik erlaubt dabei eine gezielte Fallunterscheidung zwischen INSERT, UPDATE und logischem DELETE auf Basis des Business Keys, ohne eine vollständige Neuberechnung der Zieltabelle zu erfordern. Dies reduziert sowohl die Verarbeitungslatenz als auch den Ressourcenbedarf bei inkrementellen Aktualisierungen erheblich.

Die Ausführungssteuerung des Silver-Jobs basiert auf einem Table-Update-Trigger, der bei einer Änderung an der Bronze-Delta-Tabelle automatisch ausgelöst wird. Im Gegen-

satz zu einem zeitgesteuerten Batch-Ansatz wird die Silver-Verarbeitung damit unmittelbar und ereignisgetrieben durch das Eintreffen neuer Rohdaten angestoßen. Diese Ausgestaltung verhindert unnötige Verarbeitungsläufe in Abwesenheit von Änderungen und minimiert gleichzeitig die Latenz zwischen Bronze-Ingestion und Silver-Zustandsbildung. Innerhalb des Jobs sind Silver- und nachgelagerter Gold-Tasks sequenziell abhängig. Da je Business-Objekt ein eigenständiger Silver, beziehungsweise Gold-Job betrieben wird, sind Trigger, Verarbeitung und Fehlerbehandlung vollständig pro Objekttyp isoliert.

Löschoperationen werden in Silver physisch umgesetzt. Da Bronze die vollständige Ereignishistorie weiterhin vorhält, entsteht dadurch kein Informationsverlust im Gesamtsystem. Silver bleibt somit eine reine Repräsentation des aktuellen Fachzustands und muss keine zusätzliche Historien- oder Soft-Delete-Logik mitführen. Die Nachvollziehbarkeit historischer Änderungen ist jederzeit über die Bronze-Schicht gegeben.

5.3.3 Gold-Schicht als kuratierter Datenvertrag

Die Gold-Schicht bildet den kuratierten, exportorientierten Referenzzustand und dient als stabiler Datenvertrag für nachgelagerte Systeme. Während Silver primär die strukturelle Harmonisierung und Zustandsbildung adressiert, fokussiert Gold auf die Bereitstellung einer konsumentenorientierten Sicht. Dazu zählen die Reduktion technischer Felder, die Vereinheitlichung der Struktur sowie – sofern erforderlich – die domänenübergreifende Zusammenführung ausgewählter Informationen, beispielsweise zur Bildung einer konsistenten Master-Sicht über mehrere Quellsysteme hinweg.

Die konzeptionelle Notwendigkeit einer eigenständigen Gold-Schicht ergibt sich aus dem Prinzip der Schnittstellenstabilität. Silver kann interne Änderungen an Schemastruktur, Feldbezeichnungen oder Konsolidierungslogik aufnehmen, ohne dass nachgelagerte Konsumenten davon betroffen sind. Gold definiert dagegen ein explizit konsumentenorientiertes Schema, dessen Struktur nur im Rahmen bewusst gesteuerter und rückwärtskompatibel behandelter Änderungen modifiziert wird. Diese Entkopplung entspricht dem in ADR-004 dokumentierten Prinzip des stabilen Outbound-Datenvertrags. Externe Systeme wie die Asset Administration Shell oder nachgelagerte Analyseumgebungen binden sich ausschließlich an Gold-Schemata und sind damit von internen Verarbeitungsänderungen isoliert. Durch diese Abgrenzung entsteht eine klar definierte Systemgrenze, an der fachliche Vollständigkeit, strukturelle Konsistenz und exportorientierte Aufbereitung zusammenfallen.

Im Prototyp umfasst die Gold-Transformation typischerweise die gezielte Selektion fachlich relevanter Felder, die Eliminierung integrationstechnischer Metadaten wie *ingest_ts* oder *source_topic* sowie eine konsumentenseitig optimierte Feldstruktur. Der resultierende Gold-Datensatz repräsentiert damit nicht mehr ein einzelnes Quellereignis, sondern den fachlich vollständigen und strukturell stabilen Zustand eines Business-Objekts zum Zeitpunkt der letzten Änderung.

Die konkrete Job-Konfiguration zeigt, dass die Gold-Schicht mehrere Silver-Tabellen als Eingabe verarbeitet und damit eine explizite Konsolidierungsfunktion über Fachdomänen hinweg übernimmt. Im Prototyp fließen beispielsweise für ein Equipment-Business-Objekt strukturelle Stammdaten, ergänzende Klassifikationsinformationen sowie standortbezogene Zuordnungen aus jeweils eigenständigen Silver-Tabellen in die Gold-Transformation ein. Die Gold-Schicht ist damit nicht auf eine 1:1-Projektion aus einer einzigen Silver-Tabelle beschränkt, sondern realisiert eine gezielt gesteuerte, bereichsübergreifende Zusammenführung. Das resultierende Gold-Schema stellt einen fachlich vollständigen, domänenübergreifenden Referenzzustand bereit, der ohne Kenntnis der

internen Silver-Struktur konsumierbar ist. Dieses Zusammenführungsprinzip entspricht dem in Abschnitt 5.3 beschriebenen Konzept des kuratierten Datenvertrags, in welchem Gold-Tabellen nicht nur einzelne Felder abstrahieren, sondern auch die interne Aufteilung der fachlichen Information auf mehrere Teilschemas übernehmen.

Die Befüllung erfolgt ebenfalls inkrementell und business-key-basiert. Damit bleibt auch Gold deterministisch reproduzierbar und unabhängig von der Reihenfolge der Ereignis-Ankunft. Physische Löschungen werden analog zu Silver durchgeführt, da die Historie über Bronze verfügbar bleibt und Gold ausschließlich den aktuellen kuratierten Zustand repräsentieren soll. Durch die ACID-Eigenschaften von Delta Lake ist auch die Gold-Schicht transaktionssicher befüllbar, sodass zu keinem Zeitpunkt inkonsistente Zwischenzustände für externe Konsumenten sichtbar werden. Dies ist insbesondere relevant, da Gold-Tabellen direkt als Ausgangspunkt der Kafka-Publikation dienen und eine unvollständige Aktualisierung zu fehlerhaften Exportereignissen führen würde.

5.3.4 Änderungsgetriebene Publikation aus Gold nach Kafka

Die Weitergabe an nachgelagerte Systeme erfolgt im Prototyp ausschließlich aus der Gold-Schicht. Dieses Vorgehen entspricht der in ADR-004 festgelegten Strategie, konsolidierte und kuratierte Topics als stabilen Datenvertrag für externe Konsumenten bereitzustellen.

Die Publikation aus Gold wird änderungsgetrieben umgesetzt. Ereignisse werden nur dann erzeugt, wenn sich ein Datensatz in der Gold-Schicht tatsächlich verändert. Technisch wird hierfür der Delta Change Data Feed (CDF) genutzt, der Änderungen an Delta Tables als inkrementellen Strom bereitstellt. Der CDF wertet dazu das Delta-Transaktionslog aus, ohne die gesamte Tabelle neu zu lesen, und liefert Insert-, Update- und Delete-Operationen mit präziser Zuordnung zum auslösenden Commit. Diese transaktionsbasierte Ableitung stellt sicher, dass Änderungen vollständig, geordnet und ohne redundante Wiederholung erfasst werden.

Die abgeleiteten Operationen werden anschließend in entsprechende Gold-Topics publiziert. Jede publizierte Nachricht wird dabei mit dem fachlichen Business Key als Kafka-Nachrichtenschlüssel versehen. Dadurch werden redundante Re-Exports vermieden, eine effiziente Synchronisation nachgelagerter Systeme ermöglicht und *Log-Compaction*, das Zusammenfassen von Logs, auf den Gold-Topics unterstützt. Neue Konsumenten können damit den vollständigen, kuratierten Referenzzustand rekonstruieren, indem sie alle zusammengefassten Nachrichten pro Schlüssel einlesen und dabei keinen Zugriff auf interne Delta-Tabellen benötigen. Die Publikation folgt dem Prinzip der *At-Least-Once-Zustellung*. Dabei wird auch im Fehlerfall eine Nachricht erneut publiziert, um Datenverlust zu verhindern. Da Gold-Topics ausschließlich konsolidierte Referenzzustände transportieren, sind idempotente Konsumenten für diesen Anwendungsfall grundsätzlich gut realisierbar, da mehrfach empfangene Nachrichten für denselben Schlüssel stets denselben Zielzustand beschreiben.

Die aus der Gold-Schicht erzeugten Ereignisse unterscheiden sich grundlegend von den quellsystemseitigen Raw-Ereignissen. Während Raw-Topics unmittelbar aus operativen Systemen stammen und quellsystemspezifische Snapshot-Repräsentationen enthalten, repräsentieren Gold-Topics ausschließlich konsolidierte und harmonisierte Zustandsänderungen eines bereits gebildeten Referenzzustands. Tabelle 5.5 stellt diese architektonische Differenzierung systematisch gegenüber.

Tabelle 5.5: Architektonische Abgrenzung zwischen Raw- und Gold-Topics

Kriterium	**Raw Topics**	**Gold Topics**
Ursprung	Quellsystemseitige Snapshot-Ereignisse	Aus Gold-Delta-Tables abgeleitete Zustandsänderungen
Semantik	Quellsystemspezifisch	Systemübergreifend harmonisiert
Zustandsbildung	Keine Konsolidierung, reine Ereignisprojektion	Deterministisch konsolidierter Referenzzustand
Datenqualität	Quellsystemzustand, unverändert	Bereinigt, harmonisiert, konsumentenorientiert
Historienbezug	Einzelne Snapshot-Ereignisse	Änderung am kuratierten Referenzzustand
Delete-Handling	Explizite Delete-Ereignisse (Quellsystem)	Änderungsgetrieben, auf Basis des Gold-Zustands
Exportmechanismus	Durch Quellsystem publiziert	Inkrementell via Delta Change Data Feed
Reihenfolgegarantie	Partitionenbasiert (Kafka)	Deterministisch via Delta-Transaktionslog
Zielgruppe	Integrations- und Verarbeitungsschichten	Externe Konsumenten (z. B. AAS)
Rolle	Eingangspunkt der Daten- und Integrationsinfrastruktur	Outbound-Datenvertrag des Integrationskerns

Die Gegenüberstellung verdeutlicht, dass Gold-Topics keine zweite Form quellsystemseitiger Ereignisse darstellen, sondern die inkrementelle Projektion eines bereits konsolidierten Referenzzustands. Die Entkopplung zwischen Rohdatenerfassung, Zustandsbildung und externer Bereitstellung bleibt dadurch gewahrt und stellt sicher, dass nachgelagerte Systeme ausschließlich stabilisierte Zustandsänderungen verarbeiten.

5.4 Mapping-Service und Bereitstellung der Asset Administration Shell

Mit der im vorhergehenden Abschnitt beschriebenen änderungsgetriebenen Publikation aus der Gold-Schicht ist die datenorientierte Verarbeitungskette des Prototyps abgeschlossen. Die Gold-Schicht stellt einen konsolidierten, harmonisierten und exportorientierten Referenzzustand bereit, der als stabiler Datenvertrag für nachgelagerte Systeme dient. Für die in dieser Arbeit verfolgte Zielsetzung genügt jedoch die reine Bereitstellung eines kuratierten Datenzustands noch nicht. Das Asset Management Portal soll die integrierten Informationen nicht lediglich in einer generischen Datenstruktur verfügbar machen, sondern diese als standardkonforme semantische Repräsentation eines Assets im Sinne der Asset Administration Shell bereitstellen. Zwischen dem konsolidierten Gold-Zustand und der finalen AAS-Repräsentation ist daher ein eigener Transformationsschritt erforderlich, in dem die AAS-spezifische Strukturierung und Modellierung vorgenommen wird.

Dieser Verarbeitungsschritt wird im Prototyp bewusst nicht im Integrationskern selbst umgesetzt. Die vorgelagerte Medaillen-Architektur bleibt konsequent AAS-agnostisch und beschränkt sich auf die Aufnahme, Persistierung, Harmonisierung und Konsolidierung heterogener Quellsystemdaten. Dadurch wird verhindert, dass semantische Modellierungslogik bereits in den Datenfluss der Integrationsarchitektur hineinragt und dort zu einer Vermischung unterschiedlicher Verantwortlichkeiten führt. Die in Kapitel 4 beschriebene Trennung zwischen datenorientierter Zustandsbildung und AAS-bezogener Semantik wird damit auch in der technischen Umsetzung beibehalten. Die Outbound-Verarbeitung beginnt erst an der Schnittstelle, an der ein stabiler und fachlich verwertbarer Asset-Zustand bereits vorliegt.

Die zentrale Rolle übernimmt hierbei der Mapping-Service als eigenständiges Software-Artefakt der prototypischen Umsetzung. Er bildet die operative Transformationsinstanz zwischen den aus der Gold-Schicht publizierten Asset-Snapshot-Ereignissen und der konkreten Repräsentation dieser Zustände als Asset Administration Shell. Seine Aufgabe besteht nicht in einer bloßen Weiterleitung der erzeugten Nachrichten, sondern in der kontrollierten semantischen Abbildung eines AAS-agnostischen Referenzzustands auf die Zielstruktur der Verwaltungsschale. Der Mapping-Service ist damit derjenige Baustein, in dem die semantische Zielmodellierung technisch realisiert wird. Während die Daten- und Integrationsinfrastruktur einen konsistenten Zustand bereitstellt, entscheidet erst der Mapping-Service, wie dieser Zustand im Sinne der AAS zu interpretieren, zu strukturieren und auf konkrete AAS-Artefakte abzubilden ist.

Die Wahl dieser Architekturaufteilung ist für die entwickelte Systemlösung von zentraler Bedeutung. Würde die semantische Modellierung bereits in den Quellsystemen erfolgen, müssten diese Kenntnis über die Zielstruktur der Asset Administration Shell besitzen und Änderungen an der semantischen Modellierung unmittelbar nachvollziehen. Würde die AAS-Logik hingegen in den Integrationskern verlagert, wäre die Daten- und Integrationsinfrastruktur nicht mehr generisch nutzbar, sondern fest an eine spezifische Zielrepräsentation gekoppelt. Durch die Auslagerung in einen eigenständigen Mapping-Service wird die semantische Abbildung stattdessen an einem zentralen Ort gebündelt. Dies ermöglicht es, sowohl die vorgelagerte Datenintegration als auch die nachgelagerte semantische Zielstruktur unabhängig voneinander weiterzuentwickeln. Der Mapping-Service übernimmt somit nicht nur eine technische, sondern eine explizit architektonische Vermittlungsfunktion zwischen zwei bewusst getrennten Systemdomänen.

Die Bereitstellung der Asset Administration Shell realisiert dabei die in ADR-007 dokumentierte Entscheidung zur Nutzung des Eclipse BaSyx-Stacks als Referenzimplementierung und setzt die in ADR-008 festgelegte Mapping-Strategie operativ um. Ziel dieser Schicht ist es, die im Integrationskern gebildeten, konsolidierten und kuratierten Asset-Zustände in eine standardkonforme, semantisch strukturierte Repräsentation zu überführen und diese über eine einheitliche Zugriffsschicht bereitzustellen. Die AAS-Schicht ist dabei konsequent als Outbound-Komponente ausgeprägt. Sie konsumiert ausschließlich konsolidierte Exportdaten aus den Gold-Topics (ADR-004). Dadurch bleibt die in Kapitel 4 hergeleitete Verantwortungstrennung erhalten, nach der die Integrationsarchitektur einen stabilen Datenvertrag bereitstellt, während die semantische Modellierung und Publikation ausschließlich innerhalb der AAS-Schicht stattfindet.

Abbildung 5.5 zeigt den prototypischen Ablauf der Synchronisation zwischen den konsumierten Kafka-Ereignissen und den im BaSyx konfigurierten AAS-Ressourcen. Die Darstellung umfasst den vollständigen Verarbeitungspfad innerhalb des Mapping-Service. Diese reicht von der Aufnahme eingehender Ereignisse über die interne Aggregation und Existenzprüfung bis hin zu den konkreten Schreibzugriffen auf BaSyx-API, BaSyx-Repository, Submodel-Repository, AAS-Registry sowie der zugrundeliegenden BaSyx-

Datenbank. Der Ablauf ist im Prototyp bewusst so gestaltet, dass jeder verarbeitete Kafka-Impuls in einen wohldefinierten Zielzustand im BaSyx-Kontext überführt wird und damit eine reproduzierbare Aktualisierung von AAS und Submodellen ermöglicht.

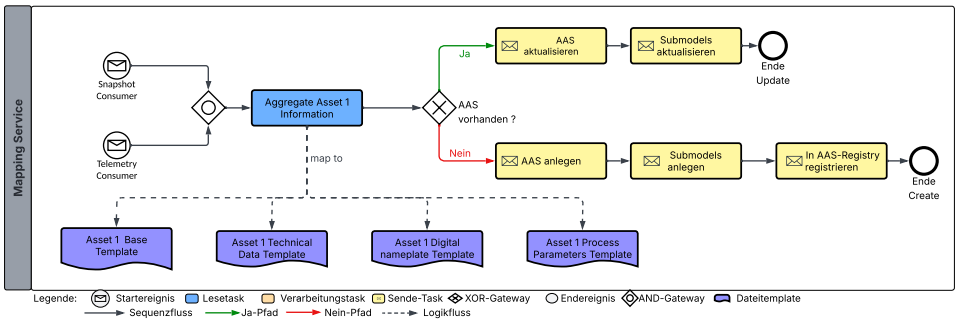

Abbildung 5.5: BPMN-Prozessdiagramm: Funktionsablauf der AAS-Schicht zur Erstellung und Aktualisierung von Assets und Submodellen (eigene Darstellung)

Die Verarbeitung eingehender Ereignisse ist im Prototyp in zwei Konsumpfade aufgeteilt, die in Abbildung 5.5 als *Snapshot Consumer* und *Telemetry Consumer* dargestellt sind. Snapshot-Nachrichten repräsentieren den kuratierten Referenzzustand eines Assets und bilden die Grundlage für die konsistente Strukturierung der AAS sowie der zugeordneten Submodells. Telemetrie-Nachrichten transportieren dagegen zustands- oder prozessnahe Aktualisierungen mit typischerweise höherer Frequenz und geringerem fachlichem Umfang. Die Trennung der Konsumpfade ist dabei nicht als fachliche Fragmentierung des Datenvertrags zu verstehen, sondern als technische Entkopplung heterogener Aktualisierungscharakteristika. Telemetrie-Updates sollen nicht durch umfangreichere Snapshot-Verarbeitungen blockiert werden, während Snapshot-Änderungen weiterhin die maßgebliche Referenz für strukturprägende und stammdatennahen Inhalte bilden. Eine Vermischung der Datenklassen auf Topic-Ebene wird vermieden. Die Zusammenführung erfolgt ausschließlich innerhalb des Mapping-Service auf Basis eines asset-zentrierten Bezugsschlüssels.

Der Mapping-Service bildet die zentrale Transformations- und Orchestrierungskomponente der AAS-Schicht. Seine Verantwortung liegt darin, eingehende Kafka-Nachrichten in eine AAS-konforme Zielrepräsentation zu projizieren und diese Zielrepräsentation idempotent mit dem BaSyx-Kontext zu synchronisieren. Der Service implementiert dabei keine zusätzliche fachliche Zustandsbildung im Sinne der Medaillen-Architektur (ADR-005), sondern setzt den im Gold bereitgestellten Referenzzustand als Ausgangspunkt voraus. Das Mapping ist folglich als Projektion aus einem konsolidierten Datenvertrag auf eine standardisierte semantische Repräsentation ausgeprägt. Diese Ausprägung ist entscheidend für die Abgrenzung der Verantwortlichkeiten. Während Gold die konsolidierte Sicht, inklusive Löschlogik und Harmonisierung, bereitstellt, beschränkt sich die AAS-Schicht auf die konsistente, standardkonforme Strukturierung und Bereitstellung.

Die Projektion erfolgt im Prototyp template-basiert. Die in Abbildung 5.5 angedeuteten Vorlagen repräsentieren stabile Zielstrukturen für AAS und Submodells, die unabhängig von einzelnen Quellsystemattributen definiert sind. In den Templates sind insbesondere die Hierarchie der Submodell-Elemente, die Bezeichner, Datentypen sowie die Zuordnung der Felder zu Submodell-Gruppen festgelegt. Der Mapping-Service befüllt diese

Templates ausschließlich mit den aus den konsumierten Ereignissen abgeleiteten Werten. Durch diese Trennung von Strukturdefinition und Wertableitung wird verhindert, dass quellsystemspezifische Feldnamen oder Integrationsmetadaten in die AAS-Struktur durchschlagen. Gleichzeitig entsteht eine konsistente Zielstruktur, die über REST-API und UI stabil adressierbar bleibt und sich kontrolliert erweitern lässt, ohne den Datenvertrag der Integrationsarchitektur zu verändern.

Die semantische Abbildung orientiert sich dabei an der Trennung zwischen Asset-Identifikation, fachlicher Beschreibung und laufzeitnaher Zustandsrepräsentation. Attribute, die der eindeutigen Identifizierbarkeit eines Assets dienen, werden in der Verwaltungsschale selbst geführt. Beschreibende Stammdaten — beispielsweise Maschinentyp, Seriennummer oder organisatorische Zuordnungen — werden in inhaltlich passende Submodelle überführt. Laufzeitnahe Informationen wie Betriebszustände oder maschinennahe Statuswerte werden separat modelliert, um ihre funktionale Rolle von den stabileren Stammdaten abzugrenzen. Die semantische Modellierung folgt damit nicht einfach der Reihenfolge der Eingangsattribute, sondern einer bewussten fachlichen Strukturierungslogik, die für gleichartige Assets konsistent wiederverwendbar bleibt.

Tabelle 5.6 zeigt exemplarisch, wie ausgewählte Attribute eines konsolidierten Gold-Snapshots im Mapping-Service auf Zielstrukturen der Asset Administration Shell abgebildet werden. Sichtbar wird insbesondere, dass nicht nur eine technische Feldzuordnung erfolgt, sondern eine inhaltliche Entscheidung darüber getroffen wird, welche Information auf welcher semantischen Ebene der Verwaltungsschale repräsentiert werden soll.

