Semantic Integration Patterns for Manufacturing

Deliverable 2.2b

des Forschungsprojekts i-Twin

Dietmar Glachs, Georg Güntner
Salzburg Research

Jänner 2024

Das Projekt i-Twin wird gefördert vom BMK und von der FFG aus Mitteln des Programms IKT der Zukunft

Inhaltsverzeichnis

1 Zusammenfassung

Publizierbare Version

Moderne Fertigungs-Netzwerke sind durch eine hohe Vernetzung der Anlagen und der IT-/OT-Systeme gekennzeichnet. Die Heterogenität der Systeme und die Verwendung einer Vielzahl von industriellen Kommunikationsprotokollen und Standards stellen komplexe Anforderungen an die Integration der Anwendungen. Im Kontext der Digitalisierung bilden digitale Zwillinge ein technologisches Schlüsselkonzept für die Sammlung und Weitergabe von Anlagen- und Prozessdaten. Sie bieten daher eine geeignete Basis für die „Standardisierung“ der Anwendungsintegration. Wenn es gelingt, hier durch die Entwicklung von Patterns die Komplexität zu reduzieren, kann der Integrationsaufwand erheblich reduziert werden. Dies bildet die Hypothese des Forschungsprojekts „i-Twin“ und der dort propagierten „Semantic Integration Patterns“

Der vorliegende Bericht beschreibt das Konzept der Semantic Integration Patterns: Diese Patterns beruhen gemäß dem System-Design auf einer Middleware Plattform für die semantische Anwendungsintegration („i-Asset Plattform“). Der Bericht stellt daher zunächst die wesentlichen Komponenten der Plattform vor und entwickelt darauf basierend die Kennzeichen und Design-Prinzipien für die Semantic Integration Patterns. Dabei wird auf die Rolle der Application Connectors bei der semantischen Anwendungsintegration über Datenobjekte eingegangen. Die Datenobjekte beruhen auf dem Informationsmodell der Asset Administration Shell bzw. deren Teilmodellen und deren Prozess-Modell. Der letzte Teil des Berichts ist der Umsetzung der Semantic Integration Patterns gewidmet. Beispielsweise wird ein Pattern zum Senden und Bearbeiten von Störmeldungen („Alert Messages“) vorgestellt.

Der vorliegende Bericht basiert auf dem System-Design und der Architektur der Middleware-Plattform (D2.4). Eine Spezialisierung der Semantic Integration Patterns für analytische Systeme erfolgt in einem weiteren Bericht (D2.3).

i-Twin

i-Twin erforscht Interoperabilitätskonzepte für daten-getriebene digitale Zwillinge in der Fertigungsindustrie. Das Projekt propagiert eine Open-Source-Middleware für die Integration von Fertigungs-IT-Systemen und vernetzten Anlagen auf der Grundlage von Semantic Integration Patterns. Das vorrangige Ziel von i-Twin ist es, den Integrationsaufwand zu reduzieren und den Austausch von Stamm- und Betriebsdaten in Fertigungsnetzwerken zu ermöglichen. Die Ergebnisse werden in einem Forschungslabor und in einem industriellen Asset-Management-Szenario validiert.

Das Projektkonsortium unter der Leitung der Salzburg Research verbindet die Forschungsinteressen von drei Systemanbietern (H&H Systems: CMMS, COPA-DATA: OT Software Plattform, IcoSense: Edge-Nodes) und eines Industrieunternehmens (INNIO Jenbacher: diskrete Fertigung) mit der Expertise der beteiligten Forschungspartner (Universität Salzburg: Data Science, Salzburg Research: Motion Data Intelligence).

Das Projekt i-Twin wird gefördert vom BMK (Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie) und von der FFG (Österreichische Forschungsförderungsgesellschaft mbH) aus Mitteln des Programms IKT der Zukunft.

2 Anwendungsintegration mit Semantic Integration Patterns

2.1 Integration als Herausforderung der Digitalisierung

Die Digitalisierung hat sich in der Industrie zu einem wesentlichen Treiber für die Verbesserung der Produktivität, Nachhaltigkeit und Ressourceneffizienz entwickelt. Seit rund einem Jahrzehnt formierten sich große Initiativen, die die Digitale Transformation durch die Unterstützung von Standardisierungs-Vorhaben, die Sammlung von Erfolgsberichten („Good Practice“), die Veranstaltung von Fachkongressen zur Vernetzung der Akteure und zum Austausch von Methodik und Technologie unterstützten. Beispiele dieser im Grunde sehr ähnlichen industriellen Digitalisierungs-Initiativen sind: die deutsche “Plattform Industrie 4.0”[1], die Schweizer Netzwerk-Plattform “Industrie 2025”[2], das amerikanische “Industry IoT Consortium”[3] oder das chinesische Programm “Made in China 2025”[4].

Eine für unsere Betrachtungen wesentliche Folge der industriellen Digitalisierung ist die Vernetzung der Anlagen und Maschinen und der Fertigungs-IT-Systeme: Industrie 4.0 verwandelte fast jede neue Maschine und Komponente in ein intelligentes vernetztes Asset. Und dieser Trend setzt sich auch durch digitales „Retrofitting“ – d.h. durch die nachträgliche Einführung digitaler Kommunikationskomponenten und Sensorik – bei den Bestandsanlagen fort.

Dies führt zu einer Auflösung der klassischen Automatisierungspyramide (Åkerman, 2018) und resultiert in vernetzten Fertigungsnetzwerken, in denen praktisch alle Assets untereinander und mit den Fertigungs-IT-Systemen verbunden sind. Assets liefern kontinuierlich, bei Zustandsänderungen oder in festgelegten Intervallen Prozess- und Zustandsdaten an jene IT-Systeme, die diese Daten für die Monitoring-, Überwachungs-, Steuerungs-Prozesse verwenden. Die Heterogenität und die Vielzahl von gebräuchlichen Protokollen und Standards stellen komplexe Anforderungen an die Kommunikationsprotokolle und an die Interoperabilität in Bezug auf die industriellen Assets und die IT-Systeme.

Beispielhafte Szenarien zur Veranschaulichung dieser Interoperabilitätsanforderungen sind im Folgenden angeführt:

- Verschiedene IT-Systeme, die Anlageninformationen verwalten (z. B. ERP, CMMS, Edge Nodes), wollen sicherstellen, dass sie über dieselbe Maschine "sprechen" und gleichzeitig redundante und veraltete Informationen vermeiden.
- Ein Edge Controller fragt die jüngste Wartungshistorie einer bestimmten Anlage ab, um das Betriebs- und Instandhaltungspersonal über die jüngsten Wartungsaktivitäten zu informieren.
- Ein Management-Dashboard fragt die Gründe für Ausfallzeiten ab, um die Overall Equipment Efficiency (OEE) zu berechnen.
- Ein Analysedienst möchte ausgewählte Maschinendaten nutzen, um Machine-Learning-Modelle in Abhängigkeit von dem in der Produktion verwendeten Material zu entwickeln. Die trainierten Modelle müssen regelmäßig für die Echtzeitanalyse auf Edge Nodes aktualisiert werden.

[1] Plattform Industrie 4.0: https://www.plattform-i40.de/
[2] Industrie 2025: https://www.industrie2025.ch/
[3] Industry IoT Consortium: https://www.iiconsortium.org/
[4] Made in China 2025: https://de.wikipedia.org/wiki/Made_in_China_2025

Zur Lösung dieser Anforderungen entwickelten sich – teils mit starker Unterstützung durch die eingangs erwähnten Initiativen – eine Reihe von Informationsmodellen, die einen wichtigen Schritt zur Reduktion der Komplexität der Integrationsaufgaben bildeten. Diese Informationsmodelle decken idealerweise den kompletten Lebenszyklus im Asset Management ab. Die Kompatibilität zwischen diesen teils standardisierten, teils proprietären oder branchenspezifischen Modellen ist nach wie vor begrenzt, da derartige Standards nur langsam angenommen werden und oft für spezielle Branchen oder Domänen entwickelt werden. Viele der neu entstandenen Interoperabilitätskonzepte tendieren dazu, domänenspezifische, proprietäre, geschlossene Ansätze für die Erstellung, Umwandlung, den Import, Export und die Synchronisierung von Datensätzen und Nachrichten zu verwenden.

In modernen Fertigungsnetzwerken haben sich **digitale Zwillinge** („Digital Twins“) - digitale Abbilder von physischen Assets - als technologisches Schlüsselkonzept für Maschinen und Infrastrukturkomponenten etabliert. Informationsmodelle für datengetriebene digitale Zwillinge zielen darauf ab, folgende Kategorien von Daten zu beschreiben:

- die Stammdaten der Fertigungsanlagen und ihrer Komponenten,
- die Konfigurationsparameter für den Betrieb der Anlage und ihrer Komponenten sowie
- die dynamischen Sensordaten der Anlagen, die dann für eine Vielzahl von begleitenden Prozessen wie Monitoring, Analytik, Prognose (z.B. vorausschauende Wartung) und Optimierung (z.B. KPI-Ermittlung, Wartungsplanung, Erhöhung der Anlagenverfügbarkeit) genutzt werden.

Digitale Zwillinge konsumieren und liefern im Idealfall Informationen an alle angeschlossenen Anwendungen in den operativen Fertigungssystemen. Daher stehen digitale Zwillinge und zugehörige Informationsmodelle naturgemäß im Mittelpunkt von Interoperabilitätsüberlegungen: Lösungen zur Bereitstellung offener, standardbasierter, selbstbeschreibender (semantischer) Schnittstellen zwischen den Teilnehmern eines Fertigungs-Ökosystems haben ein hohes Potenzial zur Reduzierung des Integrationsaufwands.

An dieser Stelle setzt unser Forschungsprojekt "**i-Twin**" an: Das 2022 gestartete Projekt untersucht Interoperabilitätskonzepte für datengetriebene digitale Zwillinge in der Fertigungsindustrie. Das Projekt propagiert eine Open-Source-Middleware für die Integration von Betriebsführungssystemen und vernetzten Anlagen auf Basis eines Konzepts namens "Semantic Integration Patterns". Es zielt darauf ab, den Integrationsaufwand zu reduzieren und einen sicheren und zuverlässigen Austausch von Stamm- und Betriebsdaten in Fertigungsnetzwerken zu ermöglichen.

Die folgenden Abschnitte geben einen Überblick über die Architektur einer Plattform für die semantische Anwendungsintegration („i-Asset Plattform“, s. Abschnitt 3) und des dieser Plattform zugrundeliegenden zentralen Konzepts der „Semantic Integration Patterns“, welche auf dem Informationsmodell der Asset Administration Shell, RAMI4.0/AAS, (DIN SPEC 91345, 2016) beruhen (s. Abschnitt 4). Eine erste detailliertere Darstellung der technologischen Konzepte und Design-Richtlinien erfolgt in den Abschnitten 5 und **Fehler! Verweisquelle konnte nicht gefunden werden.**.

2.2 Plattform für semantische Anwendungsintegration

Unser Ansatz zur Bewältigung einer Vielzahl von Kommunikationsanforderungen und Interoperabilitätsproblemen in heterogenen industriellen Fertigungssystemen propagiert ein datenzentriertes, integriertes Asset Management durch die Integration bestehender Software-Systeme in einer Middleware Plattform mit der Bezeichnung „**i-Asset Plattform**“. Der Name

der Plattform stammt vom Forschungsprojekt i-Asset[5] („Innovationsnetzwerk Digital Asset Management“), in dem die Plattform initiiert und als Grundlage für das digitale Asset Management entwickelt wurde.

In mittelständischen Unternehmen finden sich oftmals verschiedene Softwaresysteme, meist unterschiedlicher Anbieter, wie Dokumentenmanagementsysteme (DMS), Instandhaltungsmanagementsysteme (CMMS), Analytik-Systeme (z.B. für Predictive Maintenance) sowie weiterer (spezialisierte) Planungs- und Kontrollsysteme für die Produktion (z.B. ERP, MES, BDE, QA). In jeder dieser Anwendungen wird eine Vielzahl von Daten erhoben und verarbeitet, jedoch kann das Potential dieser Daten oft nicht gänzlich ausgeschöpft werden, da eine Datenintegration mittels individuell erstellter Schnittstellen kostenintensiv und mit hohem zeitlichem Aufwand verbunden ist.

Um diese Integrationshürde zu überwinden, entsteht die i-Asset Plattform als Bindeglied zwischen allen spezialisierten IT-Systemen und den Assets. Die i-Asset Plattform ermöglicht einen standardisierten Datenaustausch und forciert die lose Kopplung der angeschlossenen Systeme mit einem datenstrom-zentrierten Architekturansatz. Jegliche Kommunikation zwischen angeschlossenen Systemen erfolgt über eine zentrale Datenleitung, die von den Anwendungen beschrieben bzw. ausgelesen werden kann.

Abbildung 1 zeigt die generelle Architektur der i-Asset Plattform, welche den Anwendungs-Layer, den **i-Twin Data Integration Layer** sowie die im Edge-Layer eingebunden Assets und Devices umfasst. Diese Komponenten manifestieren gemeinsam den **Digital Twin** von realen Anlagen für die Zwecke des digitalen Asset Managements.

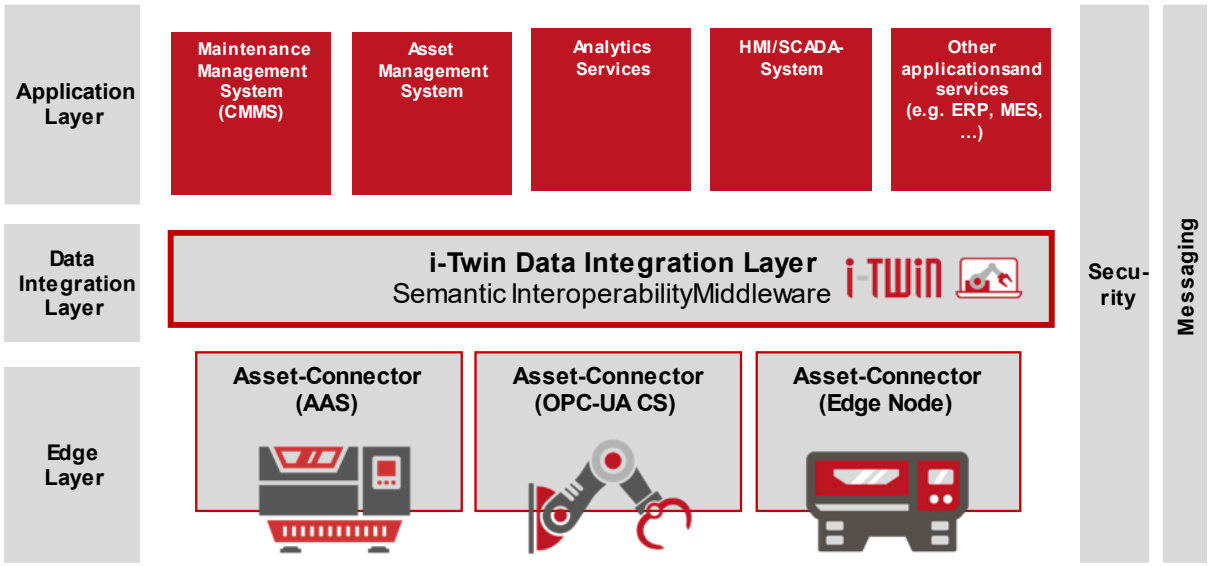


Abbildung 1: Übersicht über die Architektur der i-Asset Plattform

Der i-Twin Data Integration Layer einschließlich generischer (Asset- und Application-) Connectors ermöglicht die semantische Interoperabilität zwischen Fertigungsanwendungen und Anlagen. Die Lösung basiert auf den folgenden Gestaltungsprinzipien:

- Digital Factory: Die Norm „Digital Factory Framework” (IEC 62832-1, 2020) definiert eine Reihe von Modellelementen und Regeln für die Modellierung von Produktionssystemen.
- Semantic Integration Patterns (Details siehe Abschnitt 4) für die minimalinvasive Integration von Fertigungsanwendungen und OT-Softwareplattformen sowie für Analysedienste

[5] i-Asset („Innovationsnetzwerk Digital Asset Management“): https://srfg.at/i-asset

auf der Grundlage verfügbarer Standards für den Austausch von Modellen für maschinelles Lernen und KI.
- Ein Messaging System für semantisch beschriebene Datenströme.
- Ein Security- und Identity-Management-Service zum Schutz der in der Middleware verarbeiteten Daten.

Aus technologischer Sicht stützt sich die vorgeschlagene Lösung auf die folgenden Entwurfskriterien:

- Kanonisches Metadatenformat: Die Verwendung der Asset Administration Shell (AAS) als Metamodell für die Beschreibung von Assets, ihrer Eigenschaften und Fähigkeiten in einem konsistenten Format, welches die Verwendung von semantischem Markup ermöglicht (DIN SPEC 91345, 2016)
- Semantisches Markup: Für die semantische Anreicherung der Asset-Eigenschaften wird der Standard IEC 61360 (CDD, 2017) verwendet. ECLASS[6] stellt bereits einen Datensatz von ca. 46.000 Konzepten für Produkte und Dienstleistungen aus verschiedenen Anwendungsbereichen zur Verfügung. Darüber hinaus muss die Verwaltung und Bereitstellung von unternehmens- und domänenspezifischen Konzepten (z.B. eine Wartungsontologie) unterstützt werden.
- Identity-Management: Ein zentrales Identity- und Rechtemanagement auf Basis modernster Sicherheitsmechanismen stellt sicher, dass Datenpakete nur für autorisierte Teilnehmer und Anwendungen mit gültigen Zugriffstoken zugänglich sind.

Der Data Integration Layer, seine Komponenten und die ihm Zugrunde liegenden Entwurfskriterien werden im Abschnitt 3 detailliert beschrieben.

2.3 Semantic Integration Patterns

Die Systemarchitektur der semantischen Integrationsplattform („i-Asset Plattform“, s. Abschnitt 3) sieht als Bindeglied zwischen dem Data Integration Layer auf der einen Seite und den Anwendungen und Assets auf der anderen Seite so genannte Connectors („**Application Connectors**“, „**Asset Connectors**“, s. Abbildung 3) vor. Diese Interface-Elemente haben die Funktion, das jeweilige Asset bzw. eine Anwendung im Fertigungsnetzwerk als Industrie 4.0 (I4.0) Komponente für andere Teilnehmer sichtbar zu machen und zu integrieren. Als reaktive I4.0 Komponente werden Maschinen bezeichnet, die Auskunft über ihren Zustand geben können und auch in der Lage sind, Operationen auszuführen. Proaktive I4.0 Komponenten können mit ihrer Umgebung kommunizieren.

