Hochschule Neu-Ulm
University of Applied Sciences

Technische Hochschule Ulm

Bachelorarbeit

im Bachelorstudiengang

Wirtschaftsingenieurwesen

an der Hochschule für angewandte Wissenschaften Neu-Ulm

Entwicklung einer Industrie 4.0 Verwaltungsschale für eine Festo MPS-Station mit Integration von Asset Skills

Erstkorrektorin: Prof. Dr. -Ing. Lisa Ollinger

Zweitkorrektor: Prof. Dr. -Ing. Hartwig Baumgärtel

Verfasserin: Cigdem Verkhov (Matrikel-Nr.: 254604)

Von-Hünefeld-Straße 24, 89231 Neu-Ulm

Abgabedatum: 16.04.2025

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne unerlaubte Hilfe Dritter angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Darüber hinaus erkläre ich hiermit, dass ich folgende KI-Werkzeuge und -Modelle verwendet habe:

ChatGPT von OpenAI.

Der Einsatz dieses Werkzeugs beschränkte sich auf folgende Bereiche: Ideenfindung, Verbesserung und Erläuterung von Quellcode sowie sprachliches Korrekturlesen einzelner Textabschnitte. Die Verantwortung für den gesamten Inhalt der Arbeit liegt ausschließlich bei mir.

Neu-Ulm, 16.04.2025 

Ort, Datum Cigdem Verkhov

Abstract

Diese Bachelorarbeit behandelt die Entwicklung einer Industrie 4.0-fähigen Verwaltungsschale vom Typ 2 für eine Festo MPS-Station mit Sortierfunktion an der Technischen Hochschule Ulm. Ziel ist die Erstellung einer digitalen Repräsentation, die sowohl statische Informationen als auch dynamische Fähigkeiten in Form von ausführbaren Skills integriert. Diese Fähigkeiten werden auf Basis des Skill-basierten Engineerings über ein standardisiertes Kommunikationsprotokoll eingebunden. Untersucht werden verschiedene Ansätze zur Realisierung von Verwaltungsschalen. Ein geeigneter Implementierungsweg wird identifiziert und prototypisch umgesetzt. Die entwickelte Verwaltungsschale beinhaltet standardisierte Teilmodelle sowie Erweiterungen zur Ausführung der Skills, die eine modulare Steuerung der Station ermöglichen und Industrie 4.0-Anforderungen erfüllen. Die Arbeit schafft eine übertragbare Basis, um Verwaltungsschalen auch für andere Stationen mit unterschiedlichen Funktionalitäten weiterzuentwickeln und in zukünftige Automatisierungskonzepte zu integrieren.

This bachelor's thesis focuses on the development of an Industry 4.0-compliant Asset Administration Shell of type 2 for a Festo MPS station with sorting functionality at Ulm University of Applied Sciences. The objective is to create a digital representation that integrates both static information and dynamic capabilities in the form of executable skills. These capabilities are incorporated based on the concept of skill-based engineering via a standardized communication protocol. Various approaches to the realization of Asset Administration Shells are examined. A suitable implementation method is identified and prototypically implemented. The developed Asset Administration Shell includes standardized submodels as well as extensions for the execution of skills, enabling modular control of the station and fulfilling the requirements of Industry 4.0. This work establishes a transferable foundation for further development of Asset Administration Shells for additional stations with varying functionalities and for integration into future automation concepts.

Inhalt

Abkürzungsverzeichnis

SkE	Skill-based Engineering
OPC UA	Open Platform Communications Unified Architecture
SkEs	Skill-based Engineerings
AAS	Asset Administration Shell
RAMI	Referenzarchitekturmodell
IEC	International Electrotechnical Commission
CAD	Computer Aided Design
AASs	Asset Administration Shells
XML	Extensible Markup Language
JSON	JavaScript Object Notation
UI	User Interface
VDI	Verein Deutscher Ingenieure
VDE	Verband der Elektrotechnik, Elektronik und Informationstechnik
GUI	Graphical User Interface
REST	Representational State Transfer
API	Application Programming Interface
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
IDs	Identifiers
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
URI	Uniform Resource Identifier
YAML	Yet Another Markup Language
JPEG	Joint Photographic Experts Group

CSV	Comma Separated Values
SPS	Speicherprogrammierbare Steuerung
MQTT	Message Queuing Telemetry Transport
M2M	Machine-to-Machine
AASX	Asset Administration Shell Package Echange
IRDIS	International Registration Data Identifier
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie e. V.
VDMA	Verband Deutscher Maschinen- und Anlagenbau
IDTA	Industrial Digital Twin Association
CRUD	Create Read Update Delete
mDNS	Multicast Domain Name System
IT-OT Systeme	Informations- und Betriebstechnologie-Systeme
idShorts	Identifikation Shorts
TIA	Totally Integrated Automation
FB	Funktionsbaustein
UI	User Interface

1. Einleitung

1.1 Motivation

Mit dem Fortschritt der digitalen Transformation gewinnt Industrie 4.0 zunehmend an Bedeutung für die industrielle Produktion. Ein zentrales Element ist die Verwaltungsschale, die als digitale Repräsentation physischer Anlagen deren Integration in digitale Systeme ermöglicht. Sie schafft die Grundlage für vernetzte, flexible und transparente Prozesse und leistet einen wichtigen Beitrag zur modernen Automatisierung.(1)

Skill-based Engineering (SkE) ergänzt dieses Konzept, indem es Maschinenfunktionen als modulare, wiederverwendbare Skills beschreibt. Die Kombination beider Ansätze eröffnet neue Möglichkeiten, Funktionen strukturiert zu modellieren, gezielt aufzurufen und in digitale Ökosysteme einzubetten. Trotz verfügbarer Werkzeuge zur Erstellung digitaler Repräsentationen fehlt es bislang an etablierten, praxisorientierten Vorgehensmodellen für die konkrete Umsetzung an realen Anlagen. Diese Arbeit greift dieses Defizit auf und untersucht, wie Verwaltungsschalen in Kombination mit dem SkE zur strukturierten Maschinenansteuerung genutzt werden können.

1.2 Ziele

Ziel dieser Bachelorarbeit ist es, den Aufbau und die Nutzung einer digitalen Repräsentation für ein physisches Asset aufzuzeigen. Anhand eines realen Anwendungsszenarios soll demonstriert werden, wie sich Funktionen einer Anlage strukturiert modellieren und zur Steuerung nutzen lassen. Im Mittelpunkt steht die exemplarische Entwicklung einer Verwaltungsschale des Typs 2, deren Struktur an den Vorgaben der Plattform Industrie 4.0 ausgerichtet ist. Die Arbeit untersucht die methodischen und technischen Anforderungen für die Modellierung, Erweiterung und Integration dieser digitalen Repräsentation in bestehende Systeme. Ein zentrales Ziel ist es, Maschinenfunktionen als aufrufbare digitale Fähigkeiten abzubilden und in eine benutzerfreundliche Struktur zu überführen. Dabei soll ein flexibles, wiederverwendbares Steuerungskonzept entstehen, das sich nahtlos in moderne Industrie-4.0-Umgebungen integrieren lässt.

1.3 Aufgabenstellung

Im Rahmen dieser Arbeit wird eine Verwaltungsschale für die Festo MPS-Station „Sortieren“ entwickelt. Die Station befindet sich im Automatisierungslabor der Technischen Hochschule Ulm und wird über eine Siemens S7-Steuerung betrieben. Diese Steuerung soll mithilfe des Kommunikationsprotokolls Open Platform Communications Unified Architecture (OPC UA) mit der Verwaltungsschale verbunden werden. Die Steuerungsfunktionen werden im Sinne des Skill-based Engineerings (SkEs) als modular aufrufbare Skills umgesetzt und über die Verwaltungsschale abgebildet. Die Skills werden dabei als ausführbare Teilmodell-Elemente innerhalb geeigneter Teilmodelle implementiert, sodass sie direkt über die Verwaltungsschale angesteuert werden können. Ziel ist es, durch eine standardisierte digitale Schnittstelle eine flexible und transparente Steuerung einzelner Maschinenfunktionen zu ermöglichen. Die Umsetzung erfolgt mit dem BaSyx.NET-Framework, der Automatisierungsumgebung TIA-Portal sowie einer benutzerfreundlichen Web-Oberfläche, die auf Komponenten von BaSyx basiert.

Zentrale Arbeitsschritte sind zunächst die Recherche und Einarbeitung in den aktuellen Stand der Technik zu Verwaltungsschalen, digitalen Zwillingen und SkE sowie in die technische Funktionsweise der Festo MPS-Station und der Siemens-Steuerung. Im Anschluss daran erfolgt die Konzeption eines Informationsmodells auf Basis der Verwaltungsschalen-Spezifikation, einschließlich der Identifikation relevanter Datenpunkte und Skills. Darauf aufbauend wird eine geeignete Softwarearchitektur entworfen, die sowohl die Verwaltungsschale als auch die skill-basierte Steuerung berücksichtigt. In der Implementierungsphase werden die definierten Skills in der Steuerung abgebildet, über OPC UA angebunden und in der Verwaltungsschale als Teilmodellelemente ausführbar dargestellt. Abschließend erfolgt die Testphase, in der die Kommunikation zwischen Verwaltungsschale, Steuerung und MPS-Station überprüft wird. Die Ergebnisse werden bewertet und Optimierungspotenziale dokumentiert.

2. Stand der Technik

In diesem Kapitel werden die grundlegenden technischen und konzeptionellen Aspekte der Verwaltungsschale im Kontext von Industrie 4.0 dargestellt. Der Fokus liegt auf der Verwaltungsschale des Typs 2.

2.1 Definition Verwaltungsschale

Die Verwaltungsschale, auf Englisch auch Asset Administrationshell (AAS) genannt(2) bezeichnet die Umsetzung eines digitalen Zwillings für die Industrie 4.0. Sie ermöglicht einen standardisierten und herstellerübergreifenden Datenaustausch entlang des gesamten Lebenszyklus von Produkten bzw. Assets jeglicher Art. Diese Daten werden in Form von Teilmodellen mit ihren jeweiligen Eigenschaften in der AAS abgelegt, wodurch sie für unterschiedliche Zwecke genutzt werden können.

Unter anderem können sie standardisiert gefunden und verarbeitet als auch modelliert und weitergeleitet werden.(3) Die nachfolgende Abbildung 1 verdeutlicht das Zusammenspiel zwischen dem physischen Asset, der AAS und dem digitalen Zwilling. Dabei wird gezeigt, wie die AAS als Vermittler zwischen realer und digitaler Welt fungiert und den interoperablen Datenaustausch ermöglicht.

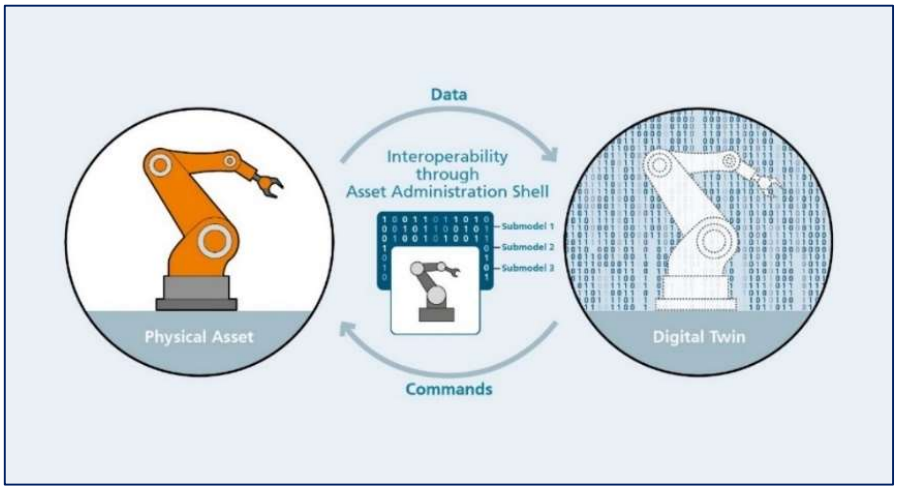


Abbildung 1: Interoperabilität zwischen physischem Asset und digitalem Zwilling über die AAS

2.2 Rolle der AAS im Referenzarchitekturmodell Industrie 4.0

Das Referenzarchitekturmodell Industrie 4.0, auch bekannt als RAMI 4.0, ist ein dreidimensionales Koordinatensystem, das die zentralen Aspekte der Industrie 4.0 strukturiert abbildet. Es dient dazu, komplexe Zusammenhänge in kleinere und überschaubare Einheiten zu unterteilen, um eine systematische Herangehensweise an Industrie-4.0-Projekte zu ermöglichen. Auf der linken horizontalen Achse wird der Lebenszyklus von Produkten und Anlagen dargestellt. Diese Darstellung basiert auf der internationalen Norm „International

Electrotechnical Commission (IEC) 62890, welche die Integration von Unternehmens-IT mit dem Produktlebenszyklus beschreibt. Die rechte horizontale Achse bildet die Hierarchieebenen eines Unternehmens ab, wie sie in der Norm IEC 62264 beschrieben sind. Diese Achse umfasst die verschiedenen Ebenen der Automatisierung, beginnend bei der Feldebene über die Steuerungs- und Leitebene bis hin zur Betriebs- und Unternehmensebene. Die Struktur und Dimensionen des Referenzarchitekturmodells Industrie 4.0 sind in Abbildung 2 veranschaulicht.

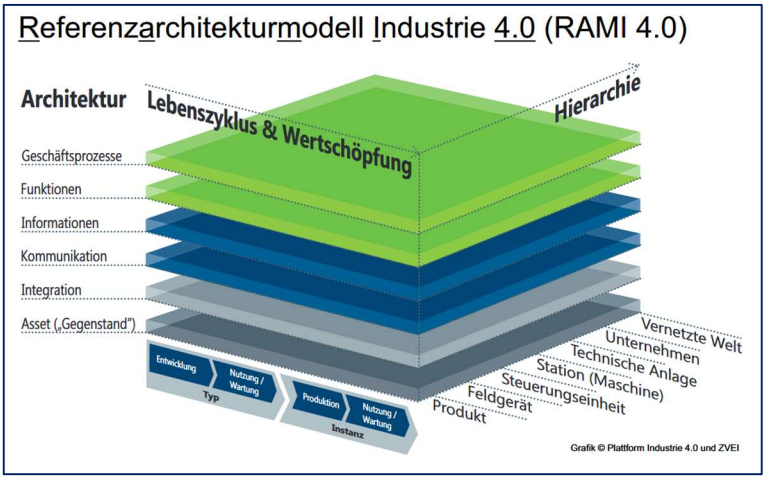


Abbildung 2: RAMI 4.0

Position der AAS im Kontext von Industrie 4.0

Die AAS stellt das digitale Abbild eines Assets dar. Ein Asset kann ein physischer, digitaler oder logischer Gegenstand sein, der im Unternehmen beispielsweise für interne Prozesse oder die Zusammenarbeit mit Zulieferern und Kunden relevant ist. Dazu zählen unter anderem Anlagen, Roboter, Prozesse oder Produkte.(2)

Gemeinsam mit dem zugehörigen Asset bildet die AAS eine sogenannte Industrie 4.0 Komponente und übernimmt darin eine zentrale Rolle, da sie im Architekturkonzept von Industrie 4.0, insbesondere im Referenzarchitekturmodell RAMI 4.0, als standardisierte digitale Repräsentation eines Assets definiert ist. Im Kontext von Industrie 4.0 übernimmt die AAS die Rolle einer standardisierten Schnittstelle für die Kommunikation zwischen verschiedenen Komponenten. Sie ermöglicht den einheitlichen Zugriff auf die Eigenschaften und Funktionen eines Assets und bildet damit die Grundlage für Interoperabilität und Automatisierung. Die in der AAS enthaltenen Teilmodelle strukturieren die Informationen in sogenannten Merkmalen, die entweder auf Typenebene oder für konkrete Instanzen definiert sind.(4) Die folgende Abbildung 3 zeigt den Aufbau einer Industrie 4.0 Komponente bestehend aus einem Asset und der zugehörigen AAS.

Abbildung 3: I4.0-Komponente AAS und Gegenstand

Mehrwert der AAS für die Industrie 4.0

Eine AAS bietet zahlreiche Vorteile im Kontext der Industrie 4.0. Ein zentrales Problem besteht aktuell darin, dass kein einheitlicher Standard für die Bereitstellung von Dateiformaten für Maschinenparameter existiert. Die AAS schafft hier Abhilfe, indem sie eine standardisierte Struktur bereitstellt, in der diese Informationen einheitlich erfasst und gespeichert werden können. Dadurch wird die Kommunikation zwischen unterschiedlichen Datenformaten verschiedener Hersteller erheblich erleichtert. Gleichzeitig vereinfacht die AAS die Datenintegration und ermöglicht eine effizientere Datenanalyse. Gerade für Anlagenbetreiber stellt dies einen entscheidenden Vorteil dar, da sich neue Maschinen und Komponenten dank standardisierter Schnittstellen deutlich einfacher in bestehende Produktionslinien integrieren lassen. Die laufenden Prozesse können dabei weiterhin zuverlässig überwacht und bei Bedarf gezielt optimiert werden und das ohne aufwendige Anpassungen an herstellerspezifische Systeme.(5)

Darüber hinaus bietet die AAS eine standardisierte Methode zur Verwaltung von Anlagen, die Wartungsprozesse effizienter gestaltet und ungeplante Ausfallzeiten reduziert. Dies führt zu einer erhöhten Zuverlässigkeit industrieller Anlagen und unterstützt Unternehmen dabei, ihre Produktionsziele wirtschaftlich und zuverlässig zu erreichen. Auch im Hinblick auf die Datenanalyse und -optimierung eröffnet die AAS neue Potenziale. Sie ermöglicht eine strukturierte Speicherung sowie den gezielten Zugriff auf Asset-Daten. Dadurch wird die gemeinsame Datennutzung innerhalb eines Unternehmens und über Systemgrenzen hinweg erheblich vereinfacht. Unternehmen können so Trends und Muster in der Anlagenleistung erkennen, den Betrieb gezielt optimieren und die Effizienz nachhaltig steigern. Durch die Bereitstellung einer standardisierten Methode zur Beschreibung und Verwaltung von Assets unterstützt die AAS darüber hinaus die Erfüllung gesetzlicher Anforderungen und den Nachweis der Einhaltung branchenüblicher Standards.

Nicht zuletzt verschafft der Einsatz der AAS einen klaren Wettbewerbsvorteil, da Unternehmen die Effizienz und Zuverlässigkeit ihrer Asset-Management-Prozesse deutlich verbessern können, insbesondere im Vergleich zu jenen, die weiterhin auf manuelle oder uneinheitliche Ansätze setzen.(6)

2.3 Struktur der AAS

Die in Abbildung 4 dargestellte Struktur der AAS zeigt den grundlegenden Aufbau einer AAS, wie er im AAS-Metamodell beschrieben ist. Eine AAS besteht aus zwei zentralen Komponenten, dem Header und dem Body. Im Header werden alle relevanten Identifikationsdaten gespeichert. Dazu zählen sowohl die Identifikation der AAS selbst als auch die des zugehörigen Assets. Diese Informationen sind notwendig, um die AAS eindeutig zuordnen, referenzieren und systemübergreifend austauschen zu können.

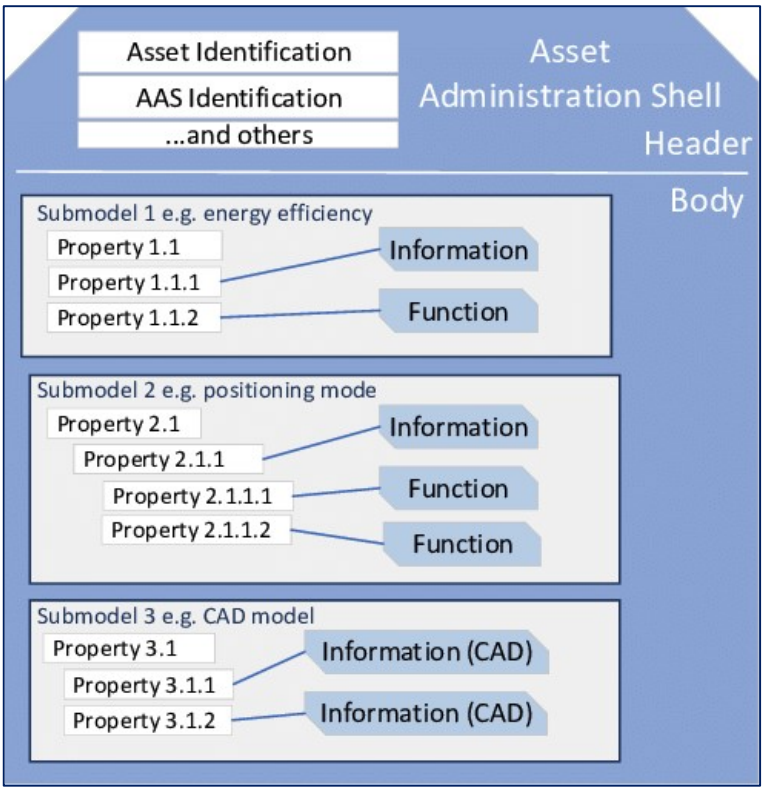


Abbildung 4: Grobstruktur der AAS mit Header und Body gemäß AAS-Metamodell

Der Body umfasst eine Sammlung sogenannter Teilmodelle. Diese Teilmodelle repräsentieren unterschiedliche Aspekte des Assets, wie beispielsweise Energieeffizienz, Positionierungsmodi oder Computer Aided Design (CAD)-Modelle. Jedes Teilmodell enthält eine hierarchisch strukturierte Menge an Eigenschaften, auch Properties genannt, die sowohl statische Informationen als auch Funktionen abbilden können. Dadurch lassen sich nicht nur beschreibende Merkmale hinterlegen, sondern auch konkrete Fähigkeiten und Verhaltensweisen des Assets digital darstellen.

Wie in der Abbildung 4 exemplarisch gezeigt, enthält beispielsweise das Teilmodell zur Energieeffizienz verschiedene Properties, denen Informationen und Funktionen zugeordnet sind. Das Teilmodell „Positioning Mode“ bildet weitere Funktionen ab, während im Teilmodell 3 ein konkretes CAD-Modell als digitale Datei integriert ist. Diese Struktur ermöglicht es, sowohl einfache Informationen als auch komplexe Dateien systematisch zu erfassen und standardisiert bereitzustellen. Durch diese modulare und klar strukturierte Aufteilung können Asset Administration Shells (AASs) flexibel erweitert, wiederverwendet und interoperabel zwischen verschiedenen Systemen ausgetauscht werden. Das Metamodell legt dabei die formale Grundlage zur Beschreibung aller Elemente der AAS wie Teilmodelle, Properties, Funktionen und semantische Informationen fest.(7)

Teilmodelle der AAS

Grundsätzlich verfolgt das Konzept der Industrie 4.0 Komponente das Ziel, dass jedes Asset über eine eigene AAS verfügt, welche das Asset über seinen gesamten Lebenszyklus hinweg minimal, aber ausreichend beschreibt. Diese Beschreibung erfolgt in Form von Teilmodellen, die unterschiedliche Aspekte wie Identifikation, Sicherheit, Energiemanagement oder spezifische Prozessfähigkeiten abbilden. Wie bereits zuvor erwähnt, bestehen Teilmodelle aus Properties, die im Kontext von Industrie 4.0 ein übergreifendes Konstrukt darstellen, mit dem sich statische Merkmale, Parameter, Methoden, Fähigkeiten, Operationen, Zustände und weitere Eigenschaften eines Assets modellieren lassen. Der Zugriff auf diese Merkmale ist jedoch nicht uneingeschränkt möglich, da nicht jeder Partner innerhalb eines Wertschöpfungsnetzwerks oder einer Organisationseinheit automatisch Zugriff auf alle Informationen erhält. Teilmodelle sollten so standardisiert werden, dass für jeden Aspekt nur ein einheitliches Teilmodell existiert. Durch diese Standardisierung wird eine Vergleichbarkeit und Austauschbarkeit von Industrie-4.0-Komponenten innerhalb eines Prozesses ermöglicht. Die jeweiligen Teilmodelle enthalten spezifische Merkmale, die den jeweiligen Aspekt eindeutig beschreiben und so eine klare Identifikation sowie eine effiziente Integration in digitale Prozesse unterstützen. Für die Kommunikation zwischen verschiedenen Industrie 4.0-Komponenten können bestimmte Merkmale als standardisierte Grundlage angenommen werden. In einem solchen Fall könnte ein Teilmodell zum Thema Energieeffizienz beispielsweise die Fähigkeit eines Assets beschreiben, in Betriebsphasen mit geringem Energiebedarf den Stromverbrauch automatisch zu reduzieren.(4)

2.4 Typen der AAS

Derzeit werden zwischen drei Typen der AAS unterschieden:

- AAS-Typ-1: Diese AAS besteht aus serialisierten Dateien, beispielsweise im „Extensible Markup Language“ (XML)- oder „JavaScript Object Notation“ (JSON)-Format, welche

statische Informationen enthalten. Sie können als Datei weitergegeben werden und dienen als transportierbare Repräsentation eines Assets.(8) Dabei handelt es sich um Interaktionen, bei denen Informationen strukturiert und ausgetauscht werden, ohne dass eine direkte Verbindung zwischen der AAS und dem Asset besteht.(9)

- AAS-Typ-2: Eine AAS des Typen zwei existiert hingegen als Laufzeitinstanz. Dieser Typ wird auf einem Server gehostet und enthält sowohl statische Informationen als auch dynamische Informationen, die im laufenden Betrieb aus anderen Komponenten stammen können. Dadurch bietet sie eine Userinterface (UI) für verschiedene Anwendungszwecke wie zum Beispiel für Geräteservices, Live-Daten von Sensoren oder Produkten sowie für Informationen zur Echtzeitverfügbarkeit und zu den Lieferzeiten von Ersatzteilen. Des Weiteren kann sie nicht nur Eigenschaften und Operationen bereitstellen, sondern auch auf sich ändernde Bedingungen reagieren, indem sie entsprechende Ereignisse signalisiert. Auch hier wird das Datenmodell durch das AAS-Metamodell definiert. Darüber hinaus spezifiziert die AAS eine generische Laufzeitschnittstelle, die den Zugriff auf Eigenschaften, Operationen und Ereignisse ermöglicht. Die Typ-2-AAS kann somit einheitliche Schnittstellen zu heterogenen Systemen realisieren.
- AAS-Typ-3: Die AAS-Typ-3 erweitert die Funktionalität der AAS-Typ-2, indem sie zusätzlich ein aktives Verhalten implementiert, was bedeutet, dass sie eigenständig kommunizieren und Verhandlungen führen kann. Die Spezifikation des Vereins Deutscher Ingenieure (VDI) und des Verbands der Elektrotechnik Elektronik Informationstechnik (VDE) mit der Nummer 2139 definiert eine entsprechende Sprache zur Umsetzung von AASs des Typs 3.(8)

Im folgenden Verlauf dieser Arbeit liegt der Schwerpunkt auf der AAS des Typen 2.

2.5 Ausgewählte Infrastrukturkomponenten einer AAS-Typ-2

Im Folgenden werden ausgewählte Infrastrukturkomponenten einer AAS-Typ-2 erläutert, um einen grundlegenden Überblick über deren Funktion innerhalb eines AAS-Ökosystems zu geben. Diese Infrastrukturkomponenten sind in der Abbildung 5 dargestellt.