Tabelle 5.6: Beispielhafte Zuordnung konsolidierter Eingangsattribute im Mapping-Service

Eingangsattribut	**Zielstruktur**	**Zielattribut**	**Bemerkung**
`assetId`	Asset Administration Shell	`id`	Eindeutige Identifikation des Assets
`machineType`	Submodell Identification	`typeName`	Technische Klassifikation
`serialNumber`	Submodell Identification	`serialNumber`	Systemübergreifender Referenzschlüssel
`plant`	Submodell OperationalContext	`plant`	Organisatorische Werkszuordnung
`costCenter`	Submodell OperationalContext	`costCenter`	Betriebswirtschaftlicher ERP-Bezug
`operatingState`	Submodell OperationalState	`state`	Laufzeitnaher Betriebszustand
`lastStateChangeTs`	Submodell OperationalState	`timestamp`	Zeitpunkt der letzten Zustandsänderung
`manufacturer`	Submodell TechnicalDescription	`manufacturer`	Beschreibendes Stammdatenmerkmal

Die Tabelle verdeutlicht zugleich, dass der Mapping-Service eine bewusst zentralisierte semantische Steuerung ermöglicht. Änderungen an der Struktur eines Submodells, an Namenskonventionen oder an der Zuordnung einzelner Attribute müssen nicht in den Quellsystemen oder in der Integrationsarchitektur vorgenommen werden, sondern können im Transformationsbaustein gebündelt umgesetzt werden. Dies ist insbesondere für die Weiterentwicklung des Portals relevant, da die semantische Sicht auf ein Asset im

Verlauf eines Projekts typischerweise verfeinert, erweitert oder neu strukturiert wird. Die entwickelte Lösung trägt diesem Umstand dadurch Rechnung, dass die Mapping-Entscheidungen an einer Stelle konzentriert werden, an der sie kontrollierbar, nachvollziehbar und assetübergreifend konsistent fortgeschrieben werden können.

Der operative Ablauf im Mapping-Service beginnt mit einem Aggregationsschritt, der in Abbildung 5.5 als *Aggregate Asset* modelliert ist. In diesem Schritt werden die für ein Asset relevanten Informationen in einer Arbeitsrepräsentation zusammengeführt. Diese Arbeitsrepräsentation enthält insbesondere einen stabilen Asset-Identifier, der aus dem Gold-Datenvertrag abgeleitet wird, sowie die für die Template-Befüllung erforderlichen Attributgruppen. Die Aggregation ist dabei als technisches Bündelungsprinzip ausgeprägt und dient der deterministischen Steuerung der nachfolgenden Operationen. Sie ersetzt keine Konsolidierung, sondern stellt sicher, dass die Transformation unabhängig vom jeweiligen Eingangsereignis (Snapshot oder Telemetrie) auf einer einheitlichen Struktur operiert.

Auf Basis dieser Arbeitsrepräsentation wird anschließend geprüft, ob die AAS bereits im BaSyx-Kontext existiert (Entscheidung *Asset exists* in Abbildung 5.5). Diese Existenzprüfung erfolgt über die BaSyx-API anhand des eindeutigen Identifikators der AAS. Die Prüfung erfüllt im Prototyp eine doppelte Funktion. Einerseits dient sie der Ableitung des korrekten Lebenszykluspfads (Initialisierung versus Aktualisierung). Andererseits ist sie Bestandteil eines idempotenten Konsistenzmodells, das Kafka-bedingte Mehrfachzustellungen und Replays berücksichtigt. Durch die Verwendung stabiler Identifikatoren wird eine eindeutige Zuordnung zwischen Gold-Referenzzustand und AAS-Ressource sichergestellt, ohne dass transportbezogene Kafka-Metadaten (z. B. Offset oder Partition) in die Semantik einfließen.

Ist eine AAS-Instanz bereits vorhanden, wird der Update-Pfad ausgeführt. Dabei wird zwischen der Aktualisierung der AAS selbst und der Aktualisierung der zugehörigen Submodells unterschieden. Die AAS-Aktualisierung adressiert im Prototyp primär referenzielle Metadaten und Identitätsinformationen, während die fachlichen Detailinformationen in Submodells gekapselt sind. Telemetrie-Events führen in diesem Modell typischerweise zu gezielten Updates zustandsnaher Submodell-Elemente, ohne dass die gesamte AAS-Struktur reinitialisiert werden muss. Snapshot-Events können sowohl Änderungen in stammdatennahen Informationen als auch strukturell relevante Aktualisierungen in Submodelle auslösen. Der Update-Pfad ist damit so ausgeprägt, dass die AAS-Schicht Änderungen aus dem Gold-Datenvertrag deterministisch nachvollzieht und dabei die Abgrenzung zwischen Shell-Metadaten und Submodell-Inhalten konsistent einhält.

Existiert die AAS-Instanz nicht, wird der Create-Pfad ausgeführt. Zunächst wird die AAS als eigenständige Ressource erzeugt (*Create Asset*), anschließend werden die benötigten Submodelle erstellt (*Create Submodules*) und mit der AAS verknüpft. Diese Reihenfolge folgt dem Ressourcenmodell des BaSyx-Stacks, in dem AAS und Submodelle getrennt geführt werden und ihre Beziehung über explizite Referenzen hergestellt wird. Die explizite Erzeugung und Verknüpfung unterstützt zudem die Erweiterbarkeit des Prototyps, da zusätzliche Submodelle später ergänzt werden können, ohne die bestehende AAS-Instanz strukturell zu migrieren. Der Create-Pfad ist damit nicht als einmaliger Initialisierungsschritt verstanden, sondern als deterministische Herleitung eines Zielzustands aus dem Datenvertrag, die bei Bedarf reproduzierbar erneut ausgeführt werden kann.

Die gesamte Synchronisation ist im Prototyp auf Idempotenz ausgelegt. Da Kafka eine erneute Zustellung von Nachrichten (z. B. durch Consumer-Restarts oder Replay-Szenarien) nicht grundsätzlich ausschließt, muss die AAS-Schicht bei wiederholter Verarbeitung derselben fachlichen Änderung denselben Zielzustand herstellen. Dies wird durch

das Zusammenspiel aus stabilen Identifikatoren, Existenzprüfung und template-basierter Projektion erreicht. Wiederholte Create-Versuche werden durch die Existenzprüfung abgefangen, während wiederholte Update-Operationen aufgrund der deterministischen Ableitung aus dem Gold-Referenzzustand zu einem identischen Ergebnis führen. Partielle Fehler in einzelnen Schritten sind dadurch im prototypischen Konsistenzmodell durch erneute Verarbeitung korrigierbar, ohne eine zusätzliche zustandsbehaftete Orchestrierungskomponente einzuführen.

Die Nutzung des BaSyx-Stacks erfüllt dabei eine klar abgegrenzte Rolle. BaSyx übernimmt die standardkonforme Verwaltung, Persistierung und Zugänglichmachung der erzeugten AAS-Artefakte über AAS-Repository, Submodel-Repository, AAS-Registry und die zugrundeliegende BaSyx-Datenbank. Die eigentliche fachlich-semantische Transformationsleistung verbleibt jedoch im Mapping-Service. Der Beitrag der entwickelten Lösung besteht nicht in der bloßen Verwendung eines vorhandenen AAS-Frameworks, sondern in der Gestaltung eines vorgelagerten Software-Bausteins, der konsolidierte Asset-Zustände regelbasiert in eine standardkonforme semantische Zielstruktur überführt. Der BaSyx-Stack bildet die operative Infrastruktur, nicht jedoch den Ort der eigentlichen Transformationslogik.

Durch diese Aufteilung entsteht eine klare End-to-End-Verarbeitung von der Änderung im Quellsystem bis zur aktualisierten semantischen Repräsentation im Portal. Änderungen werden zunächst quellsystemnah erfasst, anschließend in der Bronze-Schicht protokolliert, in der Silver-Schicht deterministisch konsolidiert, in der Gold-Schicht kuratiert und schließlich über den Mapping-Service in AAS-Artefakte transformiert. Der Mapping-Service stellt damit das entscheidende Bindeglied zwischen dem datenzentrierten Integrationskern und der für Nutzer und nachgelagerte Anwendungen relevanten semantischen Sicht auf das Asset dar. Erst durch diesen Schritt wird aus einem konsolidierten Datenzustand eine nutzbare, standardisierte Asset-Repräsentation. Die prototypische Umsetzung zeigt somit, dass die in der Architekturkonzeption vorgesehene Trennung zwischen Datenintegration und semantischer Modellierung nicht nur theoretisch sinnvoll, sondern auch technisch realisierbar ist. Der Mapping-Service konkretisiert die in ADR-008 festgelegte Mapping-Strategie in Form eines eigenständigen Software-Artefakts und schafft damit die Voraussetzung, dass die transformierten Asset-Zustände als operative Asset Administration Shells innerhalb des Asset Management Portals bereitgestellt und genutzt werden können.

5.5 Validierung der Architektur anhand des Anwedungsfalls: Fertigungsmaschine

Die prototypisch umgesetzte Systemarchitektur wird in diesem Abschnitt anhand eines konkreten industriellen Anwendungsfalls strukturell validiert. Als Anwendungsfall dient eine 5-Achs-CNC-Fertigungsmaschine vom Typ *VCE-1250/5* der *PrecisionTech AG*, die im Produktionswerk P001 betrieben wird. Der Anwendungsfall ist repräsentativ für die in Kapitel 3 beschriebene Problemklasse des physischen Assets, welches in mehreren heterogenen Quellsystemen partiell abgebildet ist, wobei sich dessen Daten in Schnittstellencharakteristik, Aktualisierungsfrequenz und semantischer Granularität unterscheiden. Ziel ist es, aus diesen verteilten Teilinformationen eine vollständige, standardkonforme Verwaltungsschale zu erzeugen.

Gegenstand der Validierung ist nicht die funktionale Vollständigkeit im Sinne eines Systemtests, sondern die Überprüfung der in Kapitel 4 hergeleiteten Architekturprinzipien

anhand eines konkreten Datendurchlaufs. Die drei Kernaspekte, welche die Fähigkeit der Medaillien-Architektur heterogene Quelldaten schrittweise in ein schema-stabiles, quellsystemunabhängiges Austauschformat zu überführen, die Rolle des Mapping-Service als das semantisch verantwortliche Artefakt an der Grenze zwischen Daten- und AAS-Schicht und die durchgehende Nachvollziehbarkeit der Transformationskette vom Rohereignis bis zur persistierten Verwaltungsschale umfassen, stehen dabei im Fokus.

Tabelle 5.7 gibt einen Überblick über die vier beteiligten Quellsysteme, ihre Datenklassen, Schnittstellentypen und AAS-Zielsubmodelle. Die Gegenüberstellung verdeutlicht, dass die Submodelle *Digital Nameplate* und *TechnicalData* durch Daten aus zwei verschiedenen Quellsystemen gespeist werden und das *Process Parameters*-Submodell durch den Gold-Schicht-Stream-Join aus OPC UA und CAD/CAM konsolidiert wird. Erst die Aggregationslogik der Gold-Schicht macht diese Anforderung erfüllbar und belegt damit die strukturelle Notwendigkeit dieser Architekturschicht.

Tabelle 5.7: Quellsysteme und Zielsubmodelle im Anwendungsfall Fertigungsmaschine

Quellsystem	**Datenklasse**	**Schnittstelle**	**Zielsubmodell(e)**
ERP-System	Gerätestammdaten	Kafka (Event-Driven)	Digital Nameplate, TechnicalData
TDM-System	Technische Stammdaten	REST (Polling)	TechnicalData, Digital Nameplate
OPC UA	Maschinenzustandsdaten	OPC UA Subscription	Process Parameters
CAD/CAM-System	NC-Programmdaten	Kafka (Event-Driven)	Process Parameters

Die Trennung in zwei Ereignisklassen (Asset-Snapshot-Daten aus ERP und TDM einerseits sowie Telemetriedaten aus OPC UA und CAD/CAM andererseits) ist keine willkürliche Gruppierung, sondern folgt einer bewussten Architekturentscheidung. Stammdaten ändern sich selten und bestimmen die strukturelle Identität eines Assets. Telemetriedaten hingegen sind hochfrequent und beschreiben dessen transienten Zustand. Eine gemeinsame Verarbeitungspipeline würde entweder die Stammdatenverarbeitung durch unnötige Schreiboperationen belasten oder die Aktualität der Zustandsdaten durch die längere Konsolidierungskette der Snapshot-Daten einschränken. Die Trennung operationalisiert damit das in Abschnitt 4.4 hergeleitete Prinzip der Entkopplung nach Datenklasse.

5.5.1 Stammdaten der Fertigungsmaschine

Das ERP-System ist die primäre Quelle für die Gerätestammdaten der Fertigungsmaschine. Es verwaltet Assets unter dem fachlichen Begriff *Equipment* und ist über eine ereignisbasierte Kafka-Schnittstelle angebunden. Änderungen an relevanten Datensätzen werden durch den ERP-seitigen Producer unmittelbar an den Kafka-Broker publiziert. Ein dedizierter Consumer-Adapter abonniert die entsprechenden Topics und überführt eingehende Ereignisse ohne inhaltliche Filterung oder Transformation in der Bronze-Schicht. Dieser push-basierte Integrationsmechanismus unterscheidet das ERP-System grundlegend von polling-basierten Quellsystemen, da Änderungsereignisse nahezu verzögerungsfrei zur Verfügung stehen, ohne dass ein Worker-Prozess periodisch auf Änderungen prüfen muss. Der Originalzustand der Quelldaten bleibt vollständig erhalten und ermöglicht eine spätere Nachverarbeitung ohne erneuten Quellzugriff.

Tabelle 5.8 listet die für diesen Anwendungsfall relevanten ERP-Felder mit ihren Datentypen, anonymisierten Beispielwerten und der jeweiligen AAS-Zielzuordnung auf.

Tabelle 5.8: Relevante ERP-Felder der Fertigungsmaschine VCE-1250/5

Feldname	**Typ**	**Beispielwert**	**Zielsubmodell**
equipment_id	string	EQ-78421	Digital Nameplate
inventory_number	string	ANL-78421	Digital Nameplate
manufacturer	string	PrecisionTech AG	Digital Nameplate
manufacturer_serial_no	string	PT-2019-00423	Digital Nameplate
year_of_construction	int	2019	Digital Nameplate
month_of_construction	int	2	(nicht übernommen)
equipment_description	string	5-Achs-Bearb.zentrum VCE-1250/5	Digital Nameplate
country_of_manufa	string	CH	Digital Nameplate
plant_id	string	P001	Digital Nameplate
workplace_id	string	WP-042	Digital Nameplate
cost_nr	string	K-5520	(intern, entfällt)
class_id	string	27-01-04-01	Digital Nameplate
acquisition_date	date	2019-04-23	Digital Nameplate

Die Feldauswahl in Tabelle 5.8 verdeutlicht ein für ERP-Systeme typisches Charakteristikum, indem das Quellsystem betriebswirtschaftliche und technische Informationen in einer gemeinsamen Datenstruktur verwaltet. Felder wie *cost_ nr* und *vendor_ number* sind für Beschaffungs- und Buchhaltungsprozesse relevant, besitzen jedoch kein Äquivalent im IDTA-Standardsubmodell *Digital Nameplate*. Das Feld *month_ of_ construction* illustriert einen weiteren Fall semantischer Redundanz, denn der IDTA-Standard sieht ausschließlich das Konstruktionsjahr vor, eine monatsgenaue Angabe ist nicht modelliert.

Die Verantwortung für diese Filterentscheidungen liegt bewusst in der Silver-ETL-Schicht und nicht im Mapping-Service. Diese Zuordnung folgt dem in Abschnitt 4.5 beschriebenen Prinzip der Schichtenverantwortung. Die Silver-Schicht erzeugt ein schema-stabiles, auf AAS-Relevanz reduziertes Ereignisformat, sodass der Mapping-Service ausschließlich semantische Zuordnungen vornimmt und nicht mit quellsystemspezifischen Artefakten konfrontiert wird. Die Filterlogik ist damit explizit lokalisiert und unabhängig von der Mapping-Konfiguration änderbar.

Das TDM-System verwaltet die maschinenspezifischen technischen Eigenschaften, die steuerungsrelevante Parameter, Werkzeugmagazin-Konfigurationen, Bestell- und Artikelinformationen sowie Versionsstände von Hard- und Software beinhalten. Im Unterschied zum ERP-System, das über eine ereignisbasierte Kafka-Schnittstelle angebunden ist, stellt das TDM-System ausschließlich eine REST-Schnittstelle bereit. Die Anbindung erfolgt daher über einen Hangfire-gesteuerten Worker-Prozess, der die Schnittstelle periodisch abfragt und eingehende Datensätze anhand eines Zeitstempelvergleichs auf Änderungen prüft. Erkannte Änderungen werden als atomare Bronze-Events persistiert. Dieser polling-basierte Ansatz ist für das TDM-System angemessen, da sich dessen Daten ausschließlich bei klar abgrenzbaren technischen Interventionen ändern, etwa bei Softwareaktualisierungen der Maschinensteuerung, Hardwarewechseln oder Magazin-Umrüstvorgängen. Die effektiv übertragene Ereignisrate ist damit so gering, dass der

Overhead periodischen Pollings gegenüber einer ereignisbasierten Integration vernachlässigbar ist.

Da TDM-Daten ausschließlich technische Informationen ohne betriebswirtschaftliche Anteile enthalten, ist die Filtermenge in der Silver-Schicht geringer als beim ERP-System. Dabei haben alle TDM-Felder ein Äquivalent in den Zielsubmodellen, welche nachfolgende Tabelle 5.9 veranschaulicht.

Tabelle 5.9: Relevante TDM-Felder der Fertigungsmaschine VCE-1250/5

Feldname	**Typ**	**Beispielwert**	**Zielsubmodell**
product_id	string	VCE-1250-5A	Digital Nameplate
article_number	string	4012-VCE-1250	Digital Nameplate
order_code	string	VCE1250-5EU	Digital Nameplate
control_unit	string	SinuTech NC 840D	TechnicalData
magazine_type	string	Kettenmagazin-60	TechnicalData
presetting_postprocessor	string	TDM-PostPro v4.2	TechnicalData
tool_max_diameter	double	200.0 (mm)	TechnicalData
tool_max_length	double	400.0 (mm)	TechnicalData
tool_max_weight	double	10.5 (kg)	TechnicalData
tool_count	int	60	TechnicalData
hardware_revision	string	Rev-B	Nameplate + TechnicalData
firmware_revision	string	V2.4.1	Nameplate + TechnicalData
software_version	string	CNC-SW 9.2.0	Nameplate + TechnicalData

Besondere Aufmerksamkeit verdienen die Felder *hardware_revision*, *firmware_revision* und *software_version*. Beide IDTA-Standards, *Digital Nameplate* und *TechnicalData*, sehen diese Versionsangaben vor, sodass ein einzelnes Gold-Feld in zwei Zielsubmodelle abgebildet werden muss. Dies stellt eine strukturelle Anforderung an den Mapping-Service dar, die über eine einfache 1:1-Zuordnung hinausgeht. Die deklarative Mapping-Konfiguration adressiert diesen Fall durch eine 1:n-Regelstruktur. Hierbei wird dasselbe Quellfeld auf mehrere Zieleinträge referenziert, ohne dass die Transformationslogik im Mapping-Service dafür angepasst werden muss. Das *Digital Nameplate* führt diese Felder als versionierende Identifikationsmerkmale, das *TechnicalData*-Submodell als Teil des *GeneralInformation*-Blocks. Die Konsistenz beider Submodelle ist damit durch die Mapping-Konfiguration garantiert, nicht durch eine laufzeit-basierte Synchronisationslogik.

Die Fertigungsmaschine stellt ihre Zustandsdaten reaktiv über einen OPC UA-Server bereit. Ein dedizierter OPC UA-Client registriert für die relevanten Knoten *MonitoredItems* mit konfigurierbarem Abtastintervall und Deadband-Schwellwert. Wertänderungen werden vom Server aktiv an den Client notifiziert und unmittelbar als atomare Ereignisse auf der Bronze-Schicht publiziert. Der Anwedungsfall vereint damit alle drei im Architekturkapitel beschriebenen Integrationsmuster in einem einzigen Asset. Die Anbindung des ERP-Systems und des CAD/CAM-Systems erfolgt über Kafka-Events. Hingegen sind das TDM-System über REST-Polling mit einem Hangfire-Worker und die Maschinendaten mittels OPC-UA über serverseitige Subscriptions integriert. Während TDM-Events

durch einen Worker-Prozess periodisch ausgelöst werden, entsteht ein OPC UA-Bronze-Event ausschließlich dann, wenn sich ein überwachter Zustand tatsächlich ändert. Die Ereignisrate ist damit direkt an das Maschinenverhalten gekoppelt, nicht an einen konfigurierten Timer.

Um die Ende-zu-Ende-Latenz vom Maschinenereignis bis zur Bronze-Persistenz messbar zu halten, trägt jeder OPC UA-Bronze-Event zwei Zeitstempel. Dabei enthält *event_ts* den originalen OPC UA-Zeitstempel der Wertänderung an der Maschinensteuerung und *ingestion_ts* den Zeitpunkt der Persistenz in der Bronze-Schicht. Ihre Differenz ist ein direktes Maß für die Ingestion-Latenz und ermöglicht eine nachträgliche Qualitätsbewertung der Echtzeitfähigkeit des Systems ohne Eingriff in die Verarbeitungslogik.

Tabelle 5.10: Abonnierte OPC UA-Knoten der Fertigungsmaschine

Node ID	**Typ**	**Beschreibung**	**Beispielwert**
ns=2;s=MachineStatus	Int16	Maschinenstatus (codiert)	2
ns=2;s=MachineStatus2	Boolean	Sekundärer Statusindikator	true
ns=2;s=NCProgName	String	Aktives NC-Programm	_N_00781234_MPF
ns=2;s=ActiveBlock	String	Aktuell ausgeführter NC-Satz	N1200 G01 X-245.5 Y12.8 F800
ns=2;s=FeedRateOverride	Float	Vorschub-Override (%)	8.0
ns=2;s=NCAssembly	Int32	NC-Bausteinnummer	75285
ns=2;s=ActiveToolNumber	String	Aktive Werkzeug-ID	T042

Die abonnierten Knoten umfassen die Statusvariablen *MachineStatus* und *MachineStatus2*. Während der Erste Knoten den primären Betriebszustand als ganzzahligen Code übermittelt, ist der Zweite ein boolescher Indikator, der denselben Zustand als binäres Flag spiegelt und für die Kompatibilität mit Downstream-Systemen mitgeführt wird. Beide Statusvariablen werden im Mapping-Service als Properties in die *ResourceParameters*-Sub-Collection des Process Parameters-Submodells übernommen: *MachineStatus* als *xs:int* und *MachineStatus2* als *xs:boolean*.