Die Application Connectors bilden die Grundlage für die Umsetzung eines zentralen Konzepts von i-Twin, der „**Semantic Integration Patterns**“. Die wesentlichen Kennzeichen dieser Semantic Integration Patterns seien vorab im Überblick dargestellt:

- **Repräsentation von vordefinierten Kommunikationskanälen** der Teilnehmer (Assets, Anwendungen) eines Fertigungsnetzwerks
  Beispiel: *Weiterleitung einer Störmeldung („Alarm“) von einer Anlage an ein Instandhaltungsmanagementsystem (CMMS)*
- **Systemübergreifende Verbindung** verschiedener Anwendungsdomänen
  Beispiel: *Anforderung eines Machine Learning Modells zur Ausführung auf einem Edge-Device*

[6] https://eclass.eu/

- **Semantische Auszeichnung und Spezifikation** sowohl der Kommunikations-Endpunkte (synchron, asynchron) der Teilnehmer eines Fertigungsnetzwerks als auch der ausgetauschten Nachrichten („Message-Payload“)
  Beispiel: *Anzeige der Wartungshistorie auf dem Bedienfeld einer Anlage*
- Verwendung **existierender Industriestandards**
  Beispiel: *IEC 61360, RAMI4.0/AAS*

Asset Connectors bieten eine standardkonforme I4.0-Schnittstelle, die einen sicheren Zugriff auf Echtzeitdaten ermöglicht, aber auch Anfragen zum Aufruf von Methoden akzeptiert. Die Connectors interagieren direkt mit der Steuerung oder dem Edge Controller der Anlage oder verwenden OPC UA oder ähnliche Protokolle, um Status- und Sensorinformationen von der Anlage zu erhalten (z.B. um den Wert einer Eigenschaft zu erhalten oder zu aktualisieren oder um einen Steuerbefehl aufzurufen. Auf die gleiche Weise stellen Application Connectors eine standardkonforme Schnittstelle für Anwendungen zur Verfügung wie Asset Connectors. Sie transformieren sowohl die Anwendungsmethoden als auch die ausgetauschten Daten in die standardisierte I4.0-Welt. Ziel ist es, Methodenaufrufe wieder in die standardisierte I4.0-Welt zu ermöglichen, natürlich unter Anwendung von Sicherheitseinstellungen.

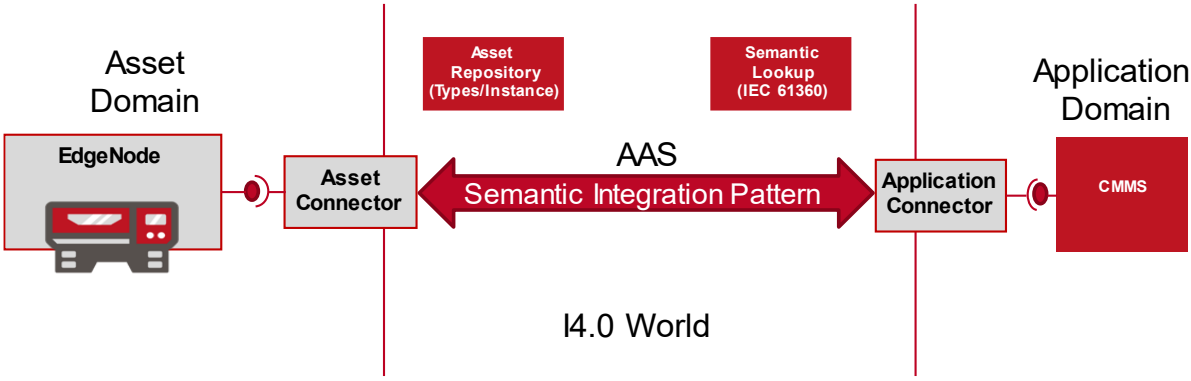


Abbildung 2: Integration von Anlagen und Anwendungen mittels Semantic Integration Patterns

Beide Arten von Connectors erfordern ein AAS-Teilmodell, das die Runtime-API (AAS Part 2, 2023) und, noch wichtiger, die Struktur der ausgetauschten Daten (AAS Part 1, 2023) festlegt. Eine Anlage oder Anwendung, die ihre Informationen auf diese Weise offenlegt, wird als aktive I4.0-Komponente bezeichnet, die einen standardisierten Zugang und standardisierte Daten bietet. Die AAS-Spezifikation liefert die Modellelemente zur Beschreibung der Eigenschaften eines Assets (bzw. einer Anwendung) und seiner Kommunikationsfähigkeiten.

Der von uns vorgeschlagene Ansatz geht über den Stand der Technik aktueller Enterprise Integration Patterns (EIP, https://www.enterpriseintegrationpatterns.com/) hinaus: Die **Semantic Integration Patterns** erweitern traditionelle EIP-Konzept, indem sie die ausgetauschten Daten bzw. die Message-Payload allen Teilnehmern explizit bekannt machen.

Ein Beispiel: Aktive I4.0-Komponenten, die ein Asset repräsentieren, wollen deren Wartungshistorie von einem CMMS abrufen. Damit diese Anfrage erfolgreich bearbeitet werden kann, muss die I4.0-Komponente a) wissen, wie sie mit dem CMMS interagieren kann, b) den genauen Methodennamen einschließlich der erforderlichen Anfrageparameter kennen und c) die Struktur der vom CMMS zurückgegebenen Datensätze der Wartungshistorie kennen. Auf herkömmliche Weise führt dies zu einem massiven Integrationsaufwand für jede benötigte Funktionalität. Außerdem muss bei jeder Änderung einer angeschlossenen Anwendung der Integrationsaufwand wiederholt werden.

Zur Verringerung solcher Integrationshürden dient ein Semantic Integration Pattern: Zunächst wird das AAS-Metamodell verwendet, um die von den Anwendungstypen bereitgestellten, gemeinsam genutzten Funktionen zu definieren. Wie im Beispiel angedeutet, kann die Bereitstellung der Wartungshistorie als eine Funktionalität betrachtet werden, die von den meisten (oder allen) CMMS bereitgestellt wird. Bei der Modellierung des Anwendungstyps für CMMS ist es naheliegend, diese Methode vorzusehen. Durch die Verwendung von Modellvererbung mit dem AAS-Metamodell werden die generischen Definitionen neu definiert und schließlich von einem konkreten CMMS instanziiert. Dies gilt sowohl für die AAS-Modellelemente als auch für das verknüpfte semantische Informationsmodell.

Daher gelten für die Semantic Integration Patterns die folgenden Gestaltungsprinzipien:

- Weitgehende Verwendung von Modellvererbung
- Ausgiebige Verwendung von Typ/Instanz-Beziehungen
- Zwingende Verwendung von semantischen Referenzen für alle Elemente eines Semantic Integration Pattern. Bei Modell-Vererbung zeigt diese Referenz auf das jeweils übergeordnete Element.

Auf diese Weise muss eine I4.0-Komponente lediglich die semantische Referenz der angeforderten Methode kennen. Anhand der semantischen Referenz kann überprüft werden, ob die fragliche Methode aktiviert ist, z. B., ob ein konkretes CMMS-System in die Plattform integriert ist, welche die angeforderte Methode auch anbietet. Falls ein CMMS vorhanden ist, erhält die I4.0-Komponente die vollständigen Angaben zu den Parametern der Methodenanforderung und auch die Antwortdatenstrukturen. Es ist dann möglich, den Request mit Hilfe der Asset Repository Komponente zu erstellen und aufzurufen. Die I4.0-Komponente muss dabei nicht wissen, welches CMMS-System im Einsatz ist oder wo es erreicht werden kann. Durch die semantische Definition jedes ausgetauschten Datenobjekts liegt auch die Antwort der Methode in einer standardisierten, I4.0-konformen Struktur vor. Der semantische Connector transformiert schließlich die ausgetauschten Daten in anwendungsspezifische, typsichere Datenobjekte.

Die Application Connectors bilden die Grundlage für die Umsetzung von Semantic Integration Patterns. Sie übernehmen die aus dem Asset Repository gewonnenen Struktursettings und instanziieren die entsprechenden AAS-Modelle. Application Connectors verwenden die in der instanziierten AAS-Modellstruktur enthaltene „semanticId“ und verwenden die semantische Suche (Semantic Lookup), um die angeforderten Datenstrukturen zu erhalten, um die Daten zu validieren und um die Daten in proprietäre Schnittstellen zu transformieren. Dieser transparente Austausch von strukturellen Datendefinitionen funktioniert für Methodenaufrufe und asynchrone Datenströme. Damit werden die folgenden Gestaltungsprinzipien in Bezug auf die Vernetzung erfüllt:

- Anwendbarkeit in vernetzten Produktionsumgebungen
- Einhaltung bestehender und neuer Industriestandards (insbesondere RAMI4.0), was den Datenaustausch betrifft.
- Unterstützung von datengesteuerten digitalen Zwillingen bei der Integration verteilter Datenquellen ("Mikrodaten-Ökosystem" für Anlagen)

3 i-Asset Plattform

Das Hauptaugenmerk in diesem Bericht liegt auf der Definition von Semantic Integration Patterns. In Abschnitt 2.1 wurden die Anforderungen an Hinsichtlich Datenintegration definiert und eine konzeptionelle Architektur mit einem Data Integration Layer vorgestellt, welcher die Anwendungen (Office-Floor) mit den produzierenden Maschinen (Shop-Floor) verbindet. Eine erste Detaillierung des Data Integration Layer findet sich in Abbildung 3. Dieser Abschnitt erläutert jeden Baustein der i-Asset Plattform hinsichtlich seines Beitrags zur Etablierung von Semantic Integration Patterns.

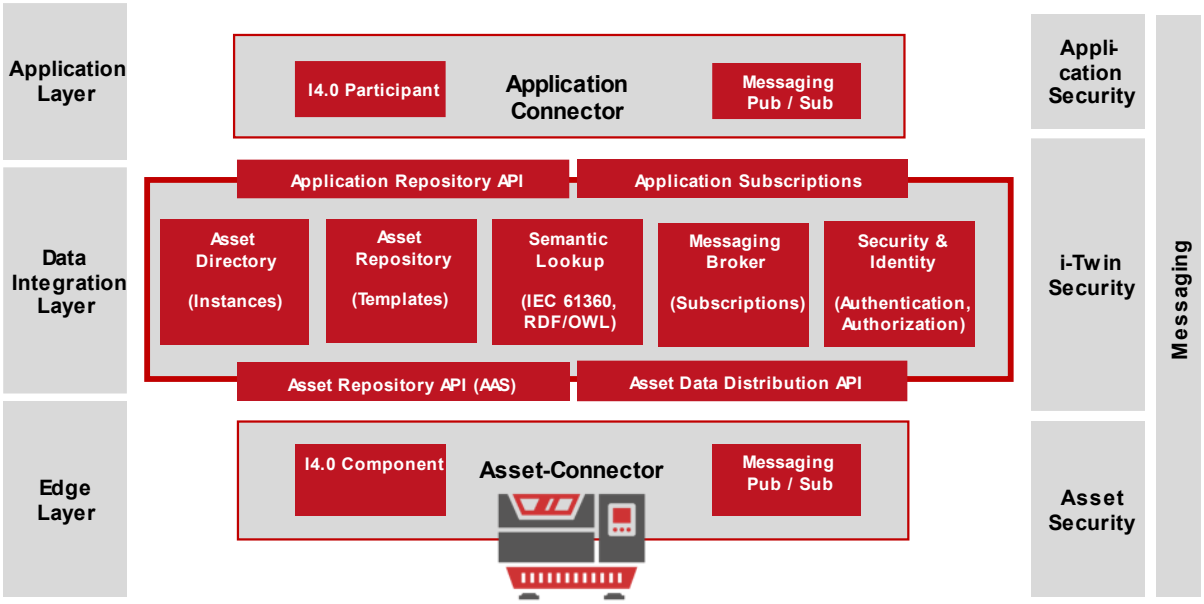


Abbildung 3: i-Twin Data Integration Layer

Für den Data Integration Layer wurden folgenden Hauptbausteine identifiziert:

- Asset Directory: Bildet einen Verzeichnisdienst für die aktiven Asset Instanzen. Diese registrieren sich mit ihren jeweiligen Service-Endpunkten. Zusätzlich werden auch die realisierten Semantic Integration Patterns zu den Asset Instanzen gespeichert.
- Asset Repository: Enthält die Meta-Daten aller verwalteten Assets. Hier werden Asset-Typen (generelle Beschreibungen eines Assets) und Asset-Instanzen (Informationen zu konkreten Maschinen) abgelegt.
- Semantic Lookup (IEC 61360): Stellt zusätzliche, allgemein gültige Informationen über die Asset Meta-Daten bereit. Dies umfasst gültige Wertebereiche oder -listen für Attribute, welche Einheit ein (Sensor)Wert liefert usw. Es können global genutzte Taxonomien wie ECLASS bzw. CDD ebenso verwendet werden wie auch eigene Taxonomien aufgebaut werden.
- Distribution Network (Subscriptions): Mit Hilfe des Data Integration Layer sollen Assets in die Lage versetzt werden, ihre mittels Sensorik erhobenen Daten an beliebige Empfänger zu versenden. Im Distribution Network werden die Datenkanäle verwaltet und die Verbindung zwischen Sender (Sensor) und Empfänger (Anwendungen) hergestellt.
- Security & Identity Management: Dieser Baustein stellt die erforderlichen Security-Mechanismen bereit, um die Kommunikation zwischen den einzelnen Assets bzw. Anwendungen abzusichern.
- Asset Repository API (AAS): Der Zugriff auf gespeicherte Asset-Informationen erfolgt über eine einheitliche Schnittstelle.

- Asset Distribution API: Assets und Anwendungen benötigen (autorisierten) Zugriff auf die für sie relevanten Datenkanäle
- Connectoren: Stellen die Laufzeitumgebung für Asset Instanzen zur Verfügung. Mit Hilfe von Connectoren wird eine Verbindung zu den produzierenden Assets (Edge Layer) und den Anwendungen (Application Layer) hergestellt. Das verbindende Element ist dabei die Asset Administration Shell, welche die Funktionalitäten von Assets aber auch Anwendungen beschreibt. Aus Sicht des Data Integration Layer werden zwei Formen von Connectoren unterschieden:
  - Asset Connector: Mit Hilfe des Asset Connectors können produzierende Maschinen zu I4.0 Komponenten aufgewertet werden. Eine I4.0 Komponente stellt das funktionale Abbild einer Maschine, eines Assets dar und ermöglicht (über einheitlichen Zugriffswege) den Zugriff auf die aktuell an der Maschine anliegenden Informationen.
  - Application Connector: Technologisch ident zum Asset Connector stellt dieser einheitliche Zugriffsmechanismen zu den Informationen, Operationen einer angeschlossenen Anwendung bereit.

Mit dem Konzept des Asset- bzw. dem Application-Connector wird einerseits die Verbindung zwischen Office-Floor (Applications) und Shop-Floor (Assets, Edge Devices) hergestellt und andererseits eine Anlaufstelle für semantische Integration der einzelnen Teilnehmer im System geschaffen.

3.1 Data Integration Layer

Der Data Integration Layer dient als Datendrehscheibe und soll die Kommunikation zwischen Assets und Anwendungen ermöglichen. Dem Data Integration Layer kommt dabei eine Vermittlerrolle zwischen den einzelnen Assets und Anwendungen zu, die dabei transportierten Daten sind für den Data Integration Layer ein „geschlossenes“ Datenpaket, welches nur von den jeweiligen Empfängern eingesehen und interpretiert werden kann. Dennoch benötigt der Data Integration Layer ein umfassendes Datenmanagement, um diese Vermittlerrolle ausfüllen zu können. Für dieses Datenmanagement dient das Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) wie in Abbildung 4 dargestellt.

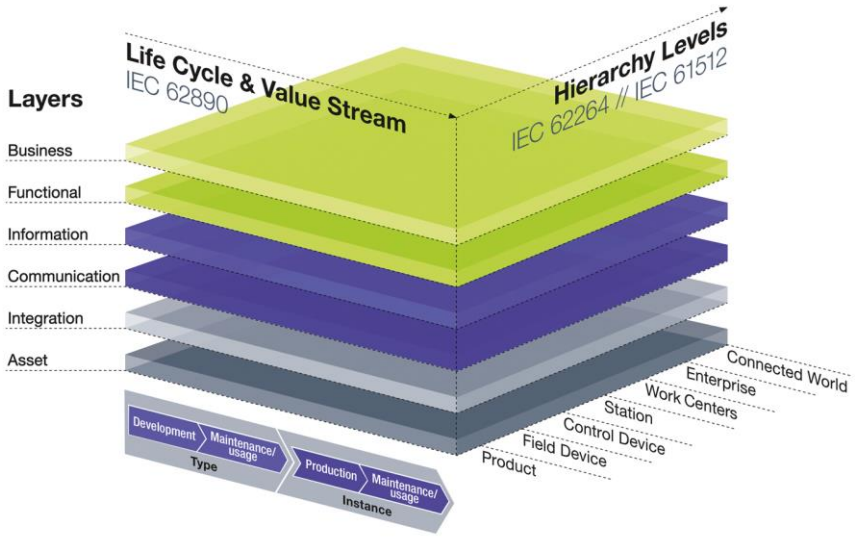


Abbildung 4: RAMI 4.0 Modell, gem. (Heidel, Hoffmeister, Hankel, & Döbrich, 2017)

Aus Sicht des Data Integration Layer RAMI 4.0 ist vor allem die Hierarchie-Ebene (IEC 62264 // IEC 61512) wesentlich. Einzelne, von Assets oder Anwendungen erzeugte Datenpakete müssen zielgerichtet an die jeweiligen Empfänger weitergeleitet werden. Dabei können Datenpakete an einzelne, vollständig identifizierte Empfänger gerichtet sein. Aber es können auch einzelne Nachrichten an mehrere Empfänger einer Hierarchie-Ebene oder eines definierten Typs gerichtet werden. Der Sender kann sich dabei auf das korrekte Versenden der Datenpakete konzentrieren, die Zustellung der Daten übernimmt der Data Integration Layer.

Einige Beispiele für Kommunikationsanforderungen hierfür sind:

- Ein Asset sendet seinen Status-Update an „seine“ Arbeitsstation.
- Ein Asset sendet eine Wartungsanforderung an ein CMMS, da eine geplante Wartung (max. Betriebsdauer erreicht) ansteht.
- Ein Analysetool zur Maschinenoptimierung sendet eine neue Rezeptur für die Betriebsparameter an alle Maschinen eines bestimmten Typs.
- Ein Asset stellt seine Sensor-Daten im System zur Verfügung. Für Predictive Maintenance-Aufgaben werden ausgewählte Sensor-Daten an ein externes Analytics-Tool gesendet.

Diese Liste ließe sich noch beliebig weiterführen. Sie zeigt aber bereits, dass es zur Verwaltung der Kommunikationskanäle die Hierarchie-Ebenen gemäß RAMI 4.0 benötigt, um die Datenströme entsprechend zu benennen und zu befüllen. Im Data Integration Layer übernimmt das **Distribution Network** (siehe 3.1.4) diese Aufgabe und erlaubt die Definition von Unternehmen und dessen Unterteilung in Work Center bzw. Stationen.

Zusätzlich zu den Kommunikationskanälen werden noch weitere Informationen benötigt um sowohl den Sender eines Datenpakets ausreichend zu beschreiben und auch, um letztlich die Semantik des transportierten Datenpakets zu spezifizieren. Das i-Twin Konsortium hat sich für diese Aufgabe auf das Meta-Modell der Plattform-Industrie 4.0, die Asset Administration Shell (AAS), verständigt, welche in Abbildung 5 vereinfacht dargestellt ist.