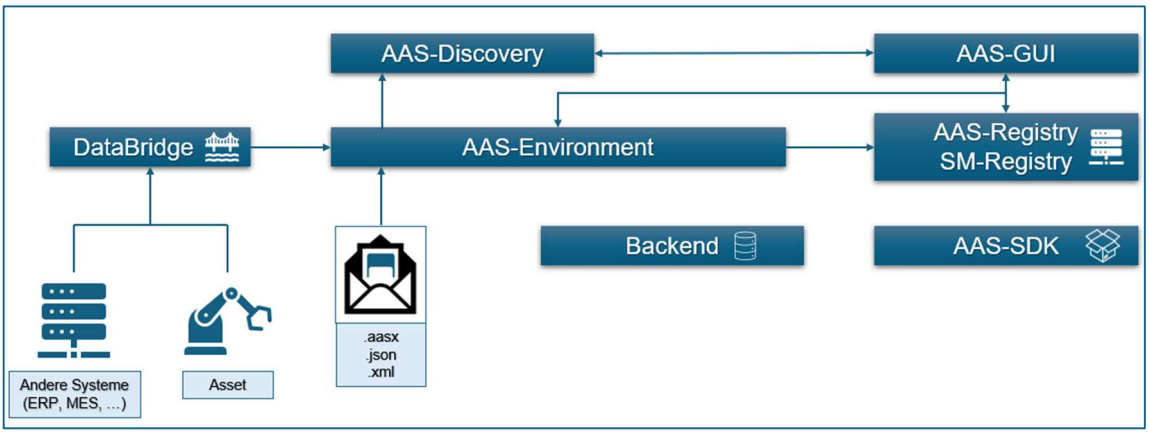


Abbildung 5: Infrastrukturkomponenten einer AAS-Typ-2(10)

Das AAS-Environment steht im Zentrum der Architektur und es empfängt Daten von der DataBridge, kommuniziert mit dem Backend zur Datenablage, überträgt Informationen an die Registry und unterstützt die AAS-GUI (Graphical Userinterface) sowie den Discovery-Service mit den erforderlichen Daten. Diese Komponente fasst das AAS-Repository, das Teilmodell-Repository sowie das Concept Description Repository in einer zentralen Laufzeitkomponente zusammen.(10) Die Repository ist eine Komponente, die eine Representational State Transfer (REST)- Application Programming Interface (API) zur Interaktion mit AASs bereitstellt. Bei einer REST-Architektur wird ein Konzept für die Gestaltung verteilter Systeme und definiert Prinzipien beschreiben, die eine einfache und skalierbare Kommunikation über Netzwerke ermöglichen.(11) Die API ist dabei eine definierte Schnittstelle, die aus Befehlen, Protokollen, Objekten und Funktionen besteht und von Entwicklerinnen und Entwicklern genutzt wird, um Software zu erstellen oder mit anderen Systemen zu interagieren. Sie regelt die Zugriffspunkte zum Server und ermöglicht die Kommunikation zwischen verschiedenen Komponenten.(12) Die Hauptfunktion eines Repository besteht darin, Instanzen zu erstellen, abzurufen, zu aktualisieren und zu löschen. Es dient somit als zentrale Schnittstelle für das Management digitaler Repräsentationen von Assets.(13) Eine Concept Description ist ein Wörterbuch, welches die semantischen Bedeutungen von Teilmodellen oder Teilmodel-Elementen enthält. Es dient dazu, Begriffe eindeutig zu definieren und maschinenlesbar zu beschreiben, damit verschiedene Systeme dieselbe Bedeutung interpretieren können.(14) Die Komponente DataBridge überträgt strukturierte Daten aus dem Asset oder aus Systemen wie Enterprise Resource Planning (ERP) oder Manufacturing Execution System (MES) direkt in das AAS-Environment. Durch die Bereitstellung als leicht einsetzbare Standardkomponente auf DockerHub lässt sich die DataBridge unkompliziert in eigene Anwendungsfälle integrieren.(15). Der AAS Discovery Service ist über das AAS-Environment angebunden und kann mit der AAS-GUI kombiniert werden. Es ermöglicht eine gezielte Suche nach Asset-Identifiers (IDs) zugehöriger AASs sowie die Ermittlung von AAS-IDs auf Basis bekannter Asset-IDs. Dies ist besonders hilfreich beim Arbeiten mit

hierarchischen Strukturen. Auf diese Weise können AASs, die in einem solchen Teilmodell referenzierte Assets enthalten, schnell gefunden und gezielt angesteuert werden.(13) Das Backend ist mit dem AAS-Environment verbunden und wird genutzt, um Daten dauerhaft oder temporär abzulegen und wieder bereitzustellen.(14) Mögliche Speicherlösungen sind beispielsweise „InMemory“, bei der die Daten temporär im Arbeitsspeicher abgelegt werden, oder eine persistente Speicherung in einer Datenbank wie MongoDB.(16) Die AAS-GUI-Komponente unterstützt die benutzerfreundliche Darstellung und Verwaltung von AASs und ermöglicht eine nahtlose Integration in digitale Zwillinge.(17) Sie kommuniziert sowohl mit dem AAS-Environment als auch mit der Registry. Das AAS-Environment überträgt die relevanten Registrierungsinformationen an die Registry, welche wiederum auch von der AAS-GUI zur Darstellung genutzt wird. Die Registry dient der Registrierung aller vorhandenen AASs und ermöglicht das gezielte Auffinden einer AAS sowie ihrer Teilmodelle anhand eindeutiger Identifikationen. Darüber hinaus können in der Registry auch die zugehörigen Endpunkte der AASs und ihrer Teilmodelle hinterlegt werden. Auf diese Weise lassen sich Informationen zu einer AAS und ihren Teilmodellen über die Registry jederzeit abrufen und aktualisieren.(18) Das Software Development Kit (SDK) kann mit allen Hauptkomponenten interagieren, insbesondere mit dem AAS-Environment und der Registry und dient als technische Grundlage für individuelle Erweiterungen und Anwendungen. Zur Implementierung von AASs stehen verschiedene SDKs zur Verfügung.(19)

2.6 Schnittstellen der AAS-Typ-2

Die AAS vom Typ 2 nutzt standardisierte API-Schnittstellen, um einen strukturierten Datenaustausch zwischen dem digitalen Zwilling und externen Systemen zu ermöglichen. Über diese Schnittstellen lassen sich sowohl Informationen auslesen und verändern als auch Funktionen des zugrunde liegenden Assets gezielt ansteuern. Ziel dieser Schnittstellenarchitektur ist es, eine hohe Interoperabilität zwischen heterogenen Systemen zu gewährleisten und die nahtlose Integration in bestehende Industrie-4.0-Infrastrukturen zu ermöglichen. Die API-basierte Kommunikation stellt somit einen zentralen Ansatz dar, um AASs dynamisch und flexibel in moderne IT-Systeme einzubinden.[1] Je nach Anwendung kommen unterschiedliche Kommunikationsprotokolle zum Einsatz, die im Folgenden näher erläutert werden.

HTTP-REST

Das Hypertext Transfer Protocol (HTTP) dient der Übertragung von Hypertext-Dokumenten und bildet die Grundlage für die Kommunikation zwischen Webservern und Webbrowsern. Heute wird es vor allem für den Austausch von Webseiteninhalten verwendet.(11) REST-

[1] IDTA-01002-3-0_SpecificationAssetAdministrationShell_Part2_API.pdf

basierte Schnittstellen, nutzen HTTP als Transportprotokoll und finden breite Anwendung in webbasierten Systemen sowie in industriellen Kommunikationslösungen, beispielsweise im Industrial Internet of Things (IoT). Das grundlegende Ziel von REST besteht darin, dem Client eine repräsentative Darstellung des aktuellen Zustands einer Ressource zu übermitteln. Ein zentrales Prinzip dabei ist die Zustandslosigkeit, bei der der Server keine Informationen über vorherige Anfragen oder Verbindungen zum Client speichert. Stattdessen liefert er auf jede Anfrage stets nur den aktuellen Zustand der jeweiligen Ressource. Diese Ressourcen sind über eindeutige Zugriffspunkte, sogenannte Uniform Resource Identifiers (URIs), adressierbar.

REST erlaubt es, Ressourcen in verschiedenen Datenformaten darzustellen, etwa in JSON, XML, Ain't Markup Language (YAML) oder auch als Binär- oder Bilddateien wie Joint Photographic Experts Group (JPEG). Die eigentliche Ressource bleibt dabei auf dem Server. Der Client erhält lediglich deren Repräsentation. Dies ermöglicht eine hohe Flexibilität, da die Daten auf Serverseite z. B. in einer Datenbank, einer Comma-Separated Values (CSV)-Datei oder einer objektorientierten Struktur gespeichert sein können. Anwendungen oder Schnittstellen, die diesen Prinzipien folgen, werden als RESTful bezeichnet. Für eine RESTful Kommunikation müssen bestimmte Rahmenbedingungen eingehalten werden. Eine Voraussetzung ist die Kommunikation zwischen dem Server und dem Client, bei der in der Client-Anfrage eine URI und eine HTTP-Methode enthalten sind. Diese Methoden sind Aktionen, die vom Client beim Server angefordert werden. Standardmäßig gibt es sechs Methoden (GET, PUT, POST, DELETE, HEAD und OPTIONS), die je nach Situation verwendet werden können. Die Methode „GET“ wird genutzt, um den Zustand einer Ressource zu empfangen. Die Methode „PUT“ dient der Aktualisierung einer Ressource und „DELETE“ dem Löschen einer Ressource. Eine weitere Rahmenbedingung ist wie bereits erwähnt, die Zustandslosigkeit. Der Server speichert nicht, mit welchem Client er interagiert hat. Dadurch wird das Speichern des jeweiligen Zustands auf den Client verlagert, was eine kompakte Implementierung auf Serverseite ermöglicht. Wenn der Server cachefähige Ressourcen anbietet, enthalten diese Versionsnummern, sodass der Client Informationen zur Gültigkeit und zum Verlauf der Ressource erhalten kann. Eine weitere Voraussetzung ist, dass die Ebenen, die zwischen dem Client und dem Server liegen, in der Lage sein sollten, miteinander zu kommunizieren, auch wenn beispielsweise Proxys vorhanden sind. Auch wenn mehrere Ebenen wie etwa Sicherheits-, Caching- oder Lastverteilungsebenen existieren, sollte die Nachrichtenübertragung zwischen Client und Server dadurch nicht beeinträchtigt werden. Als letzte Rahmenbedingung sollte eine einheitliche Schnittstelle vorhanden sein, die ebenfalls bestimmte Anforderungen mit sich bringt. Dazu gehört, dass jede Ressource eine eindeutige Kennung in Form eines URIs besitzt. Sobald der Client eine Darstellung

einer Ressource inklusive ihrer Metadaten erhalten hat, muss es möglich sein, eine Ressource hinzuzufügen, zu löschen oder zu ändern.

Die ausgetauschten Nachrichten sollten selbstbeschreibend sein und Informationen zur Verarbeitung der Ressourcen enthalten. Zudem sind die Ressourcen miteinander verknüpft, sodass der Client Informationen zu den URIs aller Ressourcen erhalten kann und gezielt darauf zugreifen kann.(20)

OPC UA

Das Kommunikationsprotokoll OPC UA ist eine plattformunabhängige, serviceorientierte Kommunikationsarchitektur, die im Jahr 2008 veröffentlicht wurde. Ziel ihrer Entwicklung war es, die Funktionalität der bisherigen OPC-Classic-Spezifikationen in einem einheitlichen und erweiterbaren Framework zu vereinen. OPC UA stellt einen Industriestandard dar, der sowohl den zuverlässigen Austausch von Daten als auch die Modellierung komplexer Informationen unterstützt und sich besonders für Anwendungen im Umfeld von Industrie 4.0 eignet. Dazu zählen unter anderem die Discovery-Funktion, mit der OPC-Server auf lokalen PCs oder Netzwerken erkannt werden können, ein hierarchisch aufgebauter Adressraum sowie die Möglichkeit, Daten auf Abruf zu lesen oder zu schreiben. Weitere Funktionen umfassen Abonnements, mit denen Datenänderungen überwacht werden können, sowie die Verarbeitung von Events und das Ausführen von Methoden, die auf dem Server definiert sind. Da OPC UA vollständig plattformunabhängig ist, kann es sowohl auf klassischen PCs als auch auf eingebetteten Systemen wie Mikrocontrollern oder speicherprogrammierbaren Steuerungen (SPS) eingesetzt werden.

Unterstützt werden eine Vielzahl von Betriebssystemen wie Windows, Linux, Android oder macOS. Diese Eigenschaft macht OPC UA besonders flexibel in Bezug auf die Integration in heterogene Systemlandschaften. Ein weiterer zentraler Aspekt ist die umfassende Sicherheitsarchitektur. OPC UA bietet Mechanismen wie Verschlüsselung, Nachrichtensignierung, Benutzerauthentifizierung und Rechtemanagement. Auch sicherheitsrelevante Aspekte wie Transportprotokolle, sitzungsbasierte Verschlüsselung, die korrekte Reihenfolge von Datenpaketen sowie die Zugriffsüberwachung, auch als Auditing bezeichnet, sind Bestandteil der Spezifikation. Dadurch eignet sich OPC UA auch für sicherheitskritische industrielle Anwendungen.

Die Erweiterbarkeit von OPC UA erlaubt es, neue Technologien, Sicherheitsmechanismen oder Informationsmodelle zu integrieren, ohne die Kompatibilität zu bestehenden Systemen zu verlieren. Dies macht OPC UA zu einer zukunftssicheren Lösung. Ein zentrales Element ist außerdem die Informationsmodellierung. OPC UA ermöglicht die objektorientierte Abbildung von Datenstrukturen, Zuständen, Methoden und Ereignissen. Über standardisierte

Zugriffsmechanismen wie Browsing, Lese- und Schreiboperationen, Methodenaufrufe und Ereignisbenachrichtigungen können Clients auf diese Informationen zugreifen. Dabei unterstützt OPC UA sowohl das klassische Client-Server-Modell als auch das „Publish-Subscribe-Modell“, wodurch eine Vielzahl von Kommunikationsszenarien abgedeckt wird.(21)

MQTT

Neben HTTP und OPC UA wird auch MQTT (Message Queuing Telemetry Transport) als Kommunikationsprotokoll in Typ-2-AAS eingesetzt. Es handelt sich dabei um ein leichtgewichtiges Nachrichtenprotokoll, das speziell für den Einsatz in Netzwerken mit eingeschränkter Bandbreite, hoher Latenz und begrenzten Ressourcen entwickelt wurde, wie sie häufig bei IoT-Geräten zu finden sind. Aufgrund dieser Eigenschaften eignet sich MQTT besonders gut für die Machine-to-Machine(M2M) -Kommunikation in verteilten Systemen. Die Kommunikation innerhalb von MQTT folgt dem „Publisher-Subscriber-Prinzip“ und wird durch einen zentralen Vermittlungsdienst, den sogenannten Broker, organisiert. Der MQTT-Broker nimmt eine zentrale Rolle ein, da er die Verwaltung der Nachrichten übernimmt, die von den Publishern an definierte Topics gesendet werden. Ein Topic ist dabei ein hierarchisch strukturierter Kanal, der zur thematischen Einordnung der Nachrichten dient. Clients, die an bestimmten Informationen interessiert sind, abonnieren ein entsprechendes Topic und erhalten automatisch alle Nachrichten, die diesem Thema zugeordnet sind. Die Besonderheit dieses Kommunikationsmodells besteht darin, dass Sender und Empfänger nicht direkt miteinander kommunizieren müssen. Der Broker übernimmt die Aufgabe, eingehende Nachrichten zu filtern, die entsprechenden Abonnenten zu identifizieren und die Daten gezielt weiterzuleiten. Ein weiterer Vorteil des MQTT-Protokolls liegt in seiner Fähigkeit, auch bei instabilen Netzwerkverbindungen zuverlässig zu arbeiten. Sollte ein Client vorübergehend nicht erreichbar sein, kann der Broker die Nachrichten zwischenspeichern und bei erneuter Verbindung zustellen. Darüber hinaus ermöglicht MQTT die Übertragung mit verschiedenen Qualitätsstufen, den sogenannten „Quality-of-Service“-Stufen, die das Verhalten bei der Zustellung definieren. Die Nachrichten selbst bestehen aus einer Nutzlast, der sogenannten Payload, deren Struktur frei wählbar ist. In der Praxis werden jedoch meist standardisierte Formate wie JSON oder XML verwendet, um die Kompatibilität mit anderen Systemen zu gewährleisten. Der MQTT-Broker übernimmt neben der Nachrichtenverteilung auch sicherheitsrelevante Aufgaben wie die Authentifizierung und Autorisierung der Clients. In industriellen Anwendungen kommen häufig spezialisierte MQTT-Broker wie HiveMQ oder Mosquitto zum Einsatz. Auch Cloud-Plattformen wie Microsoft Azure und Amazon Web Services stellen mit Azure IoT Hub beziehungsweise AWS IoT Core eigene MQTT-Broker für die Integration von IoT-Systemen in cloudbasierte Architekturen bereit.

Insgesamt stellt MQTT eine effiziente, skalierbare und ressourcenschonende Lösung dar, die sich ideal für die Anbindung von AAS an moderne, vernetzte Produktions- und Cloud-Umgebungen eignet.(22) Zusammenfassen können die Einsatzgebiete der zuvor genannten Kommunikationsprotokolle wie in Abbildung 6 visualisiert werden.

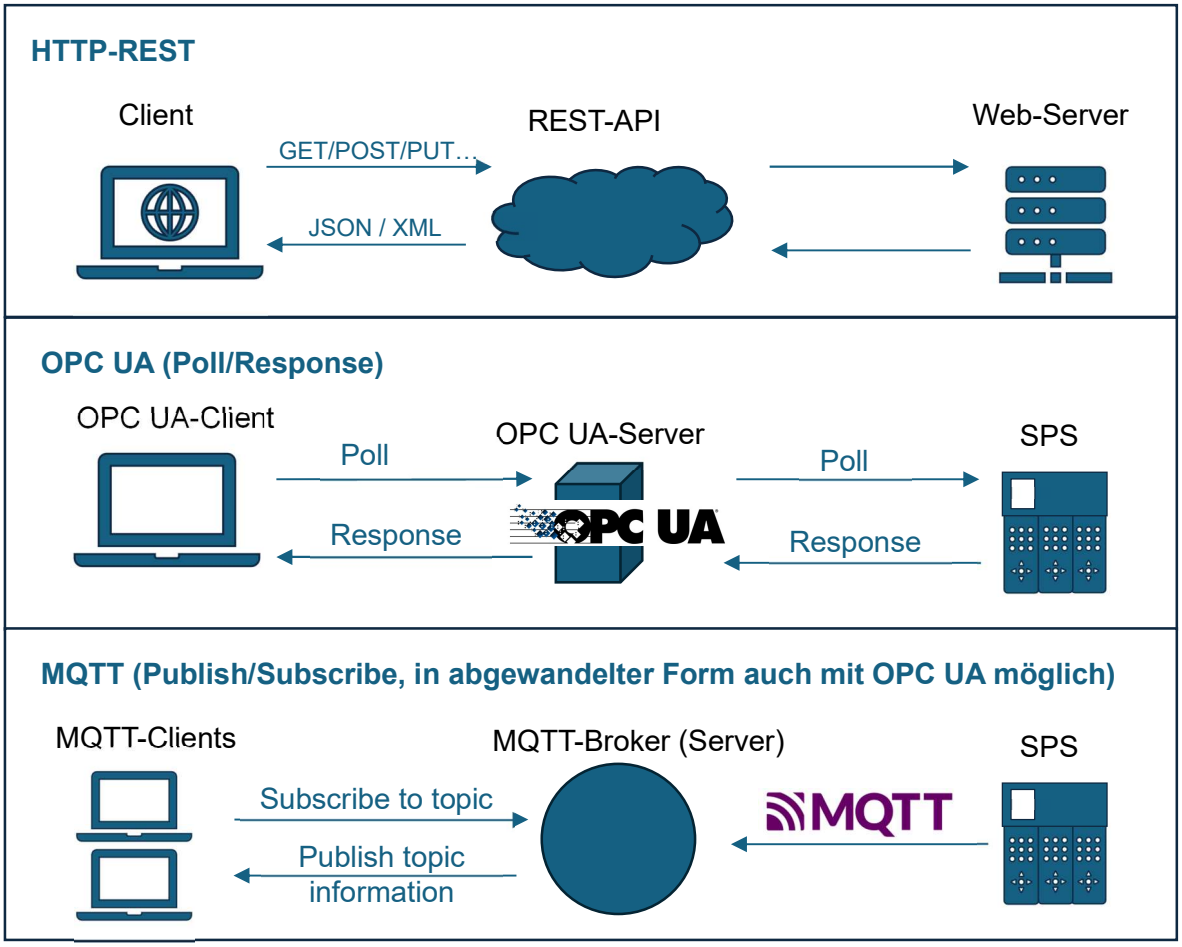


Abbildung 6:Übersicht über HTTP-REST, OPC UA und MQTT(23,24)

2.7 Technische Realisierungsansätze für AASs

Dieser Abschnitt gibt einen Überblick über etablierte Werkzeuge und Ansätze zur technischen Realisierung von AASs in Industrie-4.0-Umgebungen. Eine Möglichkeit zur Realisierung einer AAS ist der AASX Package Explorer. Dieser ist ein frei verfügbares Open-Source-Tool zur Erstellung, Bearbeitung und Visualisierung von AASs im Asset Administration Shell Exchange (AASX)-, XML- und JSON-Format. Entwickelt wurde das Tool im Rahmen der Plattform Industrie 4.0 und steht unter der Eclipse Public License 2.0 zum kostenlosen Download bereit. Es richtet sich in erster Linie an Unternehmen und Entwickler, die sich niederschwellig mit dem Konzept der AAS vertraut machen möchten.

Der AASX Package Explorer ermöglicht die Definition von AASs, Teilmodellen, Eigenschaften und Operationen. Darüber hinaus können eCl@ss International Registration Data

Identifier (IRDIs) für standardisierte Konzeptbeschreibungen automatisch integriert werden. Zusätzlich bietet das Tool Import- und Exportfunktionen zu anderen Formaten wie AutomationML, OPC UA oder BMEcat. Da es sich jedoch um ein statisches Tool handelt, ist der AASX Package Explorer primär für die Entwicklung von AASs des Typs 1 geeignet. Eine aktive Laufzeitumgebung zur Anbindung an industrielle Systeme, die Ausführung von OPC UA Skills oder die Kommunikation über REST-APIs, wie sie bei einer AAS-Typ 2 erforderlich ist, wird durch das Tool nicht abgedeckt. Nichtsdestotrotz eignet sich der Package Explorer zur Modellierung und strukturellen Vorbereitung von AAS-Inhalten, die anschließend in ein Typ-2-kompatibles Backend überführt werden können.(25) Die Abbildung 7 zeigt eine beispielhafte AAS für das Asset „ExampleMotor“ im AASX Package Explorer. Die AAS enthält mehrere strukturierte Teilmodelle, in denen Informationen wie Hersteller, maximale Drehzahl, Drehmoment und zugehörige Dokumente anhand von IDs, Kurzbezeichnungen (idShorts) und standardisierten Metadaten übersichtlich modelliert sind.

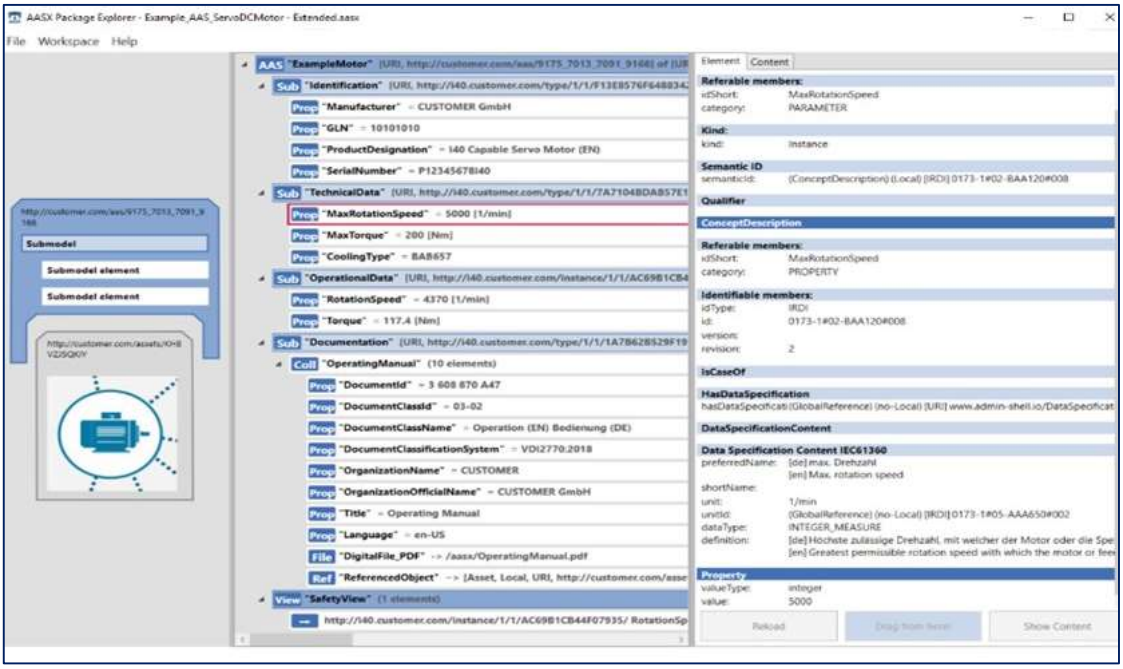


Abbildung 7: AASX Package Explorer Übersicht

Um den steigenden Anforderungen an Datenintegration, Transparenz und Automatisierung gerecht zu werden, entwickelte die OPC Foundation in Zusammenarbeit mit dem Zentralverband der Elektro- und Digitalindustrie (ZVEI) und dem Verband Deutscher Maschinen- und Anlagenbau (VDMA) die OPC UA „Companion Specification“ OPC 30270. Diese Spezifikation ermöglicht es, das Metamodell der AAS direkt in einen OPC UA Server zu integrieren. Damit leistet die OPC Foundation einen wichtigen Beitrag zur Standardisierung und erleichtert die praktische Umsetzung der AAS in industriellen Anwendungen. Durch die Verwendung dieser Spezifikation kann ein OPC UA Server so erweitert werden, dass er selbst als AAS fungiert. Die Struktur der AAS, welche aus der Shell selbst, ihren Teilmodellen,

Eigenschaften und Operationen besteht, wird dabei mithilfe standardisierter OPC UA Objekttypen, Variablentypen und Referenzen innerhalb des Adressraums des Servers abgebildet. Die „Companion Specification“ definiert hierzu unter anderem die Typen „AASAssetType“, „AASSubmodelType“ und „AASPropertyType“, welche die zentralen Elemente des AAS-Metamodells repräsentieren. Diese direkte Integration ermöglicht die Bereitstellung AAS-konformer Informationen über OPC UA, ohne dass zusätzliche Middleware-Lösungen erforderlich sind. Der OPC UA Server wird somit selbst zum Träger des digitalen Zwillings im Sinne der Industrie 4.0. Dies schafft die Grundlage für eine konsistente, standardisierte Kommunikation und erleichtert den Austausch von Informationen zwischen vernetzten Komponenten innerhalb moderner Produktionsumgebungen.(26) Die Umsetzung einer AAS auf Basis der OPC UA „Companion Specification“ OPC 30270 stellt somit eine Möglichkeit dar, eine vollständige AAS vom Typ 1 direkt im Addressraum eines OPC UA Servers abzubilden.(27)

Ein weiterer Realisierungsansatz für AASs wird durch das Open-Source-Framework Eclipse BaSyx bereitgestellt. Es stellt verschiedene Komponenten zur Verfügung, mit denen AASs erstellt, verwaltet und in Industrie 4.0 Anwendungen integriert werden können. Dabei unterstützt BaSyx sowohl einfache statische AASs als auch komplexere Strukturen mit Live-Daten und operativen Funktionen. Ein zentrales Merkmal ist die Möglichkeit, eine Verbindung zwischen einem physischen System und seinem digitalen Abbild herzustellen. Dies erlaubt es, Zustände und Prozesse in Echtzeit zu überwachen oder zu steuern. Die Kommunikation erfolgt über standardisierte Protokolle wie OPC UA oder MQTT, was eine Integration in bestehende Automatisierungslösungen erleichtert. Die BaSyx-Plattform ist modular aufgebaut. Sie enthält unter anderem Komponenten für das Verwalten von Teilmodellen, für die zentrale Registrierung von AASs sowie Schnittstellen zur Datenverarbeitung und Authentifizierung. Die einzelnen Module können flexibel kombiniert und an die jeweilige Systemarchitektur angepasst werden. Eclipse BaSyx ist quelloffen und richtet sich sowohl an Entwickler in der Forschung als auch an Anwender in der industriellen Praxis. Es eignet sich insbesondere für Szenarien, in denen die AAS aktiv mit anderen Systemen kommunizieren oder Methoden aufrufen soll, wie es bei AASs des Typs 2 erforderlich ist.(28)

Nach dem Herunterladen und Starten des offiziellen Starterkits von Eclipse BaSyx öffnet sich eine klar strukturierte Benutzeroberfläche, die als zentraler Einstiegspunkt in das BaSyx-System dient.(29) Die Abbildung 8 zeigt die Startseite der BaSyx-Webanwendung, von der aus direkt auf Dokumentation, Komponenten, Community sowie das AAS-Dataspace-Testumfeld zugegriffen werden kann.