Der ganzzahlige Statuscode wird ohne Auflösung durch Silver- und Gold-Schicht als *machine_status* weitergereicht und im Mapping-Service direkt als *xs:int*-Property in die *ResourceParameters*-Sub-Collection übernommen. Tabelle 5.11 dokumentiert die Bedeutung der einzelnen Statuscodes als Referenz für die Interpretation der AAS-Werte zur Laufzeit.

Tabelle 5.11: Statuscode-Mapping für den OPC UA-Maschinenstatus

Code	**Statusbezeichnung**	**Bedeutung**
0	Stillstand	Maschine steht still, kein aktiver Betrieb
1	Einrichten	Rüst- oder Einrichtbetrieb aktiv
2	Laeuft	NC-Programm wird ausgeführt
3	Wird_beladen	Werkstückwechsel oder Beladungsvorgang
4	Stoerung	Maschinenstörung oder Fehler

Das CAD/CAM-System verwaltet die NC-Programmdaten der Fertigungsmaschine. Es stellt für jedes NC-Programm eine Programmbeschreibung, sowie eine simulierte Zykluszeit bereit, die als Grundlage für die obligatorischen IDTA-02031-1-Felder *ProcessDescription* und *PlannedProcessTime* dient. Das CAD/CAM-System verfügt über native Ereignisfähigkeit und ist wie das ERP-System über Kafka-Raw-Topics in die Daten- und Integrationsinfrastruktur eingebunden. Änderungen an einem NC-Programm führen zur Publikation eines vollständigen Snapshot-Ereignisses und durchlaufen anschließend die Medaillen-Pipeline. Die von CAM gelieferten Felder sind in Tabelle 5.12 dokumentiert.

Tabelle 5.12: Relevante CAD/CAM-Felder des NC-Programms für die Fertigungsmaschine VCE-1250/5

Feldname	**Typ**	**Beispielwert**	**Zielsubmodell**
ncprog_name	string	_N_00781234_MPF	(Korrelationsschlüssel)
program_description	string	5-Achs-Fräsbearbeitung: Schruppen...	Process Parameters
simulated_cycle_time_s	double	863.0	Process Parameters

Das Feld *ncprog_name* dient in der Gold-Schicht als Korrelationsschlüssel für den Stream-Join mit dem OPC UA-Telemetriestrom. Die *simulated_cycle_time_s* wird in der Silver-Schicht von Sekunden in eine ISO-8601-konforme Dauer (*xs:duration*) umgewandelt und als *cam.planned_process_time* in den TELEMETRY-Gold-Event übernommen.

5.5.2 Ereignisverarbeitung in der Medaillen-Architektur

Die vier Quellsysteme liefern strukturell heterogene Daten, die in der Medaillen-Architektur schrittweise in einen stabilen, AAS-kompatiblen Zustand überführt werden. Jede Verarbeitungsstufe übernimmt dabei eine klar abgegrenzte Verantwortung, sodass Operationen ausschließlich in der dafür vorgesehenen Schicht getroffen werden.

Für jedes der vier Quellsysteme existiert eine dedizierte Bronze-Tabelle, in die eingehende Ereignisse ohne jede inhaltliche Transformation persistiert werden. Alle Bronze-Events teilen eine gemeinsame Envelope-Struktur aus systemübergreifenden Metadaten-Feldern, während der Nutzlast-Anteil der jeweiligen Originalstruktur des Quellsystems entspricht und verweist damit auf die in Tabellen 5.8, 5.9, 5.10 und 5.12 dokumentierten Quellfelder. Tabelle 5.13 stellt die Envelope-Felder der vier Quellsysteme gegenüber.

Tabelle 5.13: Bronze-Event-Envelope im Vergleich der vier Quellsysteme

Feld	Typ	ERP	TDM	OPC UA	CAD/CAM
`event_ts`	timestamp	2025-11-03T06:42:11Z	2025-11-03T06:43:05Z	2025-11-03T07:30:15.412Z	2025-11-03T07:25:00Z
`ingestion_ts`	timestamp	n. v.	n. v.	2025-11-03T07:30:15.748Z	n. v.
`source`	string	`ERP`	`TDM`	`OPC_UA`	`CAM`
`equipment_id`	string	EQ-78421	EQ-78421	EQ-78421	n. v.*
`[Nutzlast]`	object	gem. Tab. 5.8	gem. Tab. 5.9	gem. Tab. 5.10	gem. Tab. 5.12

* CAD/CAM-Events verwenden *ncprog_ name* als primären Identifikator. Der Asset-Bezug wird in der Gold-Schicht über den Stream-Join mit dem OPC UA-Telemetriestrom hergestellt.

Das Feld *ingestion_ ts* wird ausschließlich im OPC UA-Event mitgeführt. Seine Differenz zu *event_ ts* von im vorliegenden Beispiel rund 336 ms quantifiziert die Ingestion-Latenz vom Maschinenereignis bis zur Bronze-Persistenz und ermöglicht eine nachträgliche Bewertung der Echtzeitfähigkeit ohne Eingriff in die Verarbeitungslogik. Das Feld *equipment_ id* ist in ERP, TDM und OPC UA identisch belegt und fungiert als systemübergreifender Korrelationsschlüssel für den ASSET_SNAPSHOT-Pfad. Das CAD/CAM-System verwendet stattdessen *ncprog_ name* als primären Identifikator, da seine Ereignisse NC-Programm-spezifisch und nicht direkt Equipment-spezifisch sind.

Die Silver-Schicht überführt die vier heterogenen Bronze-Strukturen in ein einheitliches, schema-stabiles Format. Die angewendeten Transformationen lassen sich in fünf Kategorien einteilen, die in Tabelle 5.14 quellsystemübergreifend gegenübergestellt sind.

Tabelle 5.14: Silver-Schicht-Transformationen: Kategorisierung je Quellsystem

Quelle	**Transformationstyp**	**Beispiel**
ERP	Feldumbenennung	`equipment_id` → `asset_id`, `manufacturer` → `manufacturer_name`, `country_of_manufa` → `country_of_origin`, u. a.
ERP	Feldeliminierung	`cost_nr`, `vendor_number`, `language_key`, `month_of_construction`
ERP	Typkonvertierung	`year_of_construction`: `int` → `string`
TDM	Feldeliminierung	`machine_id` (interner Bezeichner, kein AAS-Äquivalent)
TDM	Einheit-Explizierung	`tool_max_diameter` → `tool_max_diameter_mm`, analog für `tool_max_length` und `tool_max_weight`
OPC UA	snake_case-Norm.	`NCProgName` → `ncprog_name`, `ActiveBlock` → `act_block`, `FeedRateOverride` → `fr_ovr`, `NCAssembly` → `nc_assembly`, u. a.
CAD/CAM	Feldumbenennung	`ProgramName` → `ncprog_name`, `ProgramDescription` → `program_description`
CAD/CAM	Typkonvertierung	`simulated_cycle_time_s`: `double` → `xs:duration` (ISO-8601-Konv.)

Die Kategorisierung in Tabelle 5.14 zeigt, dass ERP die höchste Transformationsdichte aufweist. Dort fallen Umbenennung, Filterung und Typkorrektur gleichzeitig an, weil das ERP-System betriebswirtschaftliche und technische Felder in einer gemeinsamen Struktur verwaltet. TDM erfordert primär Einheits-Explizierungen, da die Quelldaten physikalische Größen ohne Einheitenangabe liefern. Das OPC UA-System erfordert primär eine snake_case-Normalisierung der CamelCase-Knotenbezeichnungen in einheitliche Feldbezeichner. Das CAD/CAM-System erfordert neben einer Feldumbenennung eine Typkonvertierung der simulierten Zykluszeit von Sekunden in eine ISO-8601-konforme Dauer. Die Verantwortung für alle diese Transformationen liegt bewusst in der Silver-Schicht und nicht im Mapping-Service, der dadurch vollständig auf semantische Zuordnungen beschränkt bleibt.

Die Gold-Schicht übernimmt eine doppelte Aufgabe. Er abstrahiert von den quellsystemspezifischen Silver-Events und erzeugt einen stabilen, AAS-kompatiblen Datenvertrag, der den nachgelagerten Mapping-Service von Integrationsdetails entkoppelt.

Für die Fertigungsmaschine werden auf Basis der vier Quellsysteme zwei semantisch unterschiedliche Ereignistypen, wie in ADR-004 beschrieben, definiert. *ASSET_SNAPSHOT* fasst die Stammdaten aus ERP und TDM zu einer konsolidierten Ereignisstruktur zusammen, die als Datenbasis für die Submodelle *Digital Nameplate* und *TechnicalData* dient. *TELEMETRY* aggregiert OPC UA-Maschinendaten und CAD/CAM-Programmdaten und bildet im vorliegenden Prototyp die Grundlage für das *Process Parameters*-Submodell. Dabei ist anzumerken, dass IDTA 02031 das *ProcessParameters*-Submodell explizit auf Basis eines PPR-Modells (Product–Process–Resource) konzipiert und dafür vier obligatorische Sub-Collections vorsieht: *ProductParameters*,

ProcessParameters, *ResourceParameters* sowie *ProcessBoM*. Diese sind semantisch auf unterschiedliche Quellen ausgelegt: Ressourcenparameter entstammen der Maschinensteuerung (OPC UA), Prozessparameter wie NC-Programm-spezifische Konfigurationswerte stammen typischerweise aus CAM- oder MES-Systemen, und Produktparameter aus ERP- oder PLM-Systemen. Der vorliegende Prototyp implementiert davon die SubCollections *ResourceParameters* und *ProcessParameters* sowie die innerhalb jedes *Process__00__*-Eintrags obligatorischen Felder *ProcessDescription* (MultiLanguageProperty) und *PlannedProcessTime* (Property). Das CAM-System verfügt über native Ereignisfähigkeit und ist wie das ERP-System über Kafka-Raw-Topics in die Daten- und Integrationsinfrastruktur eingebunden. Änderungen werden als vollständige Snapshot-Ereignisse publiziert und durchlaufen die Medaillen-Pipeline von Bronze über Silver bis zur Gold-Schicht. Die Gold-Schicht aggregiert den OPC UA-Telemetriestrom und den CAM-Gold-Strom zu einem gemeinsamen *TELEMETRY*-Event. Als Korrelationsschlüssel dient *ncprog_name*, das den aktiv ausgeführten NC-Programmnamen eindeutig identifiziert. Die Trennung in zwei Ereignistypen ist aus mehreren Gründen architektonisch motiviert. Erstens unterscheiden sich die Änderungscharakteristika grundlegend, da Stammdaten sich selten ändern und eine Aggregation über mehrere Quellsysteme erfordern, während die im Prototyp erfassten Betriebsdaten hochfrequent sind und vorrangig aus der Maschinensteuerung stammen. Zweitens werden durch die Führung beider Ereignistypen durch die Medaillen-Architektur konkurrierende Schreibzugriffe auf das BaSyx-Repository vermieden. Da ausschließlich der Mapping-Service als einziger Konsument beider Gold-Topics auf BaSyx schreibt, können *ASSET_SNAPSHOT*- und *TELEMETRY*-Updates nicht gleichzeitig auf dieselbe AAS-Instanz treffen. Ein direkter Schreibpfad aus dem OPC UA-Adapter würde diesen Schutz aufheben und zu Race Conditions bei der AAS-Aktualisierung führen. Drittens ermöglicht der gemeinsame Korrelationsschlüssel *asset_id*, der in beiden Gold-Pfaden präsent ist, dem Mapping-Service ein konsistentes Lifecycle-Management der AAS-Instanz. Ob ein eingehendes Event eine neue AAS anlegen, eine bestehende aktualisieren oder löschen soll, wird anhand desselben Identifikators entschieden, unabhängig davon, ob der Auslöser ein Stammdaten- oder ein Telemetrie-Event war.

Die Erzeugung des *ASSET_SNAPSHOT*-Events erfordert eine quellübergreifende Aggregation. Während das ERP-System betriebswirtschaftlich verwaltete Stammdaten, wie Inventarnummer, Werk, Herstellungsland liefert, steuert das TDM-System technische Gerätedaten, wie Steuerungseinheit, Werkzeugmagazin, Revisionsstände bei. Als Korrelationsschlüssel dient das in der Silver-Schicht vereinheitlichte Feld *asset_id*. Trifft ein Silver-Event nur eines der beiden Quellsysteme ein, zieht die Aggregationslogik den zuletzt persistierten Stand des anderen Systems hinzu, sodass stets ein vollständiger *ASSET_SNAPSHOT* emittiert wird.

Tabelle 5.15 zeigt den vollständigen Feldsatz mit expliziter Quellzuordnung. Von insgesamt 19 Nameplate-Feldern stammen vierzehn aus dem ERP- und fünf aus dem TDM-System. Dies belegt konkret die Notwendigkeit der systemübergreifenden Aggregation, die in der Gold-Schicht architektonisch verankert ist.

Tabelle 5.15: Gold-Event ASSET_SNAPSHOT: Feldsatz mit Quellzuordnung

Gold-Feld	**Wert**	**Quelle**
nameplate		
manufacturer_name	PrecisionTech AG	ERP
product_designation	5-Achs-Bearbeitungszentrum VCE-1250/5	ERP
serial_number	PT-2019-00423	ERP
year_of_construction	2019	ERP
country_of_origin	CH	ERP
inventory_number	ANL-78421	ERP
plant	P001	ERP
classification_code	27-01-04-01	ERP
classification_system	ECLASS	ERP
classification_system_version	14.0	ERP
address_street	Industriestraße 1	ERP
address_zip	1234	ERP
address_city	Musterstadt	ERP
address_country	AT	ERP
article_number	4012-VCE-1250	TDM
order_code	VCE1250-5EU	TDM
hardware_revision	Rev-B	TDM
firmware_revision	V2.4.1	TDM
software_version	CNC-SW 9.2.0	TDM
technical_data		
control_unit	SinuTech NC 840D	TDM
magazine_type	Kettenmagazin-60	TDM
presetting_postprocessor	TDM-PostPro v4.2	TDM
tool_max_diameter_mm	200.0	TDM
tool_max_length_mm	400.0	TDM
tool_max_weight_kg	10.5	TDM
tool_count	60	TDM

Der *TELEMETRY*-Event folgt einem anderen Entstehungsmuster als der *ASSET_SNAPSHOT*. Das CAM-System ist wie das ERP-System über Kafka-Raw-Topics in die Daten- und Integrationsinfrastruktur eingebunden und durchläuft die Medaillen-Pipeline (Bronze → Silver → Gold). Die Gold-Schicht aggregiert den OPC UA-Telemetriestrom und den CAM-Gold-Strom. Als Korrelationsschlüssel dient *nc-prog_name*, das den aktiven NC-Programmnamen aus der OPC UA-Telemetrie mit dem zugehörigen CAM-Gold-Eintrag verknüpft. Dieses Muster entspricht strukturell der quellübergreifenden Aggregation von ERP und TDM im *ASSET_SNAPSHOT*. Die Gold-Schicht reichert das Ergebnis um Metadaten (*asset_id*, *event_type*, *gold_timestamp*) an. Die semantische Bedeutung der Felder wurde in der Silver-Schicht durch snake_case-Normalisierung hergestellt. Die Gold-Schicht verändert keine Werte mehr, sondern

schreibt ausschließlich den Schnittstellenvertrag fest.

Tabelle 5.16 zeigt den vollständigen Feldsatz mit Quellzuordnung, Rückverweis auf das Bronze-Ursprungsfeld und die angewendete Transformation. Die OPC UA-Felder wurden durch snake_case-Normalisierung aus den Knotenbezeichnungen abgeleitet. Die CAM-Felder werden über einen Gold-Schicht-Stream-Join via *ncprog_name* eingebracht.

Tabelle 5.16: Gold-Event TELEMETRY: Feldsatz mit Silver-Transformation

Gold-Feld	**Wert**	**Bronze-Ursprungsfeld**	**Trans-formation**
OPC UA (Maschinensteuerung)			
machine_status	2	MachineStatus	snake_case-Norm.
machine_status2	true	MachineStatus2	snake_case-Norm.
ncprog_name	_N_00781234_MPF	NCProgName	snake_case-Norm.
act_block	N1200 G01 X-245.5 Y12.8 F800	ActiveBlock	snake_case-Norm.
fr_ovr	8.0	FeedRateOverride	Abkürzung
nc_assembly	75285	NCAssembly	snake_case-Norm.
CAM-System (Gold-Join via ncprog_name*)*			
cam.process_description	5-Achs-Fräsb. VCE-1250/5	ProgramDescription	Gold-Join & Norm.
cam.planned_process_time	PT14M23S	SimulatedCycleTime_s	Gold-Join & ISO-8601-Konv.
Metadaten			
event_ts	2025-11-03T07:30:15.412Z	event_ts	n. v.
ingestion_ts	2025-11-03T07:30:15.748Z	ingestion_ts	n. v.
source_table	silver_telemetry	–	Gold-Metadaten

5.5.3 Mapping-Service: Semantische Modellierung und Submodellaufbau

Der Mapping-Service bildet die letzte Transformationsstufe der Verarbeitungskette: Er konsumiert die typenreinen Gold-Events und überführt diese in strukturkonforme AAS-Submodelle gemäß den jeweiligen IDTA-Spezifikationen. Die zentrale Entwurfsentscheidung besteht darin, die semantische Abbildungslogik vollständig aus dem ausführbaren Quellcode auszulagern und in ein deklaratives, versionierbares Mapping-Artefakt zu überführen. Dieses Artefakt beschreibt die Zuordnung jedes Gold-Feldes zu einem AAS-Submodell-Element inklusive Elementtyp und semantischer Referenz und kann unabhängig vom Mapping-Service selbst weiterentwickelt werden. Die Konsequenz dieser

Trennung ist, dass Änderungen an Submodellstrukturen, Feldnamen oder normativen Referenzen ausschließlich Konfigurationsarbeit erfordern und keine Neukompilierung der Komponente nach sich ziehen.

Für die Wertzuweisung innerhalb der Mapping-Regeln wird *Jinja2* als Templating-Engine eingesetzt. Das *value*-Attribut jeder Regel akzeptiert Jinja2-Ausdrücke, die zur Laufzeit gegen das eingehende Gold-Event ausgewertet werden. Dieser Mechanismus erlaubt feldspezifische Wert-Transformationen wie Typkonvertierungen, Formatierungen oder Fallback-Definitionen für optionale Felder vollständig innerhalb der Konfigurationsschicht, ohne den Mapping-Service-Quellcode zu modifizieren. Jinja2 wirkt damit als interpretierbare Erweiterungsschicht zwischen dem starren Gold-Datenvertrag und den normativen Anforderungen der Ziel-Submodelle. Ein repräsentativer Ausschnitt dieser Mapping-Konfiguration für das Digital Nameplate ist in Anhang B (Listing B.1) dokumentiert.

Das Listing illustriert zwei konkrete Ausprägungen des Jinja2-Mechanismus: Die Typkonvertierung / *string* für *YearOfConstruction* ist notwendig, weil das IDTA-Namensschild den Baujahr-Wert als *xs:string* vorschreibt, während das Gold-Event ihn als numerischen Typ führt. Der Fallback-Ausdruck / *default('n/a')* bei *HardwareVersion* behandelt die Möglichkeit, dass das TDM-System keinen Revisionseintrag liefert, ohne eine Ausnahmebehandlung im Service-Code zu erfordern. Beide Fälle sind Belege dafür, dass die Grenze zwischen der normativen Struktur des Ziel-Submodells und dem konkreten Feldsatz des Gold-Events nicht immer spannungsfrei ist und dass die Jinja2-Schicht diese Impedanz zielgerichtet auflöst.

Die beschriebene Konfigurationsstruktur entspricht dem in Abschnitt 4.5 dargelegten Prinzip der Erweiterbarkeit durch Konfiguration. Auf ihrer Grundlage erzeugt der Mapping-Service für die Fertigungsmaschine drei IDTA-konforme Submodelle.

Das Digital Nameplate-Submodell folgt dem IDTA-Standard *Digital Nameplate for Industrial Equipment* in Version 3.0.1 (Oktober 2025). Tabelle 5.17 zeigt die vollständige Quell-Ziel-Zuordnung. Das Feld *nameplate.inventory_number* wird als Erweiterungselement ohne normierte *semanticId* aufgenommen, da es einen unternehmensinternen Bezeichner ohne standardisiertes Äquivalent darstellt. Die AAS-Spezifikation erlaubt dies explizit.

Tabelle 5.17: Mapping-Tabelle: Gold-Felder zu Digital Nameplate-Elementen

Gold-Feld / Quelle	**AAS-Element (idShort)**	**Typ**
(berechnet: machine_libref + uuid)	URIOfTheProduct	P
nameplate.manufacturer_name	ManufacturerName	MLP
nameplate.product_designation	ManufacturerProductDesignation	MLP
nameplate.address_street	AddressInformation/Street	MLP
nameplate.address_zip	AddressInformation/Zipcode	MLP
nameplate.address_city	AddressInformation/CityTown	MLP
nameplate.address_country	AddressInformation/NationalCode	MLP
nameplate.order_code	OrderCodeOfManufacturer	P
nameplate.article_number	ProductArticleNumberOfManufacturer	P
nameplate.serial_number	SerialNumber	P
nameplate.year_of_construction	YearOfConstruction	P
nameplate.country_of_origin	CountryOfOrigin	P
nameplate.hardware_revision	HardwareVersion	P
nameplate.firmware_revision	FirmwareVersion	P
nameplate.software_version	SoftwareVersion	P
nameplate.inventory_number	InventoryNumber	P

MLP = MultiLanguageProperty, P = Property

Markings (SML): im Template vorgesehen, im Prototyp nicht befüllt (markings = null)

Die vollständige JSON-Serialisierung des erzeugten Digital Nameplate-Submodells ist in Anhang B (Listing B.2) dokumentiert.

Das serialisierte Submodell wird unter der Kennung *urn:example:sm:EQ-78421:nameplate* im BaSyx Submodel Repository persistiert. Die Registrierung im AAS Registry und AAS Discovery Service erlaubt es nachgelagerten Systemen, das Submodell anhand der Asset-ID aufzufinden, ohne die konkrete Endpunkt-URL zu kennen. Dies ist eine Voraussetzung für die lose Kopplung im verteilten AAS-Ökosystem.