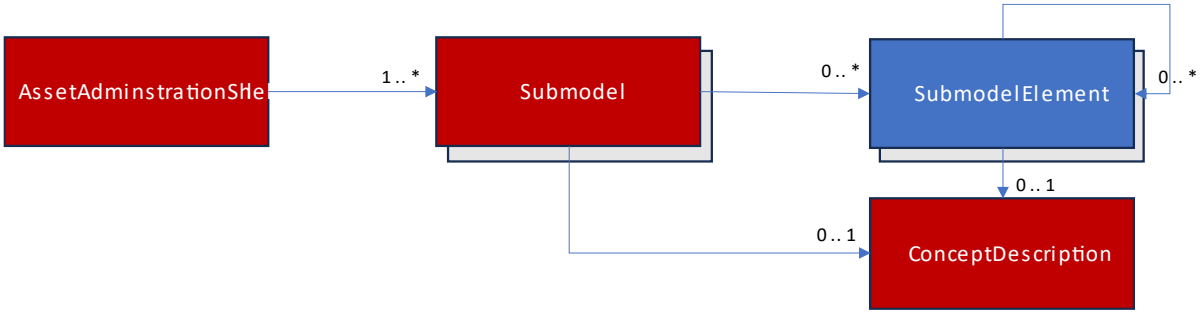


Abbildung 5: Asset Administration Shell - Informationsbausteine

Eine Verwaltungsschale oder `Asset Administration Shell` (AAS) repräsentiert alle Details zu einem `Asset` und stellt somit den obersten Einstiegspunkt für die Verarbeitung von AAS-Informationen dar. Für die Verarbeitung von Detail-Informationen verweist die Verwaltungsschale auf ein oder mehrere Teilmodelle (`Submodel`). Diese wiederum enthalten alle Details in Form von Teilmodell-Elementen (`SubmodelElement`), die innerhalb eines Teilmodells organisiert sind. Teilmodell-Elemente liegen in unterschiedlichen Ausprägungen vor, um die jeweiligen Aufgaben abdecken zu können. Teilmodell-Elemente können durch Konzeptbe-

schreibungen (`ConceptDescription`) zusätzliche Informationen aus allgemein gültigen Taxonomien wie z.B. ECLASS[7] oder dem Common Data Dictionary[8] erhalten. Alle von einem Asset bzw. seinen Teilmodellen verwendeten Konzeptbeschreibungen werden in `ConceptDescription`-Repositories gesammelt und den Asset Instanzen zur Verfügung gestellt. Die Verwaltung und Bereitstellung von Asset Administration Shells ist dabei im **Asset Repository** angesiedelt, wobei für die Einbettung von allgemein gültigen Meta-Informationen (Semantik) das **Semantic Lookup** herangezogen wird.

Jedwede Kommunikation wird mittels Security Protokollen (OAuth2, OpenID Connect) abgesichert. Die hierfür erforderlichen Security Token bzw. die Möglichkeit diese Tokens zu überprüfen, wird durch das **Security & Identity Management** bereitgestellt.

Die einzelnen Bausteine werden nachfolgend noch weiter detailliert beschrieben.

3.1.1 Asset Directory (Instances)

Eine I4.0 Instanz zeichnet sich dadurch aus, dass sie

- identifizierbar ist, d.h. sie ist mir ihrer eindeutigen ID im Netzwerk sichtbar, und
- ihre Daten und Funktionalitäten anhand von standardisierten Schnittstellen (AAS Part 2, 2023) zur Verfügung stellt.

Das Asset Directory erfüllt die erforderliche Vermittlerrolle um aktive I4.0 Instanzen im Verbund anhand des eindeutigen Asset Identifier finden und kontaktieren zu können. Im Directory werden die bereitgestellten (http-REST) Service-Endpunkte für die jeweils aktiven Instanzen verwaltet. Für die Realisierung von Semantic Integration Patterns erlaubt das Asset Directory, Asset Instanzen anhand der „bereitgestellten“ Patterns zu identifizieren.

3.1.2 Asset Repository (Templates)

Dieser Baustein hat die Aufgabe die innerhalb der Plattform verwendeten Asset Administration Shells zu persistieren bzw. diese Daten gemäß den Spezifikationen der Plattform Industrie 4.0 (AAS Part 1, 2023) zur Verfügung zu stellen.

Das Asset Repository verwaltet alle Informationen zu den im Systemverbund vorhandenen Assets. Jedes Asset – das kann eine produzierende Maschine, ein einzelnes Bauteil aber auch eine IT-Anwendung sein – wird dabei durch eine Asset Administration Shell beschrieben. Um eine möglichst effiziente Verwaltung dieser Asset Administration Shells zu erreichen, wird in der Datenmodellierung zwischen Asset Templates und konkreten Asset Instanzen unterschieden, auch Konzepte der Vererbung im Sinne von „ist abgeleitet von“ werden dabei unterstützt. Asset Instanzen können auf diese Weise viele Informationen und vor allem die (Teilmodell)-Struktur vom jeweiligen Asset-Template erhalten. Asset Templates definieren sinngemäß statische Informationen und vor allem die Struktur für die damit abgebildeten Assets.

Asset Instanzen hingegen müssen jedoch nicht nur die Daten und Struktur eines Assets transportieren, sondern aktiv mit weiteren Assets interagieren. Dies bedingt, dass Asset Instanzen als Industrie 4.0 (I4.0) Komponente in einem Systemverbund aktiviert werden. Als I4.0 Komponente wird ein Asset bezeichnet, wenn es seine Informationen und Zustandsdaten gemäß dem Meta-Modell der Plattform Industrie 4.0 (AAS Part 1, 2023) zur Verfügung stellen kann.

[7] ECLASS - https://www.eclass.eu/index.html
[8] Common Data Dictionary - https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet

Eine I4.0 Komponente wird aktiv bezeichnet, wenn es entsprechend der Definitionen in (AAS Part 2, 2023) mit der Außenwelt kommunizieren kann. Dieser Baustein ist in Abschnitt 3.2.1 weiter detailliert.

Das Asset Repository stellt somit das Meta-Daten-Management für Asset Typen bereit. I4.0 Komponenten erhalten vom Asset Repository jene Informationen, die sie für die Kommunikation mit anderen Teilnehmern/Asset Instanzen benötigen. Für die konsistente Beschreibung der jeweiligen Asset Administration Shells ist eine semantische Einordnung der einzelnen Elemente innerhalb einer Asset Administration Shell unabdingbar. Das Meta-Modell sieht hierfür eine Integration von gemeinsam genutzten Taxonomien bzw. Vokabularen vor. Diese Informationen werden in einem eigenen Semantic Lookup Service innerhalb des Data Integration Layers verwaltet, welcher zusätzliche semantische Informationen (verschiedene Bezeichner, Datentyp, Einheit, Wertebereiche) zu den Elementen der Asset Administration Shell bereitstellt. Das Asset Repository hat dabei die Aufgabe, in der Anfragebeantwortung auch diese gemeinsam genutzten Informationen in die Antwort zu integrieren. Daher ergibt sich für die Architektur des Asset Repository das in Abbildung 6 dargestellte Bild.

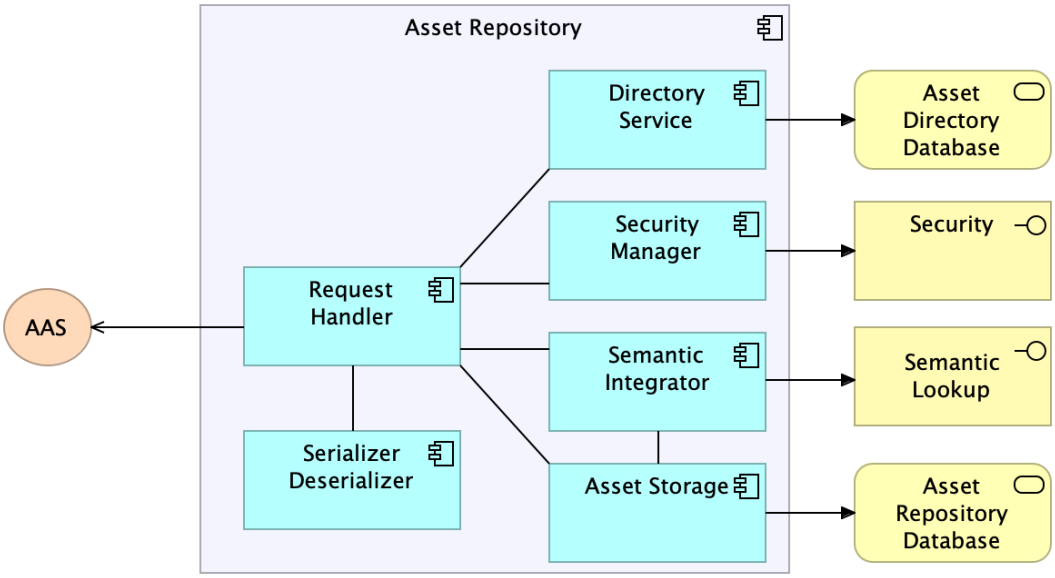


Abbildung 6: Asset Repository (Components)

Die innere Struktur des Asset Repository Service ergibt sich wie folgt:

Request Handler: Dieser Baustein stellt die API gemäß Interaktionsmodell zur Verfügung (AAS Part 2, 2023). Alle Anfragen werden zunächst vom Security-Manager geprüft und bei POST/PUT Befehlen in das interne, speicherbare Datenmodell des Asset Storage transformiert und dort gespeichert. Bei GET-Befehlen hat dieser Baustein die Aufgabe, die gesuchten Elemente zu identifizieren und dem AAS Meta-Modell entsprechend zu exportieren. (AAS Part 1, 2023)

- Serializer/Deserializer: Kapselt die Transformation der ein- bzw. ausgehenden Daten in das speicherbare Modell.
- Directory Service: Speichert die Service Endpoints der aktiven I4.0 Komponenten bzw. Asset Instanzen
- Asset Storage: Stellt die Persistenz für das Asset Repository zur Verfügung.
- Semantic Integrator: Kommuniziert mit dem externen Semantic Lookup Service und fügt die mittels `semanticId` referenzierten Zusatzinformationen in die Anfragebeantwortung ein.

- Security Manager: wird bei der Anfragebeantwortung konsultiert, um die Authentifizierung und auch Autorisierung zu überprüfen. Zu diesem Zweck wird das Security & Identity Management konsultiert.

Für die weitere Entwicklung des Asset Repository Moduls ist nun das interne Datenmodell für die Asset Administration Shell von Bedeutung. Mit Hilfe dieses Modells können die Details der Asset Administration Shell, unabhängig ob Typ oder Instanz, in der Datenbank abgelegt werden. Das Modell ist somit auch für die funktionale Sicht auf ein Asset im Sinne einer I4.0 Komponente von Bedeutung.

3.1.3 Semantic Lookup (IEC 61360)

Die dabei ausgetauschten Daten müssen immer vollständig sein, damit ein Asset, eine Anwendung alle benötigten Informationen für die Verarbeitung und Interpretation der Daten enthält, dies betrifft primär die Elemente `ConceptDescription` wenn diese als Ziel einer semantischen Referenz definiert sind. `ConceptDescriptions` definieren zusätzlich eine weitere Eigenschaft `isCaseOf` um die vollständige semantische Beschreibung des Elements zu benennen. Die Verwaltung der vollständigen, semantischen Beschreibung erfolgt mit Hilfe des Semantic Lookup Services innerhalb des Data Integration Layer.

In der i-Asset Registry sind unterschiedliche Elemente mit der Eigenschaft `HasSemantics` bzw. auch mit der Eigenschaft `HasDataSpecification` versehen. `HasSemantics` dient dabei zur Klassifikation dieser Elemente mit global verfügbaren Systemen wie ECLASS bzw. dem IEC Common Data Dictionary (CDD). `HasDataSpecification` wird in der i-Asset Plattform als Template für relevante Eigenschaften verwendet.

Mit dem *Semantic Lookup Service* steht eine Komponente zur Verfügung um die Daten von ECLASS bzw. CDD zu speichern bzw. auch um eigene Konzepte zu definieren und anderen zur Verfügung zu stellen.

Das Datenmodell für das Semantic Lookup Service orientiert sich am Schema für ECLASS und ist in Abbildung 7 dargestellt.

Die einzelnen Entitäten sind nachfolgend kurz beschrieben:

- Klassifikationsklassen (`concept_class`): Dem Namen entsprechend werden hier die Klassen für Produkte bzw. Services eingetragen. Im ECLASS Standard sind bereits eine Vielzahl von Klassifikationen definiert. Die Einträge sind dabei hierarchisch angeordnet wobei jede Hierarchiestufe eine Verfeinerung des übergeordneten Begriffs darstellt. Über eine Beziehungsrelation sind auch gängige Eigenschaften zur jeweiligen Klasse definiert.
- Eigenschaften (`concept_property`): In dieser Tabelle werden bekannte Eigenschaften detailliert beschrieben. Es wird dabei festgelegt, welcher Datentyp der Eigenschaft zugrunde liegt bzw. ob es für diese Eigenschaft auch eine definierte Einheit gibt.
- Einheiten (`concept_property_unit`): Enthält existierende Einheitensysteme (Grad Celsius, Meter etc.).
- Werte (`concept_property_value`): Enthält eine Werteliste für Eigenschaften. Als vereinfachtes Beispiel: Eine Eigenschaft vom Datentyp „BOOLEAN“ kann nur die Werte Wahr oder Falsch aufnehmen.

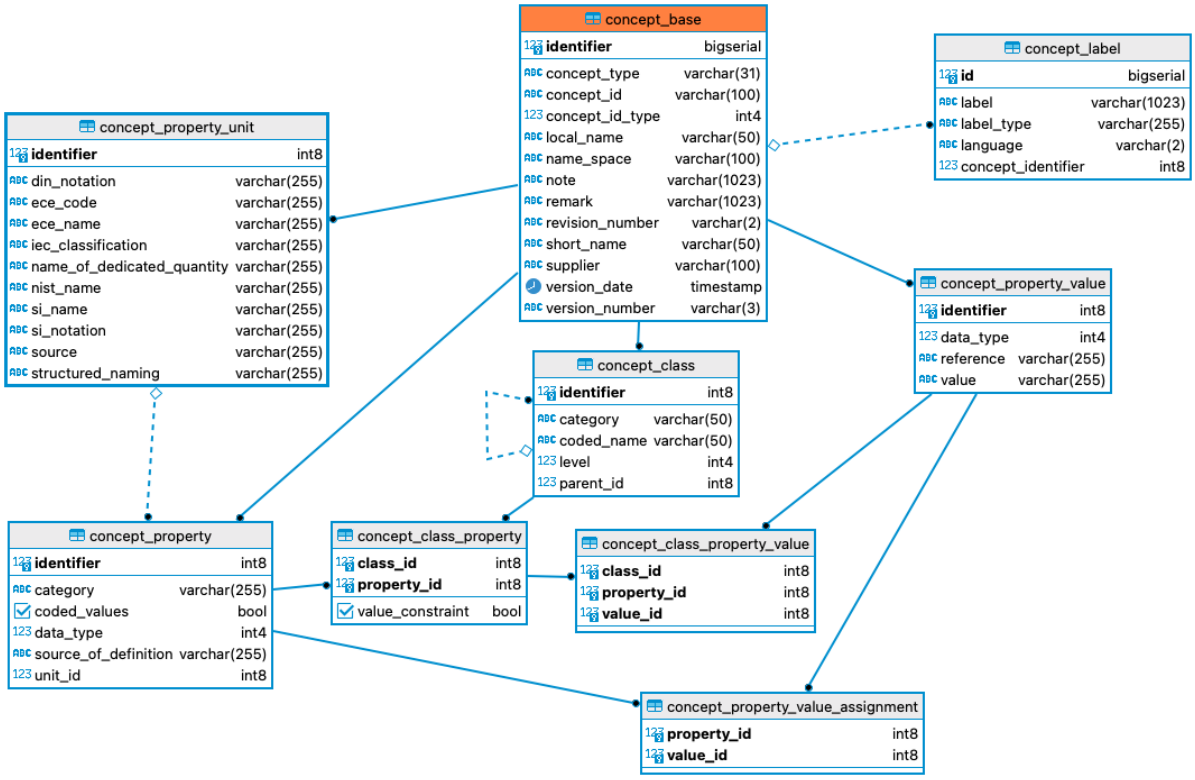


Abbildung 7: Data Specification Service Datenmodell

Mittels Beziehungsrelationen werden Klassen mit Eigenschaften bzw. auch erlaubten Wertelisten kombiniert. Folgende Beziehungsrelationen sind im Schema definiert:

- Eigenschaften zu Klassen (`concept_class_property`): Anhand dieser Beziehung wird festgelegt, welche Attribute für eine Klasse zulässig/bekannt sind.
- Werte zu Eigenschaften (`concept_property_value_assignment`): Vordefinierte Wertelisten, z.B. RAL-Codes. Die Werteliste ist den Eigenschaften direkt zugeordnet.
- Werte zu Eigenschaften (`concept_class_property_value`): Ermöglicht die klassenabhängige Definition einer Werteliste zu Eigenschaften.

3.1.4 Distribution Network (Subscriptions)

Ein Ziel der System-Architektur ist es, die Kommunikation zwischen Assets und Anwendungen möglichst einfach zu gestalten, indem sie über einen Data Integration Layer miteinander verbunden werden. Das Distribution Network orientiert sich an der von RAMI4.0 vorgegebenen Hierarchie-Achse (IEC 62264) und die organisiert die Kommunikationskanäle als in abstrakten logischen Systemen, welche unter den Überbegriffen „Domain“ und *„Enterprise“* angesiedelt sind. Ein System deckt dabei die RAMI 4.0 Hierarchie-Ebenen *„Workcenter“* und „*Station*“ ab (vgl. Abbildung 4 auf Seite 11). Die physischen Assets und deren digitalen Repräsentanten, die I4.0 Komponenten stellen nach RAMI4.0 das „Control Device“ dar und kapseln die einzelnen Field-Devices anhand der Asset Administration Shell – siehe Abschnitt 3.2.1 zur Beschreibung des Asset- bzw. Application-Connectoren und ihren Kommunikationsaufgaben. I4.0 Komponenten können sinngemäß Nachrichten senden und empfangen. Das Distribution Network als Baustein des Data Integration Layer hat die Aufgabe, die dafür genutzten Datenkanäle bereit zu stellen und gleichzeitig die Kontrolle über die Zustellung einzelner Nachrichten zu behalten.

Um diese Kontrolle zu gewährleisten, ist zunächst ein Datenmodell erforderlich, welche diese Entitäten Enterprise, System und auch die partizipierenden I4.0 Komponenten umfasst. Hinzu kommen weitere Entitäten zur Definition von Datenströmen und deren Subscriptions. Abbildung 8 zeigt das logische Daten- bzw. Objektmodell.