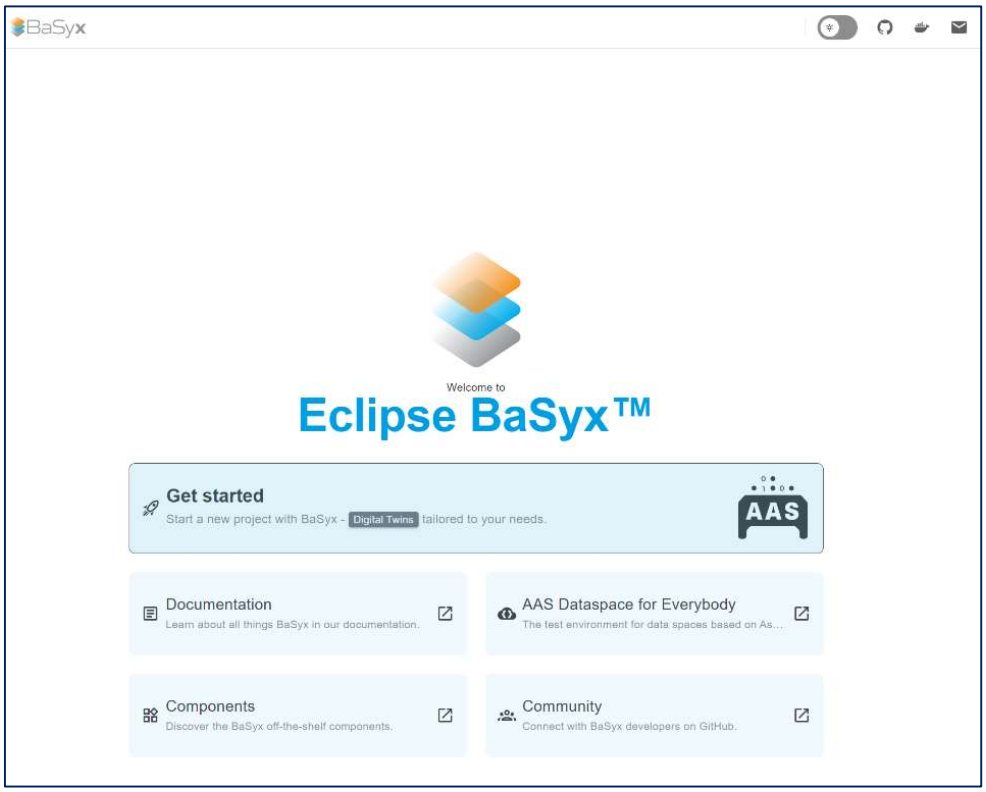


Abbildung 8: Eclipse BaSyx Benutzeroberfläche

Im Hintergrund basiert der Quick Start auf standardisierten Docker-Komponenten, die in der Version 2 von Eclipse BaSyx bereitgestellt werden. Durch den modularen Aufbau lassen sich alle notwendigen Dienste wie Registry, AAS-Server oder Benutzeroberfläche über vordefinierte Container starten. Die Starterumgebung bietet somit eine einfache Möglichkeit, BaSyx lokal auszuführen sowie erste digitale Zwillinge zu erzeugen und zu verwalten.

Eine weitere Möglichkeit zur Umsetzung einer AAS des Typen 2 bietet die REST-API-Spezifikation der Industrial Digital Twin Association (IDTA). Auf einer öffentlich zugänglichen Swagger-Oberfläche stellt die IDTA die vollständige Sammlung der REST-Endpunkte zur AAS gemäß Typ 2 bereit. Diese API-Spezifikation ermöglicht es, eine AAS vollständig über HTTP-Kommandos zu erzeugen, zu lesen, zu aktualisieren oder zu löschen, ohne zusätzliches Frontend oder eigene Backend-Logik. Über die bereitgestellte Weboberfläche lassen sich die einzelnen Endpunkte direkt inspizieren, testen und als OpenAPI-Definition exportieren. Diese Definition kann wiederum in eigene Anwendungen oder Tools integriert werden. Die REST-API deckt zentrale Funktionen wie das Anlegen einer AAS, das Hinzufügen von Teilmodellen oder das Bearbeiten einzelner Properties ab. Sie eignet sich besonders für Entwickler, die eine eigene Infrastruktur aufbauen möchten oder die AAS in bestehende Webarchitekturen einbinden wollen.(30) Die Abbildung 9 zeigt die Swagger-Umgebung mit der vollständigen API-Spezifikation zur AAS. Über die Benutzeroberfläche lassen sich alle verfügbaren REST-Endpunkte einsehen und direkt testen. Zusätzlich kann über die

Schaltfläche „Codegen“ Beispielcode für verschiedene Programmiersprachen wie C#, Java oder Python heruntergeladen werden. Dieser automatisch erzeugte Code dient als Vorlage für die eigene Implementierung und erleichtert den Einstieg in die Nutzung der Schnittstelle.

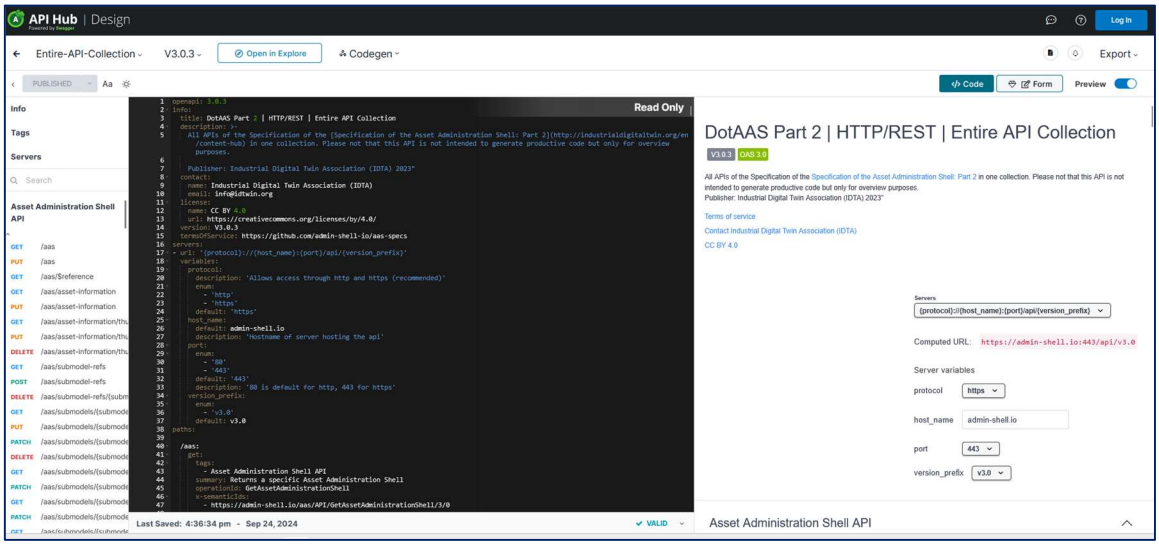


Abbildung 9: IDTA REST-API Spezifikation für AAS-Typ 2 über Swagger

Einen weiteren Ansatz zur Erstellung und Bereitstellung von AASs bietet das BaSyx.NET Framework des Eclipse-Projekts BaSyx an. Es handelt sich dabei um eine offizielle, vollständig in C#/.NET implementierte Version des Eclipse-BaSyx-Ökosystems, die auf den Industriestandards der Plattform Industrie 4.0 basiert. Das Framework orientiert sich an den Spezifikationen Typ 1 und Typ 2 und unterstützt die Erstellung, Verwaltung und Kommunikation von AASs gemäß dem AAS-Datenmodell. Das BaSyx.NET Framework stellt zahlreiche Komponenten zur Verfügung, mit denen sich AASs flexibel und modular umsetzen lassen. Zentrale Elemente sind unter anderem ein vollständig implementiertes AAS-SDK, ein standardkonformer AAS-HTTP-Server, ein Registry-Server zur Verwaltung verteilter AAS, ein AASX-File-Loader, sowie eine integrierte Web-Oberfläche zur Visualisierung und Konfiguration. Mit Hilfe des SDKs können AAS und Teilmodelle direkt in .NET erstellt oder aus dem standardisierten AASX-Dateiformat geladen werden. Dieses Format ist kompatibel mit dem AASX Package Explorer, sodass AASs modelliert, gespeichert und anschließend im .NET-Framework eingebunden und gehostet werden können. Über die integrierten HTTP-Server lassen sich die AASs über eine standardisierte REST-Schnittstelle bereitstellen. Dabei werden alle Create, Read, Update, Delete (CRUD)-Operationen auf AASs, Teilmodellen und Teilmodell-Elementen unterstützt. Sowohl Lesevorgänge als auch Schreibvorgänge sind möglich. Zusätzlich enthalten die Server eine automatisch generierte Swagger-Dokumentation sowie eine benutzerfreundliche Web-UI, mit der AASs direkt im Browser angezeigt und getestet werden können.

Ein weiterer zentraler Bestandteil des Frameworks ist der Registry-Server, der als Infrastrukturkomponente für die zentrale Verwaltung von AASs dient. AAS-Server können sich beim Start automatisch bei der Registry registrieren, wodurch eine dynamische Auffindbarkeit aller AAS im Netzwerk ermöglicht wird. Alternativ oder ergänzend unterstützt das Framework auch die automatische Erkennung im lokalen Netzwerk über das Multicast Domain Name System (mDNS). Dadurch ist es möglich, Dienste wie AAS-Server und Registry-Server auch ohne vorherige manuelle Konfiguration zu finden und zu verwenden. Zur Laufzeit kann das Framework auch als AASX File Server fungieren. Diese Funktion erlaubt es, eine komplette AASX-Datei zu hosten, wodurch nicht nur die darin enthaltene AAS, sondern auch alle zugehörigen Teilmodelle und eingebetteten Dateien über HTTP zugänglich gemacht werden. Damit lassen sich auch Dokumente, Bilder oder sonstige Zusatzressourcen innerhalb der AAS strukturiert bereitstellen.

Neben REST bietet das Framework auch Schnittstellen zu industriellen Protokollen. Für die Kommunikation mit Steuerungen oder Maschinen können Komponenten wie OPC UA und MQTT eingebunden werden. Diese Integration macht das Framework besonders geeignet für den Einsatz im Industrial IoT, da sowohl Information Technology (IT)- als auch Operational Technology (OT)-Systeme damit verbunden werden können. Das GitHub-Repository stellt eine Vielzahl an Beispielprojekten zur Verfügung, mit denen typische Anwendungsfälle der AAS demonstriert werden. Anhand dieser Beispiele wird aufgezeigt, wie sich mit geringem Aufwand eine eigene AAS entwickeln und bereitstellen lässt. Darüber hinaus werden auch weiterführende Szenarien wie komplexe Teilmodelle, externe Repositories oder kombinierte Infrastrukturen durch die bereitgestellten Anwendungen abgedeckt. Im Gegensatz zu Java-basierten BaSyx-Implementierungen setzt das BaSyx.NET Framework auf modularen .NET Core Code, der sowohl plattformunabhängig als auch leicht in bestehende .NET-Projekte integrierbar ist. Dadurch eignet sich das Framework besonders für Entwickler und Entwicklerinnen im .NET-Umfeld, die ohne großen Mehraufwand auf die Welt der Industrie 4.0 und digitalen Zwillinge zugreifen möchten.

Insgesamt stellt das BaSyx.NET Framework eine umfangreiche und praxisnahe Lösung zur Umsetzung von AASs dar. Es deckt sowohl die Modellierung, Bereitstellung, Kommunikation als auch Integration in bestehende industrielle Infrastrukturen ab und bietet somit eine vollständig einsetzbare Technologie für Industrie 4.0-Anwendungen auf Bagsis von .NET.(31) Die folgende Abbildung veranschaulicht die zentralen Komponenten des BaSyx.NET Frameworks und deren Zusammenspiel im Kontext der AAS.

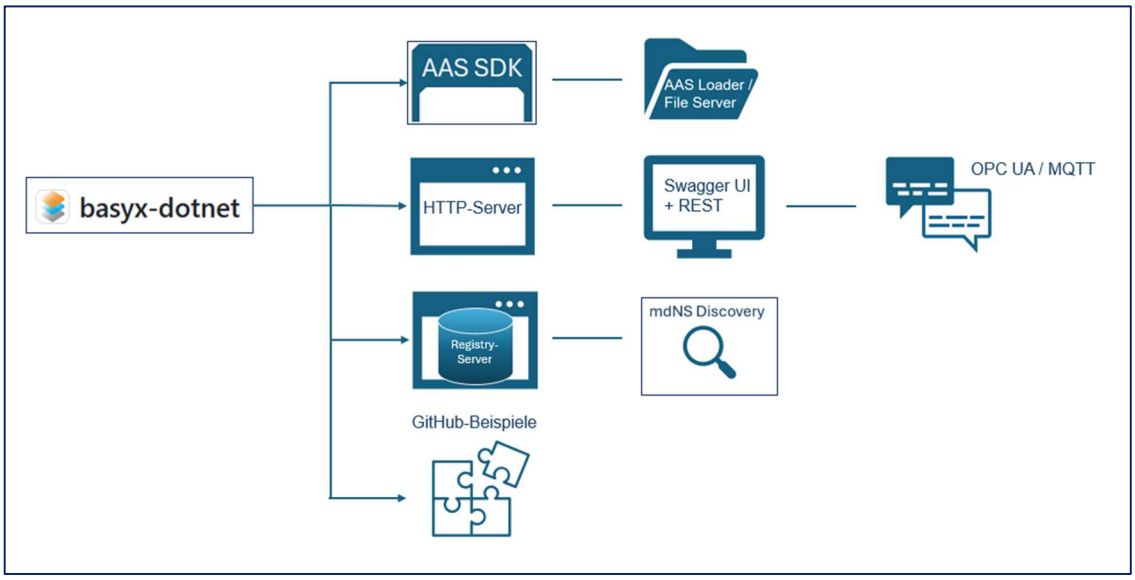


Abbildung 10: Übersicht der Hauptkomponenten des BaSyx.NET Frameworks zur Erstellung, Bereitstellung und Verwaltung von AASs nach den Spezifikationen der Plattform Industrie 4.0.(31)

Neben bestehenden Tools und Frameworks besteht auch die Möglichkeit, eine AAS vollständig selbst zu implementieren. Dabei kann auf verschiedene Open-Source-Repositories zurückgegriffen werden, die entweder generische Funktionalitäten oder speziell auf die AAS ausgerichtete Komponenten bereitstellen. Ein Beispiel hierfür ist das GitHub-Repository der OPC Foundation, das eine .NET-basierte Client- und Server-Implementierung für das OPC UA Kommunikationsprotokoll zur Verfügung stellt.(32) Dieses Repository bietet unter anderem die Möglichkeit, Methoden aufzurufen, Werte zu lesen und zu schreiben sowie eine stabile Session-Verwaltung mit automatischer Reconnect-Logik umzusetzen. Diese Funktionen sind für eine AAS des Typen 2 essenziell. Darüber hinaus existieren weitere Repositories, die den Aufbau von AAS-Strukturen oder sogar komplette Serverumgebungen ermöglichen. Je nach Anwendungsfall können diese Repositories als Grundlage genutzt oder modular erweitert werden, um eigene Anforderungen zu realisieren. Die eigenständige Entwicklung bietet insbesondere dann Vorteile, wenn spezielle Anpassungen oder tiefere Integrationen erforderlich sind, etwa zur direkten Verknüpfung mit speicherprogrammierbaren Steuerungen oder zur Einbindung domänenspezifischer Datenmodelle. Gleichzeitig erfordert dieser Ansatz fundierte Kenntnisse in der Softwareentwicklung sowie ein gutes Verständnis des AAS-Metamodells und der zugrunde liegenden Kommunikationsprotokolle.

2.8 Skill-based Engineering

SkE ist ein zentraler Ansatz der Industrie 4.0, der den Fokus auf die Beschreibung von Fähigkeiten und deren konkrete Umsetzung in Form von Skills legt. Es handelt sich dabei um ein flexibles und wiederverwendbares Konzept, das Systeme nicht mehr ausschließlich auf bestimmte Hardware festlegt, sondern auf die Fähigkeiten, die eine Ressource

bereitstellen kann. Fähigkeiten sind dabei abstrakte, implementationsunabhängige Beschreibungen der Funktion eines Assets mit dem Ziel, eine bestimmte Wirkung in der physischen oder virtuellen Welt zu erzielen. Skills hingegen stellen die konkrete, assetbezogene Umsetzung einer Fähigkeit dar und sind ein zentrales Element im Konzept des SkE. Ein wesentlicher Aspekt dieses Ansatzes ist die Unterscheidung zwischen Basic Skills und kombinierten Skills. Als Basic Skills werden in dieser Arbeit grundlegende, nicht weiter unterteilbare Maschinenfunktionen definiert. Sie führen jeweils eine spezifische, abgeschlossene Aktion aus, wie beispielsweise das Starten eines Förderbands oder das Einfahren eines Stoppers. Kombinierte Skills hingegen setzen sich aus mehreren Basic Skills zusammen und ermöglichen dadurch die Ausführung komplexerer Aufgaben, die mehrere Einzelschritte umfassen.(33) So kann beispielsweise ein kombinierter Skill zur Werkstückklassifizierung verschiedene Sensordaten auswerten und in Abhängigkeit des Ergebnisses eine entsprechende Weiche ansteuern. Auch Kombinationen mehrerer kombinierter Skills sind denkbar, um besonders umfangreiche Abläufe zu realisieren. SkE verfolgt das Ziel, Produktionssysteme auf Basis der benötigten Fähigkeiten zu planen und nicht auf Basis konkreter Hardwarekomponenten. Dies führt zu einer höheren Flexibilität im Engineering-Prozess, etwa beim Austausch einzelner Komponenten oder der Anpassung an neue Produktvarianten. Die Verwaltung der Fähigkeiten und Skills erfolgt häufig über die AAS. Sie beinhaltet standardisierte Teilmodelle, in denen Fähigkeiten beschrieben und die zugehörigen Skills über Schnittstellen, wie etwa OPC UA-Methoden, abgerufen werden können. Dadurch wird eine interoperable und automatisierte Integration von Funktionen über System- und Herstellergrenzen hinweg ermöglicht.(34)

2.9 Fachliche Einordnung und funktionale Abgrenzung des digitalen Produktpasses zur AAS

Der digitale Produktpass ist ein von der Europäischen Union initiiertes Konzept zur digitalen Erfassung und Bereitstellung produktbezogener Informationen über den gesamten Lebenszyklus hinweg. Er fungiert als digitaler Ausweis eines physischen Produkts und soll insbesondere Transparenz, Rückverfolgbarkeit und Nachhaltigkeit entlang der gesamten Wertschöpfungskette fördern. Ab dem Jahr 2027 wird der digitale Produktpass schrittweise für verschiedene Produktgruppen verpflichtend eingeführt, darunter Elektrofahrzeuge, Industriebatterien und elektronische Geräte. Unternehmen werden dadurch dazu verpflichtet, präzise Daten zu Materialzusammensetzungen, Produktionsprozessen, Energieverbrauch, Umweltauswirkungen und Recyclingfähigkeit strukturiert zu erfassen und digital bereitzustellen. Ziel ist es, gesetzlichen Anforderungen zu entsprechen und gleichzeitig zentrale strategische Vorteile zu erzielen. Dazu zählen die Einhaltung von Compliance-Vorgaben, ein aktiver Beitrag zur Kreislaufwirtschaft, die Stärkung der Marktposition durch

transparente Produktinformationen, die Optimierung globaler Lieferketten sowie die Förderung eines nachhaltigen Produktdesigns. Obwohl es inhaltliche Überschneidungen mit der AAS gibt, etwa bei der strukturierten Abbildung technischer Informationen, verfolgt der digitale Produktpass primär regulatorische und ökologische Zielsetzungen. Eine AAS hingegen ist derzeit an keine gesetzliche Pflicht gebunden und dient in erster Linie der operativen Maschinen- und Systemintegration, beispielsweise zur Ansteuerung von Funktionen oder zur standardisierten Datenbereitstellung innerhalb eines Industrie 4.0 Kontexts. Im Gegensatz zum Produktpass bildet sie jedoch keine Rückverfolgbarkeit entlang der Lieferkette ab.(35)

3. Konzeption der AAS

Aufbauend auf den theoretischen Grundlagen aus Kapitel 2 wird in diesem Kapitel das Konzept zur konkreten Umsetzung einer AAS für die Festo MPS-Station „Sortieren“ vorgestellt. Das Konzept wird anhand von Anforderungen erstellt, die auf Basis des vergangenen Kapitels ermittelt werden.

3.1 Beschreibung des physischen Assets

Die MPS-Station mit der Artikelnummer 8046325 der Firma Festo Didactic SE dient der automatisierten Sortierung von Werkstücken anhand ihrer Material- und Farbeigenschaften auf drei separaten Rutschen.

Funktion

Zu Beginn des Sortierprozesses erkennt eine Gabellichtschranke, wenn ein Werkstück auf das Förderband gelegt wird. Anschließend prüfen verschiedene Sensoren die Beschaffenheit der Werkstücke. Ein induktiver Näherungsschalter identifiziert metallische Werkstücke, ein Reflexlichttaster detektiert rote Werkstücke, während Werkstücke mit schwarzer Oberfläche direkt durch die Gabellichtschranke erkannt werden. Basierend auf der erfassten Material- oder Farbinformation wird eine der beiden elektrisch angetriebenen Weichen angesteuert, um das Werkstück in die passende Rutsche zu leiten. Der Weitertransport erfolgt über das Förderband, das die Werkstücke entsprechend der Weichenstellung einer von drei Rutschen zuführt. Am Ende jeder Rutsche befindet sich zusätzlich eine Reflex-Lichtschranke, die den Füllstand überwacht und eine Überfüllung verhindert. Die gesamte Station wird durch eine speicherprogrammierbare Steuerung des Typs SIMATIC S7-1500 gesteuert. Diese Steuerung ist vollständig in das Engineering-System Siemens TIA-Portal in der Version 16 integriert, über das sämtliche Programmierungen und Konfigurationen der Station vorgenommen werden. Die Abbildung 11 zeigt links die Festo MPS-Station

„Sortieren“ mit ihrer Sensorik und Aktorik sowie rechts die SIMATIC S7-1500 Steuerung mit der Netzwerkverkabelung im Schaltschrank.

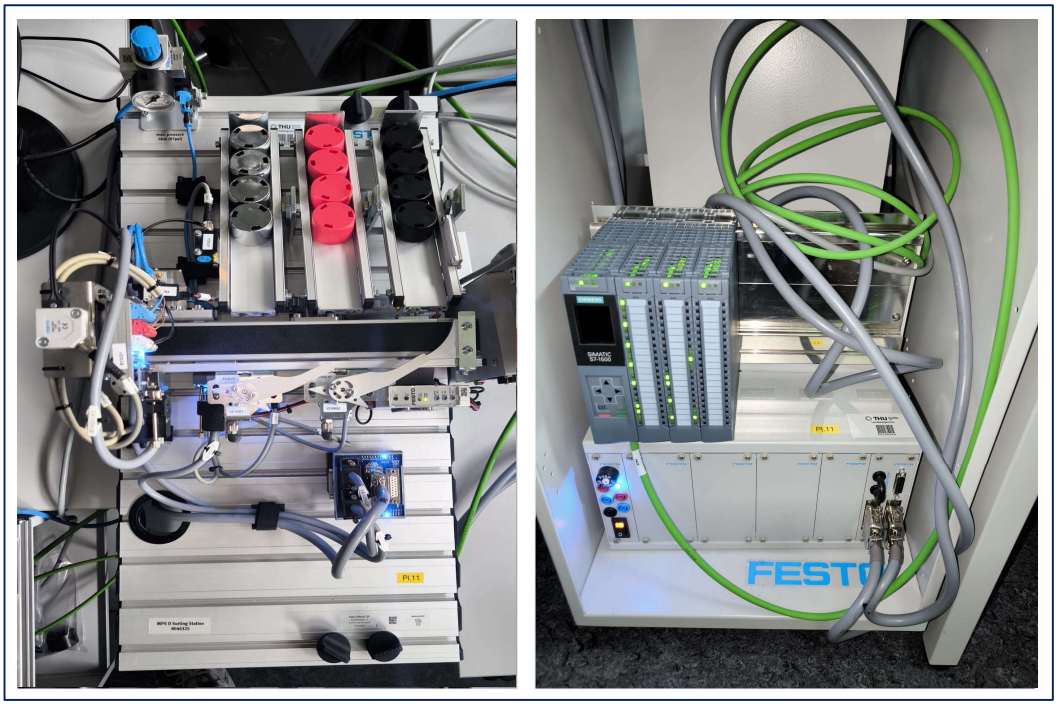


Abbildung 11: Festo MPS-Station Sortieren im Automatisierungslabor der Technischen Hochschule Ulm

3.2 Allgemeine Anforderungen

Die in dieser Arbeit zu entwickelnde AAS soll dem Typ 2 entsprechen. Sie soll sowohl strukturierte Informationen abbilden als auch Steuerungsfunktionen auslösen können. Ein modularer Aufbau sowie die Wiederverwendbarkeit der Struktur stellen zentrale Anforderungen dar (**A1**), um die AAS auch auf weitere Stationen im Automatisierungslabor der Technischen Hochschule Ulm übertragen zu können. Darüber hinaus soll der Aufbau praxisnah gestaltet sein, sodass die AAS leicht in bestehende Automatisierungssysteme integrierbar ist und sich in realen Anwendungsszenarien bewährt (**A2**). Zusätzlich soll die AAS über eine GUI verfügen, um eine einfache Interaktion mit den enthaltenen Teilmodellen zu ermöglichen (**A3**). Des Weiteren soll die AAS eine direkte Kommunikation mit der SPS der Sortierstation ermöglichen, über die sich Skills ausführen lassen, die auf Sensoren und Aktoren der Anlage zugreifen (**A4**). Dabei soll die Steuerung zuverlässig und mit geringer Reaktionszeit auf entsprechende Anfragen reagieren können. Für die Kommunikation zwischen der AAS und der Steuerung wird eine Schnittstelle benötigt, die einen standardisierten und zuverlässigen Datenaustausch ermöglicht. Die Steuerungsfunktionen sollen über eindeutig identifizierbare Aufrufpunkte angesteuert werden können. Dabei ist insbesondere darauf zu achten, dass die Verbindung stabil und sicher ausgelegt ist, um eine zuverlässige Ausführung der Funktionen zu gewährleisten (**A5**).

3.3 Teilmodelle

Die AAS umfasst elf Teilmodelle, die verschiedene Aspekte des Assets abbilden. Dazu gehören:

- **1. Teilmodell – Asset Identifikation**: Dieses Teilmodell soll Herstellerinformationen, die Seriennummer sowie die Typbezeichnung der Station enthalten. Es bildet damit das digitale Äquivalent zum physischen Typenschild und dient der automatisierten Identifikation der Station.
- **2. Teilmodell – Informationen zur Sortierstation**: Dieses Teilmodell soll allgemeine Informationen zur Station bereitstellen, darunter ein Bild des Aufbaus, eine Beschreibung der Funktion sowie Hinweise zur Inbetriebnahme. Die Inhalte orientieren sich an den Informationen des Festo Infoportals.
- **3. Teilmodell – Dokumente und Medien**: Dieses Teilmodell soll exemplarisch relevante Dokumente und Medien enthalten, die aus dem Festo Infoportal stammen. Ziel ist eine strukturierte und digitale Bereitstellung.
- **4. Teilmodell – Technische Daten**: Dieses Teilmodell soll technische Parameter wie Versorgungsspannung, Maße und Anschlussdaten enthalten. Die Informationen sollen in strukturierter Form als Teilmodellelemente abgebildet werden.
- **5. Teilmodell – Basic Skills aus JSON**: Dieses Teilmodell soll dynamisch generiert werden und Basic Skills enthalten, die aus einer externen JSON-Datei geladen werden. Die Ausführung der Skills soll über eine standardisierte Kommunikationsschnittstelle erfolgen. In diesem Teilmodell werden ausschließlich Skills abgebildet, die sensorabhängig sind und deren Ausführung auf Echtzeitdaten basiert.
- **6. Teilmodell – Basic Skills mit zusätzlichen Logiken**: Dieses Teilmodell soll Skills enthalten, die über einfache Eingabeparameter angesteuert werden können und dadurch logische Verzweigungen ermöglichen. Die Skills erwarten in der Regel einen binären Wert, wobei 0 eine Aktion wie Stoppen oder Einfahren und 1 das Starten oder Ausfahren auslöst. Vorgesehene Funktionen sind unter anderem das Steuern des Stoppers, der Weichen sowie des Förderbands. Durch die parametergesteuerte Ausführung kann die gewünschte Funktion gezielt aufgerufen werden. Dies vereinfacht die Bedienung, reduziert die Anzahl an Einzel-Skills und ermöglicht eine flexible und einheitliche Integration in die AAS.
- **7. Teilmodell – Kombinierter Skill aus JSON**: Dieses dynamisch eingebundene Teilmodell soll einen kombinierten Skill enthalten, der zur Erkennung der Werkstückfarbe dient. Der Skill „Werkstückfarbe_detektieren“ basiert auf der logischen Verknüpfung mehrerer Sensorwerte zur Ableitung der tatsächlichen Farbe. Die Definition und

Einbindung erfolgt über eine externe JSON-Datei, wodurch eine flexible Erweiterung und Anpassung ohne direkten Eingriff in den Quellcode ermöglicht wird.