Das TechnicalData-Submodell folgt dem IDTA-Standard *Generic Frame for Technical Data for Industrial Equipment* (IDTA 02003, Version 2.0). Der Standard definiert den Pflichtabschnitt *GeneralInformation* sowie die optionalen Abschnitte *ProductClassifications* und *TechnicalPropertyAreas*, die im vorliegenden Prototyp vollständig befüllt werden. Bemerkenswert ist, dass *GeneralInformation* mehrere Felder aufnimmt, die inhaltlich auch im Digital Nameplate geführt werden (z. B. *ManufacturerName*, *ManufacturerProductDesignation*). Diese normativ vorgesehene Redundanz ist keine Inkonsistenz, sondern Ausdruck der Konformitätsanforderung des IDTA 02003. Konsumenten des TechnicalData-Submodells sollen alle identifikationsrelevanten Informationen in einem Submodell vollständig vorfinden, ohne das Nameplate-Submodell zusätzlich abfragen zu müssen. Die Versionsfelder *HardwareVersion*, *FirmwareVersion* und *SoftwareVersion* gehören nicht zum normativen Bestand von *GeneralInformation*, der durch IDTA 02003 auf sechs Elemente festgelegt ist. Sie werden daher standardkonform in *TechnicalPropertyAreas* abgelegt. Tabelle 5.18 zeigt die vollständige Gold-Feld-Zuordnung zu den Submodell-Pfaden, wobei das Gold-Feld-Präfix *nameplate* mit *np* und im AAS-Pfad die Pfade GeneralInformation/, ProductClassifications/ und TechnicalPropertyAreas/ mit *GI*, *PC* und *TP* für bessere Lesbarkeit abgekürzt sind.

Tabelle 5.18: Mapping-Tabelle: Gold-Felder zu TechnicalData-Elementen

Gold-Feld	**AAS-Pfad**	**Typ**	**Quelle**
np.manufacturer_name	GI/ManufacturerName	P	ERP
np.product_designation	GI/ManufacturerProductDesignation	MLP	ERP
np.article_number	GI/ManufacturerArticleNumber	P	TDM
np.order_code	GI/ManufacturerOrderCode	P	TDM
np.classification_system	PC/ClassificationSystem	P	ERP
np.classification_system_version	PC/VersionOfClassificationSystem	P	ERP
np.classification_code	PC/ProductClassId	P	ERP
np.hardware_revision	TP/HardwareVersion	P	TDM
np.firmware_revision	TP/FirmwareVersion	P	TDM
np.software_version	TP/SoftwareVersion	P	TDM
technical_data.control_unit	TP/ControlUnit	P	TDM
technical_data.magazine_type	TP/MagazineType	P	TDM
technical_data.tool_max_dia._mm	TP/MaxToolDiameter	P	TDM
technical_data.tool_max_len._mm	TP/MaxToolLength	P	TDM
technical_data.tool_max_wt._kg	TP/MaxToolWeight	P	TDM
technical_data.tool_count	TP/ToolCapacity	P	TDM

MLP = MultiLanguageProperty, P = Property

Die *TechnicalPropertyAreas*-Liste nimmt neben den Versionsfeldern (*HardwareVersion*, *FirmwareVersion*, *SoftwareVersion*) auch die maschinenspezifischen Parameter in einer eingebetteten Collection als typisierte Properties auf. Für die dimensionsbehafteten Felder *MaxToolDiameter*, *MaxToolLength* und *MaxToolWeight* wird die Maßeinheit nicht nur im Feldnamen kodiert, was bereits in der Silver-Schicht durch Einheit-Explizierung geschah, sondern zusätzlich als *Qualifier* des jeweiligen AAS-Elements hinterlegt. Diese doppelte Verankerung entspricht der AAS-Metamodell-Empfehlung (IEC 63278-1) und stellt sicher, dass die Einheit maschinenlesbar im Submodell-Element selbst verfügbar ist, unabhängig davon, ob der konsumierende Client den Feldnamen-Konventionen folgt.

Das Gold-Feld *technical_ data.presetting_ postprocessor* (Tabelle 5.15) wird im vorliegenden Prototyp nicht in das TechnicalData-Submodell übernommen, da für diesen Feldinhalt kein standardisiertes SemanticID-Äquivalent in IDTA 02003 existiert und eine Aufnahme als Extension den Rahmen des prototypischen Use Case übersteigen würde.

Die vollständige JSON-Serialisierung des erzeugten TechnicalData-Submodells ist in Anhang B (Listing B.3) dokumentiert.

Das Process Parameters-Submodell folgt dem IDTA-Standard *Process Parameters Type* (IDTA 02031-1, Version 1.0). Der Standard definiert ein generisches, parametrisches Rahmenwerk für die Beschreibung von Prozessausgangsgrößen industrieller Anlagen und lässt bewusst Raum für anwendungsspezifische Erweiterungen. Im vorliegenden Anwendungsfall wird das Submodell eingesetzt, um den operativen Zustand der Fertigungsmaschine (Maschinenstatus, aktiv ausgeführtes NC-Programm, aktiver NC-Satz sowie Vorschubkorrektur) sowie die normativen Pflichtfelder *ProcessDescription* und *PlannedProcessTime* aus dem CAM-System als standardkonforme, laufend aktualisierte Submodell-Elemente abzubilden.

Das Aktualisierungsregime unterscheidet sich fundamental von den Stammdaten-

Submodellen: Während Digital Nameplate und TechnicalData durch *ASSET_SNAPSHOT*-Events vollständig neu geschrieben werden, empfängt das Process Parameters-Submodell ausschließlich *TELEMETRY*-Events. Jede Zustandsänderung der Maschine führt zu einem gezielten *PATCH*-Aufruf auf den betroffenen Properties der BaSyx-Submodell-API, nicht zu einem vollständigen *PUT* des Submodells. Diese asymmetrische Aktualisierungsstrategie ist technisch motiviert: Die Telemetriefrequenz kann die Stammdatenaktualisierungsfrequenz um mehrere Größenordnungen übersteigen, sodass ein vollständiges Überschreiben bei jedem Telemetrie-Event einen unverhältnismäßigen Schreibdurchsatz auf dem BaSyx Repository erzeugen würde. Tabelle 5.19 zeigt die vollständige Feldzuordnung der Gold-Telemetriefelder zu den Submodell-Elementen.

Tabelle 5.19: Mapping-Tabelle: Gold-Telemetriefelder zu Process-Parameters-Elementen

Gold-Feld	**AAS-Pfad (Processes/ Process__00__)**	**Typ**	**Beschreibung**
livedata.ncprog_name	ProcessId, ProcessName	P	NC-Programmname
cam.process_description	ProcessDescription	MLP	Programmbeschreibung (CAM)
cam.planned_process_time	PlannedProcessTime	P	Geplante Zykluszeit (CAM)
livedata.event_ts	EventTimestamp	P	Ereigniszeitpunkt
livedata.ingestion_ts	IngestionTimestamp	P	Kafka-Empfangszeitpunkt
livedata.source_table	SourceTable	P	Kafka-Quelltabelle
livedata.fr_ovr	ProcessParameters/ FeedRateOverride	P	Vorschubkorrektur (%)
livedata.act_block	ProcessParameters/ ActiveBlock	P	Aktiver NC-Satz
livedata.machine_status	ResourceParameters/ MachineStatus	P	Maschinenstatus (Int)
livedata.machine_status2	ResourceParameters/ MachineStatus2	P	Erweiterter Status (Bool)
livedata.nc_assembly	ResourceParameters/ NcAssembly	P	NC-Bausteinnummer

MLP = MultiLanguageProperty

P = Property — CAM-Felder via Gold-Schicht-Join über ncprog_name

Die vollständige JSON-Serialisierung des erzeugten Process Parameters-Submodells ist in Anhang B (Listing B.4) dokumentiert. Bemerkenswert ist, dass das Submodell zwölf Elemente umfasst, wovon elf als einfache *Property*-Typen und eines – *ProcessDescription* – als *MultiLanguageProperty* realisiert sind. Letzteres ist der Natur des normativen Pflichtfelds geschuldet: Der IDTA 02031-1-Standard schreibt für *ProcessDescription* explizit den Typ *MultiLanguageProperty* vor. Die übrigen Felder sind atomar und benötigen keine mehrsprachige Repräsentation. Die normativen Felder *ProcessId*, *ProcessName*, *ProcessDescription* und *PlannedProcessTime* tragen normierte *semanticId*-Werte. Alle implementierungsinternen Properties (*EventTimestamp*, *IngestionTimestamp*, *SourceTable*

sowie *FeedRateOverride*, *ActiveBlock*, *MachineStatus*, *MachineStatus2* und *NcAssembly*) sind prototypspezifische Erweiterungen ohne normierte *semanticId*, für die der Standard bewusst Raum lässt.

5.5.4 Ergebnis: Verwaltungsschale der Fertigungsmaschine in Eclipse BaSyx

Die beschriebene Verarbeitungskette mündet in einer vollständig instanziierten Verwaltungsschale für die Fertigungsmaschine *VCE-1250/5* innerhalb der Eclipse-BaSyx-Infrastruktur. Die AAS trägt die Identifikation *urn:example:aas:EQ-78421* und aggregiert die drei erzeugten Submodelle unter einem gemeinsamen Asset-Kontext. Tabelle 5.20 fasst die resultierenden AAS-Artefakte mit ihren persistierten Identifikatoren, dem jeweils zugrunde liegenden IDTA-Standard sowie den beteiligten Datenquellen zusammen.

Tabelle 5.20: Übersicht der erzeugten AAS-Struktur für die Fertigungsmaschine EQ-78421

AAS-Artefakt	**Identifier**	**Standard**	**Datenquelle**
Asset Administration Shell	urn:example:aas:EQ-78421	IEC 63278-1	n. v.
Submodell Nameplate	urn:example:sm:EQ-78421:nameplate	IDTA 02006 v3.0.1	ERP + TDM
Submodell TechnicalData	urn:example:sm:EQ-78421:technicaldata	IDTA 02003 v2.0	ERP + TDM
Submodell ProcessParams	urn:example:sm:EQ-78421:process-parameters	IDTA 02031-1 v1.0	OPC UA + CAM

Die Validierung des Anwendungfalls zeigt, dass die entwickelte Architektur in der Lage ist, die heterogene Datenlage aus ERP-,CAD-/CAM-, TDM -System und OPC UA zu einer konsistenten, standardkonformen Verwaltungsschale zusammenzuführen. Die Medaillen-Architektur absorbiert die Quellsystemheterogenität in frühen Schichten und schafft in der Gold-Schicht einen stabilen Übergabezustand für den Mapping-Service. Dieser übernimmt die semantische Verantwortung vollständig, ohne dass die vorgelagerte Daten- und Integrationsinfrastruktur AAS-Kenntnisse besitzen muss.

Die hexagonale Strukturierung der Daten- und Integrationsarchitektur bewährt sich in diesem Anwendungsfall insofern, als die vier technologisch heterogenen Quellsysteme (Kafka-basiertes ERP, REST-basiertes TDM, OPC UA-Subscription und Kafka-basiertes CAD/CAM) als austauschbare Adapter angebunden werden, ohne die interne Verarbeitungslogik zu beeinflussen. Ebenso kann der Mapping-Service als sekundärer Adapter ausgetauscht oder um weitere Submodell-Templates erweitert werden, ohne die Daten- und Integrationsinfrastruktur zu berühren. Dieses Verhalten entspricht dem in Abschnitt 4.5 formulierten Prinzip der klaren Verantwortlichkeitszuweisung und bestätigt die strukturelle Tragfähigkeit des gewählten Architekturansatzes.

Kapitel 6

Evaluation

Die Evaluation bildet im Sinne der Design-Science-Research-Methodologie nach Wieringa [37] die abschließende Bewertungsphase des Forschungsprozesses. Gegenstand der Evaluation ist das in Kapitel 5 entwickelte und prototypisch umgesetzte Artefakt, das anhand des Anwendungsfalls Fertigungsmaschine einer systematischen Überprüfung unterzogen wird.

Die Evaluation gliedert sich in vier Teilbereiche. Zunächst wird die Erfüllung der funktionalen Anforderungen anhand der im Anwendungsfall erbrachten Nachweise überprüft (Abschnitt 6.1). Anschließend wird die formale Konformität der erzeugten AAS-Artefakte mit dem IDTA-Metamodell auf zwei Validierungsebenen belegt (Abschnitt 6.2). Der dritte Teilbereich quantifiziert die Latenzeigenschaften beider Integrationspfade als empirische Grundlage für die Bewertung der Performanzanforderung (Abschnitt 6.3). Abschließend erfolgt eine qualitative Bewertung der nicht-funktionalen Anforderungen sowie der abgeleiteten Architekturziele (Abschnitte 6.4 und 6.5).

6.1 Funktionale Anforderungen

Die in Abschnitt 3.3 definierten funktionalen Anforderungen FA1 bis FA5 werden im Folgenden den konkreten Nachweisen im Anwendungsfall gegenübergestellt. Die Bewertungsgrundlage bilden ausschließlich die in Kapitel 5 dokumentierten Datendurchläufe, Mapping-Tabellen und erzeugten Artefakte.

Die prototypische Maschinenanbindung über OPC UA (FA1) ist vollständig nachgewiesen. Die implementierte Subscription überwacht sieben Knoten der Maschinensteuerung und übermittelt Zustandsänderungen reaktiv und ohne aktiven Polling-Overhead an den Bronze Consumer (Tab. 5.10). Damit ist die serverseitig initiierte Ereignislieferung über ein offenes Industrieprotokoll im Prototyp demonstriert.

Gleiches gilt für die konsolidierte Harmonisierung heterogener Quellsysteme (FA2). Das *ASSET_SNAPSHOT*-Event aggregiert 19 Nameplate-Felder (vierzehn aus ERP, fünf aus TDM) über den gemeinsamen Korrelationsschlüssel *asset_id* und überführt sie in einen einheitlichen, quellsystemunabhängigen Datenvertrag der Gold-Schicht (Tab. 5.15). Darüber hinaus demonstriert der *TELEMETRY*-Event quellübergreifende Aggregation auf dem Betriebsdatenpfad. Die Gold-Schicht verknüpft den OPC UA-Telemetriestrom mit dem CAD/CAM-Gold-Strom über den Korrelationsschlüssel *ncprog_name* und befüllt dadurch die normativen Pflichtfelder *ProcessDescription* und *PlannedProcessTime* des Process Parameters-Submodells.

Differenzierter fällt die Bewertung von FA3 aus, die fordert, dass Datenänderungen durch die Quellsysteme signalisiert werden können. Für ERP, das CAD/CAM-System

und OPC UA ist dies erfüllt. ERP- und CAD/CAM-Änderungen gelangen über Kafka-Events in die Pipeline, OPC UA-Ereignisse werden durch die serverinitiierte Subscription direkt an den Consumer übermittelt. Das TDM-System bietet hingegen keine Push-Schnittstelle, weshalb hier ein Hangfire-gesteuerter Polling-Mechanismus mit Zeitstempelvergleich zum Einsatz kommt, der Änderungen erst im jeweils nächsten Zyklus erkennt. FA3 ist damit teilweise erfüllt, wobei diese Einschränkung ausschließlich in der Schnittstellenarchitektur des TDM-Systems begründet liegt und kein strukturelles Defizit der entwickelten Plattform darstellt.

Für FA4, die standardisierte und semantisch strukturierte Asset-Repräsentation, sprechen drei IDTA-konforme Submodelle gemäß IDTA 02006 v3.0.1, IDTA 02003 v2.0 und IDTA 02031-1 v1.0, die eine vollständige *semanticId*-Annotation auf Element- und Containerebene aufweisen. Die Konformität mit den jeweiligen IDTA-Vorgaben ist durch die Mapping-Tabellen in Abschnitt 5.5.3 dokumentiert (Tab. 5.17, 5.18, 5.19).

Die Trennung von Stamm- und Telemetriedaten (FA5) ist durch die architektonisch separierten Event-Typen ASSET_SNAPSHOT und TELEMETRY realisiert. Stammdaten werden per *PUT* vollständig überschrieben. Telemetriedaten fließen über gezielte *PATCH*-Operationen ein. Die daraus resultierende asymmetrische Aktualisierungsstrategie erfüllt die geforderte Frequenzkontrolle vollständig.

Vier der fünf funktionalen Anforderungen sind damit vollständig nachgewiesen. Die partielle Nichterfüllung von FA3 ist, wie dargelegt, quellsystembedingt und nicht der Architektur der Plattform anzulasten.

6.2 Schema- und Strukturkonformität

Die strukturelle Korrektheit der vom Mapping-Service erzeugten Submodelle wurde auf zwei unabhängigen Validierungsebenen nachgewiesen.

Auf der ersten Validierungsebene überprüft die BaSyx-Submodell-API jede eingehende PUT-Anfrage gegen ihre interne Metamodell-Implementierung gemäß IEC 63278-1. Eine erfolgreiche Persistenz mit HTTP-Status 201 belegt, dass die übermittelte JSON-Struktur dem AAS-Metamodell entspricht. Fehlerhafte Strukturen werden von BaSyx mit einem spezifizierten Fehlercode abgewiesen und nicht persistiert.

Auf der zweiten Validierungsebene wurden die drei persistierten Submodelle aus dem BaSyx-Repository exportiert und mit dem *AASX Package Explorer* gegen das normative JSON-Schema der IDTA geprüft. Der Package Explorer implementiert den offiziellen AAS-Metamodell-Validator der IDTA-Referenzimplementierung und überprüft strukturelle Anforderungen (darunter Pflichtfelder, zulässige Elementtypen und korrekte *modelType*-Angaben) sowie die Konsistenz zwischen *valueType* und tatsächlichem Wertformat. Tabelle 6.1 fasst die Validierungsergebnisse je Submodell zusammen.

Tabelle 6.1: Schema-Validierungsergebnis der erzeugten Submodelle

Submodell	Geprüfter Standard	Validierungstool	Ergebnis
Digital Nameplate	IDTA 02006 v3.0.1 / IEC 63278-1	AASX Package Explorer	konform
TechnicalData	IDTA 02003 v2.0 / IEC 63278-1	AASX Package Explorer	konform
Process Parameters	IDTA 02031-1 v1.0 / IEC 63278-1	AASX Package Explorer	konform

Alle drei Submodelle bestehen die Validierung ohne Strukturfehler. Die formale Konformität der erzeugten AAS-Artefakte mit dem IDTA-Metamodell konnte damit auf beiden Validierungsebenen und durch das normative Referenzwerkzeug nachgewiesen werden.

6.3 Quantitative Performanzbewertung

Zur quantitativen Demonstration der Performanzanforderung NFA2 werden die Latenzen beider Integrationspfade anhand von Messdaten analysiert. Als Messgrößen werden zwei komplementäre Latenzmetriken herangezogen: Die *Edge-Latenz* erfasst die Zeitspanne von der Ereignisentstehung bis zur Aufnahme durch den Bronze Consumer und ist damit ein Maß für die Reaktionsfähigkeit der Integrationsschicht. Die *End-to-End-Gesamtlatenz* (E2E-Gesamtlatenz) umfasst darüber hinaus die interne Verarbeitungszeit durch die Medaillen-Architektur inklusive Mapping und BaSyx-Persistenz und beschreibt die vollständige Latenz bis zur Aktualität des AAS-Submodells im Repository.

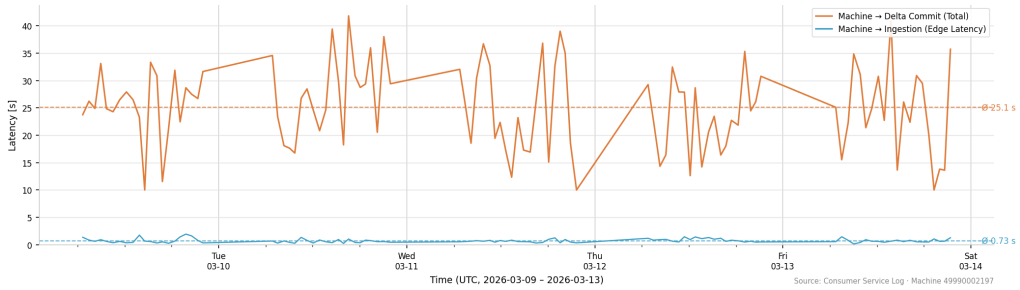

Abbildung 6.1: OPC UA-Integrationslatenz am Bronze Consumer über fünf Werktage (KW 11, 09.–13. März 2026). Edge-Latenz: vom OPC-UA-Ereignis bis zum Bronze-Consumer-Eingang. E2E-Gesamtlatenz: bis zur abgeschlossenen BaSyx-Persistenz. Mittlere Edge-Latenz: 0,73 s, mittlere E2E-Latenz: 25,1 s.

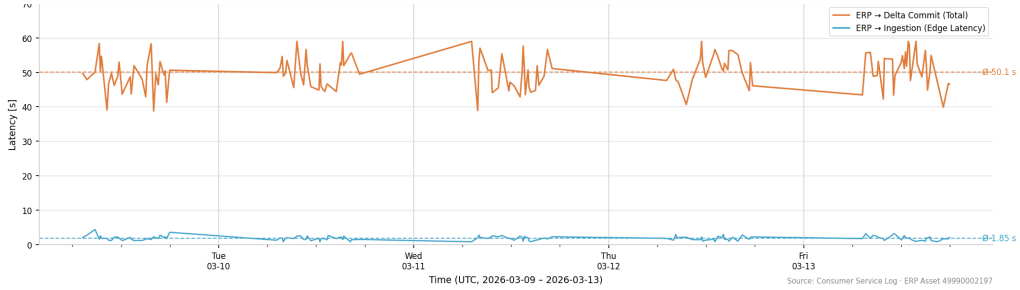

Abbildung 6.2: ERP-Kafka-Integrationslatenz am Bronze Consumer über fünf Werktage (KW 11, 09.–13. März 2026). Edge-Latenz: vom Datenbank-Trigger bis zum Bronze-Consumer-Eingang. E2E-Gesamtlatenz: bis zur abgeschlossenen BaSyx-Persistenz. Der erhöhte E2E-Mittelwert ist auf den mehrstufigen `ASSET_SNAPSHOT`-Join zurückzuführen. Mittlere Edge-Latenz: 1,92 s, mittlere E2E-Latenz: 49,4 s.

Die Abbildungen 6.1 und 6.2 zeigen den Verlauf beider Latenzmetriken über einen Beobachtungszeitraum von fünf Werktagen. Tabelle 6.2 stellt die gemessenen Integrationspfade gegenüber und führt das CAD/CAM-System als vierten Pfad auf, für den kein separates Latenz-Experiment durchgeführt wurde, da es denselben Kafka-basierten Integrationsmechanismus wie ERP verwendet und vergleichbare Charakteristika aufweist.