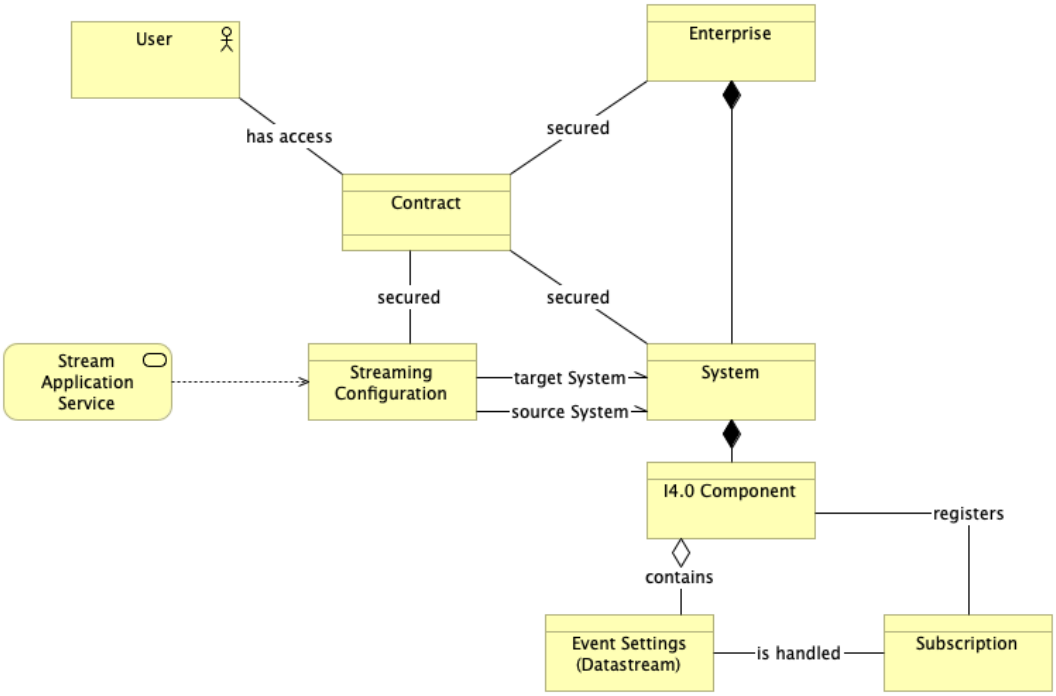


Abbildung 8: Logische Ansicht des Objektmodells des Distribution Network

Bei diesem Modell kommen folgende Überlegungen zum Tragen:

- Das Distribution Network ist ein eigenes Micro-Service mit eigener Datenhaltung. Querverbindungen zu anderen Bausteinen des Data Integration Layer werden anhand der jeweiligen Identifier hergestellt.
- In der Entität `User` werden die einzelnen Nutzer aus dem Security & Identity Management repräsentiert bzw. bereitgestellt. Mittels eines Security-Contracts werden die Zugriffsrechte des Benutzers auf `Enterprise`, `System` und `Stream Application` geregelt.
- Die Entitäten `Enterprise` und `System` sowie auch I4.0 Component ermöglichen die Abbildung der Hierarchie gem. RAMI 4.0.
- Eine `Streaming Configuration` repräsentiert Regeln für Filtering und Daten-Weiterleitung, wenn I4.0 Komponenten zweier unterschiedlicher Systeme Daten austauschen wollen.
- Innerhalb einer `I4.0 Component` sind die Event-Einstellungen (AAS: `EventElement`) definiert. Diese Entität liefert die verfügbaren Datenströme.
- Eine Anwendung oder ein Asset können sich nun auf Datenströme registrieren. Dies wird in der Relation `Subscription` abgelegt.

Anhand dieser Überlegungen wird das Datenmodell des Distribution Network verfeinert und somit die Grundlage für die Kommunikation zwischen den einzelnen I4.0 Komponenten geschaffen.

3.1.5 Security & Identity Management

Die Sicherheit der innerhalb der Plattform verwalteten Daten ist zu gewährleisten – Zugriffe auf die Daten dürfen nur autorisierten Benutzern und bekannten Anwendungen gewährt werden. Zudem dürfen einzelne Benutzer nur jene Daten einsehen und bearbeiten, für die sie berechtigt sind.

Authentifizierung und Autorisierung

Zur Sicherstellung dieser Anforderung verwendet die i-Asset Plattform zur Authentifizierung und Autorisierung den OpenID Connect[9] Standard. Hierzu wird mit Keycloak[10] ein Open Source Identity und Access Management integriert. Es können Benutzer verwaltet werden, die mit dem System interagieren (Authentifizierung) dürfen. Eine Nutzung der Anwendung ist nur nach vorheriger Anmeldung möglich. Die Benutzer sind zudem noch mit Benutzer-Rollen versehen, womit auch der Zugriff auf einzelne angebotene Funktionen geprüft wird (Autorisierung).

Damit eine gesicherte Kommunikation mit der i-Asset Plattform erfolgen kann, müssen teilnehmende Anwendungen zunächst der i-Asset Plattform aktiv bekannt gemacht werden. Hierzu wird die Anwendung im Security-Management der i-Asset Plattform als (OAuth2-)Client eingetragen. Damit wird der erforderliche Sicherheits-Key erzeugt, der für die weitere Kommunikation mit der i-Asset Plattform erforderlich ist.

Alle Micro-Services der i-Asset Plattform prüfen letztlich den Security-Token und können so die Identität des Nutzers wie auch dessen Security-Einstellungen überprüfen.

Schutz der Daten vor unerlaubtem Zugriff

Das Meta-Modell der Verwaltungsschale definiert verschiedene `Security`-Elemente vor um einen umfassenden, datenbasierten Zugriffsschutz umsetzen zu können. Siehe dazu die Spezifikationen der Plattform-Industrie[11] (AAS Part 1, 2023). Zusätzlich zum Zugriffschutz steht mit der Eigenschaft `Qualifiable` die Möglichkeit zur Verfügung, ein Teilmodell oder Teilmodell-Element mit eigenen Access-Regeln (`Constraint`) zu versehen.

3.2 Application / Edge Layer

Der Data Integration Layer dient als Vermittler zwischen Anwendungen und Assets. Entsprechend der Abbildung 3 kommunizieren Anwendungen und Assets mittels Data Integration Layer miteinander.

In den Abschnitten 3.1.1 (Request/Response) und 3.1.4 (Messaging) wurden die Kommunikationsanforderungen für die Bereitstellung von Informationen im Format der Asset Administration Shell erörtert. Das Asset Repository hat hier die Aufgabe, die Asset Administration Shell für alle verbundenen I4.0 Komponenten zu verwalten. Hierzu „registrieren“ sich alle aktiven I4.0 Komponenten im zentralen Asset Repository und geben ihren jeweiligen Service Endpoint bekannt. Das ist jene Service-Adresse, an der sie die AAS-Informationen bereitstellen. Das

[9] https://openid.net/connect/
[10] https://www.keycloak.org/
[11] https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.pdf

zentrale Asset Repository kennt somit zu jedem aktiven Asset die zugehörige Service-Adresse, kann also direkt mit dem „physikalischen“ Asset kommunizieren und entsprechende Anfragen an das Asset oder auch an eine Anwendung stellen. Dazu muss diese mit der entsprechenden Connectivität ausgestattet werden.

3.2.1 Asset-/Application-Connector

Der Asset- bzw. Application-Connector stellt das Bindeglied zwischen dem Data Integration Layer und den jeweiligen, angeschlossenen Anwendungen und Assets dar. Dieser hat dabei die Aufgabe das jeweilige Asset als Industrie 4.0 Komponente für andere Teilnehmer sichtbar zu machen. Eine I4.0 Komponente wird sinngemäß auch als die „funktionale Erweiterung“ der Asset Administration Shell verstanden.

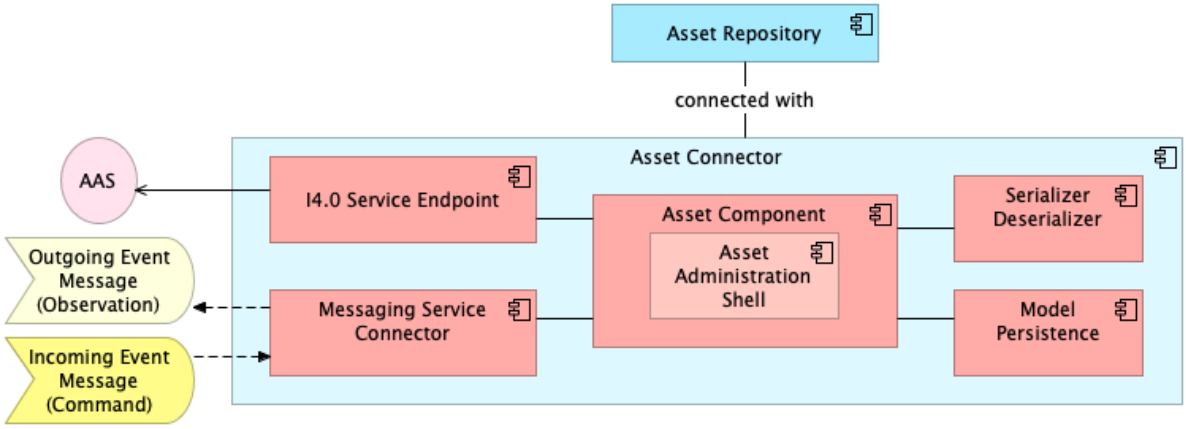


Abbildung 9: Asset Connector Sub-Components

In Abbildung 9 sind die vorgesehenen Sub-Komponenten des Assets Connectors ersichtlich, wobei gilt:

- Asset Component: Dieser Baustein bildet das funktionale Rückgrat eines Asset Connectors. Dieser wird mit einer `AssetAdministrationShell` konfiguriert.
- I4.0 Service Endpoint: Dieser Baustein hat die Aufgabe, alle Informationen für die in der Asset Component geladene `AssetAdministrationShell` gemäß den im Interaktionsmodell der Plattform Industrie 4.0 definierten Methoden bereitzustellen oder zu konsumieren (AAS Part 2, 2023). Die Darstellung der strukturierten Informationen muss dem Meta-Modell der Platform Industrie 4.0 entsprechen (AAS Part 1, 2023).
- Serializer/Deserializer: Transformiert die ein- bzw. ausgehenden Informationen vom Meta-Modell der Plattform Industrie 4.0 in das interne AAS-Modell. Das interne Modell enthält orientiert sich am Meta-Modell der Plattform Industrie 4.0, enthält jedoch wichtige funktionale Ergänzungen, um Daten vom Asset in die Anfragebeantwortung zu integrieren bzw. auch um Befehle an der Maschine ausführen zu können.
- Messaging Service Connector: Dieser Baustein bildet die Brücke zum Distribution Network. Für die Event-Elemente innerhalb einer Asset Administration Shell bzw. Teilmodell werden entsprechende Producer- bzw. Consumer-Methoden aktiviert.
- Model Persistence: Eine Asset Administration Shell ist zur Laufzeit in der Asset Component geladen, also nur im flüchtigen Speicher vorhanden. Im Sinne einer Persistenz muss dieses auch abgespeichert/exportiert werden können, um es an gleicher oder auch anderer Stelle erneut zu verwenden. Die Speicherung des Modells kann im File-System oder auch im Asset Repository erfolgen.

Um ein Asset als I4.0 Komponente zu repräsentieren und für andere I4.0 Komponenten sichtbar zu machen, sind folgende Schritte erforderlich:

1. Die Asset Instanz muss in der Asset Component geladen werden. Die Instanz kann dabei bereits lokal vorliegen, als AASX Datei oder als JSON-Dokument. Eine weitere Option ist, die Instanz auf Basis eines Asset Typs zu erstellen, der Typ wird hier vom Asset Repository bereitgestellt.
2. Die Asset Component bietet die Möglichkeit, für variable Eigenschaften (`Property`-Elemente) und `Operation`-Elemente eine Verbindung zur Steuerung des Assets herzustellen. Anhand von `Event`-Elementen wird die asynchrone Kommunikation gesteuert.
3. Für eine geladene Asset Administration Shell wird ein HTTP Service Endpoint aktiviert. Die Komponente ist nun für andere Teilnehmer im Verbund verfügbar, die Endpoint-Adresse muss jedoch im Data Integration Layer „bekannt“ gemacht werden.
4. Der `Descriptor` des Assets muss im Data Integration Layer „registriert“ werden. Der Data Integration Layer kann ab diesem Moment den Identifier des Assets mit seiner Service-Adresse verbinden.
5. Der Messaging Service Connector wird aktiviert. Dieser durchsucht die AAS nach `Event`-Elementen und richtige entsprechende Message Producer bzw. Message Consumer ein.

Abbildung 10 visualisiert die einzelnen Schritte, um eine Asset Instanz zu aktivieren. Die Zugriffe auf die Steuerung des solcherart aktivierten Assets ist abhängig von der Steuerung und kann ggf. über OPC UA oder andere Mechanismen erfolgen.

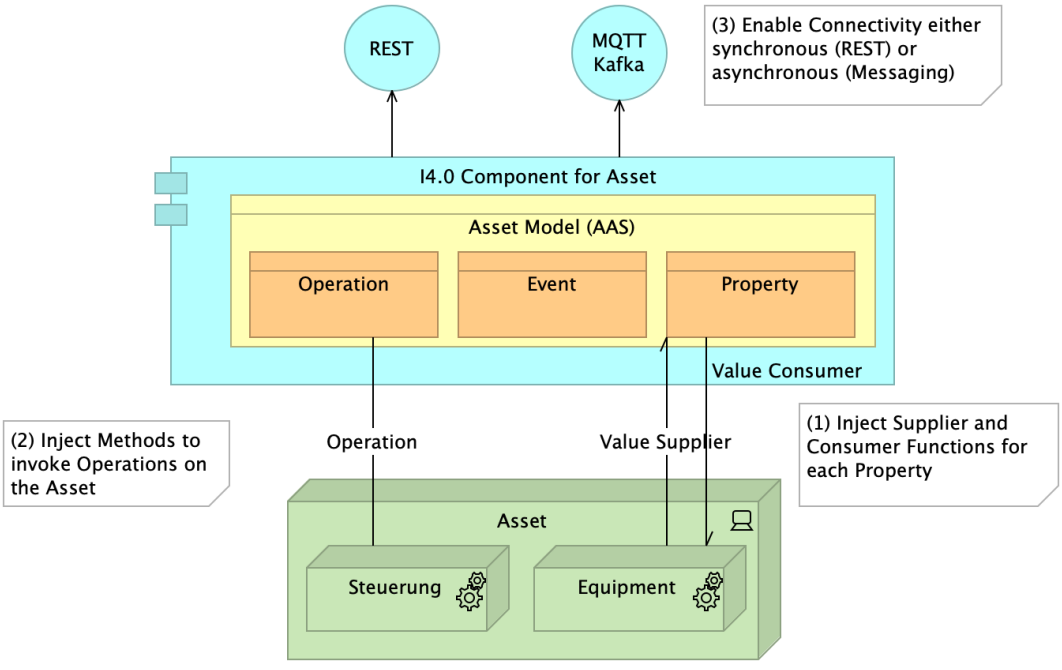


Abbildung 10: I4.0 Komponete – Aktivierung

Die I4.0 Komponente kann nun seine Struktur und auch seine Daten gemäß der Spezifikation in zur Verfügung stellen, wobei die REST API dem Interaktionsmodell entsprechen muss.

4 Semantic Integration Patterns

Mit zunehmender Vernetzung von Anwendungen und Assets steigt der hierfür erforderlich Integrationsaufwand. Ohne eine zentrale Anlaufstelle für Kommunikation und Semantik müssen alle Anwendungen eine direkte Verbindung zum Kommunikations-Partner aufbauen und aufrechterhalten. Mit steigender Vernetzung steigt auch der dafür erforderliche Integrationsaufwand. In Abbildung 11 sind Asset Management, Maintenance und Analytics als typische Vertreter von „zu integrierenden“ Anwendungen genannt.

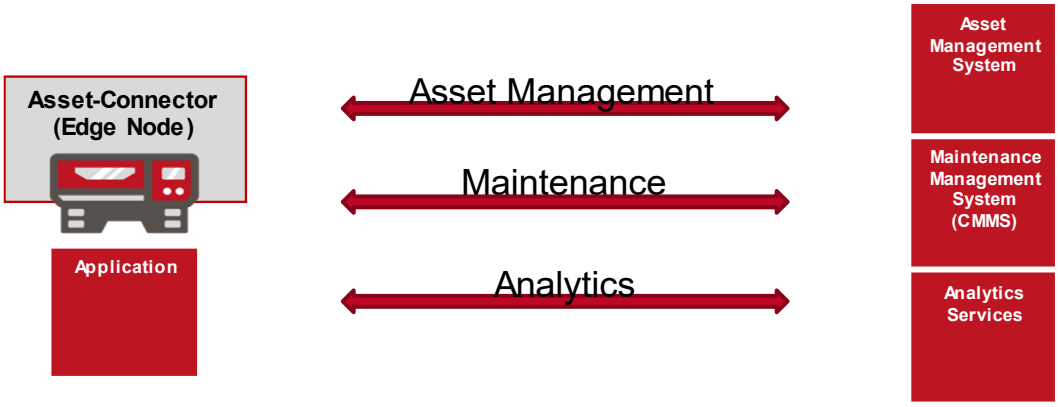


Abbildung 11: Kommunikationsanforderungen in vernetzten Produktionsumgebungen

Semantic Integration Patterns adressieren genau diese Problematik, in dem sie die vorhandenen Konzepte des in Abschnitt 3.1 beschriebenen Data Integration Layers nutzen und vordefinierte Kommunikations-Wege semantisch auf eine solche Art beschreiben, dass Anwendungen nur mehr die bestehenden Kommunikationswege nutzen, ohne dabei zu wissen zu müssen, mit wem sie sich gerade austauschen. Mit Hilfe der Semantic Integration Patterns sollen Anwendungen sich darauf verlassen können, dass eine Kommunikation immer den jeweiligen Anforderungen genügt, da die ausgetauschten Daten vom Semantic Integration Layer auf ihre Gültigkeit geprüft werden. Diese Form der Anbindung ist Abbildung 12 dargestellt.

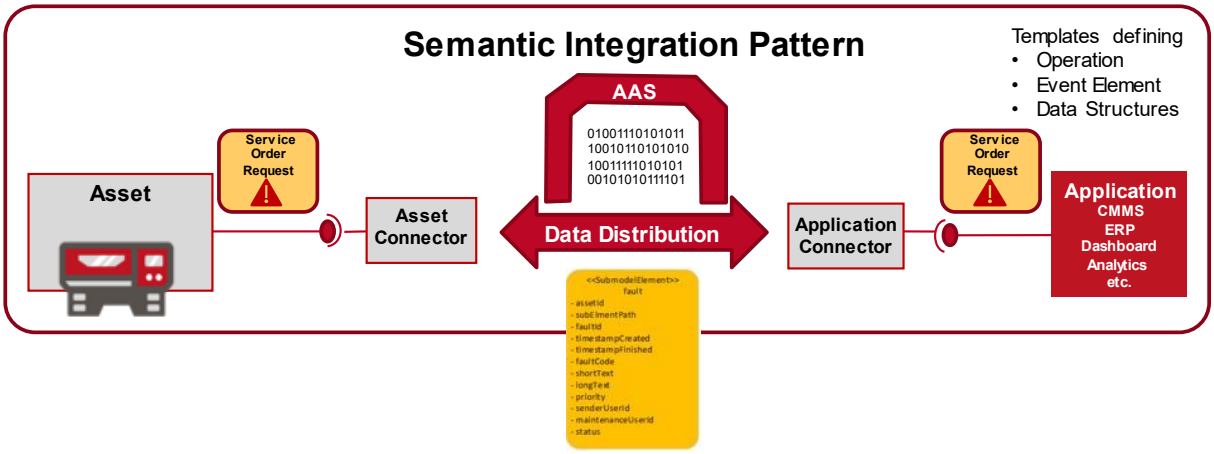


Abbildung 12: Integration von Anlagen und Anwendungen mittels Semantic Integration Patterns

Das Meta-Modell der Plattform Industrie 4.0 in Kombination mit dem Interaktionsmodell zur Kommunikation mit Industrie 4.0 Komponenten stellt hierfür die erforderliche Basis bereit, zielt jedoch vor allem darauf ab, (Produktions-)Maschinen und ihre Datenpunkte zu beschreiben.