- **8. Teilmodell – Kombinierte Skills**: Dieses Teilmodell soll mehrere kombinierte Skills enthalten, die manuell modelliert werden. Dazu gehört beispielsweise der Skill „Werkstück_in_Rutsche_i_befördern“, bei dem das Werkstück abhängig vom eingegebenen Zielwert in eine der drei Rutschen geleitet wird. Ein weiterer Skill mit erweiterter Logik ist „Werkstück_der_Farbe_j_in_Rutsche_i_befördern“, der zunächst die erkannte Werkstückfarbe überprüft und das Werkstück nur dann weiterleitet, wenn sie dem übergebenen Parameter entspricht. Zusätzlich ist der Skill „Grundeinstellung“ vorgesehen, der alle Aktoren in einen definierten Ausgangszustand versetzt. Dieses Teilmodell soll komplexe Abläufe innerhalb der Station strukturiert abbilden und gleichzeitig eine flexible Steuerung über Parameter ermöglichen.
- **9. Teilmodell – Statisch codiertes Skill-SubmodelBeispiel**: Dieses Teilmodell soll exemplarisch einen statisch codierten Skill mit dem Namen „Band_antreiben“ enthalten. Im Gegensatz zu dynamisch geladenen oder parametrisierten Skills ist dieser fest im Quellcode implementiert und enthält weder Logikverzweigungen noch externe Konfigurationen über JSON-Dateien. Er dient als Referenz für einfache Skill-Definitionen und zeigt, wie eine direkte, unverzweigte Maschinenfunktion innerhalb der AAS abgebildet werden kann.
- **10. Teilmodell – Betriebsdaten**: Dieses Teilmodell soll Betriebsdaten der Sortierstation in strukturierter Form bereitstellen. Es umfasst einerseits Zustandsdaten, wie beispielsweise den aktuellen Verbindungsstatus, und andererseits Statusinformationen zu aktiven Komponenten, etwa ob das Förderband in Betrieb ist. Die relevanten Informationen sollen über vordefinierte Datenpunkte erfasst und als Teilmodellelemente in die AAS eingebunden werden. Ziel ist eine kontinuierliche Überwachung des Systemzustands innerhalb der digitalen Repräsentation.
- **11. Teilmodell – Basic Skills**: Dieses Teilmodell soll grundlegende steuerbare Funktionen der Sortierstation abbilden, die jeweils durch eine Start- und eine Stopplogik getrennt ausgelöst werden können. Für jede Bewegung oder Aktion sind separate Steuerbefehle vorgesehen, etwa zum Ein- und Ausfahren des Stoppers, zur Ansteuerung der beiden Weichen sowie zum An- und Abstellen des Förderbands. Vorgesehen sind unter anderem folgende Skills: „Stopper_einfahren“, „Stopper_ausfahren“, „Weiche_1_ausfahren“, „Weiche_1_einfahren“, „Weiche_2_ausfahren“, „Weiche_2_einfahren“, „Band_antreiben“ und „Band_anhalten“. Die einzelnen Funktionen sollen über eindeutige Steuerpunkte ansprechbar und in die AAS integriert sein, um eine gezielte Aktor-Ansteuerung zu ermöglichen.

Struktur innerhalb der skill-basierten Teilmodelle

Alle skill-basierten Teilmodelle sollen mindestens ein Operation-Element zur Ausführung der jeweiligen Funktion sowie ein Property-Element zur Rückmeldung des Ausführungsstatus enthalten. Skills mit einfachen Eingabeparametern erwarten typischerweise Werte wie Null oder Eins, um beispielsweise eine Einfahr- oder Ausfahrbewegung zu steuern. Bei kombinierten Skills kommen Zahlenwerte wie Eins, Zwei oder Drei zum Einsatz, die innerhalb der jeweiligen Steuerlogik einem bestimmten Verzweigungspfad zugeordnet sind. Sowohl Eingabe- als auch Rückgabewerte sollen dabei eindeutig adressierbar sein, um eine gezielte Ansteuerung und eine korrekte Ausführungslogik sicherzustellen. Nach der Ausführung soll ein Statuswert bereitgestellt werden, der intern verarbeitet wird. Abhängig davon, ob der Wert beispielsweise „wahr“ oder „falsch“ ist, wird ein entsprechender Rückmeldungstext generiert und bereitgestellt. Bei einem erfolgreichen Ablauf kann dies etwa „Skill erfolgreich ausgeführt“ sein, während bei Fehlern Meldungen wie „Skill fehlgeschlagen“ oder „keine Reaktion“ angezeigt werden. Die UI liest diesen Rückmeldungstext über das zugehörige Property-Element im Teilmodell aus und stellt ihn visuell dar. Auf diese Weise wird jeder Skill-Aufruf nicht nur technisch ausgeführt, sondern auch verständlich dokumentiert und für den User transparent rückgemeldet.

3.4 Zusammenfassung der Skill-Typen

In der AAS werden unterschiedliche Skill-Typen unterschieden, die sich hinsichtlich ihrer Komplexität und Funktionsweise voneinander abgrenzen. Zu den einfachsten Varianten zählen Basic Skills mit direkter Wirkung. Diese führen eine einzelne Funktion aus, wie beispielsweise das Einfahren des Stoppers oder das Starten des Förderbands. Sie enthalten keine Entscheidungslogik und lösen eine vordefinierte Aktion unmittelbar aus. Basic Skills mit Logik und Eingabeparametern erweitern dieses Konzept um einfache Entscheidungsstrukturen. Sie erwarten bestimmte Eingabewerte, anhand derer sich das Verhalten der Funktion verändert. So kann der Skill „Band_steuern“ beispielsweise sowohl das Anhalten als auch das Starten des Förderbands auslösen, abhängig davon, ob der Eingabewert 0 oder 1 beträgt. Kombinierte Skills fassen mehrere aufeinanderfolgende Schritte zu einer übergeordneten Funktion zusammen. Ein typisches Beispiel ist der Skill „Werkstückfarbe_detektieren“. Dieser wertet mehrere Eingangssignale aus, interpretiert die Informationen und leitet daraus weitere Aktionen ab, wie beispielsweise die Aktivierung einer bestimmten Rutsche. Bei kombinierten Skills wie zum Beispiel „Werkstück_in_Rutsche_i_befördern“ ist zusätzlich eine automatische Rücksetzung nach fünf Sekunden vorgesehen, um sicherzustellen, dass die Steuerung nicht dauerhaft in einer aktiven Position verbleibt.

3.5 Strukturierung der SPS-Variablen

Bereits in der Konzeptionsphase wird eine einheitliche Benennungsstrategie für die SPS-Variablen festgelegt, um eine effiziente und fehlerfreie Implementierung zu unterstützen. Dabei kommen bewusst generische Namen wie „Tag_1“, „Tag_2“ usw. zum Einsatz. Diese Namenskonvention ermöglicht eine schnelle Wiedererkennung und erleichtert die Auswahl der Variablen in der Programmierumgebung, da die relevanten Einträge bereits bei der Eingabe automatisch vorgeschlagen werden. Zusätzlich wird für jede Variable ein eindeutiger Kommentar hinterlegt, etwa „Bandmotor“, um die Funktion der jeweiligen Steuergröße nachvollziehbar zu machen. Dadurch wird die Einbindung in Funktionsbausteine übersichtlicher und der Entwicklungsaufwand reduziert. Um die Anzahl an redundanten Einträgen in der Steuerung zu minimieren, werden die global in der Variablentabelle angelegten Variablen direkt weiterverwendet. Eine doppelte Deklaration innerhalb von Funktionsbausteinen wird bewusst vermieden, um die Komplexität der Projektstruktur gering zu halten.

Für spezielle Eingabewerte, beispielsweise zur Parametrisierung einzelner Funktionen, werden zusätzliche lokale Variablen innerhalb der jeweiligen Funktionsbausteine definiert. Diese dienen der gezielten Übergabe von Steuerparametern und gewährleisten eine klare Trennung zwischen allgemeinen Steuergrößen und funktionsspezifischen Parametern. Insgesamt soll diese strukturierte Vorgehensweise eine transparente Zuordnung von Steuerparametern ermöglichen und gleichzeitig sicherstellen, dass die Funktionen innerhalb der Steuerung eindeutig ansprechbar sind. Die gewählte Strategie bildet somit eine konzeptionelle Grundlage für die spätere Anbindung und Ausführung von Skills durch die AAS. Im Folgenden zeigt Tabelle 1 die benötigten PLC-Variablen, welche die Grundlage, für die in den Skill-basierten Teilmodellen umgesetzten Basic Skills und kombinierten Skills bilden.

Tabelle 1: PLC-Variablen der Steuerung

Variable	Datentyp	Adresse	Kommentar
Tag_1	bool	%Q4.0	Bandmotor
Tag_3	bool	%Q4.1	Weiche 1
Tag_4	bool	%Q4.2	Weiche 2
Tag_5	bool	%Q4.3	Stopper
Tag_6	bool	%I10.6	Induktiver Näherungsschalter

Tag_7	bool	%I10.5	Reflexlichttaster
Tag_8	bool	%I10.4	Gabellichtschranke

Übersicht Basic Skills

1. **Werkstück_am_Bandanfang_erkennen**

Dieser Skill ist darauf ausgelegt, mithilfe der Gabellichtschranke im Modul „Erkennen“ festzustellen, ob sich ein Werkstück am Anfang des Förderbands befindet. Die Auswertung erfolgt auf Basis eines realen Sensorwerts (Tag_8), der als Eingangsgröße dient. Bei Erkennung eines Werkstücks wird der Zustand über den Ausgabeparameter „WerkstückErkannt“ übermittelt. Tabelle 2 stellt die verwendete Variable und die zugehörigen Parameter exemplarisch dar.

Tabelle 2: Werkstück_am_Bandanfang_erkennen

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	WerkstückErkannt	Keine	WerkstückErkannt

2. **Werkstückfarbe_Metall_erkennen**

Zur Erkennung eines metallischen Werkstücks ist vorgesehen, den induktiven Näherungsschalter im Modul „Erkennen“ als Sensorquelle zu nutzen. Der Sensor (Tag_6) liefert bei Erkennung eines metallischen Objekts ein positives Signal. Dieser Zustand wird im Skill verarbeitet und über die Variable „FarbeMetall“ abgebildet. Tabelle 3 zeigt die dabei verwendete Variable und die zugehörigen Parameter.

Tabelle 3: Werkstückfarbe_Metall_erkennen

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	FarbeMetall	Keine	FarbeMetall

3. **Werkstückfarbe_Rot_erkennen**

Zur Erkennung eines roten Werkstücks ist vorgesehen, den Reflexlichttaster im Modul „Erkennen“ als alleinige Sensorquelle zu nutzen. Der Sensor (Tag_7) liefert bei Anwesenheit eines roten Werkstücks ein entsprechendes Signal, das im Skill verarbeitet und über die

Variable „FarbeRot“ dargestellt wird. Tabelle 4 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 4: Werkstückfarbe_Rot_erkennen

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	FarbeRot	Keine	FarbeRot

4. Werkstückfarbe_Schwarz_erkennen

Zur Erkennung eines schwarzen Werkstücks ist vorgesehen, die Gabellichtschranke im Modul „Erkennen“ als alleinige Sensorquelle zu nutzen. Der Sensor (Tag_8) liefert bei Anwesenheit eines schwarzen Werkstücks ein entsprechendes Signal, das im Skill verarbeitet und über die Variable „FarbeSchwarz“ abgebildet wird. Tabelle 5 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 5: Werkstückfarbe_Schwarz_erkennen

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	FarbeSchwarz	Keine	FarbeSchwarz

5. Stopper_einfahren

Der Skill „Stopper_einfahren“ ist darauf ausgelegt, über die Variable Tag_5 die Einfahrbewegung des Stoppers auszulösen. Der Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 6 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 6: Stopper_einfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_5	Keine	Tag_5

6. Stopper_ausfahren

Der Skill „Stopper_ausfahren“ ist darauf ausgelegt, über die Variable Tag_5 die Ausfahrbewegung des Stoppers zu initiieren. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 7 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 7: Stopper_ausfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_5	Keine	Tag_5

7. Weiche_1_ausfahren

Der Skill „Weiche_1_ausfahren“ ist darauf ausgelegt, über die Variable Tag_3 die Ausfahrbewegung der Weiche 1 auszulösen. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 8: Weiche_1_ausfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_3	Keine	Tag_3

8. Weiche_1_einfahren

Der Skill *Weiche_1_einfahren* ist darauf ausgelegt, über die Variable Tag_3 die Einfahrbewegung der Weiche 1 zu initiieren. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 9 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 9: Weiche_1_einfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_3	Keine	Tag_3

9. Weiche_2_ausfahren

Der Skill *Weiche_2_ausfahren* ist darauf ausgelegt, über die Variable Tag_4 die Ausfahrbewegung der Weiche 2 auszulösen. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 10 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 10: Weiche_2_ausfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter

bool	Tag_4	Keine	Tag_4

10. Weiche_2_einfahren

Der Skill „Weiche_2_einfahren“ ist darauf ausgelegt, über die Variable Tag_4 die Einfahrbewegung der Weiche 2 zu initiieren. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 11 zeigt die verwendete Variable und die zugehörigen Parameter

Tabelle 11: Weiche_2_einfahren

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_4	Keine	Tag_4

11. Band_antreiben

Der Skill Band_antreiben ist darauf ausgelegt, über die Variable Tag_1 den Antrieb des Förderbands zu aktivieren. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 12 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 12: Band_antreiben

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_1	Keine	Tag_1

12. Band_anhalten

Der Skill „Band_anhalten“ ist darauf ausgelegt, über die Variable Tag_1 den Antrieb des Förderbands zu deaktivieren. Der aktuelle Zustand der Variable wird anschließend als Ausgabeparameter bereitgestellt. Tabelle 13 zeigt die verwendete Variable und die zugehörigen Parameter.

Tabelle 13: Band_anhalten

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_1	Keine	Tag_1

Übersicht Kombinierte Skills

1. Werkstückfarbe_detektieren

Der kombinierte Skill „Werkstückfarbe_detektieren“ wertet die Signale der drei Sensoren (Tag_6, Tag_7, Tag_8) gemeinsam aus. Auf Basis definierter Logik wird die tatsächliche Farbe des Werkstücks bestimmt und als Zeichenkette (string) in der Variable „Werkstückfarbe“ gespeichert. Mögliche Rückgabewerte sind „Metall“, „Rot“, „Schwarz“ oder "Unbekannt". Tabelle 14 veranschaulicht die Rückgabewerte des Skills, die in Abhängigkeit der Sensorzustände aus der Variable „Werkstückfarbe“ resultieren.

Tabelle 14: Kombinierte Skills

Datentyp	Variable	Eingabe-parameter	Ausgabe-parameter	Bedingung für Rückgabe
string	Werkstückfarbe	Keine	Metall	Tag_6 = true Tag_7 = true Tag_8 = true
string	Werkstückfarbe	Keine	Rot	Tag_6 = false Tag_7 = true Tag_8 = true
string	Werkstückfarbe	Keine	Schwarz	Tag_6 = false Tag_7 = false Tag_8 = true
string	Werkstückfarbe	Keine	Unbe-kannt	Tag_6 = false Tag_7 = false Tag_8 = false

2. Werkstück_in_Rutsche_i_befördern

Der Skill „Werkstück_in_Rutsche_i_befördern“ ist darauf ausgelegt, ein Werkstück gezielt in eine der drei vorhandenen Rutschen zu leiten. Dies soll über einen Eingabeparameter erfolgen, der die Zielrutsche bestimmt. Unabhängig von der Auswahl wird in jedem Fall der Stopper eingefahren und das Förderband gestartet. Bei der Eingabe der Zahl 1 wird zusätzlich Weiche 1 ausgefahren, wodurch das Werkstück in Rutsche 1 gelenkt wird. Bei der Eingabe der Zahl 1 ist vorgesehen, zusätzlich Weiche 1 auszufahren, sodass das Werkstück in Rutsche 1 geleitet wird. Bei Eingabe von 2 soll Weiche 2 aktiviert werden, um eine Sortierung in Rutsche 2 zu ermöglichen. Wird der Wert 3 übergeben, bleiben beide Weichen deaktiviert und das Werkstück gelangt in Rutsche 3. Nach einer festgelegten Ausführungszeit von fünf Sekunden soll der Ablauf automatisch beendet und die Sortierstation in ihren Ausgangszustand zurückgeführt werden. Tabelle 15 zeigt die Verwendung der Variable Rutsche, die je nach Eingabewert unterschiedliche Ausgabekonstellationen erzeugt.

Tabelle 15: Werkstück_in_Rutsche_i_befördern

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
int	Rutsche	1	Tag_1: true Tag_3: true Tag_4: false Tag_5: true
int	Rutsche	2	Tag_1: true Tag_3: false Tag_4: true Tag_5: true
int	Rutsche	3	Tag_1: true Tag_3: false Tag_4: false Tag_5: true
int	Rutsche	Zahlen außer 1,2,3	Tag_1: false Tag_3: false Tag_4: false Tag_5: false

3. Werkstück_der_Farbe_j_in_Rutsche_i_befördern

Der kombinierte Skill „Werkstück_der_Farbe_j_in_Rutsche_i_befördern“ verknüpft die beiden zuvor beschriebenen Teilfunktionen. Er ist so konzipiert, dass zwei Eingabeparameter übergeben werden: ein Farbwert mit den Werten 1 für Metall, 2 für Rot oder 3 für Schwarz sowie eine Ziel-Rutschen-Nummer mit den Werten 1, 2 oder 3. Zunächst soll die tatsächliche Werkstückfarbe durch den Skill „Werkstückfarbe_detektieren“ ermittelt werden. Anschließend erfolgt ein Abgleich mit dem übergebenen Farbwert. Nur wenn beide Bedingungen erfüllt sind, also der Farbwert mit dem erkannten Wert übereinstimmt und eine gültige Ziel-Rutschen-Nummer eingegeben wurde, wird die Sortieraktion ausgelöst. Stimmen die Werte nicht überein oder ist die Zielangabe ungültig, wird keine Aktion ausgeführt. Die Steuerlogik ist so vorgesehen, dass nach erfolgreicher Ausführung eine automatische Rücksetzung in den Ausgangszustand der Sortierstation erfolgt. Tabelle 16 veranschaulicht die möglichen Eingabeparameter und deren Bedeutung, die zugehörigen Ausgabeparameter sowie die Bedingung für die Ausführung dieses kombinierten Skills.

Tabelle 16: Werkstück_der_Farbe_j_in_Rutsche_i_befördern

Daten-typ	Variable	Eingabepara-meter	Abgleich mit Variable:	Bedingung für die Ausführung
int	Farbe	1 (= Metall) 2 (= Rot) 3 (= Schwarz)	DetektierteFarbe	Nur wenn Farbe = DetektierteFarbe und

int	Rutsche	1 (= Rutsche 1) 2 (= Rutsche 2) 3 (= Rutsche 3)	EingabeRutsche	EingabeRutsche ∈ {1, 2, 3}

4. Grundeinstellung

Zur Absicherung des Systemzustands ist im Teilmodell der kombinierten Skills ein zusätzlicher Skill mit dem Namen „Grundeinstellung“ vorgesehen. Dieser soll alle relevanten Aktoren in einen definierten Ausgangszustand überführen, indem deren Steuerwerte auf „false“ zurückgesetzt werden. Tabelle 17 zeigt eine mögliche Zuordnung der benötigten Variable und Parameter dieses Skills.

Tabelle 17: Grundeinstellung

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
bool	Tag_1 Tag_3 Tag_4 Tag_5	Keine	Tag_1 Tag_3 Tag_4 Tag_5

Übersicht Basic Skills mit Erweiterung

Die zuvor beschriebenen Basic Skills mit Aktoren sind so konzipiert, dass jeweils ein Skill für das Ausfahren oder Antreiben sowie ein weiterer Skill für das Einfahren oder Anhalten vorgesehen ist. Ergänzend ist ein Skill mit der Bezeichnung „Band_steuern“ vorgesehen, der die Steuerung des Förderbands über einen Eingabeparameter realisiert. Wird der Wert 0 an die Eingabevariable „EingabeFürBandmotor“ übergeben, so wird die interne Variable „Band“ auf „false“ gesetzt. In der Folge wird die Steuervariable Tag_1 ebenfalls auf „false“ gesetzt, wodurch das Förderband anhält. Wird der Wert 1 übergeben, so wird die Variable „Band“ auf „true“ gesetzt. Dadurch wird auch Tag_1 auf „true“ gesetzt, was zum Start des Förderbands führt. Die Tabelle 18 und Tabelle 19 veranschaulichen den Aufbau des Skills „Band_steuern“ anhand der verwendeten Eingabeparameter sowie der internen Verarbeitung mit den zugehörigen Ausgabeparametern.

1. Band_steuern

Tabelle 18: Band_steuern - Eingabeparameter

Datentyp	Variable	Eingabewert	Bedeutung
int	EingabeFürBandmotor	0	Förderband

			anhalten
int	EingabeFürBandmotor	1	Förderband starten

Tabelle 19: Band_steuern - Interne Verarbeitung und Ausgabeparameter

Datentyp	Variable	Bedingung	Ausgabeparameter	Ergebnis
bool	Band	Band = 0	Tag_1	false
bool	Band	Band = 1	Tag_1	true

Analog zum Skill „Band_steuern“, der über einen Eingabeparameter die Steuerung des Förderbands ermöglicht, sollen auch für die weiteren Aktoren entsprechende parametergesteuerte Skills vorgesehen werden. Diese sind als „Weiche_1_ansteuern“, „Weiche_2_ansteuern“ und „Stopper_ansteuern“ bezeichnet. Die Tabellen: Tabelle 20, Tabelle 21, Tabelle 22, Tabelle 23, Tabelle 24, und Tabelle 25 zeigen für die Weichen und den Stopper die jeweiligen Skills im gleichen Format wie Tabelle 18 und Tabelle 19 für den Skill „Band_steuern“, bestehend aus den jeweiligen Eingabeparametern, der internen Verarbeitung, den Ausgaben und den zugehörigen Variablen.

2. Weiche_1_ansteuern

Tabelle 20: Weiche_1_ansteuern- Eingabeparameter

Datentyp	Variable	Eingabeparameter	Bedeutung
int	EingabeFür-Weiche1	0	Weiche 1 einfahren
int	EingabeFür-Weiche1	1	Weiche 2 ausfahren

Tabelle 21: Weiche_1_ansteuern - Interne Verarbeitung und Ausgabeparameter

Datentyp	Variable	Bedingung	Ausgabeparameter	Ergebnis
bool	Weiche1	Weiche1 = 0	Tag_3	false
bool	Weiche1	Weiche1 = 1	Tag_3	true

3. Weiche_2_ansteuern

Tabelle 22: Weiche_2_ansteuern- Eingabeparameter

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
int	EingabeFür-Weiche2	0	Tag_4
int	EingabeFür-Weiche2	1	Tag_4

Tabelle 23: Weiche_2_ansteuern - Interne Verarbeitung und Ausgabeparameter

Datentyp	Variable	Bedingung	Ausgabeparameter	Ergebnis
bool	Weiche2	Weiche2 = 0	Tag_4	false
bool	Weiche2	Weiche2 = 1	Tag_4	true

4. Stopper_ansteuern

Tabelle 24:Stopper_ansteuern - Eingabeparameter

Datentyp	Variable	Eingabeparameter	Ausgabeparameter
int	EingabeFür-Stopper	0	Stopper
int	EingabeFür-Stopper	1	Stopper

Tabelle 25: Stopper_ansteuern - Interne Verarbeitung und Ausgabeparameter

Datentyp	Variable	Bedingung	Ausgabeparameter	Ergebnis
bool	Stopper	Stopper = 0	Tag_5	false
bool	Stopper	Stopper = 1	Tag_5	true

Durch diese einheitliche und parameterbasierte Struktur wird die Anzahl der benötigten Skills im Teilmodell reduziert. Gleichzeitig wird die Übersichtlichkeit erhöht und die Wiederverwendbarkeit innerhalb der AAS verbessert. Darüber hinaus erlaubt dieser Aufbau eine flexible Erweiterung, beispielsweise um zeitgesteuerte Funktionen oder zusätzliche Steuerparameter zu integrieren. Auch sicherheitsrelevante Abfragen, Statusrückmeldungen oder komplexere Ablaufsteuerungen könnten in zukünftigen Ausbaustufen eingebunden werden, ohne die Grundstruktur der Skills anpassen zu müssen. Insgesamt unterstützt der parametergesteuerte Ansatz die Modularität, Skalierbarkeit und zukünftige Erweiterbarkeit der AAS.

3.6 Ablauf der Skill-Ausführung

Die Abbildung 12 zeigt den geplanten Ablauf der Skill Ausführung von der Interaktion des Users bis zur Rückmeldung durch die physische Sortierstation. Das Diagramm ist so aufgebaut, dass die Zeitachse von oben nach unten verläuft. Horizontale Nachrichten symbolisieren zeitlich geordnete Aktionen zwischen den beteiligten Komponenten, während senkrechte Linien deren zeitliche Existenz im dargestellten Ablauf visualisieren.

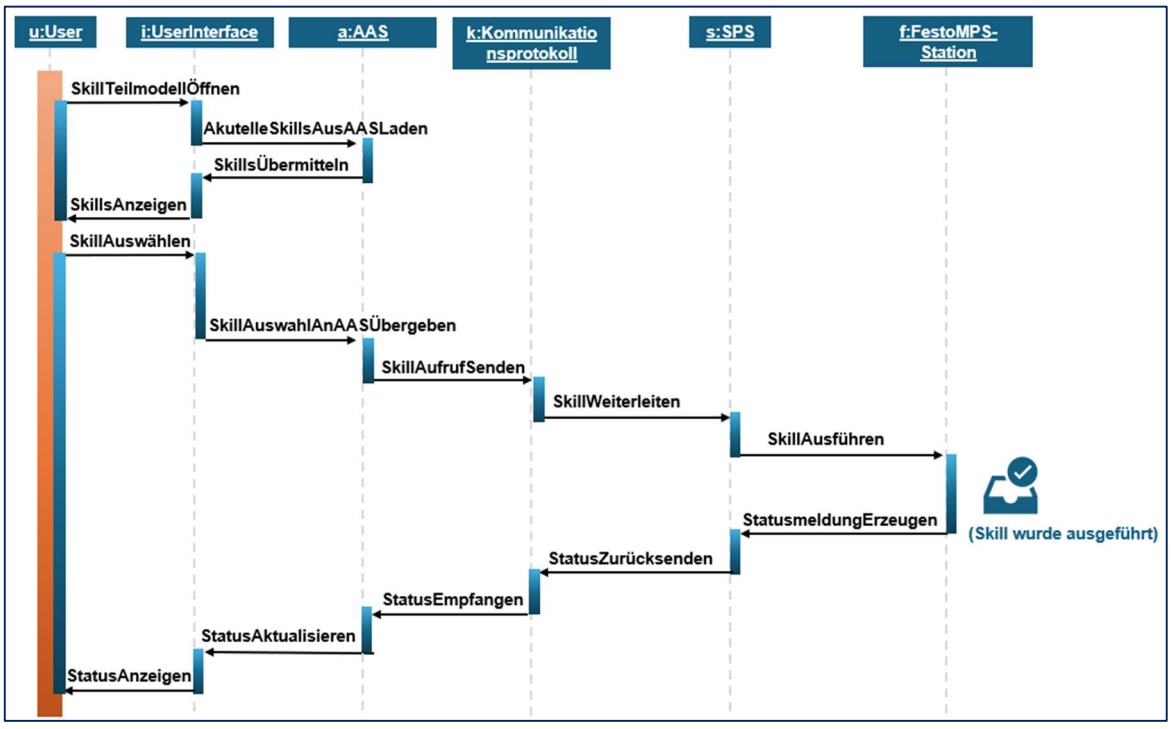


Abbildung 12: Ablauf der Skill Ausführung

Der Ablauf beginnt mit der Interaktion des Users über die UI. Dort wird ein skill-basiertes Teilmodell der AAS geöffnet und ein Skill ausgewählt. Diese Auswahl wird an das Backend übergeben, das für die Steuerlogik zuständig ist. Das Backend leitet die gewählte Funktion über eine standardisierte Schnittstelle an die zugrunde liegende Steuerungseinheit weiter. Diese interpretiert den Aufruf und führt die gewünschte Aktion an der physischen Station

aus, etwa das Bewegen eines Aktors oder das Starten eines Bandes. Nach der Ausführung erfolgt eine Rückmeldung an das Backend, welches den aktuellen Status aktualisiert. Die UI stellt diesen Status visuell dar, sodass der User den Erfolg der Ausführung nachvollziehen kann. Die AAS übernimmt dabei die zentrale Rolle als digitale Schnittstelle zwischen User, Anwendung und realem System. Die gesamte Kommunikation von der Auswahl über die Ausführung bis zur Rückmeldung erfolgt über strukturierte Teilmodelle. Dadurch wird eine klare Trennung zwischen Benutzerinteraktion, Steuerlogik und physischer Umsetzung gewährleistet.