Tabelle 6.2: Latenzvergleich der Integrationspfade

Integrationspfad	**Trigger-Typ**	**Ø Edge**	**Ø E2E**	**NFA2**
OPC UA Subscription	Serverinitiiert	0,73 s	25,1 s	erfüllt
ERP Kafka	DB-getriggert	1,92 s	49,4 s	erfüllt
CAM Kafka	Kafka-Event	n/a*	n/a*	erfüllt
TDM Polling	Zeitgesteuert	n/a	max. 2 h	bedingt erfüllt

* Kein separates Latenz-Experiment. Integrationspfad ist identisch mit ERP Kafka.

Für den OPC UA-Integrationspfad ergibt sich eine mittlere Edge-Latenz von 0,73 s und eine mittlere E2E-Gesamtlatenz von 25,1 s. Maschinenzustandsänderungen sind damit innerhalb von unter 30 Sekunden im AAS-Repository reflektiert. Der ERP-Integrationspfad weist eine mittlere Edge-Latenz von 1,92 s und eine E2E-Gesamtlatenz von 49,4 s auf. Stammdatenaktualisierungen sind damit innerhalb von unter 60 Sekunden im Repository verfügbar. Der erhöhte E2E-Mittelwert des ERP-Pfades gegenüber dem OPC UA-Pfad ist architektonisch begründet, da das ASSET_SNAPSHOT-Event erst nach vollständigem Join von ERP- und TDM-Daten in der Gold-Schicht entsteht, was die Gesamtlatenz erhöht, jedoch Voraussetzung für die semantische Vollständigkeit des erzeugten Submodells ist. Der CAD/CAM-Integrationspfad verwendet denselben Kafka-basierten Mechanismus wie ERP und wurde im Rahmen dieses Prototyps nicht separat vermessen. Für den TDM-Pfad begrenzt die Hangfire-Polling-Periode die maximale Aktualitätsverzögerung auf zwei Stunden, womit die in Abschnitt 6.1 bei FA3 festgestellte Einschränkung quantitativ untermauert wird.

6.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen NFA1 bis NFA5 beschreiben strukturelle Qualitätseigenschaften der Architektur, deren Erfüllung nicht durch einen isolierten Testfall abschließend belegbar ist, sondern eine qualitative Bewertung auf Basis der dokumentierten Architekturentscheidungen und des prototypischen Nachweises erfordert.

Die Skalierbarkeit (NFA1) ist strukturell, wenngleich nicht empirisch belegt. Die Medaillen-Architektur enthält keine Asset-spezifische Logik und ist damit konzeptionell unabhängig von der Anzahl verwalteter Verwaltungsschalen und Submodelle. Die horizontale Partitionierbarkeit des Kafka-Clusters stützt diese Eigenschaft zusätzlich, da steigende Ereignisvolumina über zusätzliche Partitionen verteilt werden können. Da der Prototyp auf einer einzelnen Asset-Instanz basiert, bleibt die quantitative Übertragbarkeit auf höhere Datenmengen einer weiterführenden Untersuchung vorbehalten.

Bei NFA2 steht kein absolutes Echtzeitgebot im Vordergrund, sondern die Forderung, eine spätere Annäherung an echtzeitnahe Verarbeitung nicht strukturell auszuschließen. Die in Abschnitt 6.3 gemessenen E2E-Gesamtlatenzen von durchschnittlich 25,1 s (OPC UA) und 49,4 s (ERP) sind dabei nicht Ausdruck einer architektonischen Schwäche, sondern der inhärente Preis für Datenvollständigkeit. Der ASSET_SNAPSHOT-Join in der Gold-Schicht sichert erst die semantische Konsistenz des Submodells. Die hexagonale Adapterstruktur ließe konzeptionell einen Fast-Path zur AAS-Schicht zu, der diese Aggregation umgeht, allerdings zulasten der inhaltlichen Vollständigkeit. NFA2 ist erfüllt.

Interoperabilität (NFA3) wird durch den konsequenten Einsatz offener Standards gewahrt: OPC UA und Kafka quellseitig, die BaSyx REST API gemäß IEC 63278-1 zielseitig. Proprietäre Protokolle sind im Datenpfad nicht enthalten. Die erzeugten Submodelle sind von jedem IEC 63278-1-konformen Client konsumierbar und weisen keine Plattformabhängigkeiten auf.

Die Wartbarkeit und Modularität der Architektur (NFA4) zeigt sich in der klaren Komponentenstruktur mit expliziten Schnittstellenverträgen. Die Adapter, Silver-ETL Pipeline, die Gold-Schicht und der Mapping-Service sind strikt getrennte Einheiten. Praktisch bedeutsam ist, dass Anpassungen an Submodellstrukturen ausschließlich die Mapping-Konfiguration betreffen und ohne Neukompilierung des Service-Codes durchführbar sind, was im Verlauf der Implementierungsarbeit mehrfach genutzt und bestätigt wurde.

Die Erweiterbarkeit (NFA5) ist strukturell verankert. Die hexagonale Adapterstruktur erlaubt die Anbindung weiterer Quellsysteme ohne Eingriffe in die interne Verarbeitungslogik. Neue Asset-Klassen werden über zusätzliche Mapping-Konfigurationen eingeführt, ohne bestehenden Service-Code zu verändern. Die ereignisbasierte Entkopplung über Kafka ermöglicht es zudem, neue Datenkonsumenten unabhängig an bestehende Topics anzubinden. Offen bleibt, ob dieser Mechanismus auch auf Asset-Klassen mit grundlegend abweichender semantischer Struktur ohne Anpassungen übertragbar ist.

6.5 Architekturziele

Die in Abschnitt 3.5 formulierten Architekturziele AZ1 bis AZ5 konkretisieren die angestrebten strukturellen Eigenschaften der Systemarchitektur und dienten als übergeordnete Leitlinien für den Architekturentwurf. Im Folgenden wird bewertet, inwieweit das entwickelte Artefakt diesen Zielen gerecht wird.

Das Ziel einer konsistenten, zentral auswertbaren Asset-Repräsentation (AZ1) wurde erreicht. Drei IDTA-konforme Submodelle sind im BaSyx-Repository persistiert und stehen über die standardisierte REST-API für beliebige IEC 63278-1-konforme Clients zur Verfügung. Die in Abschnitt 6.2 beschriebene Schema-Validierung bestätigt strukturelle Konsistenz der Submodelle mit dem normativen IDTA-Metamodell.

Die Schnittstellenheterogenität (AZ2) ist durch den durchgängigen Einsatz offener Standards nachhaltig reduziert, wobei OPC UA und Kafka quellseitig umgesetzt sind und die IDTA-Submodell-Templates und BaSyx REST API nach IEC 63278-1 zielseitig umgesetzt sind. Proprietäre Protokolle oder herstellerspezifische Middleware sind im Datenpfad nicht enthalten.

Kafka als asynchroner Entkopplungsmechanismus realisiert AZ3, das die vollständige Trennung von Quellen und Konsumenten vorschreibt. Die Event-Typen ASSET_SNAPSHOT und TELEMETRY bilden einen stabilen, quellsystemunabhängigen Datenvertrag. Der AAS-Consumer besitzt keine Kenntnisse über Herkunft oder interne Struktur der Quellsysteme, und neue Adapter lassen sich integrieren, ohne die nachgelagerte Verarbeitungslogik zu berühren.

Dass Datenintegration und semantische Modellierung sauber getrennt bleiben (AZ4), ist architektonisch verankert. Bronze-, Silver- und Gold-Schicht verantworten ausschließlich Erfassung, Filterung, Normalisierung und Aggregation. Das AAS-Domänenwissen ist vollständig im Mapping-Service konzentriert, der als einzige Komponente im Datenpfad über IDTA-Semantik verfügt. Quellsystemspezifische Details dringen damit strukturell nicht in die semantische Modellierungsschicht ein.

Die Erweiterbarkeit und Evolvierbarkeit (AZ5) ist durch das konfigurationsbasierte Mapping-Design und die hexagonale Adapterstruktur strukturell verankert. Dabei können sowohl neue Asset-Klassen über zusätzliche Mapping-Konfigurationen, als auch neue Quellsysteme als eigenständige Adapter, jeweils ohne Eingriff in bestehenden Service-Code, eingeführt beziehungsweise implementiert werden. Die Beantwortung der Frage, ob dieser Mechanismus auch auf Asset-Klassen mit grundlegend abweichender semantischer Struktur ohne Anpassungen übertragbar ist, wird weiterführenden Arbeiten überlassen.

Kapitel 7

Diskussion

Die Diskussion führt die Ergebnisse der prototypischen Umsetzung und der strukturierten Evaluation in eine reflektierte Gesamtbewertung über. Dabei werden die Implikationen der Evaluationsergebnisse aus Kapitel 6 interpretiert, die Beantwortung der Forschungsfrage noch einmal zusammengefasst und der wissenschaftliche sowie praktische Beitrag eingeordnet. Abschließend werden Grenzen der Arbeit offen benannt und Erkenntnisse aus dem Forschungs- und Entwicklungsprozess festgehalten.

7.1 Bewertung und Beantwortung der Forschungsfrage

Die Erfüllung der funktionalen Anforderungen FA1 bis FA5 und die nicht-funktionalen Anforderungen NFA1 bis NFA5 sowie die Architekturziele AZ1 bis AZ5 wurde systematisch geprüft (siehe Kapitel 6). Vier der fünf funktionalen Anforderungen sind vollständig erfüllt. FA3 ist nur teilweise erfüllt, da das TDM-System keine Push-Schnittstelle bereitstellt und Änderungen somit nicht eigenständig signalisieren kann. Diese Einschränkung liegt jedoch ausschließlich in der Schnittstellenarchitektur des Quellsystems begründet und stellt kein strukturelles Defizit der entwickelten Plattform dar.

Das in dieser Arbeit entwickelte Artefakt adressiert die Frage nach der Gestaltung einer Systemarchitektur zur Datenintegration in heterogenen IT- und OT-Systemlandschaften sodass Interoperabilität, Skalierbarkeit und semantische Konsistenz bei der Repräsentation von Asset-Zuständen gegeben ist.

Die Architektur ermöglicht zunächst eine durchgängige *Datenintegration in heterogenen IT- und OT-Systemlandschaften*. Die hexagonale Strukturierung auf Systemintegrationsebene bindet unterschiedliche Quellsysteme über technologiespezifische Adapter an, ohne dass die interne Verarbeitungslogik von den jeweiligen Schnittstellendetails abhängt. Die Medaillen-Architektur kompensiert die Variabilität der Quellsysteme in den frühen Schichten: Filterlogik und Normalisierung sind in der Silver-Schicht lokalisiert, während die Gold-Schicht einen stabilen, quellsystemunabhängigen Datenvertrag definiert. Weder der Mapping-Service noch das AAS-Repository sind damit von Integrationsdetails abhängig, was eine klare Schichtenverantwortung und die Austauschbarkeit beider Architekturseiten strukturell sicherstellt.

Bezüglich der *Interoperabilität* werden ausschließlich offene Standards eingesetzt: OPC UA und Kafka als Integrationsschnittstellen, die IDTA-Submodell-Templates als Zielformat und die BaSyx-REST-API als Bereitstellungsschnittstelle. Die erzeugten Verwaltungsschalen sind durch jeden IEC 63278-1-konformen Client konsumierbar und unabhängig von der konkreten Implementierung der Plattform.

Hinsichtlich der *Skalierbarkeit* zeigt der Anwendungsfall, dass die hexagonale Adapterstruktur eine technologisch homogene Erweiterung auf weitere Quellsysteme erlaubt.

Neue Quellsysteme werden als eigenständige Adapter implementiert, ohne dass die interne Verarbeitungslogik angepasst werden muss. Die Medaillen-Architektur ist Asset-agnostisch: Neue Asset-Klassen werden durch zusätzliche Mapping-Konfigurationen eingeführt, nicht durch Erweiterungen des Service-Codes. Die ereignisbasierte Entkopplung über Kafka ermöglicht darüber hinaus eine unabhängige Skalierung von Produzenten und Konsumenten.

Die *semantische Konsistenz* wird durch den Mapping-Service als alleinigen semantisch verantwortlichen Punkt gewährleistet. Er übernimmt die Zuordnung von Quellfeldern zu AAS-Elementen gemäß den jeweiligen IDTA-Submodell-Spezifikationen. Die deklarative Konfigurationsstruktur stellt sicher, dass diese Zuordnungen explizit dokumentiert, versionierbar und unabhängig vom ausführbaren Code änderbar sind. Die formale Validierung der erzeugten Submodelle gegen das IDTA-Metamodell bestätigt die strukturelle Konformität des Ergebnisses (Tab. 6.1).

Das entwickelte Artefakt vereint Datenintegration, Interoperabilität, Skalierbarkeit und semantische Konsistenz in einer Systemarchitektur, die anhand eines industriellen Anwendungsfalls prototypisch validiert wurde. Der Nachweis ist struktureller Natur, da Aussagen über Produktionstauglichkeit, Langzeitstabilität und Verhalten unter Last weiterführende Evaluationen erfordern, die über den Rahmen dieser Arbeit hinausgehen.

7.2 Beitrag zu Forschung und Praxis

Der wissenschaftliche Beitrag der Arbeit liegt in der gezielten, neuartigen Kombination bestehender Konzepte und deren prototypischer Erprobung in einem realistischen Umfeld. Die Medaillen-Architektur, die hexagonale Strukturierung und das AAS-Konzept sind in der Literatur bekannte Ansätze. Die vorliegende Arbeit leistet einen Beitrag, indem sie diese Konzepte in einer konkreten Architektur zusammenführt, die Schnittstellenfragen zwischen Schichten präzise beantwortet und die Entscheidungen in Form von ADRs nachvollziehbar dokumentiert. Die ADRs sind dabei nicht nur eine Dokumentationsform, sondern beinhalten Abwägungen wie die Trennung von Datenintegration und Beschreibung der Semantik, die Wahl eines deklarativen Mapping-Ansatzes und die ereignisbasierte Entkopplung über Kafka. Die Dokumentation solcher Entscheidungen ist für Folgearbeiten und vergleichbare Implementierungsprojekte direkt verwertbar, weil sie den Begründungszusammenhang explizit macht. Gerade die Rückverfolgbarkeit von Anforderungen über dokumentierte Architekturentscheidungen bis zum erzeugten AAS-Artefakt ist ein methodischer Beitrag, der über den konkreten Anwendungsfall hinaus übertragbar ist.

Aus praktischer Perspektive ist der unmittelbarste Beitrag die Demonstration, dass eine vollständige AAS-Integration über mehrere heterogene Quellsysteme mit ausschließlich quelloffenen Komponenten realisierbar ist. Eclipse BaSyx und Apache Kafka bilden zusammen eine funktionsfähige Plattform, ohne dass proprietäre Middleware oder kommerzielle AAS-Werkzeuge erforderlich sind. Für Unternehmen, die eine AAS-Adoption erwägen, aber noch keine Infrastrukturentscheidungen getroffen haben, liefert die Arbeit einen konkreten Ausgangspunkt mit dokumentierten Technologieentscheidungen und deren Begründungen.

Der konfigurationsbasierte Mapping-Ansatz hat darüber hinaus einen praktischen Wert, der über die vorliegende Implementierung hinausgeht, denn die Zuordnung von Quelldaten zu AAS-Submodell-Elementen ist eine Aufgabe, die in jedem AAS-Adoptionsprojekt anfällt. Die Auslagerung dieser Zuordnung in eine YAML-Konfiguration mit Jinja2-Ausdrücken trennt Domänenwissen von Implementierungswissen. Das bedeutet, dass

die Erweiterung um neue Asset-Klassen oder die Anpassung bestehender Submodell-Zuordnungen ohne Eingriff in den Service-Code möglich ist. Für Organisationen mit klarer Trennung zwischen Entwicklungs- und Fachdomäne ist das ein relevanter Gewinn an Wartbarkeit und Anpassungsfähigkeit.

7.3 Limitationen und Erkenntnisse

Die Validierung der Architektur erfolgte ausschließlich auf struktureller Ebene anhand eines einzigen industriellen Anwendungsfalls. Ein Lasttest unter realistischen Produktionsbedingungen, Tests mittels Fehlerinjektion und eine formale Verifikation der Systemgarantien wurden nicht durchgeführt. Aussagen über das Verhalten der Architektur bei vielen gleichzeitig aktiven Assets, hohen Kafka-Durchsätzen oder netzwerkbedingten Latenzen können auf Basis der vorliegenden Arbeit nicht getroffen werden.

Eine zentrale strukturelle Grenze betrifft die Echtzeitfähigkeit. Die Medaillen-Architektur führt mehrere Verarbeitungsstufen ein, die zwischen dem originären Maschinenereignis und der Aktualisierung des entsprechenden AAS-Submodells sequenziell durchlaufen werden. Die gemessene E2E-Gesamtlatenz (siehe Abschnitt 6.3) von durchschnittlich 25,1 s (OPC UA) ist dabei das Ergebnis dieser Mehrschichtigkeit. Bronze-zu-Silver-Transformation, Gold-Schicht-Aggregation und der abschließende BaSyx-*PATCH*-Aufruf addieren sich zu einer Gesamtlatenz, die für zeitkritische Prozesssteuerungsanwendungen nicht ausreichend sein kann. Ein direkter Fast-Path, der die Medaillen-Architektur für ausgewählte Telemetriedaten umgeht, würde diese Latenz deutlich reduzieren. Er würde jedoch gleichzeitig die Enrichment-Logik ausschließen, die für ein vollständig strukturiertes Process-Parameters-Submodell notwendig ist. Erst die Gold-Schicht verankert Telemetriedaten im Asset-Kontext und verknüpft sie mit den zugehörigen Stammdaten. Die Auflösung dieses Zielkonflikts zwischen Aktualität und Datenvollständigkeit ist eine offen gebliebene Forschungsfrage, die in einer Folgearbeit durch eine hybride Routing-Strategie adressiert werden könnte, bei der zeitkritische Telemetrie über einen direkten Pfad und kontextanreichernde Stammdaten über die vollständige Pipeline verarbeitet werden.

Die Anbindung des TDM-Systems über einen Hangfire-gesteuerten Polling-Mechanismus stellt eine Annäherung an Ereignisbasierung dar, nicht deren vollständige Realisierung. Änderungen im TDM werden erst beim nächsten Polling-Zyklus erkannt. Das Ausmaß dieser Einschränkung ist für das TDM-System vertretbar, da sich technische Stammdaten erfahrungsgemäß selten ändern. Für Quellsysteme mit höherer Änderungsfrequenz und ohne Push-Schnittstelle wäre jedoch eine andere Strategie erforderlich.

Der vorliegende Anwendungsfall validiert die Architektur für eine Asset-Klasse mit Live-Daten einer einzigen Maschineninstanz. Die konzeptionelle Abdeckung aller Fertigungsmaschinen des betrachteten Typs ist durch die Architektur vorgesehen. Die Übertragbarkeit auf weitere Asset-Klassen wie Werkzeuge, Fertigungsaufträge oder Prüfmittel ist konfigurationsbasiert möglich, aber nicht prototypisch demonstriert worden. Ob die gewählten Gold-Event-Strukturen und Mapping-Muster ohne strukturelle Änderungen auf semantisch grundlegend andere Asset-Klassen übertragbar sind, bleibt eine offene empirische Frage.

Aus den identifizierten Grenzen und dem Verlauf der Implementierungsarbeit lassen sich abschließend folgende Erkenntnisse ableiten, die für vergleichbare Entwicklungsvorhaben von Bedeutung sind.

- Die konsequente Zuordnung von Transformationsverantwortung zu spezifischen Architekturschichten hat sich als wesentlicher Stabilitätsfaktor erwiesen. Die Entschei-

dung, Feldfilterung und Normalisierung ausschließlich in der Silver-Schicht und semantische Abbildungen ausschließlich im Mapping-Service zu verorten, hat dazu geführt, dass Änderungen an einer Schicht isoliert durchgeführt werden konnten, ohne andere Schichten zu destabilisieren. Die konsequente Einhaltung der Schichtenverantwortung ist in Bezug auf die Wartbarkeit besser geeignet als eine Umsetzung der Transformationslogik an jener Stelle, wo sie technisch am einfachsten umzusetzen ist.

- Die deklarative Mapping-Konfiguration mit Jinja2 als Erweiterungsschicht hat sich für einfache bis mittelschwere Transformationen als ausgesprochen effektiv erwiesen. Die klare Trennlinie zwischen Konfiguration und Service-Code hält den Wartungsaufwand niedrig und gewährleistet Transparenz über semantische Zuordnungen. Gleichzeitig zeigte sich, dass komplexe bedingte Logik innerhalb von Jinja2-Ausdrücken schnell an Lesbarkeit verliert. Die Grenze zwischen einer sinnvoll konfigurierbaren Erweiterungsschicht und einem Programm in Konfigurationssyntax ist fließend. Für eine produktive Weiterentwicklung wäre eine explizite Richtlinie sinnvoll, ab welcher Komplexitätsschwelle ein Jinja2-Ausdruck durch eine typisierte Transformationsregel im Service-Code ersetzt werden sollte.
- Das Eclipse-BaSyx-Ökosystem bietet eine funktionsfähige Implementierungsbasis für AAS-konforme Infrastruktur, ist jedoch in seiner aktuellen Reifeform deutlich einem Forschungsprototyp näher als einem produktionstauglichen Framework. Dokumentationslücken und eine noch im Aufbau befindliche Community haben dazu geführt, dass ein erheblicher Anteil der Implementierungszeit auf Framework-Exploration entfiel. Für Folgeprojekte empfiehlt sich eine frühzeitige Tiefenanalyse der BaSyx-Komponentenarchitektur sowie der jeweiligen API-Versionskompatibilität, bevor Architekturentscheidungen auf spezifischen BaSyx-Funktionen aufgebaut werden.

Kapitel 8

Fazit und Ausblick

Die vorliegende Arbeit ging von der Beobachtung aus, dass zwischen den heterogenen Quellsystemen industrieller Produktionsumgebungen und einer standardkonformen, interoperablen Asset-Repräsentation eine architektonische Lücke klafft, die in der Literatur bislang nur partiell adressiert wird. Daraus ergab sich die Forschungsfrage, wie eine Systemarchitektur zur Datenintegration gestaltet werden kann, die eine interoperable, skalierbare und semantisch konsistente Repräsentation von Asset-Zuständen in heterogenen IT- und OT-Systemlandschaften ermöglicht. Zur Beantwortung wurde ein Design-Science-Artefakt konzipiert, prototypisch umgesetzt und anhand eines industriellen Anwendungsfalls evaluiert.