Sensor-Daten, Maschinenzustände werden dadurch in einem einheitlichen Datenformat mit einer standardisierten Schnittstelle den unterschiedlichen Anwendungen zugänglich gemacht.

In typischen Umgebungen sind jedoch nicht nur Assets zu finden, sondern auch verschiedenste Anwendungen, die auf Basis von Sensor-Daten, Maschinenzuständen etc. ihre Funktionen bereitstellen.

Semantic Integration Patterns standardisieren diese höherwertige Kommunikationsaufgaben, die in vernetzten (Produktions-)Umgebungen zwischen Anwendungen einerseits aber auch zwischen Assets und Anwendungen erforderlich sind.

4.1 Semantic Integration Patterns - Datenobjekte

Ein Semantic Integration Pattern stellt Anwendungen und Assets vordefinierte Anknüpfungspunkte zur Erledigung ihrer Kommunikationsaufgaben zur Verfügung. So möchte beispielhaft ein Asset, eine Maschine im Störfall eine entsprechende Meldung absetzen. Um dies erfolgreich erledigen zu können, muss bekannt sein

- welches Instandhaltungsmanagement-System (CMMS) überhaupt in Verwendung ist,
- wie das CMMS erreicht werden kann,
- wie eine gültige Störmeldung formuliert sein muss, welche Störursachen im CMMS akzeptiert werden,
- mit welcher Kennung das Asset im CMMS registriert ist.

In herkömmlichen Umgebungen muss ein Operator das Web-, Tablet- oder App-Interface des CMMS-Systems nutzen, und seine Anlage zunächst im CMMS suchen und die entsprechenden Informationen eingeben.

Semantic Integration Patterns überwinden diese Integrationshürde und bieten einer Maschine vordefinierte Kommunikationswege an, die von Assets und Anwendungen genutzt werden können. Ein Asset, eine Maschine wird somit in die Lage versetzt, eine Störmeldung absetzen zu können, ohne dabei vorab wissen zu müssen, welches CMMS im Einsatz ist. Das Pattern übernimmt auch die Validierung der Störmeldung und stellt die Gültigkeit einer Nachricht sicher.

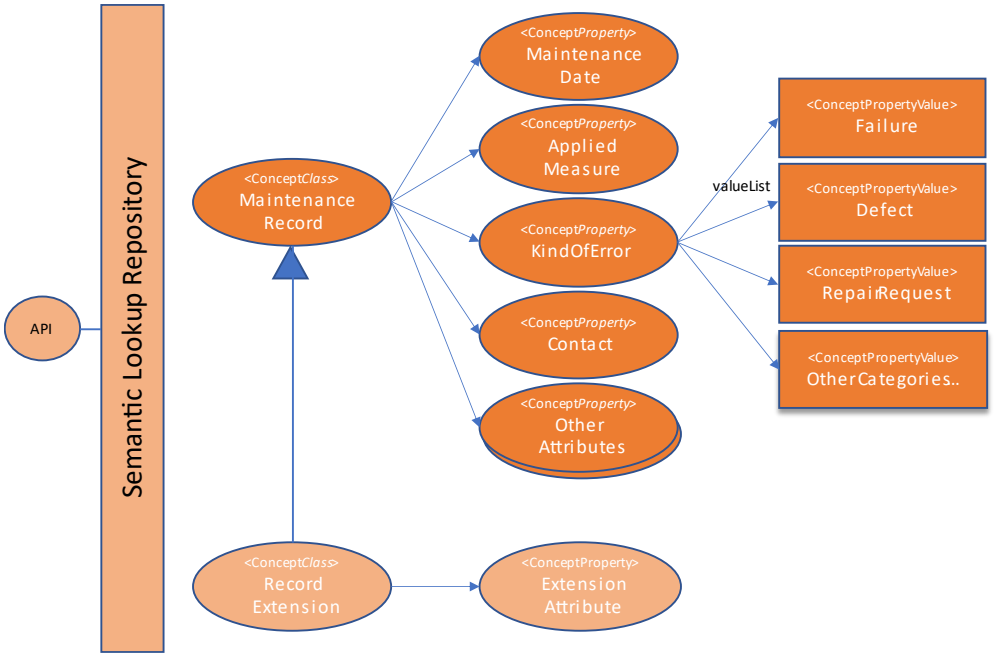


Abbildung 13: Datenobjekte & Vererbung

Für eine umfassende Validierung aller gesendeten Daten müssen alle ausgetauschten Datenobjekte vollständig beschrieben sein. Diese Beschreibung der Datenobjekte erfolgt nicht direkt im Asset Repository, sondern im Semantic Lookup Service, welches dem IEC 61360 Standard folgt. Auf diese Weise können die Definitionen von Datenobjekten ausgetauscht werden. Zudem bietet der IEC 61360 Standard das Konzept der Vererbung. Es können Basis-Datenobjekte definiert werden und mittels „abgeleiteten Datenobjekten“ erweitert werden. Abbildung 13 zeigt das Prinzip der Vererbung im Semantic Lookup Repository. Der Basis-Eintrag „Maintenance Record“ zur Beschreibung von durchgeführten Wartungsaktivitäten besitzt verschiedene Attribute. Ein abgeleitetes Datenobjekt erbt alle Attribute des Parent-Objekts und fügt weitere Attribute hinzu.

Auf diese Weise definierte Datenobjekte müssen jedoch im Kontext einer Asset Administration Shell abgebildet werden. Hierfür gilt, dass eine Konzept-Klasse im AAS Meta-Modell ein `Submodel`, oder eine `SubmodelElementCollection` sein kann. Eigenschaften werden im AAS Meta-Modell als `Property` definiert. Da `Submodel`, `SubmodelElementCollection` und auch Property die Eigenschaft `HasDataSpecification` aufweisen, kann mit den Mitteln des AAS Meta-Modells diese Beziehung abgebildet werden wie in Abbildung 14 dargestellt.

`HasDataSpecification` erlaubt somit die Erweiterung der AAS Elemente mit „Data Specification Templates“. Im AAS Meta-Modell sind bereits Templates für IEC 6130 kompatible Meta-Daten enthalten. Da alle Ausprägungen von `SubmodelElement` diese Erweiterungsmöglichkeit aufweisen, kann mit Hilfe des AAS Meta-Modells die vollständige semantische Zusatzinformation in diesen Elementen abgelegt werden. Für die saubere Trennung von Daten und Semantik empfiehlt sich die Verwendung einer `ConceptDescription`. Eine `ConceptDescription` ist „identifizierbar“, kann somit als Ziel einer `semanticId`-Referenz in Teilmodell-Elementen verwendet werden. Eine `ConceptDescription` definiert weiter mit `isCaseOf` die Herkunft der externen Information. Abbildung 14 zeigt die entsprechenden Elemente gemäß AAS Meta-Modell.

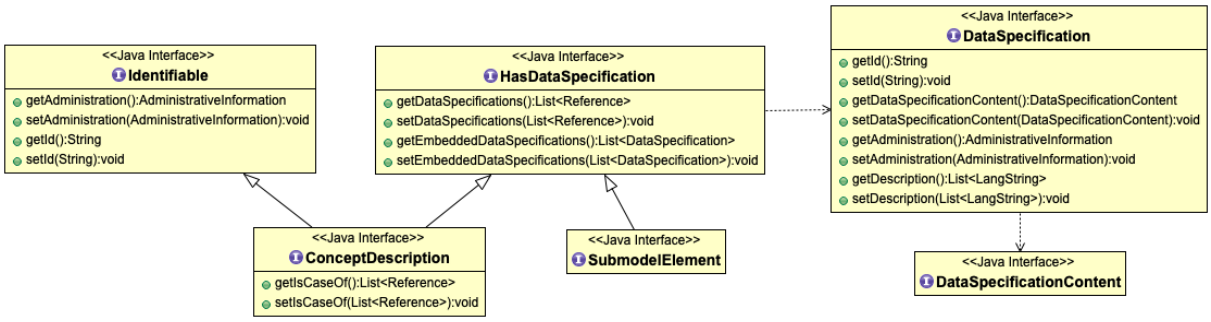


Abbildung 14: Semantic Integration

Operation und BasicEventElement verweisen AAS-Elemente verweisen schließlich per `semanticId` auf eine `ConceptDescription`, die wiederum als Abbild einer externen Definition im Semantic Lookup Service zu verstehen ist.

4.2 Definition der Kommunikation

Semantic Integration Patterns haben das Ziel, erforderliche Funktionen und Methoden, die von einzelnen Anwendungen angeboten werden, auf möglichst einfache Art und Weise für Assets und Anwendungen zugänglich zu machen. Ein wichtiges Element für die Kommunikation zwischen Assets und Anwendungen ist, dass die ausgetauschten Daten den Vorgaben entsprechen. Ein weiteres Kriterium sind jene Elemente, welche die Kommunikation zwischen Asset und Anwendung steuern.

Das Meta-Modell der Plattform Industrie 4.0 enthält hierfür Teilmodell-Elemente für synchrone und asynchrone Kommunikation:

- Synchrone Methodenaufrufe: Ausführbare Funktionen eines Assets bzw. einer Anwendung werden mit Hilfe des Teilmodell-Elements `Operation` definiert. Ausführbare Funktionen erfordern Eingabeparameter. Das Ergebnis eines Methodenaufrufs ist ebenfalls von Interesse für einen Aufrufer. Semantic Integration Patterns zielen darauf ab, diese Eingabe- und Ausgabeparameter vollständig zu definieren. Teilnehmer, die mit Hilfe eines Connectors Funktionen nutzen wollen, erhalten anhand des Semantic Integration Patterns eine vollständige Beschreibung der Methode sowie der dabei ausgetauschten Daten.
- Asynchrone Kommunikation: Ereignisse werden mit Hilfe des Teilmodell-Elements `BasicEventElement` definiert. Dieses kontrolliert, auf welche Ereignisse ein Asset bzw. eine Anwendung reagieren soll bzw. welche Ereignisse diese selbst auslösen kann. Wesentlich ist hierbei, dass der Nachrichteninhalt im `BasicEventElement` definiert ist und dieser Nachrichteninhalt ist im Semantic Integration Pattern vollständig definiert.

Beiden Elementen ist gemeinsam, dass sowohl für Methoden-Aufrufe wie auch für Ereignisse die erforderlichen Datenelemente bekannt sind und diese mittels `semanticId` derart ausgezeichnet sind, dass die auszutauschenden Daten auf ihre Gültigkeit validiert werden können und so ein gesicherter Datenaustausch etabliert wird.

Die funktionale „Berücksichtigung“ eines Semantic Integration Pattern erfolgt im Asset bzw. Application Connector.

4.3 Identifikation von Semantic Integration Patterns

Die Datenaustausch-Objekte, Kommunikationseinstellungen die letztlich von Anwendungen realisiert werden und von Assets bzw. anderen Anwendungen genutzt werden sind jeweils Teilmodell-Elemente und somit nicht direkt „identifizierbar“. Sie werden daher in Teilmodellen organisiert und im Asset Repository verwaltet und letztlich von verschiedenen `AssetAdministrationShell`’s verwendet.

Ein Teilmodell enthält eine Vielzahl von Teilmodell-Elementen, wobei auch ein hierarchischer Aufbau zulässig ist. Wesentlich an dieser Stelle ist, dass ein Teilmodell identifizierbar, ein Teilmodell-Element lediglich referenzierbar ist. Dieses kann nur anhand des Teilmodells und seinem unmittelbaren Kontext (übergeordnete Elemente) aufgefunden werden. Teilmodelle stellen somit einen Einstiegspunkt dar, um Informationen über Maschinen, Datenpunkte oder auch sonstige Informationen zur Maschine zu sammeln. Da im Regelfall viele Maschinen gleicher Bauart existieren, erlaubt das AAS-Meta-Modell die Definition von Template-Elementen. So können z.B. allgemein gültige Informationen wie eine Bedienungsanleitung oder Hersteller-Informationen einmalig abgelegt werden. Das erlaubt es, für Maschinen gleicher Bauart nicht jede einzelne Information immer wieder redundant ablegen zu müssen. Zudem bietet die Beziehung zwischen Templates und Instanzen im AAS-Meta-Modell die Möglichkeit der Vererbung, da Instanzen auf Basis von Templates erstellt werden können. Die Unterscheidung zwischen Template- und Instanz-Informationen ist ein wesentliches Element des AAS-Meta-Modells und wird durch den Stereotyp *HasKind* definiert, siehe dazu auch Abschnitt 3.1.1:

- Template-Informationen: Datenobjekte mit `HasKind=TEMPLATE` repräsentieren abstrakte oder statische Informationen, die letztlich an konkrete Anwendungs-Instanzen weitervererbt werden können. Template-Informationen können analog der objekt-orientierten Datenmodellierung keine aktive Rolle in einer I4.0 Umgebung einnehmen. Sie können sinngemäß nicht als I4.0 Komponente aktiviert werden und somit auch keine dynamischen Informationen von Assets repräsentieren. Ein Template kann auch Informationen von einem anderen Template „erben“!
- Instanz-Informationen: Datenobjekte mit `HasKind=INSTANCE` repräsentieren konkrete digitale Abbilder von Anwendungs-Eigenschaften. Diese können, müssen aber nicht von abstrakten Template-Informationen abgeleitet sein, wobei diese Typ-Instanz-Beziehung auf verschiedenen Ebenen stattfinden kann: `Asset`, `Submodel` und `SubmodelElement`. Diese Datenobjekte können aktiviert werden, d.h. innerhalb einer I4.0 Komponente stellen Objekte die Verbindung zum physischen Asset dar, indem sie sich mit der Steuerung des physischen Assets verbinden.

Die Typ-Instanz-Beziehung kann nicht nur für die Vererbung von Struktur und statischen Informationen genutzt werden. Neben den dynamischen Eigenschaften kann ein Asset-Template auch Ereignisse und Methoden definieren, die letztlich von den Instanzen „ausgestaltet“ werden müssen. Ein Typ kann somit generell als Klassifikationsmerkmal für Instanzen betrachtet werden. Vereinfacht dargestellt, stellt ein Anwendungs-Typ eine funktionale Vorlage für konkrete Anwendungen dar. Diese Beziehung wird mit der Eigenschaft `HasSemantics` realisiert. Entsprechend der Spezifikation in (AAS Part 1, 2023) erlaubt diese Eigenschaft, AAS-Elemente mit einer „primären“ semantischen Referenz auszustatten. Weiter können zusätzliche, unterstützende „supplemental“ Referenzen hinzugefügt werden. Abbildung 15 veranschaulicht die HasSemantics-Eigenschaft und zeigt auch die beiden Ausprägungen von Referenzen.

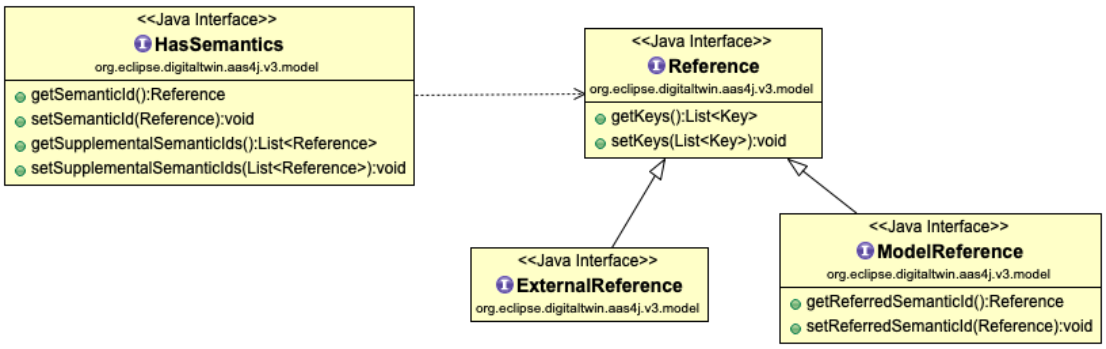


Abbildung 15: HasSemantics und Referenzen

Verweise auf Teilmodell-Elemente werden als `ModelReference` abgelegt. Wenn die referenzierten Teilmodell-Elemente ihrerseits die Eigenschaft `HasSemantics` aufweisen, kann deren `semanticId` ebenfalls in die Referenz gepackt werden. Auch bei einer mehrstufigen Verweiskette werden auf diese Weise die wesentlichen semantischen Bezeichner aus den Templates an die Instanzen weitergegeben. Die Kette an Verweisen mündet in der Regel in einer `ExternalReference`, einem Konzept aus externen Systemen wie z.B. dem SemanticLookup.

Templates können nicht aktiviert werden, sie dienen ausschließlich zur Definition von funktionalen Aspekten die letztlich durch konkrete Anwendungen realisiert werden. Auf diese Weise können auch die funktionalen Blöcke in Abbildung 1 modelliert und realisiert werden. Abbildung 16 visualisiert diese Beziehung.

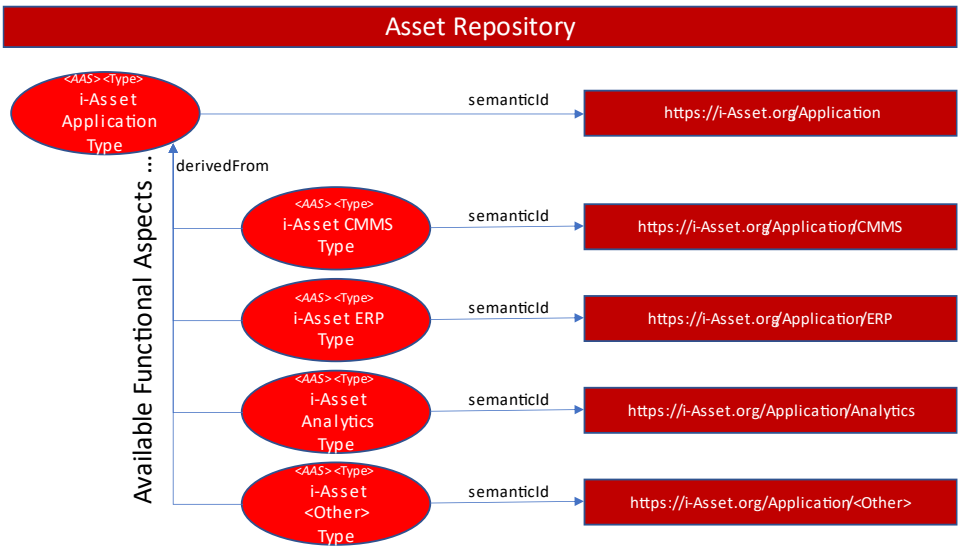


Abbildung 16: Funktionale Aspekte im Asset Repository

Wesentlich dabei ist, dass die vorhandenen funktionalen Aspekte auch zur Laufzeit gefunden werden können, daher wird zunächst ein zentraler Einstiegspunkt definiert. Alle weiteren funktionalen Aspekte leiten sich von diesem zentralen Einstiegspunkt ab und sind über die `derivedFrom` Beziehung erreichbar.

Ein Asset Repository kann somit seine existierenden funktionalen Aspekte auflisten, unabhängig davon ob nun ein (oder mehrere) CMMS vorhanden ist oder nicht.

Ist ein AAS-Template für eine Anwendung gefunden, stehen auch alle darin verlinkten Teilmodelle und die hier enthaltenen Teilmodell-Elemente zur Verfügung, die für die Abdeckung der

Kommunikations-Anforderungen erforderlich sind. Abbildung 17 zeigt ein rudimentäres Teilmodell für Maintenance. Dieses definiert die Abfrage der Wartungshistorie für ein Asset als Template.