3.7 Lösungsdarstellung des Konzepts

In diesem Abschnitt wird das entwickelte Gesamtkonzept zur Umsetzung der AAS zusammengefasst. Aufbauend auf den zuvor definierten Anforderungen (vgl. Kapitel 3.2 Allgemeine Anforderungen) wird eine modulare Struktur konzipiert, die sowohl statische als auch dynamisch generierbare Teilmodelle umfasst. Die Lösung basiert auf einer klaren Trennung zwischen der logischen Struktur der AAS, den standardisierten Kommunikationsschnittstellen und der Benutzerinteraktion über die UI. Die AAS enthält sowohl Teilmodelle mit ausführbaren Skills als auch rein informationsbasierte Teilmodelle, etwa zur Dokumentation der Sortierstation. Skills werden nach einem einheitlichen Modellierungsprinzip dargestellt. Jedes Teilmodell enthält mindestens ein Operation-Element zur Ausführung des Befehls sowie ein Property-Element zur Rückmeldung des Ausführungsstatus. Die Kommunikation mit der Steuerung der Sortierstation erfolgt über ein Kommunikationsprotokoll. Jeder Skill-Befehl wird über eine eindeutige Kennung adressiert und methodisch aufgerufen (vgl. **A4**, **A5**). Die Rückmeldung erfolgt in Echtzeit an die AAS und wird in der GUI visualisiert. Neben fest im Code hinterlegten Teilmodellen unterstützt die AAS auch dynamisch erweiterbare Skills. Die Skill-Parameter werden in JSON-Dateien abgelegt, beim Start der Anwendung eingelesen und als Teilmodelle interpretiert. Diese dynamische Erweiterung verbessert die Skalierbarkeit und Wiederverwendbarkeit der Lösung (vgl. **A1**). Die Parametrierung einzelner Skills, zum Beispiel durch Eingabevariablen, ermöglicht eine flexible Nutzung und reduziert gleichzeitig die Gesamtanzahl der benötigten Skill-Definitionen. Dadurch wird sowohl die interne Struktur als auch die Bedienung über die UI vereinfacht (vgl. **A3**). Abbildung 13 zeigt das übergeordnete Lösungskonzept mit allen beteiligten Komponenten und deren Interaktionen im Überblick.

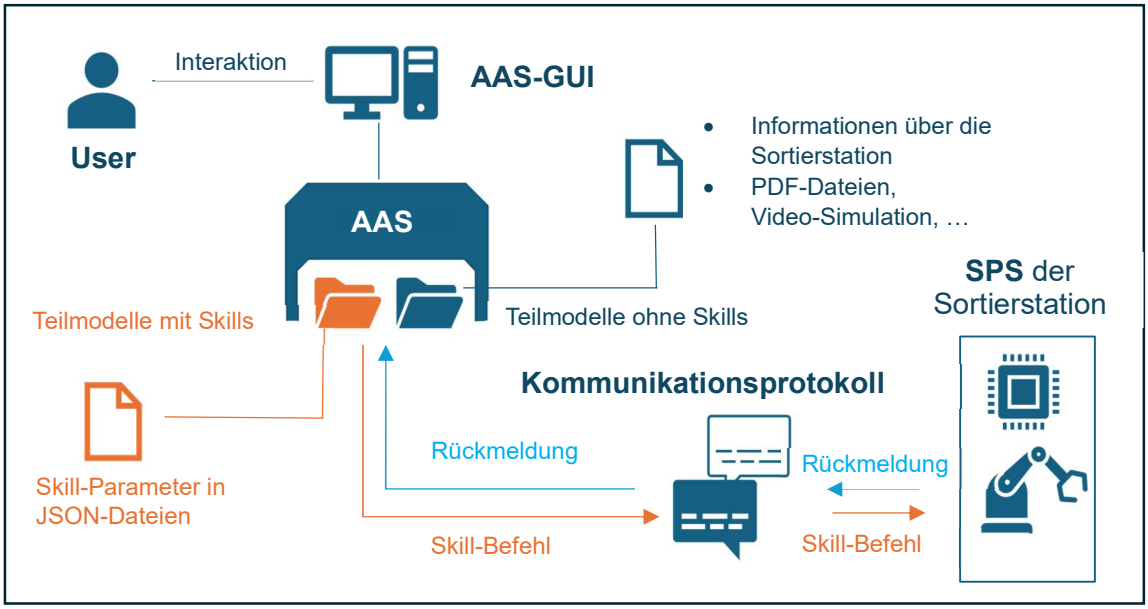


Abbildung 13: Lösungskonzept

4. Implementierung und Integration der AAS für die Sortierstation

Während Kapitel 3 die konzeptionellen Grundlagen der AAS behandelte, wird in diesem Kapitel die technische Umsetzung und Integration der AAS für die Sortierstation beschrieben.

4.1 Technische Grundlagen: Komponenten und Variablenstruktur

Übersicht der eingesetzten Software- und Hardwarekomponenten

Die Tabelle 26 zeigt, welche Software und Hardware in den einzelnen Umsetzungsschritten der AAS verwendet werden. Jede Komponente wird dabei entsprechend ihrer Funktion im Gesamtprozess zugeordnet, angefangen bei der Aktivierung des OPC UA-Servers bis hin zur Ausführung und dem Test der Skills. Dadurch wird ersichtlich, wie alle technischen Bausteine zur Umsetzung der AAS für die Sortierstation zusammenwirken.

Tabelle 26: Übersicht Software- und Hardwarekomponenten

	Software					Hardware				
Aufgabenstellung	**OPC UA Server**	**OPC UA Expert**	**OPC UA-Server Simulator**	**TIA**	**AAS-Backend**	**UI**	**SPS**	**Festo MPS-Station**	**Entwicklungs-PC (Homeoffice)**	**Edge Device (Hochschul-PC)**
OPC UA-Server Aktivierung				X						
Skills-Definition								X		
Skills-Implementierung				X						
Testen der Skills	X	X					X	X		
Entwicklung der AAS	X		X		X	X			X	X
Skill-Ausführung	X					X	X	X		X

Begründung für die Auswahl der Komponenten

Die Tabelle 27 gibt einen Überblick über die im Projekt eingesetzten Komponenten, deren Verwendung in der Entwicklungs- bzw. Laufzeitumgebung sowie die jeweilige Begründung für ihre Auswahl.

Tabelle 27: Überblick über die im Projekt eingesetzten Komponenten

	Entwicklungs-umgebung	**Laufzeit-umgebung**	**Begründung für die Auswahl**
BaSyx.NET	X		Einfache Integration in die .NET-Umgebung und standardkonforme AAS-Entwicklung mit Unterstützung für Teilmodelle, Skill-Ausführung und Industrie 4.0-Kompatibilität.
Microsoft Visual Studio	X		Gute Kompatibilität mit BaSyx.NET, etablierte IDE für C#, einfache Verwaltung von Projekten und Bibliotheken.
Kommunikations-protokoll: OPC UA	X	X	Bereits von der eingesetzten SPS unterstützt, ermöglicht standardisierten und sicheren Datenaustausch ohne zusätzliche Middleware.
TIA-Portal	X		Notwendig zur Konfiguration und Programmierung der Steuerung der Sortierstation; ermöglicht Definition der Methoden und Variablen.

OPC UA-Serversimulator von Integration Objects	X		Ideal für die Entwicklung im Homeoffice, da ein lauffähiger OPC UA-Server ohne physische SPS bereitgestellt wird.

Begründung für die Strukturierung der SPS-Variablen

Die gewählte Strukturierung der SPS-Variablen basiert auf technischen Anforderungen, die sich aus der Kommunikation mit der AAS ergeben. Insbesondere die OPC UA-Anbindung erfordert eine eindeutige Zuordnung von Variablen zu Methoden und Parametern. Zur Reduzierung der Komplexität und zur Begrenzung der veröffentlichten Knoten wird auf die erneute Deklaration innerhalb von Bausteinen verzichtet. Stattdessen greift die Implementierung direkt auf die global definierten Variablen in der PLC-Tabelle zurück. Dies berücksichtigt die Beschränkung der OPC UA-Knotenanzahl in der Siemens-Umgebung und erhöht die Übersichtlichkeit. Eingabeparameter für steuerbare Funktionen werden lokal innerhalb der Funktionsbausteine im Abschnitt „UAMethod_InParameters“ verwaltet. Diese Trennung sorgt dafür, dass die Parameter im AAS-Backend eindeutig identifiziert und über die passende Node-ID adressiert werden können. Nur so kann ein stabiler und verlässlicher Datenaustausch mit der SPS gewährleistet werden. Die gewählte Vorgehensweise trägt somit direkt zur Robustheit der OPC UA-Kommunikation bei und bildet die technische Grundlage für eine saubere Kopplung zwischen AAS und Steuerung.

4.2 Systemarchitektur zur Ansteuerung des Assets mittels AAS

Die Abbildung 14 zeigt den technischen Aufbau der in dieser Arbeit entwickelten AAS und stellt die wichtigsten Systemkomponenten sowie deren Kommunikationsbeziehungen dar. Dabei werden die einzelnen Beziehungen nummeriert und chronologisch erläutert.

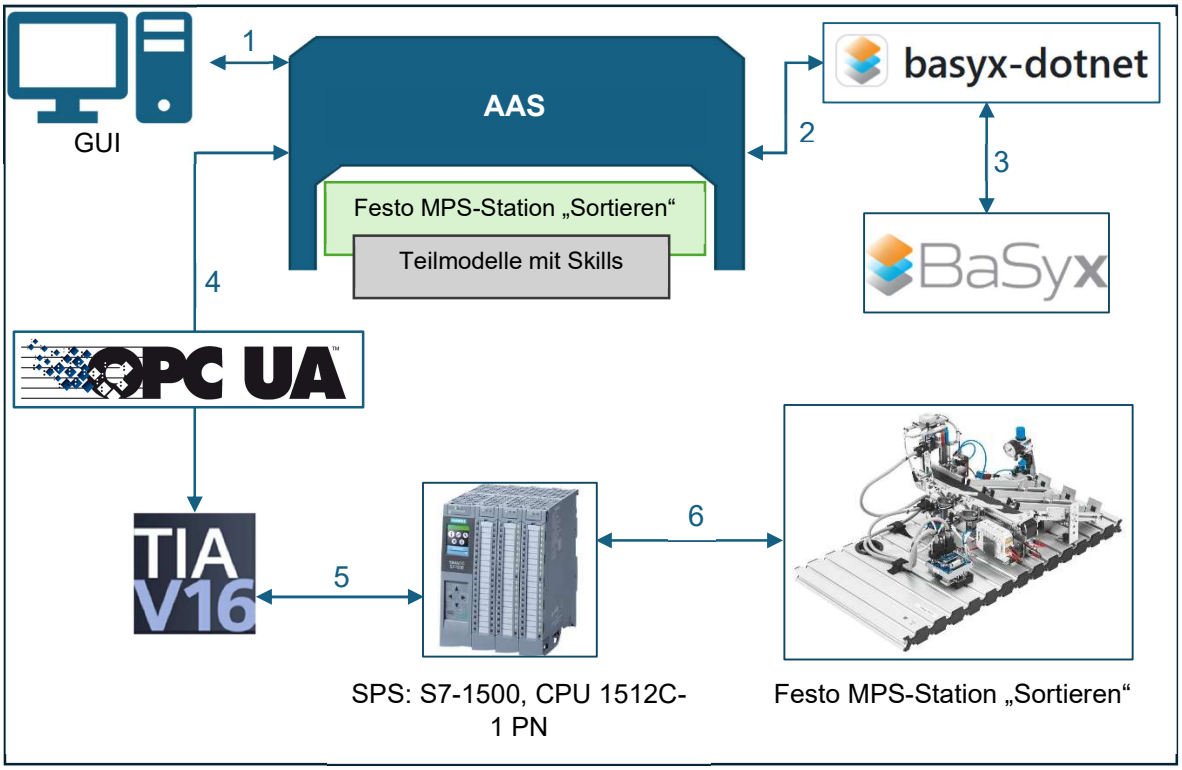


Abbildung 14: Systemarchitektur zur Ansteuerung des Assets mittels AAS(23,28,31,36–38)

- **Beziehung 1:** Das GUI ist als Weboberfläche realisiert und kommuniziert mit der AAS über REST-API. Diese Oberfläche wird durch das BaSyx.NET Framework bereitgestellt und erlaubt die Anzeige von Teilmodellen, den Abruf von Statusinformationen sowie die gezielte Ausführung von Skills. Der Zugriff erfolgt über einen Webbrowser, ohne dass zusätzliche Software erforderlich ist.
- **Beziehung 2:** Die AAS wird mit dem BaSyx.NET Framework entwickelt, das speziell für die Umsetzung von AAS in einer .NET-Umgebung konzipiert ist.
- **Beziehung 3:** Das verwendete .NET-Framework ist vollständig kompatibel mit den Spezifikationen des Eclipse BaSyx Ökosystems. Es orientiert sich an den standardisierten Datenmodellen der Plattform Industrie 4.0 und erlaubt somit eine interoperable und zukunftssichere Umsetzung der AAS.
- **Beziehung 4:** Zur Maschinenanbindung wird OPC UA eingesetzt, wobei TIA V16 als Middleware zum Einsatz kommt und über den integrierten OPC UA-Server Funktionen als aufrufbare Methoden zur Verfügung stellt. Der OPC UA-Server wird inklusive der Methoden im TIA-Portal angelegt und gestartet. Die AAS fungiert in dieser Kommunikation als OPC UA-Client, der Methoden aufruft, um das Asset zu steuern. Jeder Skill wird dabei durch eine Methode repräsentiert und ist einer eindeutigen Node-ID zugeordnet. Der Client übernimmt sowohl die Initialisierung der Verbindung als auch die Ausführung und Rückmeldung der Methode.

- **Beziehung 5:** Die SPS (S7-1500, CPU 1512C-1 PN), projektiert im TIA-Portal, verknüpft die Methoden direkt mit den Ein- und Ausgängen der SPS. Eingehende Aufrufe aus dem TIA V16 werden als Maschinenbefehle interpretiert und umgesetzt.
- **Beziehung 6:** Die SPS aktiviert die jeweiligen Aktoren der MPS-Station, sobald ein Befehl eingeht. Gleichzeitig werden Rückmeldungen von Sensoren verarbeitet und als Statusinformationen über die Beziehungen 4 und 5 OPC UA bereitgestellt. So entsteht ein vollständiger Steuerkreislauf von der Benutzeraktion bis zur physischen Ausführung und deren Rückmeldung in der AAS.

Diese Architektur erfüllt, die in Kapitel 3 definierten, allgemeinen Anforderungen an Modularität, Interoperabilität und Erweiterbarkeit. Sie gewährleistet eine standardisierte Kommunikation zwischen allen beteiligten Komponenten und ermöglicht eine flexible Erweiterung. Durch den modularen Aufbau lässt sich die Lösung nahtlos in bestehende Automatisierungssysteme integrieren und bildet gleichzeitig die Grundlage für die prototypische Umsetzung weiterer digitaler Zwillinge im Automatisierungslabor.

4.3 Vorbereitende Maßnahmen zur Umsetzung der AAS

Die Erstellung der AAS für die Sortierstation basiert auf einer systematischen Schritt-für-Schritt-Vorgehensweise, die im separaten Dokument „Anleitung zur schrittweisen Erstellung einer AAS mit BaSyx.NET“ detailliert beschrieben ist. Dieses Begleitdokument und die AAS-Anwendung inkl. ihrer Programmcodes wird unabhängig von dieser Arbeit zur Verfügung gestellt und dient als technische Anleitung zur praktischen Umsetzung. Zu Beginn wird das GitHub-Repository des BaSyx.NET Frameworks lokal geklont. Hierzu wird die Webadresse des Repositories in Visual Studio eingefügt. Nach dem erfolgreichen Klonvorgang wird Visual Studio kurzzeitig geschlossen, um die mitgelieferten Batch-Dateien auszuführen. Dabei handelt es sich um die Dateien „Setup_Basyx.bat“ und „Build_Basyx.bat“, welche die Projektumgebung einrichten und alle erforderlichen Abhängigkeiten installieren. Anschließend kann die Projektdatei „BaSyx.sln“ in Visual Studio geöffnet werden. Damit steht eine funktionsfähige Entwicklungsumgebung zur Verfügung, mit der die eigene AAS modelliert und umgesetzt werden kann. Im ersten Schritt wird im Projektmappen-Explorer ein neuer Ordner angelegt, in dem ein neues Projekt auf Basis der Vorlage „ASP.NET Core Web API“ erstellt wird. Eine beispielhafte Darstellung dieses Schritts ist in Abbildung 15 ersichtlich.

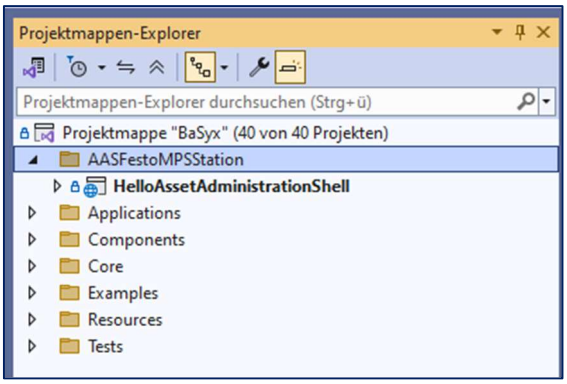


Abbildung 15: Projektmappen-Explorer

Als Grundlage dient das Beispielprojekt „HelloAssetAdministrationShell“, das eine funktionierende Referenzstruktur für eine AAS bietet. Es enthält bereits alle grundlegenden BaSyx.NET-Komponenten, die zur Modellierung und Bereitstellung einer AAS benötigt werden. In der nachfolgenden Abbildung sind die wichtigsten Elemente hervorgehoben, die in die eigene Projektstruktur der AAS übernommen werden müssen. Siehe .

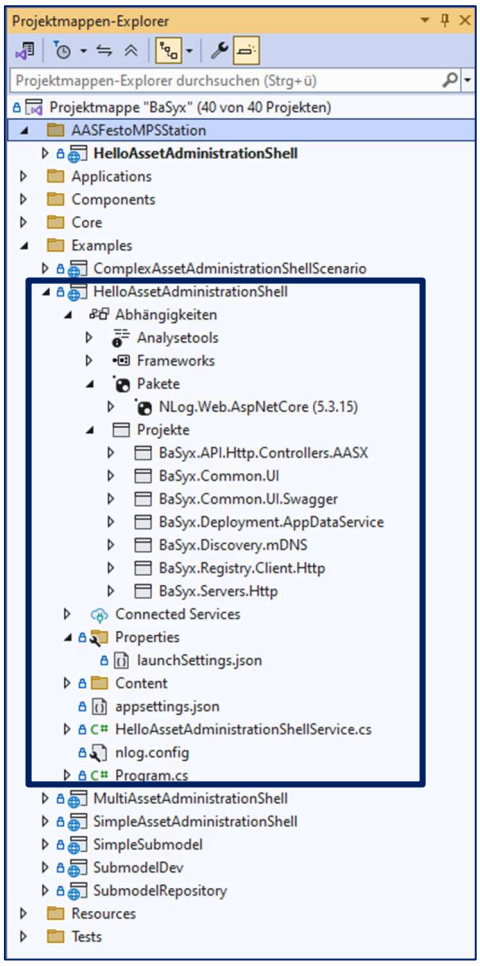


Abbildung 16: „HelloAssetAdministrationShell“ des BaSyx.NET Frameworks

Zu den relevanten Bestandteilen gehören unter anderem die eingebundenen Bibliotheken, zentrale Konfigurationsdateien sowie spezifische Serviceklassen wie die Datei „HelloAssetAdministrationShellService.cs“, die für die Verwaltung der AAS verantwortlich ist. Diese Elemente bilden die technische Basis für die eigene Implementierung und werden vollständig übernommen. Nicht benötigte Beispielinhalte aus der genannten Datei, wie vordefinierte Teilmodelle oder Beispiel-Properties, werden entfernt. Vor der Anpassung sollte zudem das eigene Projekt von automatisch generierten Inhalten bereinigt werden. Dazu zählen beispielsweise Standard-Controller, Beispielklassen oder automatisch erzeugte Konfigurationsdateien eines neuen Web API Projekts. Dies stellt sicher, dass keine unerwünschten Konflikte mit den BaSyx.NET-Komponenten entstehen. Im Anschluss daran lassen sich eigene Teilmodelle, Eigenschaften und Operationen gezielt einfügen und an die Anforderungen der realen Station anpassen.

4.4 Projektstruktur der HelloAssetAdministrationShell

Im Folgenden werden die zentralen Komponenten des Beispielprojekts erläutert. Jede Datei wird hinsichtlich ihrer Funktion, Einbindung in das Gesamtsystem und Relevanz für die Umsetzung einer eigenen AAS beschrieben.

Program.cs

Die Datei Program.cs enthält den Einstiegspunkt der Anwendung. Sie lädt die Konfigurationsdateien, startet den HTTP-Server für die AAS, bindet die bereitgestellten Services ein, aktiviert die UI sowie die Swagger-Dokumentation und initiiert die Discovery über mDNS. Diese Datei ist für das Hochfahren und das Zusammenschalten aller Systemkomponenten zuständig.

HelloAssetAdministrationShellService.cs

Diese Klasse implementiert den zentralen Service für die AAS. Sie definiert die Struktur der AAS, legt die Teilmodelle an und beschreibt deren Inhalte, wie Eigenschaften, Dateien und Operationen. Zudem werden hier sogenannte Service Provider für Teilmodelle eingerichtet, die den Zugriff auf die Teilmodelle über die HTTP-Schnittstelle ermöglichen. Der Service verknüpft die Datenstruktur mit der Zugriffsfunktionalität und stellt somit den Kern der Anwendung dar.

appsettings.json / ServerSettings.xml

Diese Dateien enthalten die Serverkonfiguration wie z. B. Ports, Pfade zu statischen Inhalten, Standardrouten und UI-Einstellungen. Sie werden beim Start geladen und bestimmen das Verhalten des Servers. So wird beispielsweise definiert, auf welcher URL die

Anwendung läuft und ob die UI automatisch geladen wird. Die Konfiguration erlaubt eine flexible Anpassung ohne Änderungen am Quellcode.

NLog.config

Die Datei NLog.config definiert das Logging-Verhalten der Anwendung. Hier wird festgelegt, welche Informationen geloggt werden und wohin die Ausgaben geschrieben werden sollen, z. B. in die Konsole. Damit wird die Nachvollziehbarkeit der Serveraktivitäten sichergestellt.

launchSettings.json

Diese Datei legt fest, mit welchen Startparametern die Anwendung im Entwicklungsumfeld ausgeführt wird. Sie enthält unter anderem die URL, auf die der Browser beim Debug-Start zugreift. Diese Datei dient hauptsächlich der Arbeit mit Visual Studio und hat keinen Einfluss auf den produktiven Betrieb.

Teilmodelle im Beispielprojekt

Im Beispielprojekt werden zwei Teilmodelle erstellt: das "HelloSubmodel" und das "AssetIdentification"-Teilmodell. Das „HelloSubmodel“ enthält ein Property, eine Datei und eine Operation, die beispielhaft die Kernfunktionalitäten einer AAS zeigen. Das AssetIdentification-Teilmodell stellt Eigenschaften wie Produkttyp, Auftragsnummer und Seriennummer bereit und zeigt, wie standardisierte IRDI-Kennungen und ConceptDescriptions eingebunden werden.

UI

Die BaSyx.NET UI ermöglicht eine interaktive Darstellung und Steuerung der AAS im Browser. Sie visualisiert die Struktur der AAS, erlaubt das Lesen und Schreiben von Properties sowie das Ausführen von Operationen. Die UI wird über Program.cs eingebunden und ist unter der Standardroute "/ui" erreichbar.

Swagger UI

Die Swagger UI stellt eine automatisch generierte Dokumentation der REST-Schnittstelle bereit. Sie listet alle verfügbaren Endpunkte auf, beschreibt deren Parameter und Rückgabewerte und erlaubt das Testen von Aufrufen direkt im Browser. Sie dient zur technischen Überprüfung und Dokumentation der HTTP-API.

mDNS-Discovery

Die Anwendung nutzt mDNS, um sich im lokalen Netzwerk bekannt zu machen. Beim Start wird die AAS als Dienst registriert, beim Herunterfahren automatisch wieder abgemeldet.

Dieser Mechanismus ermöglicht die dynamische Auffindbarkeit der AAS ohne zentrale Registry und vereinfacht die Integration in verteilte Systeme.

4.5 BaSyx.NET Kurzbezeichnungen und Identifikatoren für die AAS

Bevor die inhaltliche Erweiterung der AAS erfolgt, wird zunächst dargestellt, wie die Kurzbezeichnungen und die vollständigen Identifikatoren für die AAS, das zugehörige Asset sowie alle Teilmodelle in Anlehnung an die Struktur von BaSyx.NET vergeben werden. Siehe Tabelle 28 und Tabelle 29.

Tabelle 28: idShort und Id von AAS und Asset

	idShort	Id
AAS	AASFMPS	urn:org.eclipse.basyx:shells:AASFMPS:1.0.0
Asset	AASFMPS	urn:org.eclipse.basyx:assets:AASFMPS:1.0.0

Tabelle 29: idShort und Id der Teilmodelle

	idShort	Id
1	Asset Identifikation	urn:org.eclipse.basyx:submodels:1 Asset Identifikation:1.0.0
2	Informationen	urn:org.eclipse.basyx:submodels:2 Informationen:1.0.0
3	Dokumente und Medien	urn:org.eclipse.basyx:submodels:3 Dokumente und Medien:1.0.0
4	Technische Daten	urn:org.eclipse.basyx:submodels:4 Technische Daten:1.0.0
5	Basic Skills aus JSON	urn:org.eclipse.basyx:submodels:5 Basic Skills aus JSON:1.0.0
6	Basic Skills mit zusätzlichen Logiken	urn:org.eclipse.basyx:submodels:6 Basic Skills mit zusätzlichen Logiken:1.0.0
7	Kombinierter Skill aus JSON	urn:org.eclipse.basyx:submodels:7 Kombinierte Skills aus JSON:1.0.0
8	Kombinierte Skills	urn:org.eclipse.basyx:submodels:8 Kombinierte Skills:1.0.0
9	Statisch codiertes Skill-Submodelbeispiel	urn:org.eclipse.basyx:submodels:9 Statisch codiertes Skill-Submodelbeispiel:1.0.0
10	Betriebsdaten	urn:org.eclipse.basyx:submodels:10 Betriebsdaten:1.0.0

11	Basic Skills	urn:org.eclipse.basyx:submodels:11 BasicSkills:1.0.0

4.6 Erweiterung der Projektstruktur um nicht funktionale Aspekte

In diesem Abschnitt werden die vorgenommenen Anpassungen an der UI beschrieben, die eine klarere Zuordnung zur Sortierstation ermöglichen und die Bedienbarkeit verbessern. Darüber hinaus werden die Teilmodelle 1 bis 4 erläutert, welche ausschließlich beschreibende Informationen enthalten und unabhängig von steuerungsrelevanten Skills umgesetzt werden. Zu diesen Erweiterungen zählen unter anderem:

1. **Anpassung der UI:** In der Datei appsettings.json wird der Titel der UI auf „Asset Administration Shell UI Festo MPS-Station“ angepasst. Diese Änderung ermöglicht eine eindeutige visuelle Zuordnung der UI zum digitalen Zwilling der realen Anlage. Siehe Abbildung 17.