Das in dieser Arbeit entwickelte Artefakt adressiert das formulierte Design-Problem durch eine zweigeteilte Architektur, die Datenintegration und semantische Modellierung konsequent trennt. Die in Kapitel 6 dokumentierte Evaluation weist nach, dass die definierten funktionalen Anforderungen, nicht-funktionalen Anforderungen (NFA1–NFA5) und Architekturziele (AZ1–AZ5) auf struktureller Ebene erfüllt sind. Dieser Nachweis ist struktureller Natur und belegt die architektonische Tragfähigkeit des Entwurfs, trifft jedoch keine Aussagen über Produktionstauglichkeit oder Verhalten unter Last. Der wissenschaftliche Beitrag liegt in der gezielten Kombination bestehender Konzepte und der expliziten Dokumentation der getroffenen Architekturentscheidungen. Insbesondere der konfigurationsbasierte Mapping-Service als eigenständiges Transformationsartefakt schließt eine in der Literatur identifizierte Lücke zwischen Heterogenitätsbehandlung und standardkonformer AAS-Bereitstellung.

Die Arbeit zeigt, dass eine standardkonforme AAS-Integration über heterogene Quellsysteme hinweg mit ausschließlich quelloffenen Komponenten realisierbar ist. Der dokumentierte Entwurfs- und Entscheidungsprozess, die ADRs und der konfigurationsbasierte Mapping-Ansatz bieten eine übertragbare Grundlage für vergleichbare Vorhaben. Der Weg von der strukturellen Validierung zur Produktionstauglichkeit bleibt offen, wobei die architektonischen Voraussetzungen dafür gegeben sind.

Aus den Limitationen der vorliegenden Arbeit ergeben sich folgende Möglichkeiten für zukünftige Arbeiten.

- Zunächst wäre es möglich, die TDM-Anbindung auf eine pushbasierte Integration umzustellen, sofern das Quellsystem eine Change-Notification-API bereitstellt. Das sogenannte Hangfire-Polling würde in diesem Fall durch eine ereignisbasierte Anbindung ersetzt werden, wodurch die Anforderungen von FA3 vollständig erfüllt würden.
- Des Weiteren besteht die Notwendigkeit einer empirischen Skalierungsvalidierung. Es wird empfohlen, den Mapping-Service auf weitere Asset-Klassen, wie etwa Roboter oder Messmittel, auszurollen und das Skalierungsverhalten unter Last zu messen. Ziel dieses Vorgehens ist es, die strukturellen Skalierbarkeitsaussagen (NFA1) quantitativ zu untermauern.

- Die hexagonale Adapterstruktur erlaubt konzeptionell einen Fast-Path für latenzsensible Telemetrie, der die Gold-Aggregation umgeht und eine signifikante Reduktion der E2E-Latenz für zeitkritische Pfade ermöglichen könnte. Der identifizierte Zielkonflikt zwischen Aktualität und Datenvollständigkeit kann durch eine hybride Routing-Strategie gelöst werden (NFA2).
- Die Portabilität der erzeugten Verwaltungsschalen wurde bislang ausschließlich mit Eclipse BaSyx als AAS-Backend validiert. Eine Validierung gegen alternative, IEC-63278-1-konforme Implementierungen, wie beispielsweise den FA³ST-Service oder den AASX-Server, würde die Aussage bezüglich der Interoperabilität (NFA3) substanziell stärken.
- Es ist festzustellen, dass die gegenwärtige Architektur eine unidirektionale Datenintegration von Quellsystemen zur Verwaltungsschale aufweist. Ein Write-Back-Pfad von der AAS zu den Quellsystemen würde Closed-Loop-Szenarien wie die Parametersetzung über die Verwaltungsschale ermöglichen und den Reifegrad in Richtung AAS-Typ 3 (proaktiv) anheben.
- Die Implementierungsarbeit hat gezeigt, dass die deklarative Jinja2-Erweiterungsschicht bei steigender Transformationskomplexität an Lesbarkeitsgrenzen stößt. Eine explizite Richtlinie, ab welcher Schwelle ein Jinja2-Ausdruck durch eine typisierte Transformationsregel im Service-Code ersetzt werden sollte, würde die langfristige Wartbarkeit des Mapping-Ansatzes absichern.

Literaturverzeichnis

[1] Angelos Alexopoulos, Georgios Kalogeras, Konstantinos Koutras und Athanasios Kalogeras. 2025. Why Asset Administration Shells: A Survey on Uses and Challenges. *IEEE Access*, 13, 126582–126609. DOI: 10.1109/ACCESS.2025.3589931.

[2] Len Bass, Paul Clements und Rick Kazman. 2013. *Software Architecture in Practice*. (3rd Auflage). Addison-Wesley. ISBN: 978-0-321-81573-6.

[3] Pascal Cambatzu, Elisabeth Schmidl und Matthias Wenk. 2026. Asset Administration Shells in Production Environments: Implementation of Decentralized, Modular Agents with Event-Driven Capabilities. *Procedia CIRP*, 139, 331–336. DOI: 10.1016/j.procir.2025.08.192.

[4] Alistair Cockburn. 2005. Hexagonal Architecture. Retrieved 02. 01. 2026 from https://alistair.cockburn.us/hexagonal-architecture.

[5] Databricks. 2024. What is Medallion Architecture? Retrieved 29. 03. 2026 from https://www.databricks.com/glossary/medallion-architecture.

[6] Eclipse BaSyx. 2025. AAS Discovery Service. Retrieved 02. 01. 2026 from https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/v2/aas_discovery/index.html.

[7] Eclipse BaSyx. 2025. Registry Integration (Feature) – Submodel Repository. Retrieved 02. 01. 2026 from https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/v2/submodel_repository/features/registry-integration.html.

[8] Eclipse BaSyx. 2025. Submodel Registry. Retrieved 02. 01. 2026 from https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/v2/submodel_registry/index.html.

[9] Eclipse BaSyx. 2025. Submodel Repository. Retrieved 02. 01. 2026 from https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/v2/submodel_repository/index.html.

[10] Eclipse Foundation. 2025. BaSyx Architecture. Retrieved 02. 01. 2026 from https://eclipse.dev/basyx/architecture/.

[11] Rene-Pascal Fischer, Frank Schnicke, Bastian Beggel und Pablo Oliveira Antonino. 2022. Historical Data Storage Architecture Blueprints for the Asset Administration Shell. In *Proceedings of the 27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. DOI: 10.1109/ETFA52439.2022.9921613.

[12] Wei Guo, Torben Heinz Miny, Tobias Theodor Kleinert, Frank Schnicke, Sebastian Käbisch, Constantin Liepert, Kazeem Oladipupo und Christian Diedrich. 2024. Comparison of Data Integration Concepts for Asset Administration Shell. In *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Seiten 1–8. DOI: 10.1109/ETFA61755.2024.10710842.

[13] Gregor Hohpe. 2003. Enterprise Integration Patterns. Konferenzpräsentation, JAOO 2003. (2003). Retrieved 29. 03. 2026 from https://www.enterpriseintegrationpatterns.com/docs/jaoo_hohpeg_enterpriseintegrationpatterns.pdf.

[14] Gregor Hohpe und Bobby Woolf. 2003. Canonical Data Model. Retrieved 29. 03. 2026 from https://www.enterpriseintegrationpatterns.com/patterns/messaging/CanonicalDataModel.html.

[15] Gregor Hohpe und Bobby Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley. ISBN: 978-0321200686.

[16] IEC. 2023. IEC 63278-1: Asset Administration Shell for Industrial Applications – Part 1: Information Model. International Electrotechnical Commission, (2023). Retrieved 02. 01. 2026 from https://webstore.iec.ch/en/publication/65628.

[17] Industrial Digital Twin Association. 2023. AASX Server Reference Implementation for Asset Administration Shells. Retrieved 02. 01. 2026 from https://github.com/admin-shell-io/aasx-server.

[18] Industrial Digital Twin Association. 2025. IDTA 02003: Generic Frame for Technical Data for Industrial Equipment in Manufacturing. Technischer Bericht. Version 2.0. Industrial Digital Twin Association. Retrieved 02. 01. 2026 from https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2025/03/IDTA-02003_Generic-Frame-for-Technical-Data.pdf.

[19] Industrial Digital Twin Association. 2025. IDTA 02006: Digital Nameplate for Industrial Equipment. Technischer Bericht. Version 3.0.1. Industrial Digital Twin Association. Retrieved 02. 01. 2026 from https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2025/10/IDTA-02006-3-0-1_Submodel_Digital-Nameplate.pdf.

[20] Industrial Digital Twin Association. 2025. IDTA 02031-1: Process Parameters – Part 1: Type. Technischer Bericht. Version 1.0. Industrial Digital Twin Association. Retrieved 02. 01. 2026 from https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2025/07/IDTA-02031-1_Submodel_ProcessParameters_Type.pdf.

[21] Industrial Digital Twin Association. 2025. Submodel Templates. Retrieved 04. 05. 2026 from https://industrialdigitaltwin.org/en/content-hub/submodels.

[22] ISO/IEC/IEEE. 2011. ISO/IEC/IEEE 42010:2011 – Systems and Software Engineering – Architecture Description. International Organization for Standardization / International Electrotechnical Commission / Institute of Electrical und Electronics Engineers, (2011). DOI: 10.1109/IEEESTD.2011.6129467.

[23] Michael Jacoby, Friedrich Volz, Christian Weißenbacher und Jens Müller. 2022. FA^{3}ST Service: An Open Source Implementation of the Reactive Asset Administration Shell. In *Proceedings of the 27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. DOI: 10.1109/ETFA52439.2022.9921584.

[24] Martin Kleppmann. 2017. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media. ISBN: 978-1449373320.

[25] Jay Kreps, Neha Narkhede und Jun Rao. 2011. Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*. Athens, Greece, Seiten 1–7. Retrieved 02. 05. 2026 from https://www.semanticscholar.org/paper/ea97f112c165e4da1062c30812a41afca4dab628.

[26] Johannes Kristan, Sven Erik Jeroschewski, Milena Jäntgen und Max Grzanna. 2025. IoT Backends in the Asset Administration Shell – Four Integration Approaches. *International Journal on Software Tools for Technology Transfer*, 27, 93–102. DOI: 10.1007/s10009-025-00796-z.

[27] Jay Lee, Behrad Bagheri und Hung-An Kao. 2015. A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems. *Manufacturing Letters*, 3, 18–23. DOI: 10.1016/j.mfglet.2014.12.001.

[28] MADR Project. 2023. Markdown Architectural Decision Records. Retrieved 04. 01. 2026 from https://adr.github.io/madr/.

[29] Wolfgang Mahnke, Stefan-Helmut Leitner und Matthias Damm. 2009. *OPC Unified Architecture*. Springer. ISBN: 978-3-540-68898-3. DOI: 10.1007/978-3-540-68899-0.

[30] Robert C. Martin. 2017. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall. ISBN: 978-0-13-449416-6.

[31] Michael Nygard. 2011. Documenting Architecture Decisions. Retrieved 04. 01. 2026 from https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions .

[32] Jeffrey Palermo. 2008. The Onion Architecture: Part 1. Retrieved 04. 05. 2026 from https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/.

[33] Plattform Industrie 4.0. 2020. Details of the Asset Administration Shell, Part 1: Metamodel. Technischer Bericht. Version 3.0. Plattform Industrie 4.0. Retrieved 02. 01. 2026 from https://industrialdigitaltwin.org/wp-content/uploads/2022/06/DetailsOfTheAssetAdministrationShell_Part1_V3.0RC02_Final1.pdf.

[34] Frank Schnicke, Thomas Kuhn, Tobias Klausmann, Sten Grüner und Daniel Porta. 2022. Architecture Blueprints for the Application of the Industry 4.0 Asset Administration Shell. In *Proceedings of the 27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. DOI: 10.1109/ETFA52439.2022.9921694.

[35] Rainer Stark, Michael Wichmann und Thomas Damerau. 2021. Automated Design and Integration of Asset Administration Shells in Industrial Applications. *Sensors*, 21, 6, 2004. DOI: 10.3390/s21062004.

[36] L. Stojanovic u. a. 2021. Methodology and Tools for Digital Twin Management: The FA^{3}ST Approach. *IoT*, 2, 4, 717–740. DOI: 10.3390/iot2040036.

[37] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer. ISBN: 978-3-662-43838-1. DOI: 10.1007/978-3-662-43839-8.

[38] ZVEI. 2017. Examples of the Asset Administration Shell for Industrie 4.0 Components - Basic Part. Technischer Bericht. Zentralverband Elektrotechnik- und Elektronikindustrie e. V. https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2017/April/Asset_Administration_Shell/ZVEI_WP_Verwaltungschale_Englisch_Download_03.04.17.pdf.

Anhang A

Architecture Decision Records

Die folgenden Architecture Decision Records (ADRs) dokumentieren die wesentlichen Architekturentscheidungen im MADR-Format [28]. Sie sind unterteilt in allgemeine Architekturentscheidungen, die unabhängig vom konkreten Anwendungsfall Gültigkeit besitzen, sowie anwendungsfallspezifische Entscheidungen, die an den konkreten Kontext der Fertigungsmaschine und der AAS-Integration gebunden sind.

A.1 ADR-001: Zweigeteilte Architektur – Daten- und Integrationsarchitektur und AAS-Architektur

Status

Accepted

Kontext und Problemstellung

Die bestehende Systemlandschaft umfasst eine Vielzahl heterogener Quell- und Zielsysteme aus dem IT- und OT-Umfeld, darunter produktionsnahe Systeme, Engineering-Systeme sowie Unternehmensanwendungen. Diese Systeme unterscheiden sich insbesondere hinsichtlich Änderungsfrequenz, Datenmodellen und verfügbaren Schnittstellen.

Ziel der Architektur ist es, eine konsistente und wiederverwendbare Integration dieser Daten zu ermöglichen und gleichzeitig eine standardisierte, interoperable Bereitstellung von Asset-Informationen über die Asset Administration Shell (AAS) zu realisieren. Neben dem AAS-Anwendungsfall sollen die integrierten Daten jedoch auch für weitere Anwendungsfälle wie Analytik oder nachgelagerte Services nutzbar bleiben.

Wie soll die Gesamtarchitektur strukturiert werden, um sowohl eine wiederverwendbare Datenintegration als auch eine standardisierte AAS-Bereitstellung zu ermöglichen, ohne die beiden Anliegen eng zu koppeln?

Entscheidungstreiber

- Trennung der Verantwortlichkeiten zwischen Datenintegration und standardisierter Asset-Repräsentation
- Wiederverwendbarkeit der integrierten Daten unabhängig vom AAS-Kontext
- Änderbarkeit und Evolvierbarkeit beider Teilbereiche
- Reduktion der Kopplung zwischen Datenverarbeitung und AAS-Infrastruktur
- Abbildung realistischer Betriebsmodelle mit getrennten Deployment-Einheiten

Betrachtete Optionen

- Zweigeteilte Architektur (Daten- und Integrationsarchitektur + AAS-Architektur)
- Monolithische Architektur (Integration und AAS in einem System)
- AAS-zentrierte Integrationsarchitektur
- Reine Daten- und Integrationsarchitektur ohne AAS

Entscheidungsergebnis

Gewählte Option: „Zweigeteilte Architektur (Daten- und Integrationsarchitektur + AAS-Architektur)“, weil sie eine klare Trennung zwischen Datenintegration und AAS-Repräsentation ermöglicht und gleichzeitig die Wiederverwendbarkeit der integrierten Daten für nicht-AAS-bezogene Anwendungsfälle sicherstellt.

Die Zielarchitektur wird in zwei logisch getrennte Architekturbereiche aufgeteilt:

- **Daten- und Integrationsarchitektur**: Die Daten- und Integrationsarchitektur umfasst die Aufnahme, Integration, Harmonisierung und Veredelung der Daten aus den angebundenen Quellsystemen. Die Daten werden in Form strukturierter und kuratierter Datenprodukte bereitgestellt. Die produktive Nutzung der Daten- und Integrationsarchitektur ist unabhängig vom AAS-Anwendungsfall vorgesehen.
- **AAS-Architektur**: Die AAS-Architektur umfasst die AAS-konforme Repräsentation von Assets, deren Verwaltung sowie die standardisierte Bereitstellung über AAS-Schnittstellen. Als Datenbasis dienen ausschließlich kuratierte *Gold*-Datenprodukte aus der Daten- und Integrationsarchitektur. Die Erstellung, Aktualisierung und Löschung von AAS-Instanzen erfolgt über die entsprechenden APIs.

Die Kopplung zwischen beiden Architekturbereichen erfolgt ausschließlich über die bereitgestellten Gold-Datenprodukte. Eine direkte Kopplung der AAS-Architektur an Event-Streaming-Infrastrukturen oder Rohdatenquellen ist explizit ausgeschlossen.

Konsequenzen

+ Klare Verantwortungszuordnung zwischen Datenintegration und AAS
+ Hohe Wiederverwendbarkeit der integrierten Daten
+ Schlanker Mapping-Service mit geringem Implementierungsaufwand
– Zusätzliche Synchronisationslogik zwischen Daten- und Integrationsarchitektur und AAS
– Aktualität der AAS-Zustandsdaten ist an die Aktualisierungsfrequenz der Gold-Datenprodukte gebunden

Vor- und Nachteile der Optionen

Zweigeteilte Architektur (Daten- und Integrationsarchitektur + AAS-Architektur)

+ Klare Trennung der Verantwortlichkeiten reduziert Komplexität und erhöht Wartbarkeit
+ Integrierte Daten auch außerhalb des AAS-Kontexts nutzbar
+ Änderungen an AAS-Standards wirken sich nicht auf die Datenintegrationslogik aus
+ Gold-Datenprodukte als stabile Schnittstelle verhindern enge Kopplung

- Zusätzliche Synchronisationskomponente erforderlich
- Latenz durch indirekte Kopplung über Gold-Layer

Monolithische Architektur (Integration und AAS in einem System)

+ Reduziert initiale Systemkomplexität
- Führt zu starker Kopplung und eingeschränkter Wiederverwendbarkeit
- Erschwerte Wartbarkeit bei wachsender Funktionalität

AAS-zentrierte Integrationsarchitektur

+ Einheitliche AAS-Semantik über alle Datenflüsse
- Erzwingt frühe AAS-Semantik in allen Datenflüssen
- Erhöht Transformationsaufwand für nicht-AAS-bezogene Anwendungsfälle

Reine Daten- und Integrationsarchitektur ohne AAS

+ Maximiert Wiederverwendbarkeit der Daten
- Erfüllt nicht die Anforderungen an standardisierte und interoperable Bereitstellung von Asset-Informationen

Weitere Informationen

Diese Entscheidung bildet die Grundlage für weitere Architekturentscheidungen, insbesondere zur Integrationsstrategie (ADR-002), zur Daten- und Integrationsarchitektur (ADR-005), zur AAS-Implementierung (ADR-007) sowie zur Synchronisationslogik (ADR-008).

A.2 ADR-002: Integrationsstrategie nach Änderungscharakteristik

Status

Accepted

Kontext und Problemstellung

Innerhalb der Daten- und Integrationsarchitektur werden Daten aus unterschiedlichen Quellsystemen integriert, die sich hinsichtlich Änderungsfrequenz, fachlicher Relevanz der Änderungen, Datenvolumen sowie verfügbarer Schnittstellen deutlich unterscheiden. Insbesondere Systeme wie das ERP und PLM erzeugen fachlich relevante Änderungen ereignisbasiert, während andere Systeme wie TDM oder das MES überwiegend statische oder selten veränderliche Daten bereitstellen und primär abfrageorientiert genutzt werden.

Soll eine einheitliche Integrationsmethode für alle Quellsysteme eingesetzt werden, oder soll die Integrationsstrategie systemabhängig nach der jeweiligen Änderungscharakteristik gewählt werden?

Entscheidungstreiber

- Unterschiedliche Änderungsfrequenzen und Ereignischarakteristiken der Quellsysteme
- Fachliche Relevanz und Zeitkritikalität der Änderungen
- Anzahl potenzieller Konsumenten und Entkopplungsanforderungen
- Datenvolumen und Änderungsumfang je Quellsystem
- Latenzanforderungen je Datenquelle
- Kostenstruktur (Infrastruktur- vs. Entwicklungskosten)

Betrachtete Optionen

- Differenzierte Integrationsstrategie (Event-Streaming + REST)
- Einheitliche Event-Streaming-Integration aller Systeme
- Einheitliche REST-basierte Integration aller Systeme
- Batch-orientierte Integration

Entscheidungsergebnis

Gewählte Option: „Differenzierte Integrationsstrategie (Event-Streaming + REST)“, weil die Quellsysteme sich grundlegend in ihrer Änderungscharakteristik unterscheiden und eine bedarfsgerechte Wahl der Integrationsmethode sowohl den fachlichen Anforderungen als auch der wirtschaftlichen Abwägung am besten gerecht wird.

Die Wahl der Integrationsmethode erfolgt systemabhängig:

- **Event-Streaming** wird für Systeme mit hoher Änderungsfrequenz oder ereignisbasierten Änderungen eingesetzt:
  - **ERP**: Bei jeder Änderung eines Business Objekts wird über einen Datenbank-Trigger ein Event erzeugt, das den vollständigen Datensatz des geänderten Objekts enthält.

  - **CAD/ CAM**: Änderungen an NC-Programmen werden ereignisbasiert bereitgestellt.

- **REST-basierte Integration** wird für Systeme mit geringer oder moderater Änderungsfrequenz eingesetzt:
  - **TDM**: Statische Werkzeug-, Fixture-, NC-Programm- und Maschinenstammdaten werden periodisch erfasst (Polling-Intervall: 6 Stunden).
  - **MES**: Daten werden ebenfalls REST-basiert integriert (Polling-Intervall: 2 Stunden).