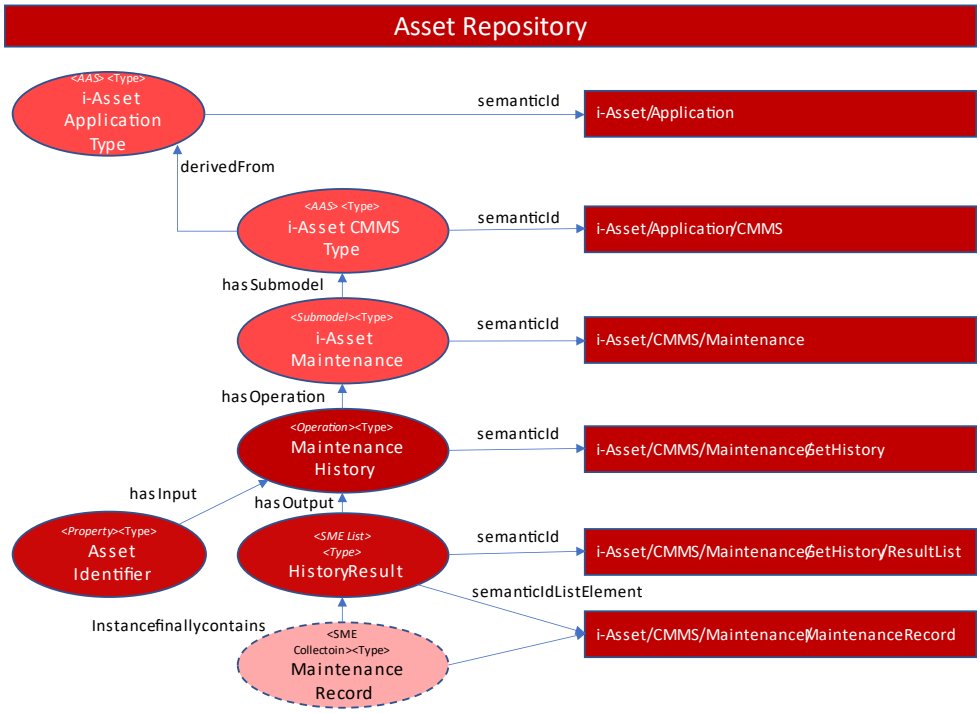


Abbildung 17: Teilmodelle und Kommunikationsmerkmale in AAS-Templates

Die Abfrage einer Wartungshistorie für ein Asset wird dabei als Methodenaufruf definiert, welcher als Eingabeparameter einen eindeutigen Bezeichner eines Assets erwartet. Als Ergebnis der Methodenausführung wird eine Liste von Wartungseinträgen aus einem CMMS erwartet. Diese Liste wird mit Hilfe des AAS-Teilmodell-Elements `SubmodelElementList` definiert. Dieses Element legt mittels Attribut `semanticIdListElement` fest, welche Elemente in der Liste enthalten sein dürfen. Diese semantische Beziehung bestimmt somit alle erforderlichen Details für implementierende Anwendungen.

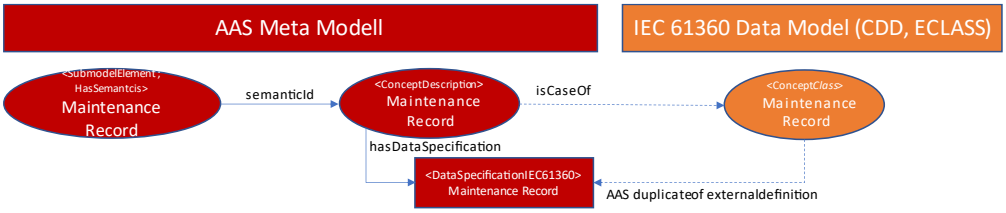


Abbildung 18: Datenintegration aus externen Systemen

In Abbildung 18 ist diese Beziehung dargestellt. Ein `SubmodelElement` aus dem AAS Meta-Modell verweist mittels `semanticId` auf eine `ConceptDescription`. Die `Concept-Description` wurde als Abbild eines externen Datensatzes (aus Semantic Lookup bzw. IEC61360 kompatiblen Repository) erstellt und mittels Beziehung `isCaseOf` in der `Concept-Description` vermerkt. Zusätzlich werden die externen Informationen in das AAS Meta-Modell übernommen und als DataSpecificationIEC61360-Objekt der `ConceptDescription` hinzugefügt. Der Zugriff auf die semantische Definition erfolgt über eine API des jeweiligen Semantic Lookup Repository. Die Synchronisation zwischen AAS und semantischer Definition aus einem angeschlossenen Semantic Lookup System übernimmt das Asset Repository.

Im Asset Repository werden somit Kommunikations-Templates verwaltet, welche von konkreten Anwendungen (CMMS, ERP, Analytics etc.) realisiert und von Assets oder anderen Anwendungen genutzt werden. Wesentlich ist dabei, dass diese Kommunikations-Templates für auch auffindbar sind.

Diese Vermittlerrolle wird vom Asset Directory bereitgestellt. Dieses speichert die Service-Endpunkte der jeweils aktiven Assets bzw. Anwendungen.

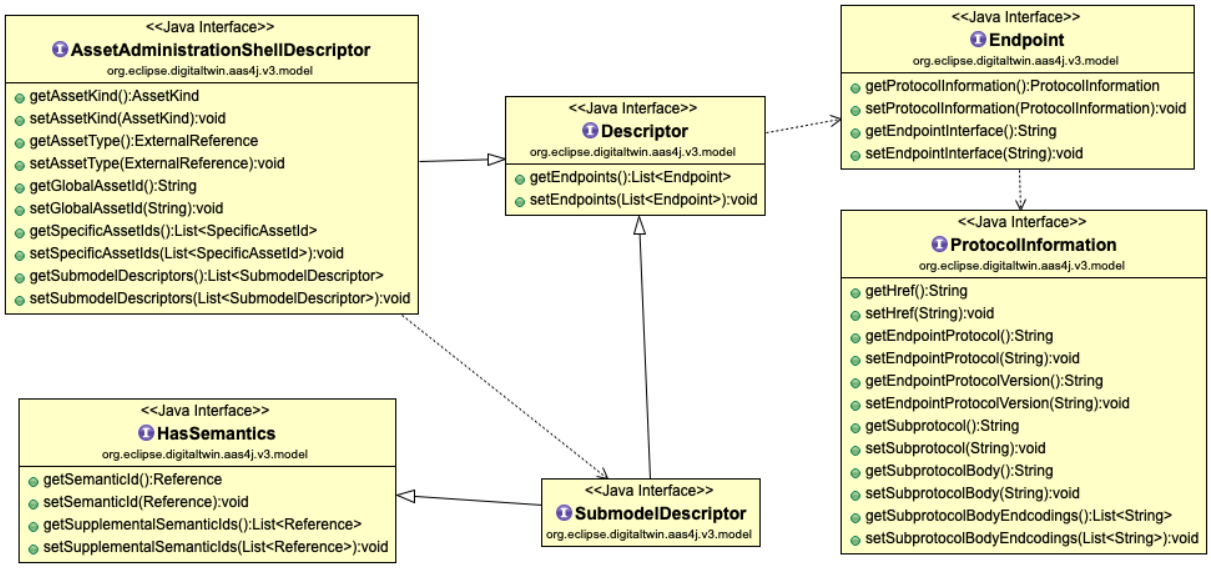


Abbildung 19: AAS und Submodel Descriptoren

Entsprechend der Spezifikation in (AAS Part 2, 2023) registrieren sich Assets/Anwendungen mit ihrem `AssetAdministrationShellDescriptor` im Asset Directory. Abbildung 19 zeigt die wesentlichen Datenbausteine für diese Deskriptoren. Die verschiedenen „Identifier“ im `AssetAdministrationShellDescriptor` erleichtern die Suche nach Assets basierend auf den Bezeichnern aus unterschiedlichen Anwendungen. So kann ein Asset anhand seiner *AAS-ID*, seiner *GlobalAssetID* oder auch anhand von weiteren *SpecificAssetID*’s im Directory identifiziert werden. Die Descriptoren enthalten eine Liste von Endpoints und Protokoll-Informationen, in denen die bereitgestellten AAS- bzw. Submodel-Interfaces gem. (AAS Part 2, 2023) aufgelistet sind

Semantic Integration Patterns werden in (ggfs. hierarchisch aufgebauten) Teilmodellen beschrieben. Wie in Abbildung 17 dargestellt, müssen die Teilmodelle, aber auch die für die Kommunikation erforderlichen Teilmodell-Elemente mit einer `semanticId` versehen sein. Um Assets anhand der „realisierten“ Patterns auffinden zu können, werden im `Submodel-Descriptor` alle „aktivitieren“ Patterns hinterlegt. Hierzu erbt der `SubmodelDescriptor` die Eigenschaft `HasSemantics` und kann so eine Liste von „Supplemental Semantic ID’s“ verwalten.

Abbildung 20 zeigt die Instanziierung eines funktionalen Aspekts, im vorliegenden Fall wird eine generische Anwendung CMMS durch eine konkrete Anwendung isproNG instanziiert. Erst durch die Realisierung des funktionalen Aspekts wird z.B. die Methode „Wartungshistorie abfragen“ im System zur Verfügung gestellt.

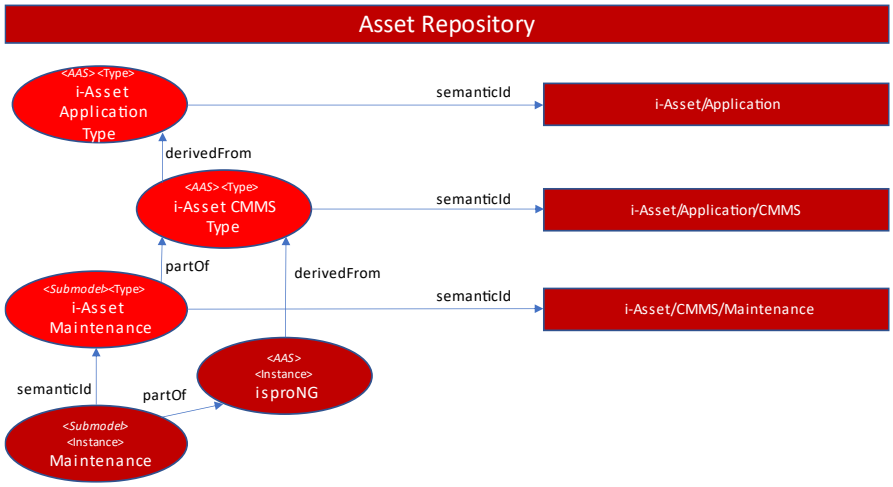


Abbildung 20: Realisierung von Semantic Integration Patterns

Damit angeschlossene Anwendungen, aber auch der Data Integration Layer als zentrale Anlaufstelle, die in verteilten Systemen verfügbare Funktionalität, hier die isproNG-Instanz, nutzen können, registriert sich diese im Asset Directory. Die Instanz-Elemente im isproNG Connector verweisen auf die definierenden AAS-Templates. Diese Verweise zeigen auf ein existierendes AAS-Element und sind daher als `ModelReference` definiert. ModelReference transportieren zusätzlich die `semanticId` des referenzierten Elements (siehe Abbildung 15). Registriert sich der isproNG Connector, so werden die wesentlichen Bezeichner (GlobalAssetId, SpecificId's) der AAS-Instanz im AAS-Deskriptor abgelegt. Zusätzlich werden auch alle Teilmodell-Deskriptoren im den AAS-Deskriptor eingefügt und im Asset Directory abgelegt. Dabei werden die semantischen Informationen zum jeweils realisierten Semantic Integration Pattern berücksichtigt. Abbildung 21 veranschaulicht diese Vorgehensweise. Im Teilmodell-Deskriptor zu „isproNG Maintenance" wird neben dem Verweis auf das „zugrundeliegende" Template (i-Asset Maintenance) auch die Operation „isproNG Maintenance History", aktiviert.

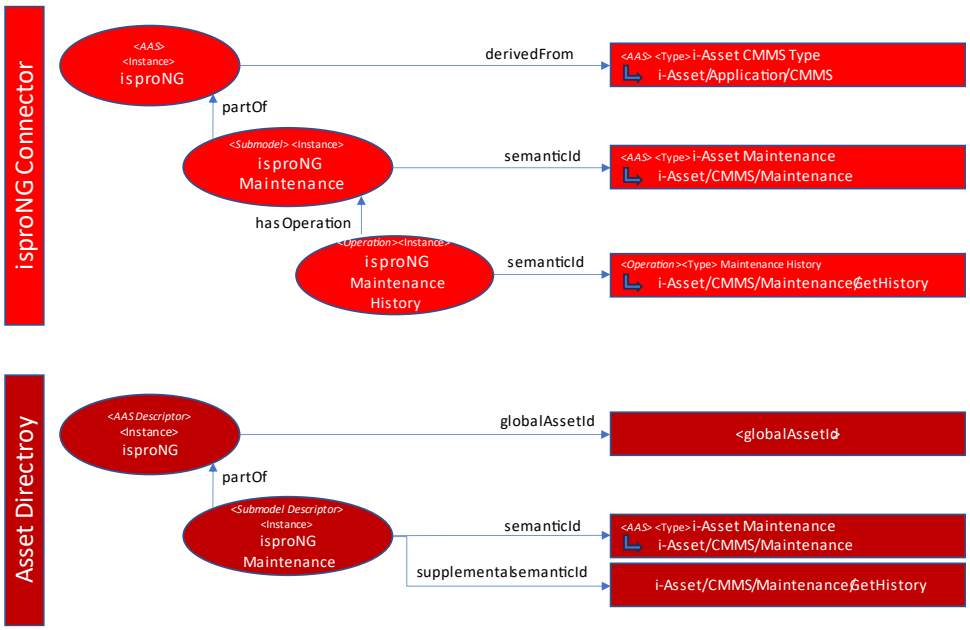


Abbildung 21: Semantic Identifier im SubmodelDescriptor

Das Asset Repository hat als zentrale Anlaufstelle alle vorhandenen Kommunikations-Templates und ist auch in der Lage, die konkreten Anwendungen für die Ausführung dieser Templates zu bestimmen.

Mit Hilfe von Semantic Integration Patterns werden somit für Anwendungen typische Kommunikationsanforderungen (Störmeldung, Wartungshistorie für CMMS; Abfrage der Anlagenstruktur für ERP o.ä.) definiert und für potentielle Nutzer angeboten. Semantic Integration Patterns können jedoch erst dann genutzt werden, wenn diese auch von zumindest einer Anwendung realisiert werden. Eine Realisierung eines Semantic Integration Patterns erfordert die Beteiligung von Application/Asset Connectoren, welche die Laufzeitumgebung für Semantic Integration Patterns darstellen.

4.4 Application Connector und Semantic Integration Patterns

Ein Asset bzw. Application Connector ist jeweils einem Asset oder einer Anwendung vorgeschaltet um die Informationen des Assets bzw. der Anwendung in die I4.0 Welt zu transformieren. Der Connector kümmert sich um die Sichtbarkeit des Assets, der Anwendung im Netzwerk und erlaubt repräsentiert sich mit einem standardisierten HTTP-REST Interface. Anfragen an einen Connector erfolgen über eine genormte Schnittstelle, demnach können alle I4.0 kompatiblen Teilnehmer mit dem Asset bzw. mit der Anwendung kommunizieren.

Neben dem REST-Interface für die synchrone Kommunikation verbindet sich ein Connector auch mit einem Message Broker, um die asynchrone Kommunikation gewährleisten zu können. Abhängig von den Einstellungen in `BasicEventElement` können Nachrichten empfangen bzw. auch gesendet werden. Abbildung 22 zeigt eine schematische Übersicht eines Connectors.

Ein Connector wird zur Laufzeit durch (zumindest) eine Asset Administration Shell gesteuert. In der AAS werden die für die Integration relevanten Methoden der Anwendung modelliert und so der Außenwelt in Form der AAS API zugänglich gemacht. Durch die Bereitstellung der Funktionen mittels REST API bzw. asynchrones Messaging kann die Anwendung angesprochen werden.

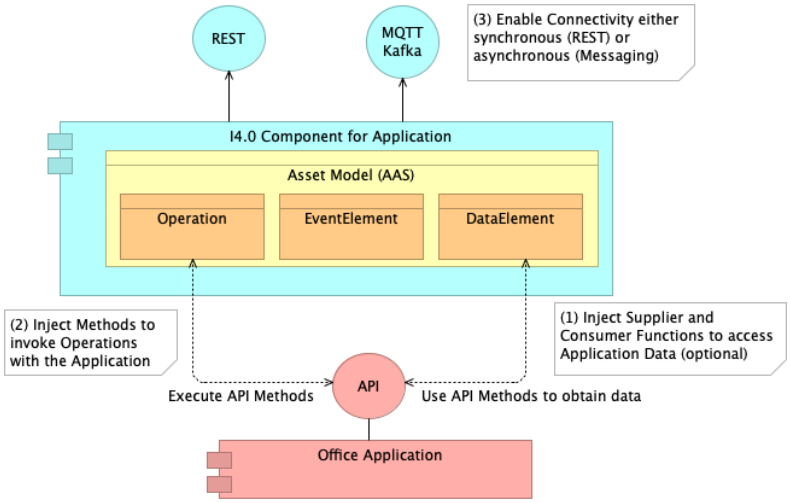


Abbildung 22: Application Connector - Übersicht

Die Vorgangsweise bei der Anbindung von Anwendungen:

(1) Für Eigenschaften (`DataElement`) werden Callback-Methoden definiert. Diese Callback-Methoden werden vom Connector aufgerufen um den aktuellen Wert der Eigenschaft vom Asset in die Anfragebeantwortung im AAS auslesen und an geeigneter Stelle in die Kommunikation anderen Anwendungen integrieren.
(2) Für Operationen (`Operation`) werden ebenfalls Callback-Methoden definiert, welche letztlich die „dahinter“ liegende Methoden der Anwendung ausführen.
(3) Die Anwendung wird als I4.0 Komponente sichtbar gemacht. Für die synchrone Kommunikation wird ein HTTP-REST Service-Endpoint aktiviert welcher die AAS API gem. der Spezifikation „Details oft he Asset Administration Shell, Part 2 – Interoperability at Runtime (AAS Part 2, 2023) zur Verfügung stellt. Diese Service API erlaubt die Abfrage bzw. Änderung einzelner Elemente einer AAS, aber auch die Ausführung[12] von Operationen – sofern die entsprechende Callback-Methode hinterlegt ist. Für die asynchrone Kommunikation werden alle `BasicEventElement`-Settings der AAS aktiviert. Es werden die konfigurierten Message Consumer für eingehende Nachrichten bzw. Message Producer für ausgehende Nachrichten erstellt.

Wie in Abbildung 22 ersichtlich, kommen den Elementen `Operation`, `BasicEventElement` und `DataElement` besondere Bedeutung bei der Aktivierung von Assets und auch Anwendungen als I4.0 Komponenten zu.