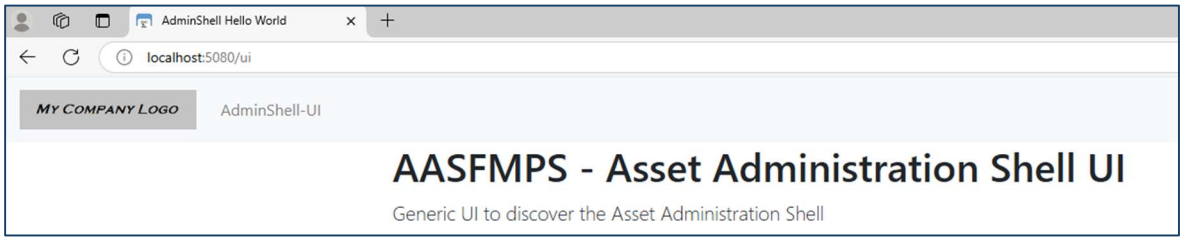


Abbildung 17: Übersicht UI-Titel

2. **Anpassung der AAS- und Asset-Informationen:** In der „HelloAssetAdministrationShellService.cs“ wird ein spezifischer Bezeichner für die AAS eingefügt. Das Framework generiert daraufhin automatisch auch den zugehörigen Bezeichner für das Asset. Zusätzlich wird eine sprachspezifische Beschreibung in deutscher Sprache ergänzt. Siehe Abbildung 18.

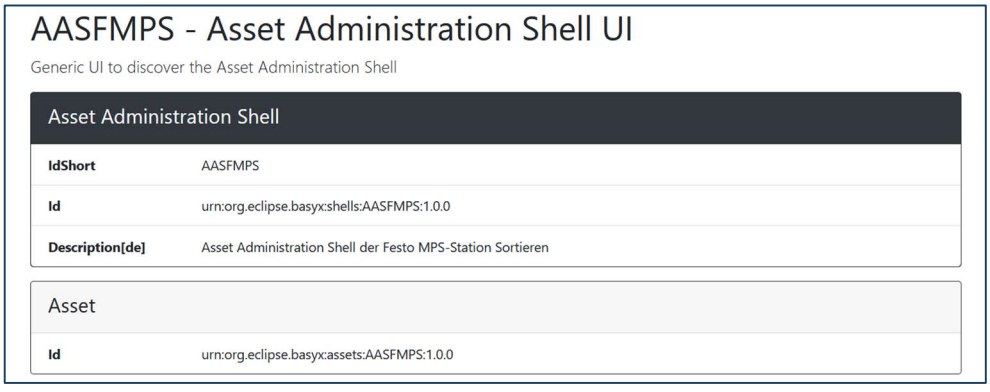


Abbildung 18: UI - AAS und Asset

3. **Einbindung statischer Teilmodelle:** In der Klasse „HelloAssetAdministrationShellService.cs“ werden insgesamt vier statische Teilmodelle direkt im Quellcode eingebunden.

Jedes dieser Teilmodelle wird über eine eigene Instanz vom Typ „SubmodelServiceProvider“ in die AAS integriert. Zu diesen statischen Teilmodellen gehören:

1. **Teilmodell Asset Identifikation:** Dieses Teilmodell bildet grundlegende Stammdaten der Station ab. Es enthält Eigenschaften wie Typbezeichnung, Bestellnummer und Seriennummer, die gezielt auf die Sortierstation angepasst wurden. Ergänzend werden Herstellerinformationen und ein Produktlink eingebunden, die der reinen Anzeige dienen. Eine Bearbeitung dieser Daten ist nicht vorgesehen, da sie ausschließlich der Orientierung und ergänzenden Beschreibung für die Benutzerinnen und Benutzer dienen.
2. **Teilmodell Informationen:** Das Teilmodell dient der strukturierten Bereitstellung allgemeiner Informationen zur Festo MPS-Station. Es umfasst mehrere Eigenschaften, die zentrale Hinweise zur Funktion, zur Inbetriebnahme und zum Aufbau der Station enthalten. Ein besonderes Element dieses Teilmodells ist ein integriertes Bild der Station. Dieses Bild wird nicht über den Content-Ordner geladen, sondern über einen festen Pfad direkt im Quellcode eingebunden.
3. **Teilmodell Dokumente und Medien:** In diesem Teilmodell werden verschiedene Dateien wie PDF-Dokumente und Videos eingebunden, die im Projektverzeichnis abgelegt sind. Darunter befinden sich beispielsweise eine Betriebsanleitung, Schaltpläne, eine Übersicht zu den SPS-Programmen sowie ein Erklärvideo zum Aufbau der Station. Die Dateien sind als separate Elemente innerhalb des Teilmodells referenziert und stehen über die Benutzeroberfläche direkt zur Ansicht zur Verfügung. Für PDF-Dateien wird eine integrierte Anzeige mit Scroll- und Zoom-Funktion angeboten, sodass die Inhalte ohne vorherigen Download betrachtet werden können.
4. **Teilmodell Technische Daten:** Dieses Teilmodell beinhaltet technische Parameter der Station, die für Aufbau, Betrieb und Integration relevant sind. Es werden unter anderem Angaben zur Versorgungsspannung, zu den Abmessungen, zu vorhandenen Schnittstellen sowie zum verwendeten Bussystem bereitgestellt. Die Eigenschaften sind als einfache Datenstrukturen umgesetzt und liegen in Form von Zeichenketten oder numerischen Werten vor. Die Erstellung des Teilmodells erfolgt direkt im Code durch die manuelle Zusammenstellung der jeweiligen Elemente.

- **Einbindung lokaler Dateien:** Im Projektverzeichnis wird der vom Framework bereits implementierte Content-Ordner verwendet, um lokale Dateien wie Anleitungen, Bilder oder Datenblätter der Sortierstation strukturiert zu hinterlegen. Dieser Ordner wird über die Programmlogik in Program.cs beim Start automatisch als Datenquelle angebunden. Die eingebundenen Dateien stehen über das Teilmodell Dokumente und Medien direkt in der UI zur Verfügung und können dort eingesehen werden.

4.7 Erweiterung der Projektstruktur um steuerbare Skills

Überblick zum Entwicklungsprozess der Skill-Integration

Die Abbildung 19 bietet einen schematischen Überblick über die zentralen Schritte zur Integration steuerbarer Skills in die AAS der Sortierstation. Gezeigt werden unter anderem die Definition der Methoden im TIA-Portal, die Anbindung über OPC UA sowie die Umsetzung im Backend. Rückkopplungspunkte zur Fehlerbehandlung und Optimierung werden ebenfalls dargestellt.

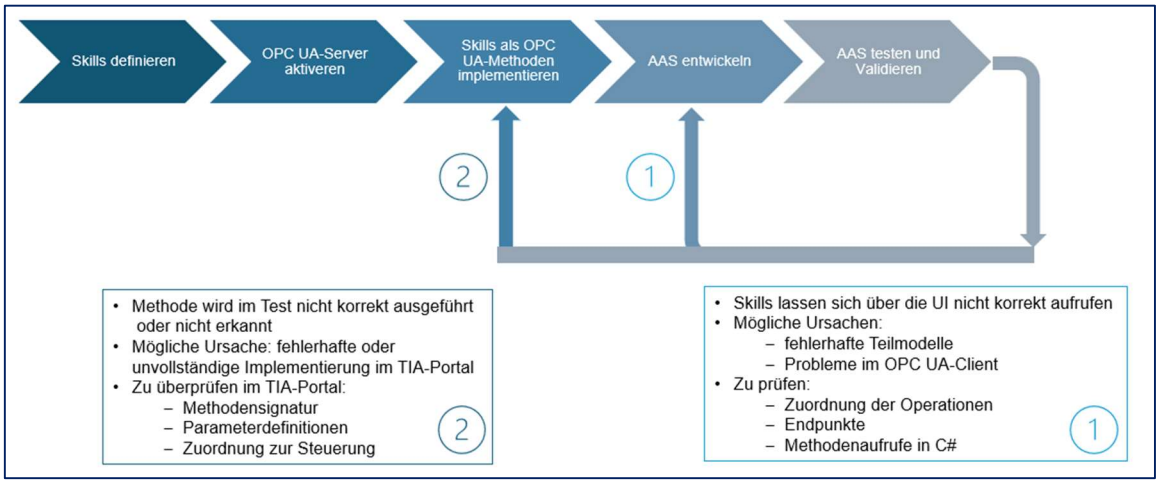


Abbildung 19: Überblick über die zentralen Schritte zur Integration steuerbarer Skills

Auf Basis dieser Übersicht wird im Folgenden die konkrete Umsetzung beschrieben. Ziel ist es, Maschinenfunktionen über definierte OPC UA-Methoden in der AAS verfügbar zu machen und über die UI steuerbar zu gestalten. Die Vorbereitung erfolgt im TIA-Portal, wo der OPC UA-Server aktiviert und Maschinenfunktionen als Methoden mit eindeutigen Node-IDs hinterlegt werden. Diese bilden die Grundlage für die Steuerung durch die AAS. Mithilfe von UA Expert werden die Methoden anschließend getestet und validiert. Zur Integration in die AAS wird in Visual Studio ein eigener OPC UA-Client entwickelt, der die Verbindung zur Steuerung herstellt, Methoden aufruft und Rückmeldungen verarbeitet. Die Skill-Ausführung kann somit direkt über die UI erfolgen. Für die dynamische Erweiterung wird ein Service namens „SkillSubmodelLoader.cs“ implementiert. Dieser liest strukturierte JSON-Dateien mit Skill-Informationen wie Name, Beschreibung und Node-IDs ein und generiert daraus zur Laufzeit Operation-Elemente für die AAS. Neue Skills können dadurch ohne Änderung des Quellcodes hinzugefügt werden. Ein OPC UA-Serversimulator von Integration Objects ermöglichte die Entwicklung und Validierung im Homeoffice. Durch die Kombination aus statischer und dynamischer Umsetzung entstand eine flexible, erweiterbare Struktur zur Steuerung realer Maschinenfunktionen über die AAS im Kontext von Industrie 4.0.

OPC UA-Methoden in Siemens S7-1500 Steuerungen

Das Unternehmen Siemens bietet mit der Firmware-Version V2.5 und ab TIA Portal V15 die Möglichkeit, das Steuerungssystem SIMATIC S7-1500 über seinen integrierten OPC UA-Server für OPC UA-Methoden bereitzustellen. Dadurch können OPC UA-Clients nicht nur auf Variablen der Steuerung zugreifen und diese überschreiben, sondern auch komplexe Funktionsabläufe direkt auslösen. Diese Technologie ermöglicht eine vollständige M2M-Kommunikation auf Basis von OPC UA und erweitert die Steuerungsmöglichkeiten durch eine standardisierte und interoperable Schnittstelle.(39)

Funktionsweise des OPC UA-Methodenaufrufs

OPC UA-Methoden auf einer SIMATIC S7-Steuerung entsprechen funktional den bekannten S7-Funktionsbausteinen. Um die OPC UA-Funktionalität jedoch vollständig bereitzustellen, müssen innerhalb eines solchen Funktionsbausteins zusätzlich zwei spezifische Systemfunktionsbausteine implementiert werden, die von Siemens mit dem TIA-Portal bereitgestellt werden. Diese sorgen für die Verbindung zwischen dem Funktionsbaustein und dem OPC UA-Server auf der Steuerung und ermöglichen den Aufruf durch externe Clients. Im Vergleich zu einem herkömmlichen Funktionsbaustein bleibt die Programmstruktur dabei weitgehend unverändert. Die Systemfunktionsbausteine werden lediglich ergänzend eingebunden, um den externen Methodenaufruf über einen OPC UA-Client zu ermöglichen. Die Abbildung 20 zeigt schematisch den Ablauf eines OPC UA-Methodenaufrufs von einem externen Client auf eine S7-1500-Steuerung, bei dem die aufgerufene Methode intern durch einen S7-Funktionsbaustein mit integrierten Systemfunktionen realisiert wird.(40)

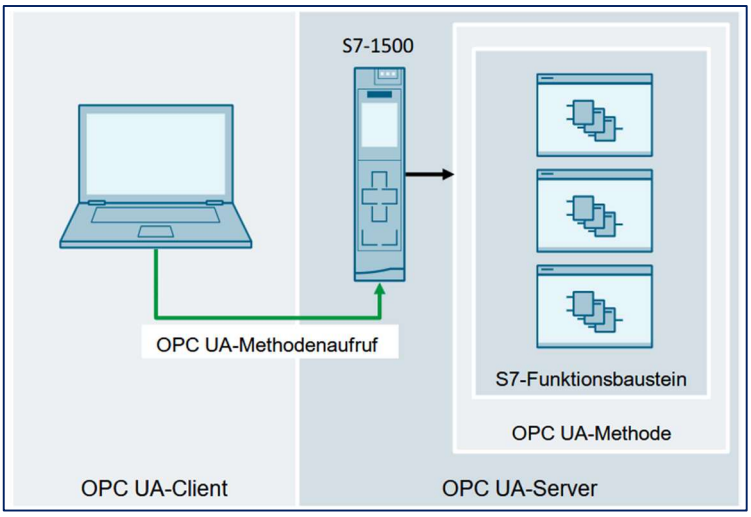


Abbildung 20: Funktionsweise Methodenaufruf(40)

Funktionsablauf

Durch die zusätzliche Implementierung der beiden Systemfunktionsbausteine „OPC_UA_ServerMethodPre“ und „OPC_UA_ServerMethodPost“ wird beim Aufruf einer OPC UA-Methode die Übergabe der Eingabeparameter an die sogenannte Pre-Instanz sichergestellt. Sobald ein OPC UA-Client eine Methode aufruft, signalisiert die Systemfunktion „OPC_UA_ServerMethodPre“ dem Anwenderprogramm, dass eine Anfrage eingegangen ist, und stellt die übermittelten Parameter bereit. Anschließend wird der im Funktionsbaustein hinterlegte Funktionscode ausgeführt. Nach der Verarbeitung übergibt die Systemfunktion „OPC_UA_ServerMethodPost“ die Ausgabewerte sowie einen Status-Code an die Schnittstelle zurück. Zusätzlich wird ein definierter Signalparameter gesetzt, der den erfolgreichen Abschluss der Ausführung kennzeichnet. Alle Daten werden anschließend an den OPC UA-Client übermittelt. Die Abbildung 21 zeigt den Ablauf eines OPC UA-Methodenaufrufs innerhalb eines S7-Funktionsbausteins, bei dem die Systemfunktionen „OPC_UA_ServerMethodPre“ und „OPC_UA_ServerMethodPost“ für die Übergabe der Ein- und Ausgabewerte sowie die Statusrückmeldung verwendet werden.(40)

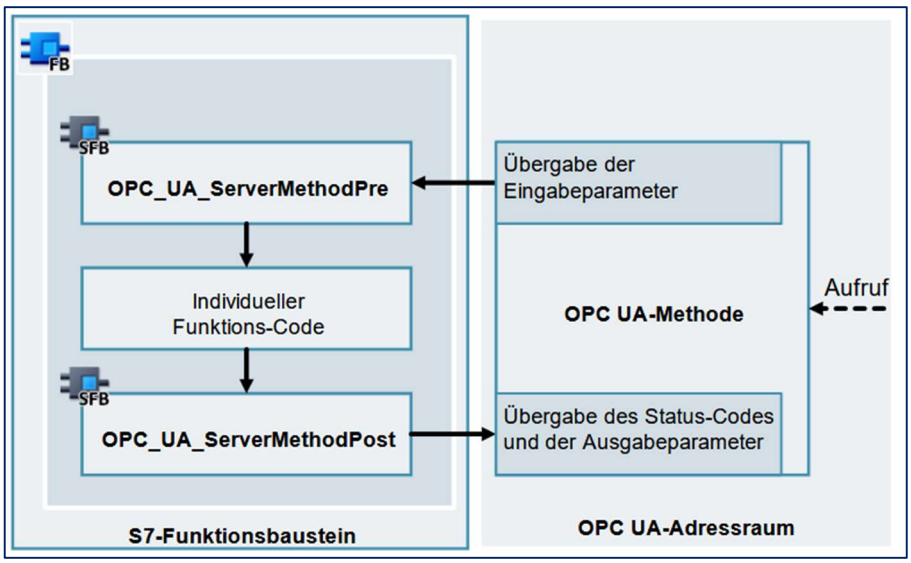


Abbildung 21: Funktionsablauf Methodenaufruf(40)

Einführung: Skills als OPC UA-Methoden implementieren

Schritt 1: OPC UA-Server aktiveren

Bevor OPC UA-Methoden verwendet werden können, muss zunächst die Grundvoraussetzung geschaffen werden, indem der OPC UA-Server auf der Steuerung aktiviert wird. Hierzu wird im TIA-Portal ein neues Projekt erstellt. Im Anschluss muss im Projektbaum unter „Geräte & Netze“ die verwendete Steuerung ausgewählt und hinzugefügt werden. In diesem Fall handelt es sich um die SIMATIC S7-1500 CPU 1215C-1 PN. Mit einem Rechtsklick auf die CPU im Projektbaum öffnet man die Eigenschaften der Steuerung. Im Bereich

OPC UA lässt sich dann durch das Setzen eines Häkchens die Option „OPC UA-Server aktivieren“ aktivieren. Weitere Einstellungen sind an dieser Stelle nicht zwingend erforderlich, es sei denn, es sollen zusätzliche Sicherheitsmechanismen konfiguriert werden. In diesem Fall kann ein Benutzer mit Passwortschutz eingerichtet werden. Bei Verwendung einer Runtime-Lizenz muss zusätzlich eine gültige Lizenz aktiviert sein. Für viele Anwendungen genügt bereits eine Small-Lizenz. Die folgende Abbildung zeigt die Konfigurationsansicht im TIA-Portal, in der die Eigenschaften der CPU aufgerufen werden, um den OPC UA-Server zu aktivieren.

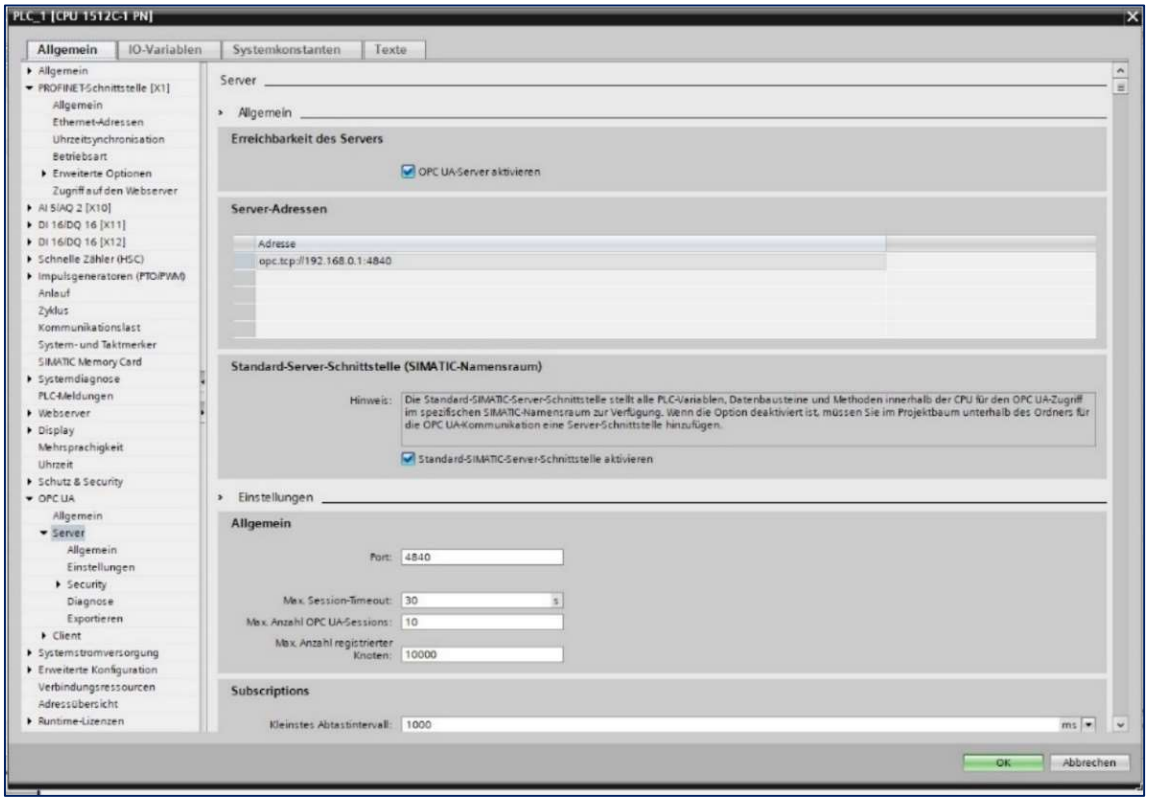


Abbildung 22: OPC UA-Server Aktivierung

Schritt 2: Erstellen einer OPC UA-Methode

Eine OPC UA-Methode wird als Funktionsbaustein (FB) im Projektbaum des TIA-Portals erstellt. Hierfür wird im Projektbaum unter dem Eintrag „Programmbausteine“ die Option „Neuen Baustein hinzufügen“ ausgewählt. Anschließend wird der Baustein als Funktionsbaustein mit der Programmiersprache SCL (Structured Control Language) angelegt.

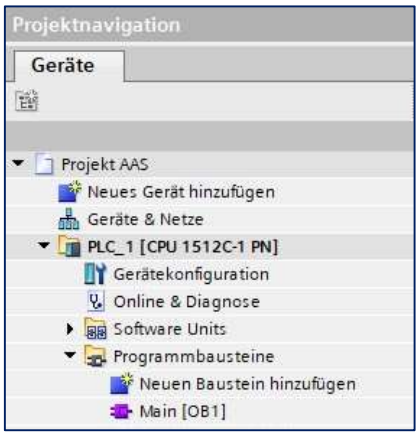


Abbildung 23: Projektnavigation TIA-Portal

Im Funktionsbaustein werden anschließend die erforderlichen Variablen für den Methodenaufruf sowie die internen Variablen für die eigentliche Methodenlogik angelegt. Daraufhin wird die sogenannte Pre-Instanz (OPC_UA_ServerMethodPre) im Funktionsbaustein angelegt. Dadurch werden die vom OPC UA-Client übermittelten Eingabeparameter entgegengenommen und an interne Variablen innerhalb des Bausteins weitergegeben. In der verwendeten Implementierung wird dies durch die Übergabe an die Variable #Readed_UAInParameters realisiert. Gleichzeitig liefern die Ausgänge Done, Busy und Error Informationen über den aktuellen Ausführungsstatus der Pre-Funktion. Der Parameter StatusCodeReturned gibt dabei an, ob die Übergabe der Eingabeparameter erfolgreich war oder ein Fehler aufgetreten ist. Die Abbildung 24 zeigt die Implementierung des OPC UA Pre-Call-Blocks, über den der Methodenaufruf vorbereitet und initialisiert wird.

```
1  // OPC UA Pre-Call (Methode wird vorbereitet)
2  #OPC_UA_ServerMethodPre_Instance(
3                                   Done => #statPreDone,
4                                   Busy => #statPreBusy,
5                                   Error => #statPreError,
6                                   Status => #statPreStatus,
7                                   UAMethod_Called => #statPreCalled,
8                                   UAMethod_InParameters := #UAMethod_InParameters
9  );
```

Abbildung 24: Pre-Call

Als nächstes wird die OPC_UA_ServerMethodPre_Instance angelegt. Diese ist dazu da, dass nach der Ausführung der eigentlichen Methodenlogik eine Rückgabe der Ergebnisse über die Post-Instanz (OPC_UA_ServerMethodPost) erfolgt. Hier werden die zuvor intern berechneten Ausgabewerte über die Variable #ResultFinished an den Parameter UAWrite_InParameters übergeben und anschließend über UAWrite_OutParameters an den OPC UA-Client zurückgespielt. Auch in diesem Schritt signalisieren die Statusvariablen Done, Busy und Error den Zustand der Verarbeitung, während StatusCodeReturned den

Ausgangszustand der Methode kennzeichnet. Über den Parameter MethodFinished wird zudem die erfolgreiche Beendigung der Methode signalisiert. Dieses strukturierte Vorgehen stellt sicher, dass OPC UA-Methoden auf der SPS eindeutig, standardkonform und fehlersicher ausgeführt und überwacht werden können. Die Abbildung 25 zeigt den Post-Call-Block, der nach dem erfolgreichen Abschluss einer OPC UA-Methode aktiviert wird und die Methode abschließend verarbeitet.

```
22  // OPC UA Post-Call (Methode erfolgreich abgeschlossen)
23 #OPC_UA_ServerMethodPost_Instance(
24                                  UAMethod_Result := #statPostResult,
25                                  UAMethod_Finished := #statPostFinished,
26                                  Done => #statPostDone,
27                                  Busy => #statPostBusy,
28                                  Error => #statPostError,
29                                  Status => #statPostStatus,
30                                  UAMethod_OutParameters := #UAMethod_OutParameters
31 );
32
```

Abbildung 25: Post-Call

Abbildung 26 zeigt exemplarisch die Umsetzung der Methodenlogik für die OPC UA-Methode „Band_antreiben“. Dabei ist die Zuweisung zwischen dem Pre-Call- und dem Post-Call-Block dargestellt, die zur Steuerung und Rückmeldung der Ausführung erforderlich ist.

```
IF #UAMethod_InParameters.statPreCalled THEN

    "Tag_1" := TRUE;                    // *Motorsteuerung mit SPS-Ausgang %Q4.0*

    #UAMethod_OutParameters.statPostCalled := TRUE;
    #statPostResult := 16#00000000;      // **OPC UA Statuscode "Good"**
    #statPostFinished := TRUE;           // **Methode als abgeschlossen markieren**

END_IF;
```

Abbildung 26: Beispiel für Methodenlogik

In diesem Zusammenhang wurde zuvor im TIA Portal eine SPS-Ausgangsvariable mit dem Namen Tag_1 angelegt, die mit einem digitalen Ausgang (z. B. %Q4.0) der Steuerung verbunden ist und zur Ansteuerung des Bandmotors dient. Wird die Methode über einen OPC UA-Client aufgerufen, erkennt das Steuerungsprogramm dies an dem gesetzten Eingangssignal „statPreCalled“, welches durch die Systemfunktion OPC_UA_ServerMethodPre übergeben wird. Sobald dieses Signal aktiv ist, wird Tag_1 auf „true“ gesetzt, wodurch das Förderband gestartet wird. Im Anschluss daran erfolgt die Rückmeldung an den OPC UA-

Client. Hierzu wird in der Methode die Variable „statPostCalled“ auf „true“ gesetzt, um anzuzeigen, dass der Methodenaufruf verarbeitet wurde. Der Wert „statPostResult := 16#00000000“ gibt den OPC UA-Statuscode „Good“ zurück, was auf eine fehlerfreie Ausführung hinweist. Abschließend wird „statPostFinished“ auf „true“ gesetzt, um die Methode als abgeschlossen zu kennzeichnen.