Konsequenzen

+ Bedarfsgerechte Wahl der Integrationsmethode je System
+ Wirtschaftlich ausgewogene Lösung hinsichtlich Infrastruktur- und Betriebskosten
+ Skalierbare und entkoppelte Datenverarbeitung für hochdynamische Quellen
– Erhöhte architektonische Komplexität durch parallele Integrationsmechanismen
– Unterschiedliche Aktualitätsniveaus je nach Datenquelle

Vor- und Nachteile der Optionen

Differenzierte Integrationsstrategie (Event-Streaming + REST)

+ Optimale Passung zwischen Systemcharakteristik und Integrationsmethode
+ Wirtschaftlich effizient durch bedarfsgerechten Ressourceneinsatz
+ Skalierbare Entkopplung für hochdynamische Quellen
– Erhöhte architektonische Komplexität durch parallele Mechanismen
– Unterschiedliche Aktualitätsniveaus je Datenquelle

Einheitliche Event-Streaming-Integration aller Systeme

+ Einheitliche Integrationsarchitektur
+ Konsistente Verarbeitungssemantik
– Unnötige Komplexität und höhere Betriebskosten für Systeme mit geringer Änderungsfrequenz
– Überdimensionierte Infrastruktur für statische Datenquellen

Einheitliche REST-basierte Integration aller Systeme

+ Reduzierte Infrastrukturkosten
+ Einfache Implementierung
– Skaliert schlecht bei hoher Änderungsfrequenz
– Erschwert entkoppelte Weiterverarbeitung durch mehrere Konsumenten

Batch-orientierte Integration

+ Vereinfacht die Integration
– Erfüllt nicht die Anforderungen an Aktualität und Reaktionsfähigkeit
– Ungeeignet für ereignisbasierte Änderungen

Weitere Informationen

Diese Entscheidung beeinflusst insbesondere die Wahl der Event-Streaming-Plattform (ADR-003), das Topic-Design (ADR-004), die Daten- und Integrationsarchitektur (ADR-005) sowie die Synchronisation der kuratierten Datenprodukte mit der AAS-Architektur (ADR-008).

A.3 ADR-003: Auswahl der Event-Streaming-Plattform

Status

Accepted

Kontext und Problemstellung

Im Rahmen der Integrationsstrategie (vgl. ADR-002) werden für Systeme mit hoher Änderungsfrequenz oder ereignisbasierten Änderungen Events erzeugt und über eine zentrale Event-Streaming-Infrastruktur an die Daten- und Integrationsarchitektur weitergeleitet. Diese Infrastruktur bildet den Backbone für die entkoppelte, skalierbare Weiterverarbeitung der Ereignisse durch unterschiedliche Konsumenten.

Welche Event-Streaming-Plattform soll als zentrale Infrastruktur für die ereignisbasierte Integration eingesetzt werden?

Entscheidungstreiber

- Skalierbarkeit und Durchsatz für hohe Ereignisraten
- Entkopplung von Produzenten und Konsumenten
- Zuverlässigkeit und Persistenz der Events
- Verfügbarkeit eines ausgereiften Ökosystems und Integrationsmöglichkeiten
- Reduktion des operativen Aufwands durch Managed Services

Betrachtete Optionen

- Apache Kafka über Confluent
- RabbitMQ oder andere klassische Message Broker
- Eigenentwickelte REST-basierte Event-Verteilung
- Batch-orientierte Datenübertragung

Entscheidungsergebnis

Gewählte Option: „Apache Kafka über Confluent“, weil Kafka die Anforderungen an Skalierbarkeit, Persistenz und entkoppelte Verarbeitung am besten erfüllt und Confluent den operativen Aufwand durch ein ausgereiftes Ökosystem und professionellen Support reduziert.

Apache Kafka übernimmt die zuverlässige Aufnahme, Persistierung und Verteilung der Events aus den angebundenen Quellsystemen (z. B. ERP, CAD/CAM) und stellt diese der Daten- und Integrationsarchitektur zur weiteren Verarbeitung zur Verfügung.

Konsequenzen

+ Skalierbare und robuste Event-Streaming-Infrastruktur
+ Einfache Erweiterbarkeit durch zusätzliche Konsumenten
+ Unterstützung von Reprocessing und Fehlerbehandlung
– Laufende Infrastruktur- und Betriebskosten
– Zusätzlicher konzeptioneller Aufwand im Vergleich zu rein REST-basierten Integrationsansätzen

Vor- und Nachteile der Optionen

Apache Kafka über Confluent

+ Horizontale Skalierung durch Partitionierung
+ Publish-Subscribe-Modell ermöglicht entkoppelte Konsumenten
+ Dauerhafte Persistenz erlaubt Reprocessing und Fehlerbehandlung
+ Ausgereiftes Ökosystem mit professionellem Support
– Laufende Infrastruktur- und Betriebskosten
– Höhere Einstiegskomplexität gegenüber einfacheren Ansätzen

RabbitMQ oder andere klassische Message Broker

+ Geeignet für punktuelle asynchrone Kommunikation
+ Geringere Einstiegskomplexität
– Keine vergleichbare Skalierbarkeit und Persistenz
– Eingeschränkte Reprocessing-Fähigkeit

Eigenentwickelte REST-basierte Event-Verteilung

+ Reduzierte Infrastrukturkosten
– Skaliert schlecht bei hoher Ereignisrate
– Starke Kopplung zwischen Produzenten und Konsumenten

Batch-orientierte Datenübertragung

+ Vereinfacht den Betrieb
– Erfüllt nicht die Anforderungen an zeitnahe Verarbeitung
– Ungeeignet für ereignisbasierte Integration

Weitere Informationen

Diese Entscheidung baut auf der Integrationsstrategie aus ADR-002 auf und beeinflusst insbesondere das Topic-Design (ADR-004), die Daten- und Integrationsarchitektur (ADR-005) sowie die Verarbeitung und Bereitstellung der Events in der Daten- und Integrationsarchitektur.

A.4 ADR-004: Topic-Zuschnitt und Trennung von Ereignis- und Zustandsdaten

Status

Accepted

Kontext und Problemstellung

Im Rahmen der ereignisbasierten Integration (vgl. ADR-002) werden Daten aus hochdynamischen Quellsystemen wie einem ERP- und einem PLM-System über die Event-Streaming-Plattform (vgl. ADR-003) in die Daten- und Integrationsarchitektur übertragen. Diese Daten unterscheiden sich wesentlich hinsichtlich Semantik, Änderungsfrequenz und fachlicher Bedeutung.

Insbesondere bei Maschinen-Assets ist zwischen häufig auftretenden ereignisbasierten Änderungen (z. B. Prozess- oder Zustandsänderungen) und vergleichsweise seltenen, zustandsorientierten Stammdaten zu unterscheiden.

Wie sollen die Event-Streaming-Daten in Topics organisiert werden, um den unterschiedlichen Semantiken und Verarbeitungsanforderungen gerecht zu werden?

Entscheidungstreiber

- Semantische Unterschiede zwischen Ereignisdaten und Zustandsdaten
- Unterschiedliche Änderungsfrequenzen je Datenart
- Vereinfachung der Weiterverarbeitung für nachgelagerte Konsumenten
- Anforderungen an differenzierte Retention- und Reprocessing-Strategien
- Erweiterbarkeit für zusätzliche Quellsysteme und Asset-Typen

Betrachtete Optionen

- Logisch getrennte Topics nach Datenart (MachineEvents + MasterData)
- Einheitliches Topic für alle Datenarten
- Topic pro Quellsystem
- Feingranulare Topics pro Asset-Typ und Ereignisart

Entscheidungsergebnis

Gewählte Option: „Logisch getrennte Topics nach Datenart (MachineEvents + MasterData)“, weil die unterschiedlichen Semantiken und Änderungsfrequenzen eine dedizierte Verarbeitung und differenzierte Retention-Strategien erfordern.

Die Event-Streaming-Daten werden wie folgt aufgeteilt:

- **MachineEvents Topics**: Enthalten ereignisbasierte Daten zu Maschinen-Assets mit hoher Änderungsfrequenz, die häufige fachliche Änderungen oder Zustandsänderungen repräsentieren.
- **MasterData Topics**: Enthalten zustandsorientierte Stammdaten zu Assets. Jeder Datensatz repräsentiert den vollständigen aktuellen Zustand eines Assets (*State Events*).

Für Maschinen-Assets existieren logisch zwei getrennte Topics (*MachineEvents* und *MasterData*). Für andere Asset-Typen wird ausschließlich ein MasterData Topic verwendet. Löschungen von Assets werden explizit als dedizierte Delete-Events abgebildet.

Innerhalb der AAS-Architektur werden die Daten nicht direkt aus den Bronze-Topics bezogen, sondern ausschließlich aus dedizierten, aufbereiteten Topics nach der Verarbeitung innerhalb der Daten- und Integrationsarchitektur.

Konsequenzen

+ Klare und stabile Datenverträge je Topic
+ Reduzierte Komplexität für Konsumenten
+ Flexible Retention- und Verarbeitungsstrategien
– Erhöhte Anzahl von Topics
– Zusätzlicher konzeptioneller Aufwand bei der Definition der Topic-Struktur

Vor- und Nachteile der Optionen

Logisch getrennte Topics nach Datenart (MachineEvents + MasterData)

+ Klare semantische Trennung reduziert Verarbeitungskomplexität
+ Differenzierte Retention- und Reprocessing-Strategien möglich
+ Konsumenten können gezielt relevante Topics konsumieren
+ Erweiterbar für weitere Asset-Typen
– Erhöhte Anzahl von Topics
– Zusätzlicher konzeptioneller Aufwand für Topic-Struktur

Einheitliches Topic für alle Datenarten

+ Reduziert Anzahl der Topics
– Komplexe Datenstrukturen innerhalb eines Topics
– Erschwerte Verarbeitung und geringe Flexibilität für Konsumenten

Topic pro Quellsystem

+ Vereinfacht die Erzeugung der Events
– Koppelt Konsumenten stark an die Struktur der Quellsysteme
– Erschwert domänenorientierte Weiterverarbeitung

Feingranulare Topics pro Asset-Typ und Ereignisart

+ Sehr präzise Trennung
– Erhöht betriebliche und konzeptionelle Komplexität erheblich
– Schwer skalierbar bei wachsender Anzahl von Asset-Typen

Weitere Informationen

Diese Entscheidung baut auf der Integrationsstrategie (ADR-002) und der Auswahl der Event-Streaming-Plattform (ADR-003) auf und beeinflusst maßgeblich die Daten- und Integrationsarchitektur (ADR-005) sowie die Synchronisation der aufbereiteten Datenprodukte mit der AAS-Architektur (ADR-008).

A.5 ADR-005: Daten- und Integrationsarchitektur nach dem Bronze/Silver/Gold-Prinzip

Status

Accepted

Kontext und Problemstellung

Innerhalb der Daten- und Integrationsarchitektur werden Daten aus unterschiedlichen Quellsystemen über verschiedene Integrationsmechanismen (vgl. ADR-002) und verarbeitet diese ereignisbasiert über eine Event-Streaming-Plattform (vgl. ADR-003) mit definiertem Topic-Zuschnitt (vgl. ADR-004).

Die integrierten Daten sollen sowohl für analytische Zwecke als auch als Grundlage für nachgelagerte Systeme, insbesondere die Asset Administration Shell (AAS), dienen. Dabei bestehen unterschiedliche Anforderungen an Datenqualität, Stabilität, Semantik und Aktualität.

Wie soll die Datenverarbeitung innerhalb der Daten- und Integrationsarchitektur strukturiert werden, um den unterschiedlichen Anforderungen an Datenqualität, Stabilität und Wiederverwendbarkeit gerecht zu werden?

Entscheidungstreiber

- Trennung von Integrations- und Nutzungssicht
- Wiederverwendbarkeit der Daten für unterschiedliche Anwendungsfälle
- Stabilität der Schnittstellen zu nachgelagerten Systemen
- Fehlerbehandlung und Reprocessing-Fähigkeit
- Evolvierbarkeit der Transformationslogik ohne Auswirkung auf Konsumenten

Betrachtete Optionen

- Mehrstufige Daten- und Integrationsarchitektur nach dem Bronze/Silver/Gold-Prinzip
- Einheitlicher Datenlayer
- Direkte Weiterverarbeitung aus Streaming-Topics
- Batch-orientierte Data-Warehouse-Architektur

Entscheidungsergebnis

Gewählte Option: „Mehrstufige Daten- und Integrationsarchitektur nach dem Bronze/Silver/Gold-Prinzip“, weil sie eine klare Strukturierung der Datenverarbeitung in Integrations-, Harmonisierungs- und Nutzungsschicht ermöglicht und stabile Datenverträge für nachgelagerte Systeme bereitstellt.

Die Daten- und Integrationsarchitektur wird wie folgt strukturiert:

- **Bronze-Layer**: Der Bronze-Layer dient der rohdatennahen Aufnahme der integrierten Daten. Events und Datensätze werden weitgehend unverändert persistiert und repräsentieren die Eingangssicht der jeweiligen Datenquelle.

- **Silver-Layer**: Der Silver-Layer übernimmt die fachliche Harmonisierung, Bereinigung und Normalisierung der Daten. Hier erfolgen unter anderem Schemaanpassungen, Konsolidierung von Datensätzen sowie die Vereinheitlichung von Identifikatoren.
- **Gold-Layer**: Der Gold-Layer stellt kuratierte, fachlich zugeschnittene Datenprodukte bereit. Diese sind stabil versioniert, auf konkrete Assets ausgerichtet und dienen als verbindliche Schnittstelle für nachgelagerte Systeme.

Innerhalb der AAS-Architektur werden ausschließlich Daten aus dem Gold-Layer bezogen. Der Gold-Layer fungiert damit als stabiler Contract zwischen Daten- und Integrationsarchitektur und AAS-Architektur.

Konsequenzen

+ Klare Strukturierung der Daten- und Integrationsarchitektur
+ Stabile Datenprodukte für nachgelagerte Systeme
+ Hohe Transparenz und Nachvollziehbarkeit der Datenverarbeitung
– Zusätzlicher konzeptioneller und technischer Aufwand
– Erhöhte Latenz durch mehrstufige Verarbeitung

Vor- und Nachteile der Optionen

Mehrstufige Daten- und Integrationsarchitektur nach dem Bronze/Silver/Gold-Prinzip

+ Klare Trennung von Integrations- und Nutzungssicht
+ Stabile und versionierbare Datenverträge im Gold-Layer
+ Persistierung der Rohdaten erlaubt Reprocessing bei Änderungen der Transformationslogik
+ Schrittweise Umsetzung von Änderungen ohne bestehende Konsumenten zu beeinträchtigen
– Zusätzlicher konzeptioneller und technischer Aufwand
– Erhöhte Latenz durch mehrstufige Verarbeitung

Einheitlicher Datenlayer

+ Reduziert initiale Komplexität
– Starke Kopplung von Integrations- und Nutzungssicht
– Eingeschränkte Wiederverwendbarkeit

Direkte Weiterverarbeitung aus Streaming-Topics

+ Reduziert Zwischenschritte und Latenz
– Erhöht Kopplung an Streaming-Details
– Erschwert stabile Datenverträge

Batch-orientierte Data-Warehouse-Architektur

+ Bewährtes Architekturmuster für analytische Szenarien
– Weniger geeignet für ereignisbasierte Integrationsszenarien
– Eingeschränkte Flexibilität bei Reprocessing-Anforderungen

Weitere Informationen

Diese Entscheidung baut auf der Integrationsstrategie (ADR-002), der Event-Streaming-Plattform (ADR-003) und dem Topic-Zuschnitt (ADR-004) auf und bildet die Grundlage für die Synchronisation der kuratierten Datenprodukte mit der AAS-Architektur (ADR-008).

A.6 ADR-006: Maschinenanbindung über OPC UA Subscriptions

Status

Accepted

Kontext und Problemstellung

Zur Erfassung von laufenden Zustands- und Prozessdaten aus dem Produktionsumfeld müssen Maschinen und Linien aus dem OT-Bereich an die Daten- und Integrationsarchitektur angebunden werden. Diese Daten weisen eine vergleichsweise hohe Änderungsfrequenz auf und besitzen eine hohe fachliche Relevanz für nachgelagerte Auswertungen sowie für die Abbildung aktueller Zustände in der Asset Administration Shell (AAS).

Gleichzeitig bestehen organisatorische und sicherheitstechnische Anforderungen, die eine klare Trennung zwischen OT- und IT-Netzwerken vorsehen. Eine direkte Kopplung der Daten- und Integrationsarchitektur an Maschinensteuerungen ist daher nicht zulässig.

Wie sollen Maschinen an die Daten- und Integrationsarchitektur angebunden werden, um Zustands- und Prozessdaten zeitnah und sicher zu erfassen, ohne die OT/IT-Netztrennung zu verletzen?

Entscheidungstreiber

- Standardisierte und herstellerunabhängige Maschinenanbindung
- Ereignisbasierte Erfassung von Zustandsänderungen
- Latenzanforderungen im Sekunden- bis Minutenbereich
- Einhaltung der OT/IT-Netztrennung und Sicherheitsanforderungen
- Skalierbarkeit für schrittweise Erweiterung um weitere Maschinen

Betrachtete Optionen

- OPC UA Subscriptions mit Gateway-Übergabepunkt
- Polling-basierte Maschinenanbindung
- Direkte Anbindung der Maschinen an die Daten- und Integrationsarchitektur
- Proprietäre Hersteller-APIs

Entscheidungsergebnis

Gewählte Option: „OPC UA Subscriptions mit Gateway-Übergabepunkt“, weil OPC UA als herstellerunabhängiger Standard eine ereignisbasierte Erfassung ermöglicht und die geforderte OT/IT-Netztrennung durch einen dedizierten Übergabepunkt eingehalten wird.

Maschinen stellen ihre Zustands- und Prozessdaten über einen OPC-UA-Server bereit. Ein dedizierter Übergabepunkt zwischen OT- und IT-Netzwerk wird durch diesen OPC-UA-Server bzw. ein zugehöriges Gateway realisiert. Im Prototyp wird der Übergabepunkt durch Orchestra umgesetzt. Orchestra übernimmt die Subscription-basierte Erfassung von OPC-UA-Änderungen, das Mapping der ValueSets in ein JSON-Zielformat sowie die Publikation der resultierenden Ereignisse in Kafka-Topics gemäß ADR-003 und ADR-004.

Die Kommunikation basiert auf Subscription-Mechanismen, um Änderungen zeitnah und effizient zu erfassen.

Konsequenzen

+ Standardisierte und herstellerunabhängige Maschinenanbindung
+ Ereignisbasierte Erfassung von Zustands- und Prozessdaten
+ Klare Trennung von OT- und IT-Bereich durch definierte Übergabepunkte
– Zusätzlicher Betriebs- und Konfigurationsaufwand für OPC-UA-Server
– Abhängigkeit von der Qualität und Vollständigkeit der bereitgestellten OPC-UA-Datenmodelle
– Zusätzlicher Betriebs- und Governance-Aufwand für die Integrationskomponente
– Potenzieller Vendor-Lock-in durch tool-spezifische Mapping- und Flow-Modelle

Vor- und Nachteile der Optionen

OPC UA Subscriptions mit Gateway-Übergabepunkt

+ Etablierter, herstellerunabhängiger Standard im industriellen Umfeld
+ Push-basierte Übertragung effizienter und aktueller als Polling
+ Aktualisierungen im Sekundenbereich möglich
+ Skalierbare Anbindung pro Maschine bzw. Linie
– Betriebs- und Konfigurationsaufwand für OPC-UA-Server und Gateway
– Abhängigkeit von Qualität der OPC-UA-Datenmodelle
– Potenzieller Vendor-Lock-in durch tool-spezifische Flow-Modelle

Polling-basierte Maschinenanbindung

+ Reduzierte Komplexität der Kommunikation
– Höhere Latenz und ineffiziente Nutzung von Netzwerkressourcen
– Änderungen zwischen Polling-Intervallen werden nicht erfasst

Direkte Anbindung der Maschinen an die Daten- und Integrationsarchitektur

+ Geringe Latenz
– Widerspricht den Anforderungen an Netztrennung und Sicherheit
– Eingeschränkte Wartbarkeit

Proprietäre Hersteller-APIs

+ Tiefe Integration mit herstellerspezifischen Funktionen
– Reduzierte Interoperabilität
– Langfristig erhöhter Integrationsaufwand

Weitere Informationen

Diese Entscheidung beeinflusst die Integrationsstrategie (ADR-002), das Event-Streaming-Setup (ADR-003) sowie den Topic-Zuschnitt für Maschinenereignisse (ADR-004) und bildet die Grundlage für die Weiterverarbeitung der Maschinendaten in der Daten- und Integrationsarchitektur.

A.7 ADR-007: Auswahl des AAS-Stacks

Status

Accepted

Kontext und Problemstellung

Zur standardisierten Repräsentation und Bereitstellung von Asset-Informationen wird in der Zielarchitektur eine Asset Administration Shell (AAS) eingesetzt (vgl. ADR-001). Die AAS soll als interoperable Zugriffsschicht für externe Systeme dienen und sowohl statische Asset-Informationen als auch laufende Zustandsdaten abbilden.

Welche AAS-Implementierung soll als technische Grundlage für die AAS-Architektur eingesetzt werden, um Standardkonformität, Modularität und Interoperabilität sicherzustellen?

Entscheidungstreiber

- Konformität mit der AAS v3 Spezifikation (IEC 63278-1)
- Modularer Aufbau mit klarer Verantwortungszuordnung
- Interoperabilität über standardisierte REST-Schnittstellen
- Unterstützung konkreter Submodel-Instanzen für realitätsnahe Validierung
- Reifegrad, Ökosystem und aktive Weiterentwicklung

Betrachtete Optionen

- Eclipse BaSyx v2 (AAS v3)
- FA^{3}ST Service (Fraunhofer)
- AASX Server
- Eigenimplementierung einer AAS-Runtime
- Kommerzielle AAS-Lösungen
- Ältere AAS-Implementierungen (AAS v2.x)

Entscheidungsergebnis

Gewählte Option: „Eclipse BaSyx v2 (AAS v3)“, weil BaSyx als etablierte Open-Source-Implementierung die AAS v3 Spezifikation umsetzt und durch seinen modularen Aufbau eine flexible, erweiterbare Architektur mit klarer Verantwortungszuordnung ermöglicht.

Der BaSyx-Stack wird mit folgenden Kernkomponenten betrieben:

- AAS Environment
- AAS Registry
- Submodel Registry
- AAS Discovery Service
- BaSyx Web UI

Der Zugriff auf die AAS erfolgt primär über die standardisierte REST-API. Zusätzlich wird eine Benutzeroberfläche für Entwicklungs-, Test- und Visualisierungszwecke eingesetzt.