4.4.1 Kommunikation und Datenaustausch

Der Datenaustausch zwischen den Anwendungen erfolgt immer über die I4.0 API bzw. über die Messaging-Einstellungen auf Basis der Asset Administration Shell. Die API einer I4.0 Komponente erlaubt jedoch mittels zusätzlicher Parameter die Art und Weise wie die Daten letztlich aufbereitet werden, zu steuern. Mit dem Request-Parameter content kann definiert werden, wie die Inhalte letztlich serialisiert werden:

- Normal: Die Meta-Daten der einzelnen Elemente werden vollständig exportiert. Dies ist der Default-Wert
- Value: Die Ausgabe der Daten reduziert sich auf den jeweiligen Wert des Elements. Es werden nur jene Elemente verarbeitet, welche entweder Unterelemente mit Werten haben oder selbst einen Wert repräsentieren. Diese Methode wird auch als `Value-Only` bezeichnet.
- Reference: Anstelle der vollständigen Meta-Daten wird eine Referenz auf das angefragte Element erzeugt.
- Trimmed: Bei dieser Form der Serialisierung werden nur ausgewählte Attribute von Elementen serialisiert, um eine kompakte Repräsentation zu erreichen.

Für die Etablierung von Semantic Integration Patterns sind die Normal- und auch die Value-Only-Serialisierung von Bedeutung. Die Unterscheidung lässt sich anhand des folgenden Beispiels darstellen. Abbildung 23 zeigt die sehr vereinfachte Element-Hierarchie zur Beschreibung ausgewählter Eigenschaften eines Roboter-Arms. Das Submodel mit der `idShort` „properties“ enthält zwei `SubmodelElementCollection`-Elemente mit darunterliegenden `Property`-Elementen.

[12] Swagger-Definition der Invoke-Operation Methode: https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-API/V1.0RC03#/Asset%20Administration%20Shell%20API/InvokeOperation

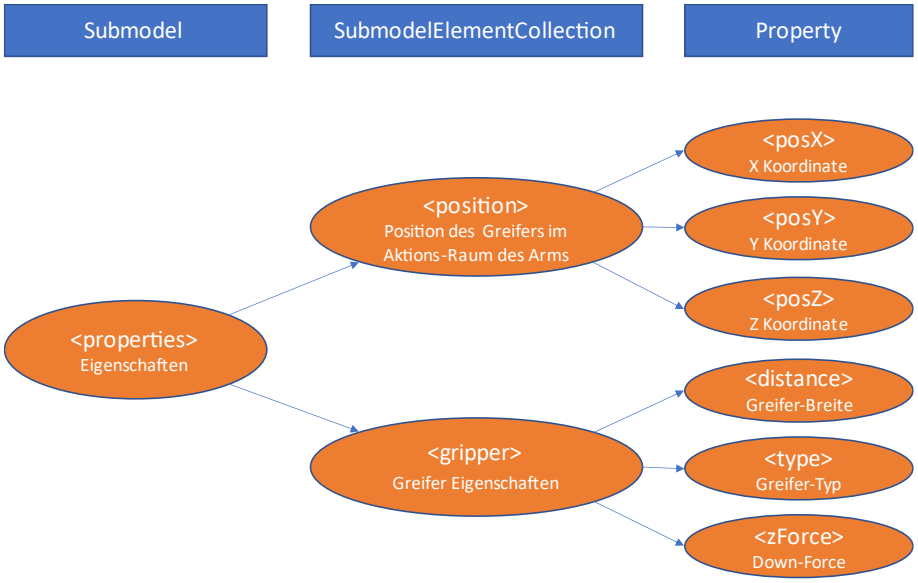


Abbildung 23: Eigenschaften Roboter-Arm

Normal Serialisierung

Eine Serialisierung der Elemente im JSON-Format ist in Listing 1 dargestellt. Für jedes der Elemente ist eine `semanticId` eingetragen, welche das Element klassifiziert, für das Teilmodell ist exemplarisch eine URI (http://iasset.salzburgresarch.at/labor/gripper) definiert. Für die `SubmodelElementCollection` bzw. auch die einzelnen `Property`-Elemente sind weitere `semanticId's` eingetragen. Weiter ist dargestellt, dass

- das `Submodel` „identifizierbar" ist – es besitzt einen global gültigen Bezeichner,
- `SubmodelElementCollection` bzw. `Property` lediglich „referenzierbar" sind – sie haben keinen global gültigen Bezeichner, sondern lediglich eine `idShort` welche nur im jeweiligen Kontext des übergeordneten Elements eindeutig ist.
- die einzelnen Property-Elemente jeweils mit Typ (`valueType`) und Wert (`value`) ausgestattet sind.
- Alle Elemente sind `Referable` und weisen daher eine textuelle Beschreibung (`description`), ggf. in mehreren Sprachen, auf.

Das Listing wurde aus Platzgründen gekürzt …

```
{   // Submodel containing "submodelElements"
    "modelType": "Submodel",
    "idShort": "properties",
    "id": "http://iasset.salzburgresearch.at/labor/panda/properties",
    "kind": "Instance",
    "semanticId": { // link to external gripper properties definition
        "keys": [
            {
                "type": "GlobalReference",
                "value": "http://iasset.salzburgresarch.at/labor/gripper"
            }
        ],
        "type": "ExternalReference"
    },
    "submodelElements": [
      { // SubmodelElement
        "modelType": "SubmodelElementCollection",
        "idShort": "position",
        "kind": "Instance",
```

```
        "semanticId": { // link to external position definition
            "keys": [
                {
                    "type": "GlobalReference",
                    "value": "http://iasset.salzburgresarch.at/labor/position"
                }
            ],
            "type": "ExternalReference"
        },
        "value": [ //
          {  // each property has it's own semanticId
            "modelType": "Property",
            "idShort": "posX",
            "kind": "Instance",
            "value": "0.234",
            "valueType": "xs:double",
            "semanticId": { … },
            "description": [
              {
                "text": "X-Koordinate",
                "language": "de"
              }
            ]
          },
          {
            "modelType": "Property",
            "idShort": "posX",
            "kind": "Instance",
            "value": "12.1",
            "valueType": "xs:double",
            "description": [
              {
                "text": "Y-Koordinate",
                "language": "de"
              }
            ]
          },
          {
            "modelType": "Property",
            "idShort": "posZ",
            "kind": "Instance",
            "value": "0.234",
            "valueType": "xs:double",
            "description": [
              {
                "text": "Z-Koordinate",
                "language": "de"
              }
            ]
          }

        ],
        "description": [
          {
            "text": "Position des Greifers im Aktions-Raum des Arms",
            "language": "de"
          }
        ],
      },
      {      // SubmodelElementCollection “gripper” collapsed
      }
    ],
    "description": [
      {
        "text": "i-Asset Labor - Panda Eigenschaften",
```

```
        "language": "de"
      }
    ],
}
```

Listing 1: Serialisierung "Normal"

Diese Datenstruktur wird im Zuge einer Kommunikation mit der Plattform oder einem Connector übertragen. Bei dieser Form der Übertragung werden alle Informationen der AAS-Elemente berücksichtigt. Dies führt zu sehr großen Dokumenten, die nicht notwendigerweise übertragen werden müssen, da beide Teilnehmer an einer Kommunikation die gleichen Strukturen nutzen. Es ist daher ausreichend, die minimalistische *ValueOnly* Serialisierung zu nutzen.

ValueOnly Serialisierung

Diese *ValueOnly*-Serialisierung ist für Semantic Integration Patterns ein wesentliches Element und in der Runtime Spezifikation der Asset Administration Shell, Abschnitt 9.4.2 ausführlich beschrieben (AAS Part 2, 2023). Diese Form der Serialisierung umfasst dabei jene Elemente die Unterelemente enthalten bzw. welche auch einzelne Daten enthalten. Die ValueOnly Darstellung gibt die Werte von Teilmodellen bzw. einzelnen Teilmodell-Elementen möglichst komprimiert aus. Dabei gilt:

- Der Wert des `Submodel` wird durch seine Unterelemente definiert, die als Key-Value-Paare ausgegeben werden. Als Key wird dabei die `idShort` des jeweiligen Unterelements verwendet. Der Wert wird schließlich durch das Element bestimmt:
  - `SubmodelElementCollection`-Elemente haben ebenfalls Unterelemente, die als Key-Value-Paare ausgegeben werden.
  - `Property`-Elemente (bzw. `DataElement`) haben keine Unterelemente, der aktuelle Wert wird analog dem definierten Datentyp ausgegeben.

Das Ergebnis der reduzierten Darstellung ist in Listing 2 im JSON-Format ersichtlich.

```
// GripperProperties
{   // GripperPosition
    "position": {
        "posX": 0.234,
        "posX": 12.1,
        "posZ": 0.234
    },
    // GripperData
    "gripper": {
        "distance" : 1.0,
        "type": "GRIPPER1",
        "zForce": 0.002
    }
}
```

Listing 2: ValueOnly Serialisierung

Das so erzeugte JSON kann nun wieder in eine Objekt-Notation überführt werden. Abbildung 24 zeigt eine einfache Java-Klasse (mit Unterobjekten) welche zur Deserialisierung der ValueOnly Ausgabe aus Listing 2 verwendet werden kann.

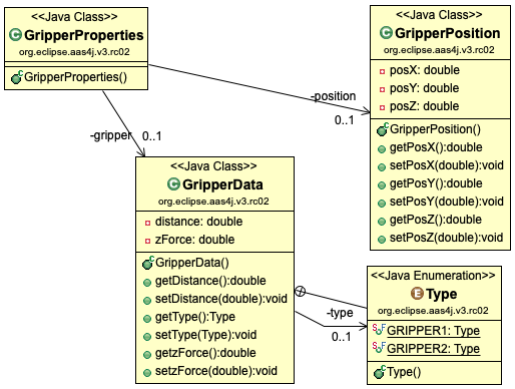


Abbildung 24: Objekt-Struktur für ValueOnly JSON

Im Zuge des Datenaustauschs sind durch die AAS immer alle Informationen vorhanden, so auch die `semanticId` des Submodel-Elements (http://iasset.salzburgresarch.at/labor/gripper) welche von einer Anwendung entsprechend ausgewertet werden kann.

4.4.2 Aktivieren von Eigenschaften

Die wesentlichen Eigenschaften einer AAS sind in Teilmodellen (`Submodel`) organisiert. Diese enthalten eine Reihe von Teilmodell-Elementen (`SubmodelElement`) um den unterschiedlichen Anforderungen zu genügen. Abbildung 25 zeigt die konkreten Ausprägungen von `SubmodelElement`.

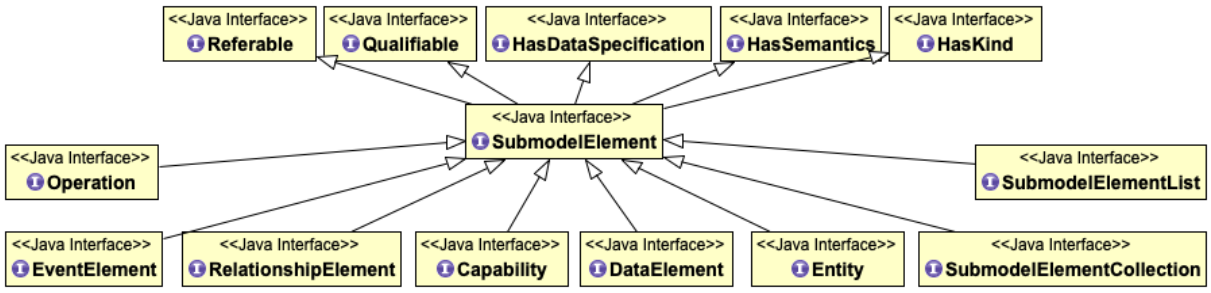


Abbildung 25: SubmodelElement – Konkrete Ausprägungen

Generell ist jedes `SubmodelElement` auch `Referable`, d.h. anhand seiner `idShort` referenzierbar. Weiter ist es `Qualifiable`, d.h. es können Einschränkungen für jedes `SubmodelElement` hinterlegt werden. Mit `HasKind` bekommt jedes `SubmodelElement` die Typ/Instanz-Unterscheidung und schließlich erlauben `HasSemantics` & `HasDataSpecification` die Klassifizierung und Ergänzung eines Elements mit zusätzlichen semantischen Informationen.

Hinsichtlich der konkreten Ausprägungen von `SubmodelElement` kann zwischen Interaktions-Elementen, Datenelementen (`DataElement`) und Struktur-Elementen unterschieden werden. Die Interaktions-Elemente (`Operation`, `EventElement`) sind nachfolgend weiter detailliert. Strukturelemente (`SubmodelElementCollection`, `SubmodelElementList`, Entity) erlauben die Modellierung von Hierarchien, diese können Unterelemente aufnehmen und Datenelemente wiederum können dem Namen entsprechend auch einzelne Werte mit unterschiedlichen Datentypen aufnehmen, diese besitzen ein entsprechendes `value`-Attribut. Abbildung 26 zeigt die weiteren Ausprägungen von `DataElement`. Diesen ist gemeinsam, dass sie einen Wert (`value`) transportieren können.

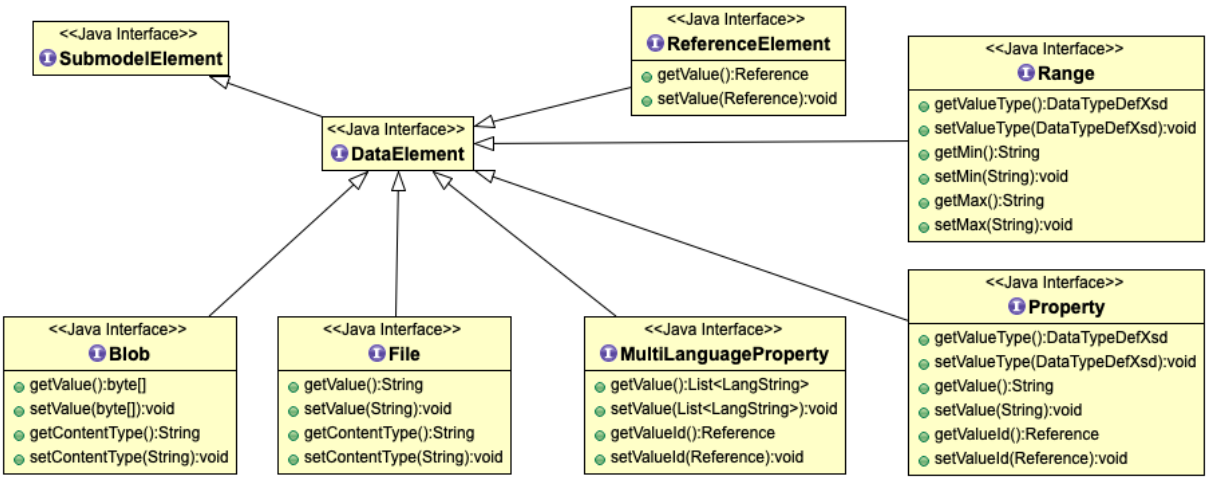


Abbildung 26: Teilmodell-Element "DataElement" mit Ausprägungen

Die Datenelemente können statisch sein, z.B. den Hersteller eines Assets benennen. Sie können aber auch einzelne dynamische Eigenschaften von Assets oder auch Anwendungen darstellen. Um die dynamischen Werte von Assets in einer I4.0 Umgebung zu repräsentieren werden diese mit Consumer- bzw. Supplier-Methoden versehen:

- Consumer-Methode: Wird in der I4.0 Umgebung aufgerufen, wenn der Wert eines Datenelements via API verändert wird. Diese Methode erhält den neuen Wert und gibt diesen an das Asset bzw. an die Anwendung weiter.
- Supplier-Methode: Wird in der I4.0 Umgebung aufgerufen, wenn das Datenelement serialisiert, d.h. der Wert des Datenelements abgerufen wird. Dabei wird z.B. die Steuerung des Assets via OPC-UA kontaktiert und der entsprechende Wert in die I4.0 Umgebung integriert.

Werden keine Consumer/Supplier-Methoden für ein Datenelement bereitgestellt, so bleibt dieses statisch.

4.4.3 Ausführen von Methoden

Office-Anwendungen stellen ihre Funktionalität primär mit Hilfe ihrer Benutzeroberfläche zur Verfügung. Zumeist wird aber auch eine eigene API, eine proprietäre Schnittstelle angeboten, die von Anwendungsintegratoren benutzt wird um auf die Funktionalität der Anwendung zugreifen zu können. Wie schon in diesem Abschnitt einleitend dargestellt, bietet eine Asset Administration Shell jene Elemente an, um mit Hilfe von I4.0 Komponenten diese Funktionalität für andere Anwendungen zu beschreiben und diese letztlich auch ausführen zu können.

Die hierfür wesentlichen Elemente `Operation` und `OperationVariable` sind nachfolgend dargestellt.

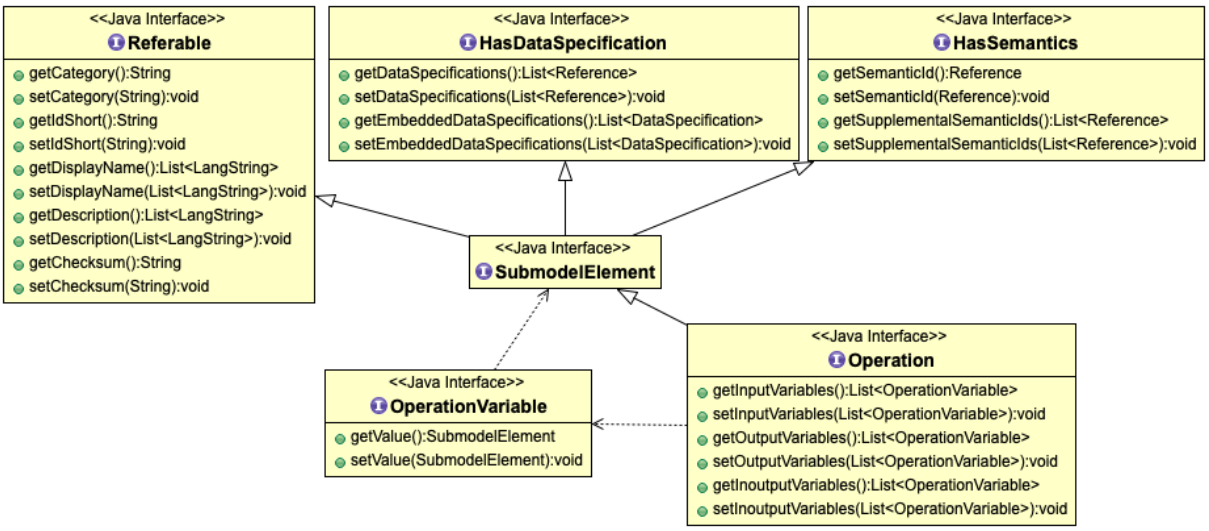


Abbildung 27: Operation Element

Wie in Abbildung 27 ersichtlich, gelten folgende Aussagen:

- Eine `Operation` ist ein `SubmodelElement`, dadurch erbt es alle Eigenschaften von `SubmodelElement` inclusive `Referable`, `HasDataSpecification`, `HasSemantics`[13]. D.h. jede `Operation` wird anhand seiner `idShort` innerhalb seines übergeordneten Elements referenziert, hat verschiedene (mehrsprachige) Bezeichner und kann mittels semantischer Referenzen zusätzliche Meta-Daten integrieren.
- Eine `Operation` wird mit `OperationVariable`-Elementen ergänzt. `OperationVariable`-Listen sind verfügbar für:
  - Input-Variablen: Erforderlich als Eingabeparameter für die `Operation`
  - Output-Variablen: Als Ergebnis der Methodenausführung
  - Inoutput-Variablen: Kombination von Input- und Output-Variablen
- Jede `OperationVariable` stellt wiederum ein `SubmodelElement` dar. Da `SubmodelElement` abstrakt ist, wird an dieser Stelle eine konkrete Ausprägung von `SubmodelElement` erwartet, welche mittels `semanticId` auf sein Typ-Element oder eine `ConceptDescription` zeigt.
- Eine `Operation` ist als `Referable`-Element in einem Teilmodell angeordnet. Innerhalb einer AAS ist eine Operation also nur ausgehend vom Teilmodell (`Submodel`) anhand seiner `idShort` auffindbar.