Schritt 3: Einbindung in die OPC UA-Kommunikation

Nachdem die Methode mit ihrer Pre-Instanz, Post-Instanz sowie der zugehörigen Logik vollständig implementiert wird, erfolgt die Einbindung in das Hauptprogramm. Dazu wird der erstellte Methodenbaustein im Projektbaum innerhalb des Main-OB1 in einem neuen Netzwerk aufgerufen. Beim Aufruf des Bausteins wird automatisch ein zugehöriger Datenbaustein generiert, welcher die für die OPC UA-Kommunikation relevanten Schnittstellenparameter enthält. Damit die Methode später über OPC UA aufgerufen werden kann, muss sie zusätzlich in einer Serverschnittstelle hinterlegt werden. Dies erfolgt im Projektbaum unter dem Punkt „OPC UA-Kommunikation“ im Abschnitt „Server-Schnittstellen“. Dort wird über „Neue Server-Schnittstelle hinzufügen“ eine neue Schnittstelle erstellt, in der anschließend der zuvor generierte Methodenbaustein bzw. der zugehörige Datenbaustein eingebunden wird. Erst durch diese Zuweisung ist sichergestellt, dass die Methode korrekt im OPC UA-Adressraum bereitgestellt wird und vom Client aus aufrufbar ist. Damit ist die Methode vollständig in die Kommunikation mit dem OPC UA-Client eingebunden und steht nach dem Laden des Projekts in die Steuerung für den praktischen Einsatz zur Verfügung.

Integration ausführbarer Teilmodelle in das Backend

Schritt 1: Einbindung der erforderlichen NuGet-Pakete für OPC UA

Für die Implementierung des OPC UA-Clients müssen zunächst die erforderlichen OPC UA-Bibliotheken über den NuGet-Paketmanager eingebunden werden. Diese ermöglichen die Kommunikation über das OPC UA-Protokoll und bilden die technische Grundlage für den Zugriff auf die Steuerung. Im Rahmen dieser Arbeit werden, die in Abbildung 27 dargestellten Pakete installiert.

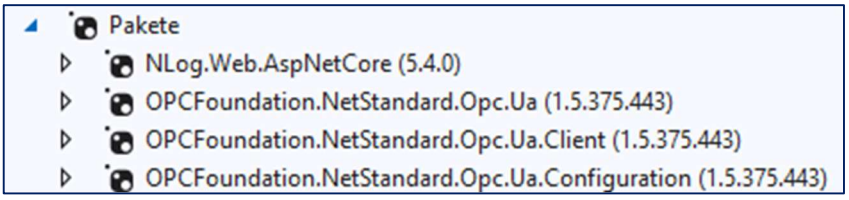


Abbildung 27: Erforderliche OPC UA-Bibliotheken

Diese Pakete enthalten alle notwendigen Klassen und Schnittstellen zur Verwaltung von Sessions, Konfiguration von Endpunkten sowie zur Ausführung von Methodenaufrufen über

OPC UA. Sie bilden die technische Grundlage für die Entwicklung des OPC UA-Clients in C#. Erst durch deren Einbindung können im weiteren Verlauf die clientseitigen Codeanweisungen korrekt ausgeführt und verwendet werden.

Schritt 2: OPC UA-Client „OpcUaClientService.cs“ erstellen

Nachdem die erforderlichen Bibliotheken installiert sind, kann die Klasse für den OPC UA-Client dem Projekt hinzugefügt werden. Der Client wird analog zu den übrigen BaSyx.NET Komponenten in C# implementiert und ist für die Kommunikation mit dem im TIA-Portal aktivierten OPC UA-Server verantwortlich. Beim Start der Anwendung initialisiert sich der Client automatisch mit der hinterlegten Serveradresse, wobei die Verbindung während der gesamten Laufzeit aufrechterhalten bleibt. Für Fehlerfälle wird im Client eine „ReconnectSession“-Logik implementiert, die bei einer Verbindungsunterbrechung automatisch einen Wiederaufbau der Verbindung einleitet. Die Hauptaufgabe des Clients besteht darin, die OPC UA-Session zu verwalten und Methoden bereitzustellen, über die gezielt einzelne Skills anhand ihrer Objekt-ID und Node-ID aufgerufen werden können. Dazu gehört insbesondere die zentrale Methode „CallSkill“, die extern aufgerufen werden kann, um einen spezifischen Maschinenbefehl zu starten.

Schritt 3: SkillSubmodelLoader.cs, „basicskills.json“ und „kombinierteskills.json“ erstellen

Die beiden JSON-Dateien „baiscskills.json“ und „kombinierteskills.json“ enthalten für die Teilmodelle fünf und sieben Eigenschaften von Basic Skills und kombinierten Skills, die keine Eingabeparameter benötigen. In den Dateien befinden sich jeweils Einträge mit den Namen der Skills, ihren Beschreibungen sowie den zugehörigen Objekt-IDs und Node-IDs. Die Klasse „SkillSubmodelLoader.cs“ liest diese Informationen beim Start der Anwendung aus und erstellt für jeden Eintrag eine Operation, die zur Laufzeit als Teilmodell-Element in die AAS eingebunden wird. Diese Operation ruft bei Aktivierung im Hintergrund die zugeordnete OPC UA-Methode über den Client auf.

Im Folgenden werden die technischen Umsetzungen der skill-basierten Teilmodelle erläutert:

1. **Teilmodell Basic Skills aus JSON:** Dieses Teilmodell basiert auf einer dynamischen Integration externer JSON-Dateien. In der Datei „basicskills.json“ sind sämtliche sensorbasierten Skills enthalten. Für jeden Skill sind Anzeigename, Beschreibung, Objekt-ID, Node-ID der OPC UA-Methode sowie die Node-ID für etwaige Eingabeparameter hinterlegt. Das Einlesen erfolgt zur Laufzeit über die Klasse „SkillSubmodelLoader.cs“, die bei Anwendungsstart sämtliche Einträge verarbeitet. Für jeden Skill wird automatisch ein Operation-Element erzeugt und in die AAS

eingebunden. Die Methode wird bei Aufruf über die UI ausgelöst. Im Hintergrund greift „SkillSubmodelLoader.cs“ auf die zentrale Methode „CallSkill“ des in „OpcUaClientService.cs“ implementierten OPC UA-Clients zu. Die Einbindung des Teilmodells erfolgt über „Program.cs“, die Interaktionslogik über „HelloAssetAdministrationShellService.cs“.

2. **Teilmodell Basic Skills mit zusätzlichen Logiken:** Teilmodell 6 enthält steuerbare Basic Skills und ist statisch in „HelloAssetAdministrationShellService.cs“ implementiert. Die Skills werden als Operationselemente mit Eingabe- und Ausgabewerten definiert. Beispielsweise erwartet der Skill „Band_steuern“ einen Integer-Wert, der in eine Eingabe-Node-ID geschrieben wird. Anschließend wird die zugehörige Methode über ihre Node-ID aufgerufen. Die Rückmeldung erfolgt über eine Status-Node-ID. Die Ansteuerung übernimmt die Methode „Callbasickills_6_1(int eingabe)“ im „OpcUaClientService.cs“. Das Ergebnis wird in der UI visualisiert. Die Einbindung des Teilmodells erfolgt über „Program.cs“.
3. **Teilmodell Kombinierte Skills aus JSON:** Dieses Teilmodell basiert ebenfalls auf einer dynamischen Einbindung über die Datei „kombinierteskills.json“. Jeder Skill-Eintrag enthält Namen, Beschreibung, Objekt-ID und Node-ID. Die Datei wird bei Anwendungsstart über „SkillSubmodelLoader.cs“ verarbeitet und automatisch in die AAS eingebunden. Die Ausführung erfolgt über die UI, wobei „SkillSubmodelLoader.cs“ erneut die Methode „CallSkill“ im „OpcUaClientService.cs“ nutzt. Die Einbindung des Teilmodells erfolgt über „Program.cs“, die Logik ist in „HelloAssetAdministrationShellService.cs“ verankert.
4. **Teilmodell Kombinierte Skills:** Dieses Teilmodell ist vollständig statisch implementiert. Sämtliche Operationen sind direkt in „HelloAssetAdministrationShellService.cs“ definiert. Die Methoden werden manuell angelegt, mit festen Node-IDs verknüpft und über „CallSkill“ im „OpcUaClientService.cs“ ausgeführt. Die Ergebnisse werden in der UI angezeigt. Die Integration erfolgt über „Program.cs“, eine dynamische Erweiterung über JSON ist hier nicht vorgesehen.
5. **Teilmodell Statisch codiertes Skill-Submodelbeispiel:** Dieses Teilmodell dient als Beispiel für die vollständig manuelle Umsetzung eines steuerbaren Skills. Der Skill „Band_antreiben“ ist in „HelloAssetAdministrationShellService.cs“ implementiert und wird über das Operationselement „9-1 Band_antreiben“ in der UI dargestellt. Die Ausführung erfolgt durch die Methode „CallSkillBandAntreiben“ im „OpcUaClientService.cs“, Rückmeldungen erscheinen in der Benutzeroberfläche. Das Teilmodell wird im Konstruktor direkt über „aas.Submodels.Add(skillBandAntreibenSubmodel)“ eingebunden und über „Program.cs“ registriert. Die Operation selbst

wird explizit erstellt, beschrieben und mit einer Logik über die Methode „OnMethodCalled“ versehen.

6. **Teilmodell Betriebsdaten:** Das Teilmodell 10 wird in der Datei „HelloAssetAdministrationShellService.cs“ vollständig als statische Struktur implementiert und erweitert die AAS um Funktionen zur reinen Datenauswertung. Es enthält zwei Operationselemente, die jeweils eine spezifische Methode im „OpcUaClientService.cs“ aufrufen und Statusinformationen aus dem OPC UA-Server abfragen. Die Integration erfolgt direkt im Quellcode, ohne zusätzliche JSON-Datei oder dynamische Ladeprozesse. Die Einbindung des Teilmodells in die AAS erfolgt über die „Program.cs“, in der es beim Start registriert und dem Shell-Objekt hinzugefügt wird. Die erste Operation mit der IdShort 10-1 Zustandsdaten ruft die Methode „ReadZustandsdatenAsync()“ aus dem OPC UA-Client auf. Diese liest die Anzahl der aktiven OPC UA-Sitzungen über die Node-ID i=2278 aus. Da es sich um eine serverinterne, standardisierte Systemvariable handelt, ist keine explizite Namespace-Angabe notwendig. Der Rückgabewert wird direkt im Property-Feld des Operationselements angezeigt. Die zweite Operation mit der IdShort 10-2 Statusinformationen greift über die Methode „ReadStatusinformationenAsync()“ auf die Node-ID ns=3;s="Tag_1" zu. Hierbei handelt es sich um eine adressierbare Maschinenvariable, die den Zustand des Bandmotors repräsentiert. Der Rückgabewert wird in einen Textstatus übersetzt und über die UI der AAS visuell dargestellt. Beide Methoden nutzen eine einheitliche Logik im „OpcUaClientService.cs“, um die definierten Node-IDs auszulesen, den jeweiligen Wert zu interpretieren und über „RegisterMethodCalledHandler“ mit dem entsprechenden Operationselement zu verbinden. Die Ergebnisse sind dadurch direkt in der AAS sichtbar. Besonders hervorzuheben ist die gezielte Verwendung unterschiedlicher Namespace-IDs. Während systeminterne Variablen mit i= „...“ angesprochen werden, stammen Maschinendaten in diesem Projekt aus ns=3 und steuerbare Methoden aus ns=4. Diese Struktur erleichtert eine klare Trennung und gezielte Adressierung der OPC UA-Knoten innerhalb der AAS.
7. **Teilmodell Basic Skills:** Das Teilmodell 11 wird vollständig statisch in der Datei „HelloAssetAdministrationShellService.cs“ implementiert und erweitert die AAS um acht steuerbare, direkt verknüpfte Maschinenfunktionen. Für jede Maschinenaktion wird ein eigenes Operationselement angelegt, das über die UI aufgerufen werden kann. Die Integration des Teilmodells in die AAS erfolgt beim Start über die Datei „Program.cs“, in der es explizit registriert und dem Shell-Objekt hinzugefügt wird. Jedes Operationselement ist mit einem zugehörigen Ausgabefeld versehen, das den Status der jeweiligen Ausführung als Zeichenkette zurückliefert. Diese Statusanzeige verwendet ein visuelles Rückmeldesystem in der UI. Die Anzeige basiert

auf der Auswertung des Rückgabewerts im OPC UA-Client. Die technische Umsetzung erfolgt über dedizierte Methoden im „OpcUaClientService.cs“. Für jeden Skill wird dort ein separater Methodenaufruf implementiert, der eine fest zugeordnete Objekt-ID und Node-ID verwendet. Diese Node-IDs entsprechen den im TIA-Portal angelegten OPC UA-Methoden und gewährleisten eine eindeutige Zuordnung zur jeweiligen Maschinenfunktion. Der zentrale Aufrufmechanismus erfolgt über die bereits etablierte Methode „CallSkill“, die für alle Operationen dieses Teilmodells verwendet wird. Im Gegensatz zu dynamisch eingebundenen Teilmodellen greift Teilmodell 11 nicht auf externe JSON-Dateien oder die Klasse „SkillSubmodelLoader.cs“ zurück. Die gesamte Struktur, einschließlich der Operationselemente und Statusanzeigen, ist fest im Quellcode definiert. Diese statische Architektur ermöglicht eine besonders zuverlässige und leicht nachvollziehbare Umsetzung, die sich ideal als Referenzmodell für weitere fest eingebundene Maschinenfunktionen eignet. Die klare Kopplung an die vorhandenen OPC UA-Methoden und die vollständige Kontrolle über die Logik im Backend machen dieses Teilmodell zu einer stabilen Vorlage für zukünftige Erweiterungen.

4.8 Bereitstellung der Laufzeitumgebung

Nach Abschluss der Implementierung AAS in Microsoft Visual Studio erfolgt im letzten Schritt die Überführung der Anwendung in eine ausführbare Laufzeitumgebung. Dieser Vorgang wird über die integrierte Veröffentlichungsfunktion von Microsoft Visual Studio realisiert, welche die Kompilierung, das Paketieren und die Bereitstellung der Anwendung in einem definierten Zielverzeichnis automatisiert durchführt. Hierzu wird das eigene Projekt innerhalb der Projektmappe per Rechtsklick ausgewählt. Im daraufhin geöffneten Kontextmenü wird die Option „Veröffentlichen …“ gewählt, wodurch ein Assistent zur Konfiguration des Veröffentlichungsziels gestartet wird. Im ersten Schritt des Assistenten wird unter „Wo soll heute veröffentlicht werden?“ das Veröffentlichungsziel festgelegt. Für eine lokale Testumgebung bietet sich die Option „Ordner“ an. Diese Auswahl ermöglicht es, die Anwendung in einem benutzerdefinierten Verzeichnis bereitzustellen, beispielsweise auf einem USB-Datenträger oder innerhalb eines bestimmten Netzwerkpfads. Nach Auswahl von „Weiter“ wird der Speicherort definiert, in dem die Inhalte abgelegt werden sollen. Mit einem Klick auf „Fertig stellen“ wird die Veröffentlichungskonfiguration gespeichert und der Veröffentlichungsprozess gestartet. Visual Studio erzeugt dabei im gewählten Zielverzeichnis eine lauffähige Version der Anwendung. Diese beinhaltet alle erforderlichen Abhängigkeiten und eine ausführbare Datei im .exe-Format. Die erstellte .exe-Datei stellt die Schnittstelle zur Laufzeitumgebung dar und kann durch Doppelklick direkt ausgeführt werden. Damit ist

es möglich, die AAS außerhalb der Entwicklungsumgebung zu starten, in eine bestehende Systemlandschaft zu integrieren oder auf andere Systeme zu übertragen.

5. Anwendung und Evaluation

5.1 Inbetriebnahme und Nutzung der AAS

Die Bereitstellung der AAS erfolgt schrittweise und beginnt mit dem Aktivieren der SPS. Dazu wird die SPS in den Betriebsmodus „RUN“ versetzt. In diesem Modus ist gewährleistet, dass der integrierte OPC UA-Server aktiv ist und die angeschlossenen Aktoren über definierte Schnittstellen angesteuert werden können. Im Anschluss wird die Anwendung gestartet, die zur Kommunikation mit der AAS dient. Dazu navigiert man zunächst in das Verzeichnis, in dem sich die ausführbare Datei namens „WebApplication 1“ befindet. Nach einem Doppelklick auf diese Datei wird die Anwendung geladen und ausgeführt. Sobald die Anwendung läuft, wird ein beliebiger Webbrowser geöffnet. In die Adresszeile des Browsers gibt man „localhost:5080“ ein. Über diese Adresse wird die GUI der AAS lokal aufgerufen. Nach dem erfolgreichen Laden der UI stehen dem User sämtliche implementierte Teilmodelle der AAS zur Verfügung. So kann beispielsweise ein Skill aus einem Skill-Teilmodell ausgewählt und direkt über die AAS-GUI ausgeführt werden, wodurch eine konkrete Aktion in der realen Anlage ausgelöst wird. Die in Abbildung 28 dargestellte Grafik veranschaulicht den zuvor beschrieben Ablauf zur Bereitstellung und Nutzung der AAS in kompakter Form. Zum Beenden der Anwendung genügt es, den verwendeten Webbrowser zu schließen und die SPS in den Betriebsmodus „STOP“ zu versetzen.

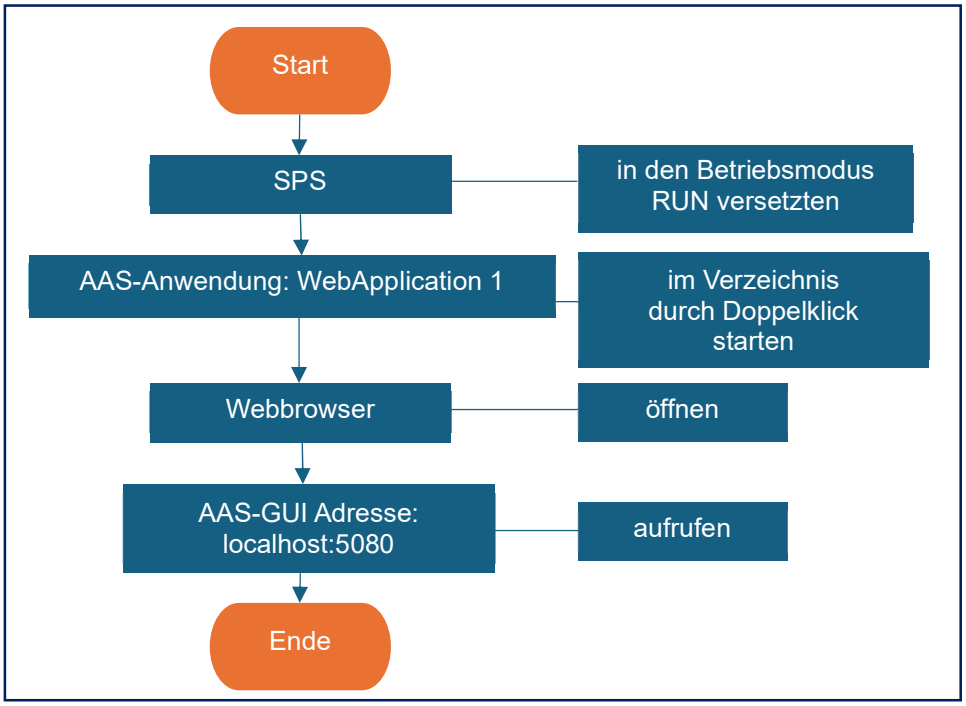


Abbildung 28: Ablauf zur Inbetriebnahme und Nutzung der AAS-Anwendung

5.2 Evaluation

In diesem Kapitel wird die entwickelte AAS hinsichtlich ihrer Funktionalität und Praxistauglichkeit evaluiert. Dabei wird überprüft, inwieweit die in Kapitel 3.2 Allgemeine Anforderungen formulierten Anforderungen erfüllt werden und ob die einzelnen Komponenten der Lösung im Zusammenspiel zuverlässig funktionieren.

Prüfung der Methoden mit UA Expert

Prüfung der Methoden mit UA Expert

Zur Überprüfung der grundsätzlichen Funktionsfähigkeit der implementierten Skills wird zunächst analysiert, ob die im TIA-Portal erstellten OPC UA-Methoden über externe Clients zuverlässig aufrufbar sind. Dieser Schritt bildet die Grundlage für die Bewertung der Anforderung **A5**, die eine stabile und sichere Kommunikation mit der SPS über OPC UA voraussetzt. Hierzu wird das Softwaretool UA Expert verwendet. Dabei handelt es sich um einen OPC UA-Testclient, der verschiedene Funktionen zur Analyse und Interaktion mit OPC UA-Servern bereitstellt. Zu den wichtigsten Merkmalen zählen ein integriertes Zertifikatsmanagement, ein Discovery-Service zur Suche nach verfügbaren OPC UA-Servern sowie die Möglichkeit, gezielt eine Verbindung zu einem Server herzustellen. Darüber hinaus erlaubt UA Expert das Einlesen von Variablenwerten, das Ausführen von OPC UA-Methoden und die Auswertung von Kommunikationsprozessen.(41) Im Rahmen dieses Projekts wird UA Expert gezielt eingesetzt, um eine Verbindung zum im TIA Portal integrierten OPC UA-Server herzustellen und einzelne Methoden manuell aufzurufen. Auf diese Weise konnte bereits im frühen Entwicklungsstadium überprüft werden, ob die Methoden wie vorgesehen ausgeführt werden können und ob die Übergabe von Eingabeparametern korrekt erfolgt und in der Steuerung verarbeitet wird. Für den Methodenaufruf wird in UA Expert das entsprechende Objekt über das Kontextmenü mit dem Befehl „Call“ aktiviert. Dabei musste das Attribut „statPreCalled“ gesetzt werden, um den Aufruf korrekt auszulösen. Anschließend lieferte UA Expert eine Statusmeldung, anhand derer die erfolgreiche Ausführung überprüft werden konnte. Die zugehörigen Node-IDs werden dokumentiert und für die spätere Integration in die C#-Implementierung verwendet.

Beobachtete Probleme bei mehrfacher Ausführung von Methoden

Während der Evaluation traten bei der wiederholten Ausführung bestimmter Skills unerwartete Probleme auf. Dies betraf insbesondere sogenannte Basic Skills, bei denen jeweils zwei OPC UA-Methoden für Start- und Stopp-Vorgänge verwendet werden, zum Beispiel „Band_antreiben“ und „Band_anhalten“. Wird ein Startbefehl wie „Band_antreiben“ ausgeführt, anschließend „Band_anhalten“ aufgerufen und danach erneut „Band_antreiben“, reagiert der OPC UA-Server nicht zuverlässig auf den erneuten Methodenaufruf. Ein ähnliches Verhalten konnte auch bei anderen Skills beobachtet werden, etwa bei

„Stopper_ausfahren“ und „Stopper_einfahren“ oder „Weiche_2_ausfahren“ und „Weiche_2_einfahren“. Die Ursache liegt vermutlich darin, dass der OPC UA-Server in Verbindung mit Siemens TIA-Portal V16 den Methodenaufruf als nicht korrekt abgeschlossen interpretiert. Der zugehörige Status bleibt dauerhaft auf „true“, da kein automatischer Rücksetzmechanismus vorhanden ist und der Abschluss der Methode in der Steuerung nicht eindeutig signalisiert wird. Infolgedessen wird ein erneuter Aufruf derselben Methode blockiert. Besonders kritisch zeigte sich dieses Verhalten nach dem Ausführen des Skills „Grundeinstellung“. Nach dem einmaligen Aufruf dieser Methode konnten keine weiteren Skills mehr über die OPC UA-Schnittstelle ausgeführt werden. Auch hier deutet alles auf einen unvollständigen oder fehlenden Rücksetzmechanismus in der Steuerungslogik hin.

Damit eine Methode mehrfach aufrufbar bleibt, muss sichergestellt werden, dass sie nach ihrer Ausführung zuverlässig in einen inaktiven Zustand zurückkehrt und vom OPC UA-Server als korrekt abgeschlossen erkannt wird. In der offiziellen Siemens Dokumentation zur Erstellung von OPC UA-Methoden wird darauf hingewiesen, dass bestimmte Variablen nach der Abarbeitung korrekt zurückgesetzt werden müssen. So wird etwa empfohlen, die Variable „UAMethod_Called“ auf „false“ zu setzen, sofern sie als Static deklariert wurde, und „UAMethod_Finished“ auf „true“ zu setzen, um das Ende der Methode korrekt zu signalisieren(40). Auch wenn dort keine expliziten Hinweise auf blockierte Methodenaufrufe zu finden sind, deutet die Beschreibung darauf hin, dass ein fehlender Rücksetzmechanismus zu unerwünschtem Verhalten führen kann. Besonders bei Methoden, die Aktoren in einem aktiven Zustand halten, wie etwa bei einem eingeschalteten Motor, kann dieses Problem auftreten, wenn im Programmcode kein klar definierter Abschlusspfad vorhanden ist. Es ist daher naheliegend, dass das im Projekt beobachtete Verhalten nicht spezifisch für die vorliegende Umsetzung ist, sondern im Zusammenhang mit der allgemeinen Methodenumsetzung in Siemens Steuerungen steht. Die beobachteten Einschränkungen bei der Wiederverwendbarkeit von Methoden, sowohl in UA Expert als auch in der Web GUI der AAS, sind somit möglicherweise auf fehlende Rücksetzmechanismen in der Steuerungslogik zurückzuführen. Im Hinblick auf die fünfte Anforderung **A5** lässt sich festhalten, dass die Verbindung zwischen dem selbst entwickelten OPC UA-Client und der SPS über die gesamte Testphase hinweg stabil war und keine Verbindungsabbrüche auftraten. Über die integrierte Web GUI konnten die implementierten Skills grundsätzlich ausgeführt werden, wobei die Kommunikation mit dem OPC UA-Server durch den Client initiiert und überwacht wurde. Aufgrund der beschriebenen Einschränkungen bei der Wiederverwendbarkeit einzelner Methoden war jedoch nicht bei allen Skills eine wiederholte Ausführung möglich. Die Anforderung **A5** ist daher nur teilweise erfüllt.

Besonderheit bei Skill-Ansteuerung über die AAS-GUI

Die Skills mit Aktoren-Ansteuerung (Write-Operation) konnten ausschließlich ausgeführt werden, wenn zuvor über UA Expert eine Initialisierung durchgeführt wurde oder ein mehrfacher Skill-Aufruf mit erneuter Session-Eröffnung erfolgte. Keine dieser Methoden ist eine optimale Lösung, daher würde die Initialisierung der Nodes aus dem OPC UA Client heraus eine wichtige Weiterentwicklung darstellen.

Bewertung der Anforderungen A1 bis A4

Modularität und Wiederverwendbarkeit der AAS-Struktur

Die Anforderung **A1** zielt auf einen modularen Aufbau und die Wiederverwendbarkeit der Struktur ab, um eine Übertragbarkeit auf weitere Stationen zu ermöglichen. Diese Anforderung wird erfüllt, da das Projekt auf der bestehenden Struktur der Beispielanwendung „HelloAssetAdministrationShell“ aus dem BaSyx.NET-Repository basiert. Die Anwendung lässt sich problemlos kopieren, erweitern und anpassen. Die Teilmodelle werden klar voneinander getrennt implementiert und können bei Bedarf durch zusätzliche Teilmodelle ergänzt oder für andere Stationen angepasst werden. Dadurch ist ein modularer und übertragbarer Aufbau der AAS gewährleistet.

Praxisnahe Integration in Automatisierungssysteme

Die Anforderung **A2** bezieht sich auf einen praxisnahen Aufbau, der eine Integration in bestehende Automatisierungssysteme ermöglichen soll. Durch die Anbindung an den im TIA-Portal V16 konfigurierten OPC UA-Server und die Nutzung von UA Expert als Testumgebung kommt eine reale Steuerung mit authentischer Kommunikationsschnittstelle zum Einsatz. Der entwickelte OPC UA-Client kommuniziert direkt mit der SPS, wodurch eine praxisorientierte Umsetzung auf Grundlage industrieller Standards erreicht wird.

Bereitstellung einer Benutzeroberfläche zur Interaktion mit Teilmodellen

Die Anforderung **A3** fordert die Bereitstellung einer UI zur Interaktion mit den Teilmodellen. Durch den Einsatz von BaSyx.NET kann auf eine integrierte Web-GUI zurückgegriffen werden, über die die Teilmodelle der AAS direkt aufgerufen und gesteuert werden. Diese ist unter der Adresse „localhost:5080“ erreichbar, wodurch keine zusätzliche Entwicklungszeit für eine eigene grafische Oberfläche erforderlich ist. Die Web-GUI ermöglicht die Anzeige aller implementierten Teilmodelle sowie die Ausführung einzelner Skills direkt über die zugehörigen Teilmodell-Elemente.