Konsequenzen

+ Standardkonforme und interoperable AAS-Implementierung
+ Modulare und erweiterbare AAS-Architektur
+ Klare Trennung zwischen Datenplattform und AAS-Zugriffsschicht
– Zusätzlicher Betriebsaufwand für mehrere AAS-Komponenten
– Einarbeitungsaufwand in Spezifikation und BaSyx-Architektur

Vor- und Nachteile der Optionen

Eclipse BaSyx v2 (AAS v3)

+ Implementiert aktuelle AAS v3 Spezifikation
+ Modularer Aufbau (Registry, Repository, Discovery) erlaubt flexible Architektur
+ Standardisierte REST-Schnittstellen für Anbindung externer Systeme
+ Etabliertes Open-Source-Projekt mit aktiver Weiterentwicklung
– Betriebsaufwand für mehrere Komponenten
– Einarbeitungsaufwand in Spezifikation und Architektur

FA³ST Service (Fraunhofer)

+ Open-Source-Implementierung mit AAS v3-Unterstützung
+ Leichtgewichtiger Einzelprozess-Ansatz mit geringem Betriebsaufwand
– Geringerer Modulierungsgrad als BaSyx – Registry, Discovery und Repository sind nicht als eigenständige Dienste trennbar
– Kleinere Community und weniger Referenzprojekte im industriellen Umfeld

AASX Server

+ Direkte Integration mit dem AASX Package Explorer-Ökosystem
+ Unterstützung des AASX-Paketformats als nativer Persistenz
– .NET-basiert – weicht vom Python/Container-Technologiestack der übrigen Architektur ab
– Primär als Begleitwerkzeug zum Package Explorer konzipiert, nicht als produktive Runtime-Plattform

Eigenimplementierung einer AAS-Runtime

+ Maximale Flexibilität
– Erheblicher Implementierungs- und Wartungsaufwand
– Standardkonformität schwer sicherzustellen

Kommerzielle AAS-Lösungen

+ Teilweise zusätzlicher Funktionsumfang
– Höherer Lock-in und geringere Transparenz
– Eingeschränkte Anpassbarkeit

Ältere AAS-Implementierungen (AAS v2.x)

+ Teilweise ausgereifter und länger erprobt
– Weniger zukunftssicher
– Unterstützen nicht vollständig die aktuellen Spezifikationen

Weitere Informationen

Diese Entscheidung baut auf der Trennung von Daten- und Integrationsarchitektur und AAS-Architektur (ADR-001) auf und bildet die Grundlage für die Synchronisation der kuratierten Datenprodukte aus dem Gold-Layer mit der AAS (ADR-008).

A.8 ADR-008: Synchronisations- und Mapping-Strategie zwischen Gold-Layer und AAS

Status

Accepted

Kontext und Problemstellung

Innerhalb der Daten- und Integrationsarchitektur werden kuratierte und fachlich zugeschnittene Datenprodukte im Gold-Layer bereitgestellt (vgl. ADR-005). Diese Datenprodukte dienen als stabile Schnittstelle für nachgelagerte Systeme und bilden die Grundlage für die AAS-Architektur (vgl. ADR-001, ADR-007).

Eine direkte Kopplung der AAS an Event-Streaming-Topics oder vorgelagerte Verarbeitungsstufen ist gemäß ADR-001 ausgeschlossen. Es ist daher eine Strategie erforderlich, die Gold-Datenprodukte auf AAS-Instanzen und Submodelle abzubilden.

Wie sollen die kuratierten Gold-Datenprodukte auf AAS-konforme Zielstrukturen projiziert werden, und wo liegt die Verantwortung für diese Transformation?

Entscheidungstreiber

- Klare Verantwortungszuordnung zwischen Zustandsbildung und semantischer Projektion
- Stabile Schnittstellen zwischen Daten- und Integrationsarchitektur und AAS-Architektur
- Zentralisierung der AAS-spezifischen Strukturierung an einer Stelle
- Idempotente Verarbeitung durch vollständige Zustandsübertragung
- Evolvierbarkeit ohne Kopplung zwischen Datenmodell- und AAS-Änderungen

Betrachtete Optionen

- Dedizierter Mapping-Service mit deklarativer Konfiguration
- Direkte Event-basierte Aktualisierung der AAS
- Fachliche Transformation innerhalb des Mapping-Service
- On-Demand-Synchronisation (Pull-Prinzip)

Entscheidungsergebnis

Gewählte Option: „Dedizierter Mapping-Service mit deklarativer Konfiguration“, weil diese Variante die Verantwortung für die semantische Projektion klar vom fachlichen Zustandsmanagement der Daten- und Integrationsarchitektur trennt und durch deklarative Mapping-Regeln eine wartbare, erweiterbare Transformation ermöglicht.

Die Synchronisation zwischen Daten- und Integrationsarchitektur und AAS-Architektur erfolgt ausschließlich auf Basis der Gold-Datenprodukte. Ein dedizierter Mapping-Service bezieht die aufbereiteten, asset-zugeschnittenen Daten aus dem Gold-Layer und projiziert diese anhand deklarativer Mapping-Regeln auf AAS-konforme Zielstrukturen. Die Abbildung erfolgt template-basiert: stabile Submodel-Strukturen werden durch Mapping-Konfigurationen definiert, Werte werden zur Laufzeit aus den Gold-Ereignissen abgeleitet. Die resultierenden AAS-Instanzen und Submodelle werden über CRUD-Operationen gegen die BaSyx-REST-APIs persistiert.

Die Gold-Datenprodukte enthalten bereits:

- eine eindeutige Asset-Identifikation,
- den vollständigen aktuellen Zustand des Assets,
- alle für die AAS relevanten Attribute.

Der Mapping-Service führt keine zusätzliche fachliche Zustandsbildung im Sinne der Medaillen-Architektur durch, sondern setzt den im Gold-Layer bereitgestellten Referenzzustand als Eingabe voraus. Seine Verantwortung liegt ausschließlich in der semantischen Projektion dieses Zustands auf die Zielstruktur der Verwaltungsschale.

Konsequenzen

+ Klare Trennung zwischen Zustandsbildung (Daten- und Integrationsarchitektur) und semantischer Modellierung (Mapping-Service)
+ Deklarative Mapping-Regeln ermöglichen Erweiterung um neue Submodelle ohne Code-Änderungen
+ Stabile und nachvollziehbare Synchronisation über Gold-Datenvertrag
– Zusätzliche Synchronisationskomponente erforderlich
– Aktualität der AAS ist an die Aktualisierungsfrequenz der Gold-Datenprodukte gebunden
– Steigende Transformationskomplexität kann Lesbarkeit der deklarativen Regeln einschränken

Vor- und Nachteile der Optionen

Dedizierter Mapping-Service mit deklarativer Konfiguration

+ Klare Verantwortungszuordnung: Mapping-Service übernimmt ausschließlich semantische Projektion
+ Gold-Layer als versionierbarer Contract verhindert enge Kopplung
+ Zentralisierung der AAS-Strukturierung an einer Stelle
+ Idempotente Verarbeitung durch vollständige Zustandsübertragung
+ Erweiterbar durch zusätzliche Mapping-Konfigurationen ohne Code-Änderungen
– Zusätzliche Synchronisationskomponente erforderlich
– Lesbarkeit der deklarativen Regeln bei steigender Komplexität eingeschränkt

Direkte Event-basierte Aktualisierung der AAS

+ Geringere Latenzen
– Starke Kopplung an Streaming-Details
– Erschwert stabile Datenverträge

Fachliche Transformation innerhalb des Mapping-Service

+ Reduziert Komplexität der Daten- und Integrationsarchitektur
– Erhöht Komplexität und Wartbarkeit der AAS-Architektur erheblich
– Verletzt die Verantwortungstrennung zwischen Zustandsbildung und Projektion

On-Demand-Synchronisation (Pull-Prinzip)

+ Reduziert Hintergrundlast
– Erfüllt nicht die Anforderungen an zeitnahe Abbildung von Zustandsänderungen
– Konsistenzprobleme bei gleichzeitigen Abfragen

Weitere Informationen

Diese Entscheidung baut auf der Daten- und Integrationsarchitektur (ADR-005) sowie dem AAS-Stack (ADR-007) auf und schließt die End-to-End-Architektur von der Daten- und Integrationsarchitektur bis zur AAS-Architektur ab.

Anhang B

Erzeugte AAS-Submodell-Serialisierungen

Die nachfolgenden Listings dokumentieren die Mapping-Konfiguration sowie die AAS-konformen JSON-Serialisierungen der vom Mapping Service erzeugten Submodelle für die Fertigungsmaschine *VCE-1250/5* (Anwendungsfall, Abschnitt 5.5.3).

B.1 Mapping-Konfiguration

```yaml
submodel: DigitalNameplate
version: "3.0.1"
mappings:
  - source: nameplate.manufacturer_name
    target: ManufacturerName
    type:   MultiLanguageProperty
    lang:   de
    value:  "{{ source.manufacturer_name }}"
    semanticId: "0112/2///61987#ABA565#009"

  - source: nameplate.serial_number
    target: SerialNumber
    type:   Property
    valueType: xs:string
    value:  "{{ source.serial_number }}"
    semanticId: "0112/2///61987#ABA951#009"

  - source: nameplate.year_of_construction
    target: YearOfConstruction
    type:   Property
    valueType: xs:string
    value:  "{{ source.year_of_construction | string }}"
    semanticId: "0112/2///61987#ABP000#002"

  - source: nameplate.hardware_revision
    target: HardwareVersion
    type:   Property
    valueType: xs:string
    value:  "{{ source.hardware_revision | default('n/a') }}"
    semanticId: "0112/2///61987#ABA926#008"
```

Listing B.1: Mapping-Konfiguration: Ausschnitt Digital Nameplate mit Jinja2-Wertausdrücken

B.2 Digital Nameplate

```
{
  "modelType": "Submodel",
  "idShort":   "Nameplate",
  "id":        "urn:example:sm:EQ-78421:nameplate",
  "semanticId": {
    "type": "ExternalReference",
    "keys": [{ "type": "GlobalReference",
               "value": "https://admin-shell.io/idta/nameplate/3/0/Nameplate
    " }]
  },
  "submodelElements": [
    {
      "modelType":  "Property",
      "idShort":    "URIOfTheProduct",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABN590#002" }] },
      "valueType":  "xs:anyURI",
      "value":      "https://example.org/product/EQ-78421/EQ-78421"
    },
    {
      "modelType": "MultiLanguageProperty",
      "idShort":   "ManufacturerName",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA565#009" }] },
      "value": [{ "language": "de", "text": "PrecisionTech AG" }]
    },
    {
      "modelType": "MultiLanguageProperty",
      "idShort":   "ManufacturerProductDesignation",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA567#009" }] },
      "value": [{ "language": "de",
                  "text": "5-Achs-Bearbeitungszentrum VCE-1250/5" }]
    },
    {
      "modelType": "SubmodelElementCollection",
      "idShort":   "AddressInformation",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference",
        "value": "https://admin-shell.io/zvei/nameplate/1/0/
    ContactInformations/AddressInformation"
      }] },
      "value": [
        { "modelType": "MultiLanguageProperty", "idShort": "Street",
          "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0173-1#02-AAO128#002" }] },
          "value": [{ "language": "en", "text": "Industriestrae 1" }] },
        { "modelType": "MultiLanguageProperty", "idShort": "Zipcode",
          "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0173-1#02-AAO129#002" }] },
          "value": [{ "language": "en", "text": "1234" }] },
        { "modelType": "MultiLanguageProperty", "idShort": "CityTown",
```

```
45          "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0173-1#02-AAO132#002" }] },
46          "value": [{ "language": "en", "text": "Musterstadt" }] },
47        { "modelType": "MultiLanguageProperty", "idShort": "NationalCode",
48          "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0173-1#02-AAO134#002" }] },
49          "value": [{ "language": "en", "text": "AT" }] }
50      ]
51    },
52    {
53      "modelType":  "Property",
54      "idShort":    "OrderCodeOfManufacturer",
55      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA950#008" }] },
56      "valueType":  "xs:string",
57      "value":      "VCE1250-5EU"
58    },
59    {
60      "modelType":  "Property",
61      "idShort":    "ProductArticleNumberOfManufacturer",
62      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA581#007" }] },
63      "valueType":  "xs:string",
64      "value":      "4012-VCE-1250"
65    },
66    {
67      "modelType":  "Property",
68      "idShort":    "SerialNumber",
69      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA951#009" }] },
70      "valueType":  "xs:string",
71      "value":      "PT-2019-00423"
72    },
73    {
74      "modelType":  "Property",
75      "idShort":    "YearOfConstruction",
76      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABP000#002" }] },
77      "valueType":  "xs:string",
78      "value":      "2019"
79    },
80    {
81      "modelType":  "Property",
82      "idShort":    "HardwareVersion",
83      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA926#008" }] },
84      "valueType":  "xs:string",
85      "value":      "Rev-B"
86    },
87    {
88      "modelType":  "Property",
89      "idShort":    "FirmwareVersion",
90      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA302#006" }] },
91      "valueType":  "xs:string",
```

```
      "value":      "V2.4.1"
    },
    {
      "modelType":  "Property",
      "idShort":    "SoftwareVersion",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABA601#008" }] },
      "valueType":  "xs:string",
      "value":      "CNC-SW 9.2.0"
    },
    {
      "modelType":  "Property",
      "idShort":    "CountryOfOrigin",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0112/2///61987#ABP462#001" }] },
      "valueType":  "xs:string",
      "value":      "CH"
    },
    {
      "modelType":  "Property",
      "idShort":    "InventoryNumber",
      "valueType":  "xs:string",
      "value":      "ANL-78421"
    }
  ]
}
```

Listing B.2: Erzeugtes Digital Nameplate-Submodell (Auszug)

B.3 Technical Data

```
{
  "modelType": "Submodel",
  "idShort":   "TechnicalData",
  "id":        "urn:example:sm:EQ-78421:technicaldata",
  "semanticId": {
    "type": "ExternalReference",
    "keys": [{ "type": "GlobalReference",
               "value": "https://admin-shell.io/IDTA/TechnicalData/Submodel
    /2/0" }]
  },
  "submodelElements": [
    {
      "modelType": "SubmodelElementCollection",
      "idShort":   "GeneralInformation",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "0173-1#02-ABK161#002/0173-1#01-AHX838#002"
    }] },
      "value": [
        {
          "modelType":  "Property",
          "idShort":    "ManufacturerName",
```

```
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
GlobalReference", "value": "0173-1#02-AAO677#004" }] },
      "valueType":  "xs:string",
      "value":      "PrecisionTech AG"
    },
    {
      "modelType": "MultiLanguageProperty",
      "idShort":   "ManufacturerProductDesignation",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
GlobalReference", "value": "0173-1#02-AAW338#003" }] },
      "value": [{ "language": "de",
                  "text": "5-Achs-Bearbeitungszentrum VCE-1250/5" }]
    },
    {
      "modelType":  "Property",
      "idShort":    "ManufacturerArticleNumber",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
GlobalReference", "value": "0173-1#02-AAO676#005" }] },
      "valueType":  "xs:string",
      "value":      "4012-VCE-1250"
    },
    {
      "modelType":  "Property",
      "idShort":    "ManufacturerOrderCode",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
GlobalReference", "value": "0173-1#02-AAO227#004" }] },
      "valueType":  "xs:string",
      "value":      "VCE1250-5EU"
    }
  ]
},
{
  "modelType": "SubmodelElementList",
  "idShort":   "ProductClassifications",
  "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
GlobalReference", "value": "0173-1#02-ABK162#002" }] },
  "typeValueListElement": "SubmodelElementCollection",
  "value": [
    {
      "modelType": "SubmodelElementCollection",
      "semanticId": {
        "type": "ExternalReference",
        "keys": [{ "type": "GlobalReference", "value": "0173-1#02-ABK162
#002/0173-1#01-AHX839#002" }]
      },
      "value": [
        {
          "modelType":  "Property",
          "idShort":    "ClassificationSystem",
          "semanticId": { "type": "ExternalReference", "keys": [{ "type
": "GlobalReference", "value": "0173-1#02-ABL424#001" }] },
          "valueType":  "xs:string",
          "value":      "ECLASS"
        },
        {
```

```
            "modelType":  "Property",
            "idShort":    "VersionOfClassificationSystem",
            "semanticId": { "type": "ExternalReference", "keys": [{ "type": "GlobalReference", "value": "0173-1#02-AAR710#003" }] },
            "valueType":  "xs:string",
            "value":      "14.0"
          },
          {
            "modelType":  "Property",
            "idShort":    "ProductClassId",
            "semanticId": { "type": "ExternalReference", "keys": [{ "type": "GlobalReference", "value": "0173-1#02-ABG776#003" }] },
            "valueType":  "xs:string",
            "value":      "27-01-04-01"
          }
        ]
      }
    ]
  },
  {
    "modelType": "SubmodelElementList",
    "idShort":   "TechnicalPropertyAreas",
    "semanticId": { "type": "ExternalReference", "keys": [{ "type": "GlobalReference", "value": "0173-1#02-ABK163#002" }] },
    "typeValueListElement": "SubmodelElementCollection",
    "value": [
      {
        "modelType": "SubmodelElementCollection",
        "semanticId": {
          "type": "ExternalReference",
          "keys": [{ "type": "GlobalReference", "value": "0173-1#02-ABL358#002/0173-1#01-AHX773#002" }]
        },
        "value": [
          {
            "modelType":  "Property",
            "idShort":    "ControlUnit",
            "valueType":  "xs:string",
            "value":      "SinuTech NC 840D"
          },
          {
            "modelType":  "Property",
            "idShort":    "MagazineType",
            "valueType":  "xs:string",
            "value":      "Kettenmagazin-60"
          },
          {
            "modelType":  "Property",
            "idShort":    "MaxToolDiameter",
            "valueType":  "xs:double",
            "value":      "200.0",
            "qualifiers": [{ "modelType": "Qualifier", "type": "Unit", "valueType": "xs:string", "value": "mm" }]
          },
          {
```

```
                "modelType":  "Property",
                "idShort":    "MaxToolLength",
                "valueType":  "xs:double",
                "value":      "400.0",
                "qualifiers": [{ "modelType": "Qualifier", "type": "Unit", "
    valueType": "xs:string", "value": "mm" }]
              },
              {
                "modelType":  "Property",
                "idShort":    "MaxToolWeight",
                "valueType":  "xs:double",
                "value":      "10.5",
                "qualifiers": [{ "modelType": "Qualifier", "type": "Unit", "
    valueType": "xs:string", "value": "kg" }]
              },
              {
                "modelType":  "Property",
                "idShort":    "ToolCapacity",
                "valueType":  "xs:integer",
                "value":      "60"
              },
              {
                "modelType":  "Property",
                "idShort":    "HardwareVersion",
                "valueType":  "xs:string",
                "value":      "Rev-B"
              },
              {
                "modelType":  "Property",
                "idShort":    "FirmwareVersion",
                "valueType":  "xs:string",
                "value":      "V2.4.1"
              },
              {
                "modelType":  "Property",
                "idShort":    "SoftwareVersion",
                "valueType":  "xs:string",
                "value":      "CNC-SW 9.2.0"
              }
            ]
          }
        ]
      }
    ]
}
```

Listing B.3: Erzeugtes TechnicalData-Submodell (Auszug)

B.4 Process Parameters

```
{
  "modelType": "Submodel",
  "idShort":   "ProcessParameters",
```

```
  "id":        "urn:example:sm:EQ-78421:process-parameters",
  "semanticId": {
    "type": "ExternalReference",
    "keys": [{ "type": "GlobalReference",
               "value": "https://admin-shell.io/idta/SubmodelTemplate/
    ProcessParameters/1/0" }]
  },
  "submodelElements": [
    {
      "modelType": "SubmodelElementCollection",
      "idShort":   "Processes",
      "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/Processes/1/0" }] },
      "value": [
        {
          "modelType": "SubmodelElementCollection",
          "idShort":   "Process__00__",
          "semanticId": { "type": "ExternalReference", "keys": [{ "type": "
    GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/Processes/1/0" }] },
          "value": [
            { "modelType": "Property", "idShort": "ProcessId",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/ProcessId/1/0" }] },
              "valueType": "xs:string", "value": "_N_00781234_MPF" },
            { "modelType": "Property", "idShort": "ProcessName",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/ProcessName/1/0" }] },
              "valueType": "xs:string", "value": "_N_00781234_MPF" },
            { "modelType": "MultiLanguageProperty", "idShort": "
    ProcessDescription",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference",
                "value": "https://admin-shell.io/idta/ProcessParameters/
    ProcessDescription/1/0" }] },
              "value": [{ "language": "de",
                "text": "5-Achs-Frsbearbeitung: Schruppen und Schlichten,
    Programm _N_00781234_MPF" }] },
            { "modelType": "Property", "idShort": "PlannedProcessTime",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference",
                "value": "https://admin-shell.io/idta/ProcessParameters/
    PlannedProcessTime/1/0" }] },
              "valueType": "xs:duration", "value": "PT14M23S" },
            { "modelType": "Property", "idShort": "EventTimestamp",
              "valueType": "xs:dateTime",
              "value": "2025-11-03T07:30:15.412Z" },
            { "modelType": "Property", "idShort": "IngestionTimestamp",
              "valueType": "xs:dateTime",
              "value": "2025-11-03T07:30:15.748Z" },
            { "modelType": "Property", "idShort": "SourceTable",
              "valueType": "xs:string", "value": "silver_telemetry" },
```

```
            {
              "modelType": "SubmodelElementCollection",
              "idShort":   "ProcessParameters",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/ProcessParameters/1/0" }] },
              "value": [
                { "modelType": "Property", "idShort": "FeedRateOverride",
                  "valueType": "xs:double", "value": "8.0" },
                { "modelType": "Property", "idShort": "ActiveBlock",
                  "valueType": "xs:string",
                  "value": "N1200 G01 X-245.5 Y12.8 F800" }
              ]
            },
            {
              "modelType": "SubmodelElementCollection",
              "idShort":   "ResourceParameters",
              "semanticId": { "type": "ExternalReference", "keys": [{ "type
    ": "GlobalReference", "value": "https://admin-shell.io/idta/
    ProcessParameters/ResourceParameters/1/0" }] },
              "value": [
                { "modelType": "Property", "idShort": "MachineStatus",
                  "valueType": "xs:int", "value": "2" },
                { "modelType": "Property", "idShort": "MachineStatus2",
                  "valueType": "xs:boolean", "value": "true" },
                { "modelType": "Property", "idShort": "NcAssembly",
                  "valueType": "xs:int", "value": "75285" }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Listing B.4: Erzeugtes Process Parameters-Submodell (Auszug)