Direkte Kommunikation mit der SPS zur Skill-Ausführung

Die Anforderung **A4** fordert eine direkte Kommunikation mit der SPS zur Ausführung der implementierten Skills. Diese wird durch die Kopplung der AAS mit dem eigens entwickelten OPC UA-Client realisiert. Jeder Skill wird über ein entsprechendes Teilmodell-Element der AAS aufgerufen, das eine spezifische OPC UA-Methode adressiert. Die Verbindung zwischen UI, AAS und SPS funktioniert zuverlässig, sodass die Steuerung der Aktoren über die AAS erfolgreich umgesetzt wird.

6. Fazit und Ausblick

Im Rahmen dieser Arbeit wird eine Basis-AAS entwickelt, die auf dem Konzept von Industrie 4.0 basiert und mit dem Framework BaSyx.NET umgesetzt wird. Methodisch orientiert sich die Anwendung an Ansätzen des Skill-basierten Engineerings, indem sie Teilmodelle zur Abbildung und Ausführung von Skills integriert. Ergänzt wird die Struktur durch weitere Teilmodelle zur Abbildung verschiedener statischer und informationsbezogener Inhalte. Ein Aspekt, der bei der Weiterentwicklung berücksichtigt werden sollte, betrifft die Lizenzbedingungen und deren Einfluss auf bestehende technische Einschränkungen. So ist beispielsweise im Labor der Technischen Hochschule Ulm die Anzahl der über OPC UA verfügbaren Methoden auf 20 begrenzt. Aufgrund dieser Begrenzung verwenden die Skills „Band_anhalten“ und „Weiche_2_einfahren“ dieselbe Node-ID wie die Methode „Grundeinstellung“, um über einen gemeinsamen Aufruf ein Rücksetzen zu ermöglichen. Die im Projekt entwickelte AAS bildet eine erweiterbare Grundlage, die bei Bedarf für weitere Einsatzzwecke angepasst und ergänzt werden kann. Ein möglicher Anwendungsfall besteht beispielsweise darin, die gesamte Anwendung auf ein externes Edge Device, etwa einen Industrie-PC, zu übertragen. Dadurch wäre der Betrieb der AAS unabhängig von einem Hochschul-PC möglich und könnte näher an der Maschine oder produktionsnah erfolgen. Die durchgeführten Tests bestätigen die grundsätzliche Funktionsfähigkeit der implementierten OPC UA-Methoden. Einschränkungen treten lediglich bei der wiederholten Ausführung bestimmter Skills auf, was auf fehlende Rücksetzmechanismen in der Steuerung zurückzuführen ist. Darüber hinaus werden im Verlauf des Projekts erste Ansätze zur Skill-Orchestrierung erprobt, um Abläufe flexibel zu kombinieren und auch außerhalb der SPS steuerbar zu machen. Die in den folgenden Abschnitten vorgestellten Erweiterungskonzepte verdeutlichen, wie sich auf Basis der entwickelten Architektur weitere Potenziale erschließen lassen, etwa durch den Einsatz von Low-Code-Plattformen wie Node-RED, durch die Integration zusätzlicher BaSyx.NET-Komponenten oder durch automatisierte Mechanismen zur Erstellung und Erweiterung von AAS-Strukturen. Zur Veranschaulichung der technischen Umsetzung sind im Anhang beispielhafte Abbildungen enthalten, darunter die Konfiguration im TIA-Portal, die grafische Darstellung der Teilmodelle in der AAS-GUI sowie die Struktur eines Teilmodells am Beispiel des Teilmodells 2.

6.1 Orchestrierung von Skills mit Node-RED

Eine alternative Möglichkeit zur Ausführung und Kombination von OPC UA-Methoden, die im Rahmen dieser Arbeit zusätzlich erstellt wird, bietet die Low-Code-Plattform Node-RED. Mit Hilfe der dort verfügbaren OPC UA-Knoten lassen sich einzelne Skills wie zum Beispiel „Band_antreiben“ gezielt auslösen und auch miteinander verknüpfen, um komplexere Abläufe außerhalb der SPS zu steuern. In Abbildung 29 wird exemplarisch gezeigt, wie ein

einfacher Trigger eingerichtet werden kann. Nach erfolgreicher Ausführung erscheint in Node-RED die Rückmeldung „Erfolgreich injiziert: timestamp“, was den korrekten Methodenaufruf bestätigt.

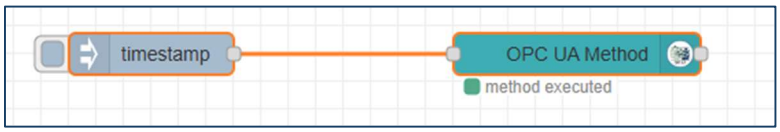


Abbildung 29: Node-RED Methodenaufruf

Abbildung 30 zeigt eine beispielhafte Konfiguration zur direkten Ansteuerung der Methode. Die Konfiguration erfolgt unkompliziert, da lediglich der Endpoint des OPC UA-Servers sowie die zugehörigen Node-IDs des Objekts und der Methode über UA Expert ermittelt und in Node-RED übernommen werden müssen. Im Eingabefeld „Arg0 Name“ wird der in UA Expert angezeigte Datentyp zusammen mit dem gewünschten Wert eingetragen. Für den Rückgabewert reicht es aus, im Feld „Out0 Name“ den Namen des in UA Expert definierten „OutputArguments“ zu hinterlegen.

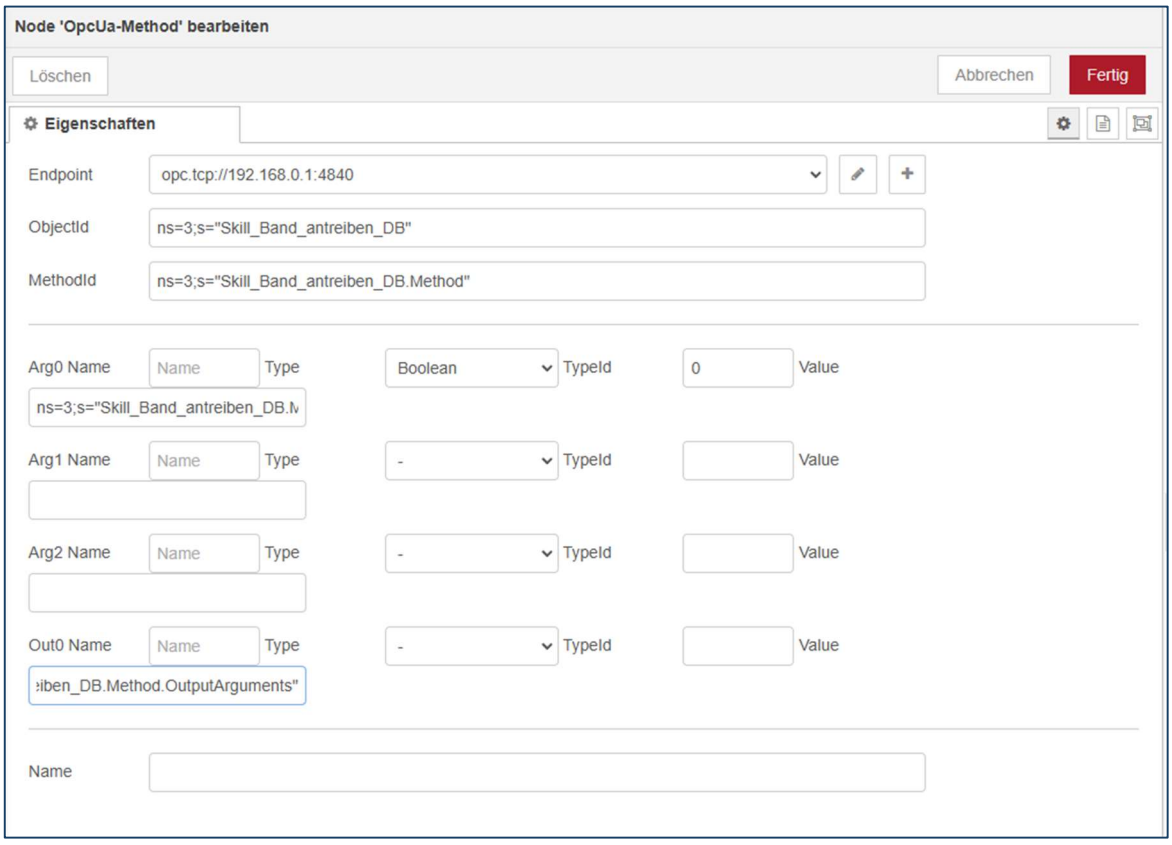


Abbildung 30: OPC UA-Methodenaufruf in Node-RED

Darüber hinaus eignet sich Node-RED auch für die Echtzeitvisualisierung und Überwachung von Prozesszuständen. So lassen sich beispielsweise der aktuelle Status eines Skills oder auftretende Fehlermeldungen grafisch in einem Dashboard darstellen. Dies

erleichtert nicht nur die Diagnose bei Fehlern, sondern erhöht auch die Transparenz im laufenden Betrieb. Besonders hervorzuheben ist die Möglichkeit, Node-RED zur Orchestrierung mehrerer Skills zu verwenden. Es lassen sich Abläufe konfigurieren, bei denen beispielsweise zunächst die Methode „Band_antreiben“ ausgeführt wird, anschließend ein Statusknoten prüft, ob der Vorgang abgeschlossen wurde, und das Ergebnis an eine API oder UI weitergeleitet wird. Auf diese Weise können Skills zu einem übergeordneten Prozess zusammengeführt werden, ohne dass die Orchestrierung innerhalb der SPS erfolgen muss. Damit bietet Node-RED eine flexible Erweiterung der Steuerungslogik sowie ein benutzerfreundliches Werkzeug zur Integration verteilter Systeme in Industrie-4.0-Umgebungen.

6.2 Nutzung weiterer AAS-Komponenten

Ein möglicher nächster Entwicklungsschritt besteht in der Integration zusätzlicher Komponenten der BaSyx-Referenzarchitektur. Dazu gehören unter anderem die Registry, das Repository sowie die Discovery-Komponente. Diese können verwendet werden, um mehrere AASs zentral auffindbar und verwaltbar zu machen. Eine Differenzierung zwischen der Registrierung vollständiger AASs und einzelner Teilmodelle ist dabei sinnvoll. Beispielsweise kann eine AAS-Registry der strukturierten Verwaltung ganzer AASs dienen, während eine Teilmodell-Registry auf die Verwaltung einzelner Teilmodelle fokussiert ist. Bei der Nutzung mehrerer ASSs innerhalb eines Systems ist zudem die Erweiterung auf den AAS-Typ-3 relevant, da dieser die Kommunikation zwischen unterschiedlichen AASs vorsieht.

6.3 Automatisierte Erstellung und Erweiterung der AAS

Für den Aufbau und die Erweiterung der AAS ist ein automatisierter Prozess auf Basis strukturierter Konfigurationsdateien möglich. Eine JSON-Datei kann verwendet werden, um die verfügbaren Skills zu definieren. Beim Start der AAS-Anwendung lässt sich prüfen, ob alle in der Datei enthaltenen Skills bereits als Teilmodelle implementiert sind. Falls dies nicht der Fall ist, können die fehlenden Skills automatisch angelegt und zur AAS hinzugefügt werden. Auch die Konfiguration des OPC UA-Servers kann in diesem Zusammenhang erweitert werden, etwa durch eine automatische Vergabe von Node-IDs für neue Skills. Auf diese Weise lässt sich der manuelle Aufwand bei der Integration weiterer Methoden reduzieren. Darüber hinaus können bestehende Skills durch zusätzliche Informationen ergänzt werden, die beispielsweise aus einer Knowledge-Base stammen. Dies betrifft unter anderem Angaben zu Parametern, Rahmenbedingungen oder Ausführungsreihenfolgen und -restriktionen. Die Beschreibung von Einschränkungen und Bedingungen für einzelne Skills in den JSON-Dateien kann für die dynamische Orchestrierung von Abläufen verwendet werden. Durch die Hinterlegung solcher Restriktionen kann bei einem externen Zugriff auf die AAS das vollständige Skill-Set inklusive seiner Bedingungen abgerufen und entsprechend

genutzt werden. Dies ermöglicht eine flexible Prozessgestaltung auf Basis der aktuellen Fähigkeiten und Zustände eines Assets.

6.4 Teilmodell mit Betriebsartensteuerung der Sortierstation

Des Weiteren könnte sich ein Teilmodell auf die Betriebsartensteuerung der Sortierstation beziehen. In diesem Teilmodell können verschiedene Zustände wie Automatikbetrieb, Handbetrieb oder Wartung nach GEMMA (Guide d'Etude des Modes de Marche et d'Arrêt) dokumentiert und bereitgestellt werden. Die Veröffentlichung dieser Informationen über die AAS unterstützt die transparente Darstellung des aktuellen Anlagenstatus und kann in Systemen mit mehreren externen Zugriffspunkten zur Koordination beitragen.

Tabellenverzeichnis

Abbildungsverzeichnis

Quellenverzeichnis

1. Cors O. Rhein-Neckar.io-Konsortium. 2024 [zitiert 7. April 2025]. Das volle Potenzial nutzen: die Relevanz der Verwaltungsschale für die Industrie 4.0. Verfügbar unter: https://rhein-neckar.io/das-volle-potenzial-nutzen-die-relevanz-der-verwaltungsschale-fuer-die-industrie-4-0/

2. Verwaltungsschale: Integrationsstecker für digitale Ökosysteme [Internet]. [zitiert 17. März 2025]. Verfügbar unter: https://www.dke.de/verwaltungsschale

3. Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA [Internet]. [zitiert 17. März 2025]. Asset Administration Shell (AAS) - Fraunhofer IPA. Verfügbar unter: https://www.ipa.fraunhofer.de/de/aktuelle-forschung/kompetenzzentrum-digitale-werkzeuge-in-der-produktion/digital-twin/asset-administration-shell.html

4. Verwaltungsschale in der Praxis.

5. Verwaltungsschale als Standard zur Administration Ihrer Assets [Internet]. elunic - professional services for the connected industry. [zitiert 22. März 2025]. Verfügbar unter: https://www.elunic.com/de/blog/verwaltungsschale-standard-administration-assets/

6. soffico.de | Orchestra [Internet]. [zitiert 22. März 2025]. Asset Administration Shell: die Verwaltungsschale | soffico. Verfügbar unter: https://soffico.de/use-cases/asset-administration-shell/

7. (PDF) Insights into Mapping Solutions Based on OPC UA Information Model Applied to the Industry 4.0 Asset Administration Shell. ResearchGate [Internet]. 9. Dezember 2024 [zitiert 22. März 2025]; Verfügbar unter: https://www.researchgate.net/publication/340651989_Insights_into_Mapping_Solutions_Based_on_OPC_UA_Information_Model_Applied_to_the_Industry_40_Asset_Administration_Shell

8. Über Asset Administration Shells - Eclipse BaSyxTM [Internet]. [zitiert 22. März 2025]. Verfügbar unter: https://wiki.basyx.org/en/latest/content/user_documentation/concepts%20and%20architecture/aas_overview.html

9. Plattform Industrie 4.0 WORD Vorlage - Functional-View.

10. Getting a Head Start in the World of AAS Dataspaces & Eclipse BASYX [Internet]. 2024 [zitiert 9. April 2025]. Verfügbar unter: https://www.youtube.com/watch?v=UvYuwEkVnVE

11. HTTP | Definition & Erklärung [Internet]. IT-SERVICE.NETWORK. [zitiert 23. März 2025]. Verfügbar unter: https://it-service.network/it-lexikon/http/

12. Talend - A Leader in Data Integration & Data Integrity [Internet]. [zitiert 9. April 2025]. Was ist eine API? Einfach erklärt! Verfügbar unter: https://www.talend.com/de/resources/was-ist-eine-api/

13. AAS-Projektarchiv - Eclipse BaSyxTM [Internet]. [zitiert 30. März 2025]. Verfügbar unter: https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/v2/aas_repository/index.html#aas-repository

14. Industry 4.0 Asset Administration Shell - 6.9 Concept description [Internet]. [zitiert 9. April 2025]. Verfügbar unter: https://reference.opcfoundation.org/I4AAS/v100/docs/6.9

15. DataBridge Component - Eclipse BaSyxTM [Internet]. [zitiert 9. April 2025]. Verfügbar unter: https://wiki.basyx.org/en/latest/content/user_documentation/basyx_components/databridge/index.html

16. MongoDB [Internet]. [zitiert 9. April 2025]. Was Ist MongoDB? Verfügbar unter: https://www.mongodb.com/de-de/company/what-is-mongodb

17. eclipsebasyx/aas-gui - Docker Image | Docker Hub [Internet]. [zitiert 9. April 2025]. Verfügbar unter: https://hub.docker.com/r/eclipsebasyx/aas-gui

18. 2021_What-is-the-AAS.pdf [Internet]. [zitiert 30. März 2025]. Verfügbar unter: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/2021_What-is-the-AAS.pdf?__blob=publicationFile&v=1

19. Downloading and Setting up BaSyx — Eclipse BaSyx™ [Internet]. [zitiert 9. April 2025]. Verfügbar unter: https://wiki.basyx.org/en/latest/content/user_documentation/user_tutorials/download/index.html

20. Team B. Vier IIoT Kommunikationsprotokolle – Teil 4: REST API [Internet]. Pepperl+Fuchs Blog. 2024 [zitiert 23. März 2025]. Verfügbar unter: https://blog.pepperl-fuchs.com/de/2024/vier-tcp-basierte-kommunikationsprotokolle-als-schluessel-fuer-das-iiot-teil-4-rest-api/

21. Unified Architecture - Landingpage [Internet]. OPC Foundation. [zitiert 23. März 2025]. Verfügbar unter: https://opcfoundation.org/about/opc-technologies/opc-ua/

22. Krohn M. Was ist MQTT? Erklärung mit industriellem Fokus [Internet]. OPC Router - The Communication Middleware. 2022 [zitiert 1. April 2025]. Verfügbar unter: https://www.opc-router.de/was-ist-mqtt/

23. FindLogoVector [Internet]. 2019 [zitiert 15. April 2025]. OPC Unified Architecture (UA) Vector Logo. Verfügbar unter: https://getvectorlogo.com/opc-unified-architecture-ua-vector-logo-svg/

24. File:Mqtt-hor.svg - Wikipedia [Internet]. 2020 [zitiert 15. April 2025]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:Mqtt-hor.svg

25. Praxisbeispiel 2: Verwaltungsschale selbst gemacht: der AASX Package Explorer [Internet]. [zitiert 1. April 2025]. Verfügbar unter: https://www.plattform-i40.de/IP/Redaktion/DE/Newsletter/2019/Ausgabe21/2019-21-Praxisbeispiel2.html

26. I4AAS - Industrie 4.0 Asset Administration Shell [Internet]. OPC Foundation. [zitiert 1. April 2025]. Verfügbar unter: https://opcfoundation.org/markets-collaboration/i4aas/

27. Industry 4.0 Asset Administration Shell [Internet]. [zitiert 1. April 2025]. Verfügbar unter: https://reference.opcfoundation.org/I4AAS/v100/docs/

28. Eclipse BaSyx [Internet]. [zitiert 1. April 2025]. Verfügbar unter: https://eclipse.dev/basyx//

29. Quick Start Guide — Eclipse BaSyx™ [Internet]. [zitiert 1. April 2025]. Verfügbar unter: https://wiki.basyx.org/en/latest/content/introduction/quickstart.html

30. 2023-07-27_IDTA_Tutorial_V3.0-Specification-AAS-Part-2_API.pdf [Internet]. [zitiert 1. April 2025]. Verfügbar unter: https://industrialdigitaltwin.org/wp-content/uploads/2023/07/2023-07-27_IDTA_Tutorial_V3.0-Specification-AAS-Part-2_API.pdf

31. eclipse-basyx/basyx-dotnet [Internet]. Eclipse BaSyx™; 2025 [zitiert 2. April 2025]. Verfügbar unter: https://github.com/eclipse-basyx/basyx-dotnet

32. OPCFoundation/UA-.NETStandard [Internet]. OPC Foundation; 2025 [zitiert 1. April 2025]. Verfügbar unter: https://github.com/OPCFoundation/UA-.NETStandard

33. Describing Capabilities of Industrie 4.0 Components.

34. Describing Capabilities of Industrie 4.0 Components.

35. Digitaler Produktpass – Für mehr Nachhaltigkeit und Zukunftssicherheit [Internet]. [zitiert 10. April 2025]. Verfügbar unter: https://www.conplement.de/digitaler-produktpass-eu

36. Siemens Software TIA Portal V16 - PLC-City [Internet]. [zitiert 15. April 2025]. Verfügbar unter: https://www.plc-city.com/shop/de/siemens-software-tia-portal-v16.html

37. webmaster@reichelt.de reichelt elektronik GIT. SIEMENS S7-1500, CPU 1512C-1 PN | Steuerungen günstig kaufen | reichelt elektronik [Internet]. [zitiert 15. April 2025]. Verfügbar unter: https://www.reichelt.de/de/de/shop/produkt/s7-1500_cpu_1512c-1_pn-267952

38. Festo Didactic InfoPortal [Internet]. [zitiert 13. März 2025]. Verfügbar unter: https://ip.festo-didactic.com/Infoportal/mps/SortingStation/DE/index.html

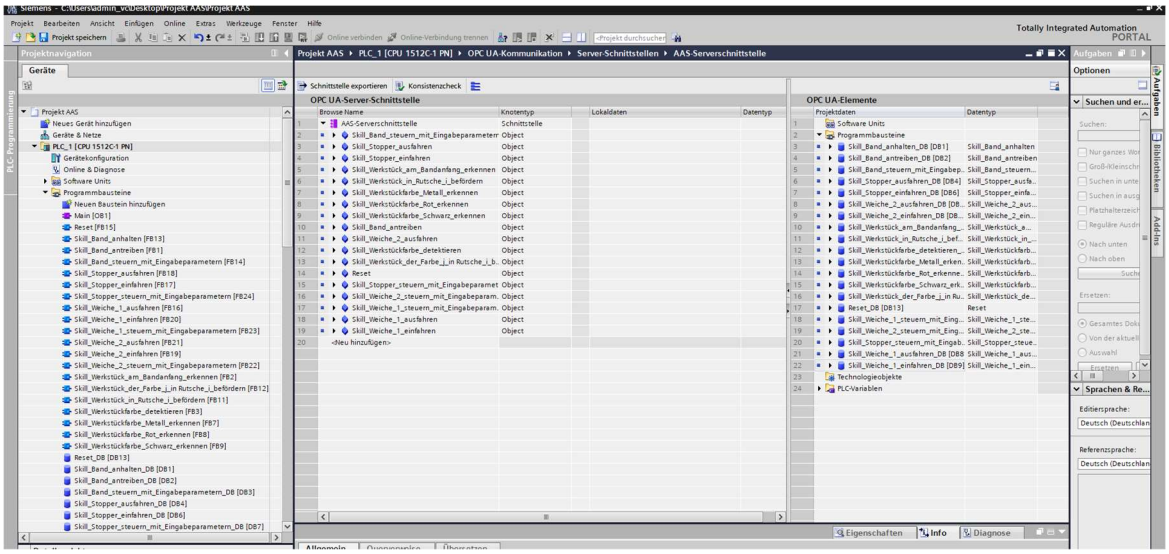
39. OPC UA-Methoden für den SIMATIC S7-1500 OPC UA-Server - ID: 109756885 - Industry Support Siemens [Internet]. [zitiert 13. März 2025]. Verfügbar unter: https://support.industry.siemens.com/cs/document/109756885/opc-ua-methoden-f%C3%BCr-den-simatic-s7-1500-opc-ua-server?dti=0&lc=de-DE

40. OPC UA-Methoden für den SIMATIC S7-1500 OPC UA-Server. 2019;

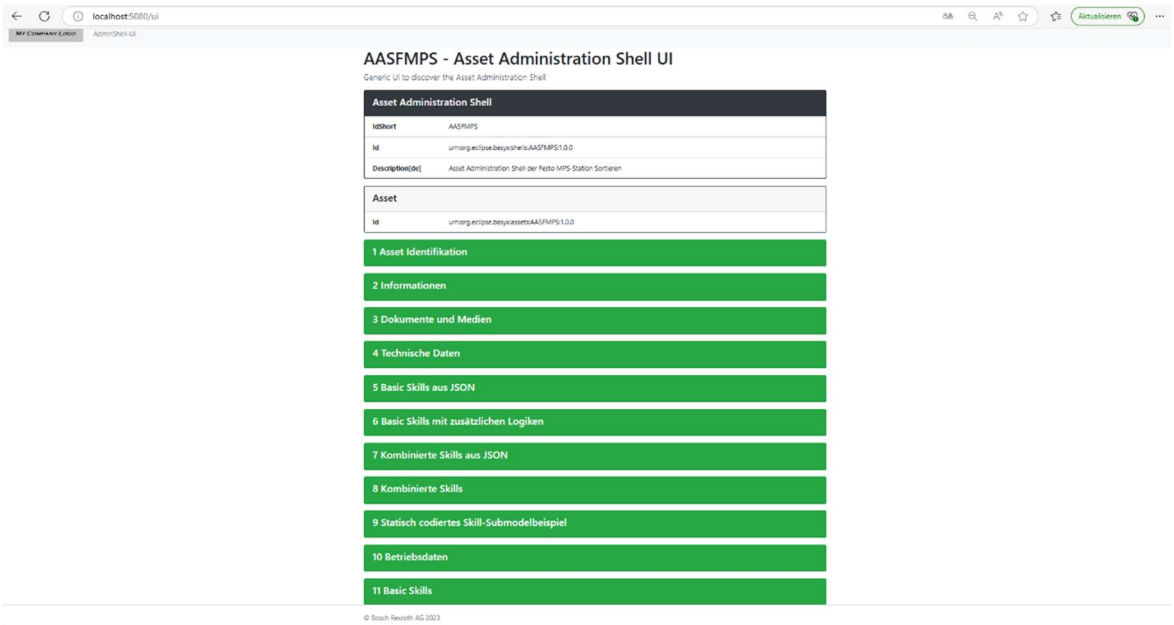
41. UaExpert „UA Referenz Client“ - Unified Automation [Internet]. [zitiert 20. März 2025]. Verfügbar unter: https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uaexpert.html

Anhang

Anhang 1: Eigene Abbildung - Screenshot des TIA-Portals mit allen Funktionsbausteinen und der AAS-Serverschnittstelle inklusive ausgewählter Datenbausteine



Anhang 2: Eigene Abbildung - Darstellung aller Teilmodelle in der AAS-GUI



Anhang 3: Eigene Abbildung - Beispielhafte Darstellung von IdShort, Id, Beschreibung und Teilmodell-Elementen anhand des Teilmodells 2

2 Informationen

IdShort	2 Informationen
Id	urn:org.eclipse.basyx:submodels:2Informationen:1.0.0
Description[de]	Informationenen über die Festo MPS-Station

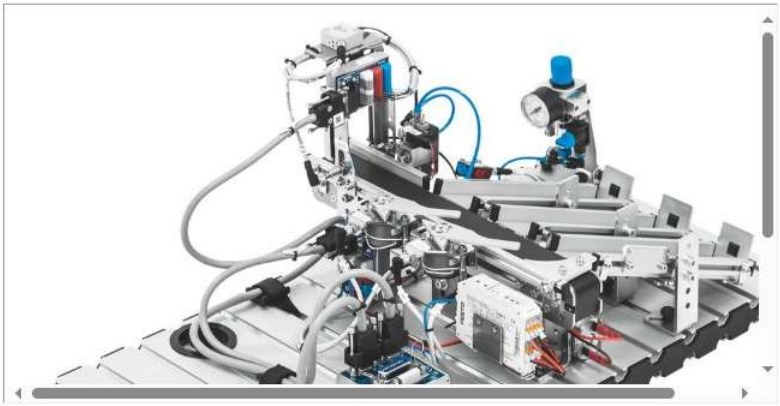
Submodel-Elements

2-1 Überblick

Description[de]	Bild, Funktion und Aufbau der Festo MPS-Station

2-1-1 Bild

Description[de]	Bild der Festo MPS-Station
ContentType	image/jpeg
Path	https://ip.festo-didactic.com/InfoPortal/MPS/SortingStation/img/image.jpg



2-1-2 Funktion

2-1-3 Aufbau