Die Verwendung der Elemente `Operation` bzw. `OperationVariable` ist nachfolgend an einer Beispiel-Methode zur Abfrage der Wartungshistorie dargestellt. Diese Operation erwartet als Input-Parameter lediglich einen Asset-Identifier. Das Teilmodell-Element, welches diesen Parameter beschreibt, ist in Listing 3 mit den wesentlichsten Eigenschaften dargestellt:

```
{   // Operation with idShort “maintenanceHistory” – Part of a Submodel
    "modelType": "Operation",
    "kind": "Instance",
    "idShort": "maintenanceHistory",
```

[13] `SubmodelElement` besitzt noch weitere Eigenschaften (`HasKind`, `Qualifiable`), die aus Platz-/Übersichtlichkeitsgründen nicht dargestellt werden!

```
    "inputVariables": [ // liste aller input-Variablen
        {   // operationvariable
            "value": { // property
                "modelType": "Property",
                "kind": "Instance",
                "semanticId": {
                    "keys": [
                        {
                            "type": "ConceptDescription",
                            "value": "0173-1#02-AA0676#003"
                        }
                    ],
                    "type": "GlobalReference"
                },
                "idShort": "assetIdentifier",
                "category": "Parameter",
                "value": "47110815",
                "valueType": "xs:integer"
            }
        }
    ],
    "outputVariables": [
        // omitted ...
    ],
    "inouutputVariables": [
        // omitted ...
    ]

}
```

Listing 3: Operation „maintenanceHistory“ (JSON)

Wie in Abbildung 22 dargestellt, wird dieses `Operation`-Element innerhalb einer I4.0 Komponente mit einer eigenen Callback-Methode versehen. Diese erhält die Input-Parameter und führt letztlich die eigentliche Methode in der Anwendung aus. Die Antwort wird wiederum von der Calllback-Methode entsprechend der Output- bzw. Inoutput-Variablen erzeugt und an den Aufrufer retourniert.

Zur Ausführung der Operation innerhalb einer I4.0 Komponente wird die entsprechende Invoke-Operation Methode gemäß AAS API verwendet, wie sie in Listing 4 dargestellt ist:

```
/aas/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke
```

Listing 4: Invoke Operation Request

Für die Ausführung des Requests müssen die beiden Pfad-Parameter (`submodelIdentifier`, `idShortPath`) ausgewertet werden. Diese http-Methode wird durch die I4.0 Komponente bereitgestellt (siehe Abbildung 22), die AAS ist somit bereits eindeutig definiert. Die weiteren wichtigen Pfad-Parameter benennen das Teilmodell (`submodelIdentifier`) und in weiterer Folge den `idShortPath` der zum gesuchten `Operation`-Element führt. Der `idShortPath` wird in seine Bestandteile aufgesplittet und (beginnend mit dem Teilmodell) rekursive abgearbeitet, bis das gesuchte `Operation`-Element erreicht ist. Ist nun eine Callback-Methode mit dem Operation-Element verbunden, so kann diese mit den im POST-Request-Body bereitgestellten Daten aufgerufen werden. Diese Callback erhält dann die Kombination aus allen Input- und Inoutput-Variablen für die Ausführung der Operation. Als Request-Antwort werden alle Output- und Inoutput-Variablen aufbereitet wobei noch zusätzlich festgelegt wird, ob die Antwort „Normal“ oder „ValueOnly“ erfolgen soll.

4.4.4 Event Handling

Innerhalb einer AAS wird das `EventElement` bzw. dessen Ausprägung `BasicEventElement` (siehe Abbildung 28) für die Modellierung der asynchronen Kommunikation genutzt. Mit dem `EventPayload` wird zudem das Nachrichtenformat definiert.

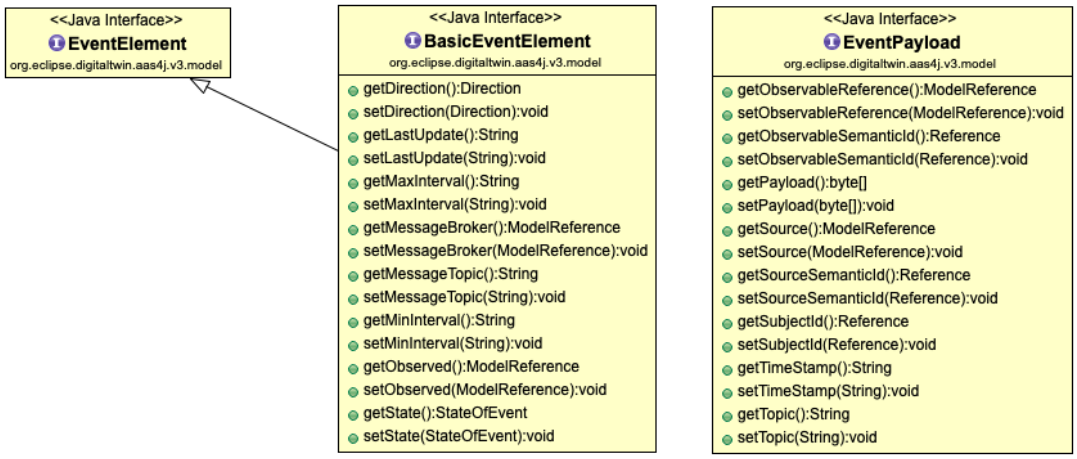


Abbildung 28: Event Settings

Abbildung 28 zeigt die wesentlichen Einstellungen für Events. Es wird je `EventElement` festgelegt, mit welchem Message Broker (Attribut `messageBroker`) der Connector interagieren soll, ob es eine eingehende oder ausgehende Kommunikation (`direction`) ist und über welches Topic (`messageTopic`) diese erfolgen soll. Mit dem Attribut `observed` wird schließlich festgelegt, welches Element innerhalb der AAS überwacht werden soll. Der Connector kann so bei Aktivierung die erforderlichen Message-Consumer (eingehende Nachrichten) und Message-Producer (ausgehende Nachrichten) bereitstellen. Das grundsätzliche Format der Nachrichten ist im `EventPayload` definiert wobei die Attribute

- `source`: Absender der Nachricht
- `sourceSemanticId`: Die semantische Auszeichnung des Absenders
- `observableReference`: Das überwachte/observed Element
- `observableSemanticId`: Die semantische Auszeichnung des überwachten/observed Element

die vollständige semantische Einordnung einer Nachricht ermöglichen. Im Attribut `sourceSemanticId` wird der Typ der Nachricht transportiert. Das Attribut `observableSemanticId` bestimmt letztlich den Inhalt der Nachricht. Das `observed`-Element wird serialisiert und im Attribut `payload` abgelegt. Der oder die Empfänger der Nachricht erhalten so den wesentlichen Inhalt, aber auch die vollständige semantische Information wie dieser zu interpretieren ist.

4.4.5 Vererbung in Semantic Integration Patterns

Semantic Integration Patterns sind erweiterbar, da sie das Prinzip der Vererbung unterstützen. Dies sowohl im Semantic Lookup System (IEC 61360) als auch im AAS Meta-Modell.

Das Prinzip der Vererbung bereits im Datenmodell des Semantic Lookup verankert und in der API berücksichtigt. Eine Abfrage einer `ConceptClass` liefert zu den direkt zugewiesenen Attributen auch alle Attribute der übergeordneten Objekte wie dies in Abbildung 29 dargestellt ist.

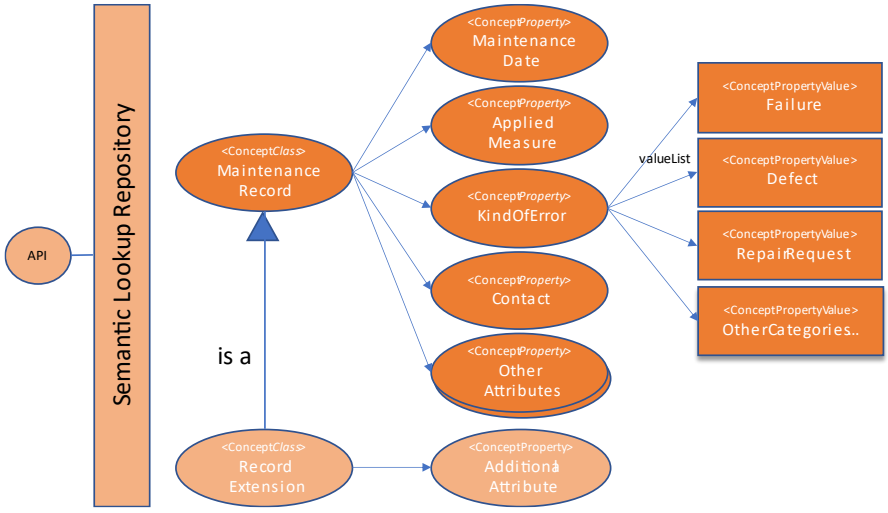


Abbildung 29: Vererbung im Semantic Lookup

Auch im Meta-Modell der Plattform Industrie 4.0 ist das Prinzip der Vererbung enthalten. Asset Administration Shells können aufeinander aufbauen, in dem sie das Attribut `derivedFrom` nutzen.

Mit dem Attribut `semanticId` und der Unterscheidung in Instanz-Element oder Template-Element steht noch eine weitere Möglichkeit zur Verfügung, eine „Vererbung“ von Informationen zu modellieren.

Hier gelten folgende Richtlinien:

- Eine `semanticId` Referenz eines `Submodel`/`SubmodelElement` wird ausgewertet, wenn
  - diese auf ein `Submodel`/`SubmodelElement` des gleichen Typs, jedoch mit `HasKind.TEMPLATE` zeigt. Ein Element mit `HasKind.INSTANCE` als Ziel einer `semanticId` Reference wird ignoriert.
  - diese auf eine `ConceptDescription` zeigt.
- Ein `Submodel`/`SubmodelElement` mit `HasKind.TEMPLATE` kann seinerseits per `semanticId` auf ein übergeordnetes Element gleichen Typs verweisen (mehrstufige Hierarchie).
- Ein `Submodel`/`SubmodelElement` kann mittels `semanticId` auf eine externe semantische Definition verweisen (`GlobalReference`) – diese Beziehung wird ausgewertet, wenn
  - ein `Submodel`, `SubmodelElementCollection` auf eine `ConceptClass` verweisen. Die zugewiesenen Attribute der `ConceptClass` müssen dann als Unterelemente in `Submodel`, `SubmodelElement` vorhanden sein.
  - ein `SubmodelElement` auf eine `ConceptProperty` verweist. Das `ConceptProperty` bestimmt den Datentyp und liefert ggf. eine Bezeichnung für das `SubmodelElement`.
- Ein `Submodel`/`SubmodelElement` kann mittels `semanticId` auf eine `ConceptDescription` verweisen (`GlobalReference`). Diese `ConceptDescription` verweist schließlich mit `isCaseOf` auf die externe semantische Definition im Semantic Lookup. Die Auswertung dieser `isCaseOf` Information erfolgt auf gleiche Weise wie bei der „direkten“ Referenz. Dies ist die bevorzugte Einbindung von externen Systemen.

4.4.6 Instanziierung auf Basis von Templates

Wird eine `AssetAdministrationShell` oder ein `Submodell` auf Basis eines Templates intanziiert, dann muss

- die `AssetAdministrationShell` mit `AssetKind.INSTANCE` per `derivedFrom` auf sein Template verweisen,
- jedes `Submodel`/`SubmodelElement` per `semanticId` auf sein Template verweisen.

Die genannten Referenzen sind so genannte `ModelReference`-Veweise. Diese zeigen auf ein existierendes AAS-Element (AAS, Teilmodell bzw. Teilmodell-Element), welche wiederum eine semantische Information bereithalten können. Selbst bei mehrstufigen Hierarchien in den Template-Elementen enthält eine Instanz so die gesamte Verweiskette bis zum Root-Element!

Bei der Instanziierung von Teilmodellen in einem Application Connector wird darauf geachtet, dass alle referenzierten semantischen Definitionen per `ModelReference` abgearbeitet werden.

4.5 Semantic Integration Patterns in sechs Schritten

In diesem Abschnitt wurden die wesentlichen Schritte bzw. AAS-Elemente behandelt, welche zur Erstellung und Nutzung von Semantic Integration Patterns erforderlich sind.

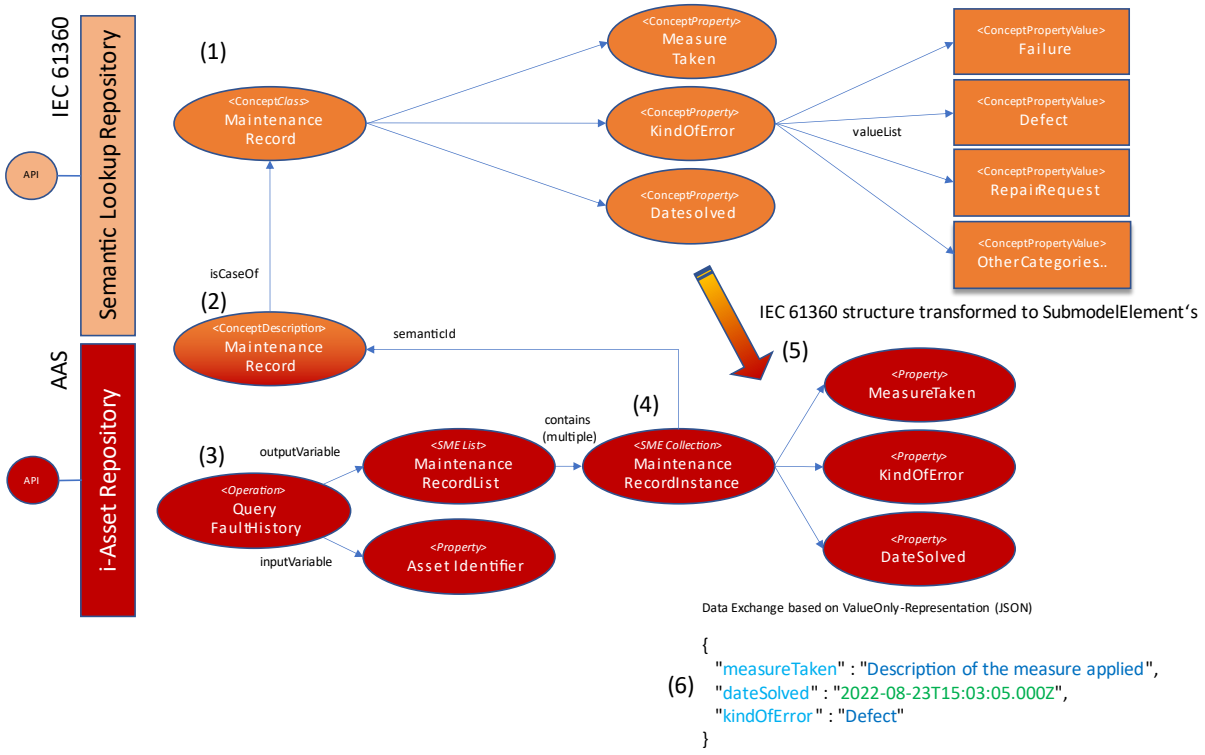


Abbildung 30: Semantic Integration Pattern in Use

Am Beispiel einer Abfrage der Wartungshistorie eines Assets aus einem CMMS sind diese Schritte in Abbildung 30 nochmals komprimiert zusammengefasst:

(1) Die für die Kommunikation erforderlichen Datenobjekte in einem externen System verwaltet. Durch die Verwendung eines externen Systems wird die Verteilung und Wiederverwendung dieser Definitionen ermöglicht.

(2) In Asset Repository werden `ConceptDescription` Elemente für diese, extern definierten Datenobjekte angelegt. Diese verweisen mittels `isCaseOf`-Referenzen auf die definierten Objekte.
(3) Im Asset Repository werden Kommunikations-Templates erstellt. Diese nutzen AAS-Teilmodell-Elemente zur Definition der Datenaustausch-Objekte. Diese Teilmodell-Elemente werden per semanticId mit den `ConceptDescription`-Elementen verbunden, welche die Datenobjekte definieren.
(4) Im Asset Connector werden diese Beziehungen ausgewertet und sichergestellt, dass die strukturellen Vorgaben gemäß `ConceptDescription` & `isCaseOf` eingehalten werden.
(5) Die vordefinierte Struktur aus gemeinsam genutzten Systemen wird auf die AAS-Teilmodell-Elemente angewandt und so sichergestellt, dass die ausgetauschten Datenobjekte den Definitionen entsprechen
(6) Die Serialisierung der AAS-Teilmodell-Elemente mittels ValueOnly-Serialisierung ermöglicht einen „reduzierten“ Datenaustausch unter gleichzeitiger Verfügbarkeit aller semantischer Definitionen. Anwendungen können mit Hilfe des Application Connectors die Datenobjekte in „reduzierter“ Form bereitstellen bzw. empfangen.

Ein Application Connector ermöglicht die Nutzung von Semantic Integration Pattern. Diese werden anhand ihrer `semanticId` im Asset Repository identifiziert und enthalten die vollständige semantische Beschreibung aller Informationen um eine Kommunikation anhand/über dieses Pattern initiieren bzw. beantworten zu können. Ein Pattern ist somit immer **vollständig** am Application Connector vorhanden.

Semantic Integration Patterns sind auch **austauschbar**, da sie mit den Mitteln der AAS-Spezifikation als AASX-Package exportiert und wieder importiert werden können.

Und sie sind auch flexibel **erweiterbar**, da sowohl die Datenobjekte (IEC 61360) als auch die AAS-Teilmodelle das Konzept der Vererbung beherrschen. Beide Standards unterstützen Versionen und Revisionen.

Unbedingte Voraussetzung für die Etablierung von Semantic Integration Patterns ist der Application Connector, der eine Laufzeitumgebung zur Verfügung stellt. Dieser sorgt dafür, dass Teilmodelle gegebenenfalls vom Asset Repository nachgeladen werden, so sie noch nicht verfügbar sind.

Der Application Connector bietet den Anwendungen auch die Möglichkeit, die transportierten ValueOnly-Daten in proprietäre, anwendungsspezifische Datenobjekte zu transformieren. Dadurch wird der Integrationsaufwand auf ein Minimum reduziert.

5 Semantic Integration Patterns: Umsetzung

5.1 Überblick

Im Folgenden werden die aus der Requirements-Analyse abgeleiteten Integration Patterns for Manufacturing in verschiedenen Reifegraden der Spezifikation. Diese Patterns werden gemäß dem Projektplan im folgenden Berichtszeitraum weiter ausgearbeitet und in der Middleware, genauer in den Connectoren, verfügbar gemacht. Eine Spezialisierung der Semantic Integration Patterns für analytische Systeme erfolgt in einem weiteren Bericht (D2.3).

Zur Beschreibung der Semantic Integration Patterns in Form von AAS-Teilmodellen und der Submodel Element Collections verwenden wir ein von der Plattform Industrie 4.0 (ZVEI) entwickeltes Template[14], welches im Folgenden angeführt wird:

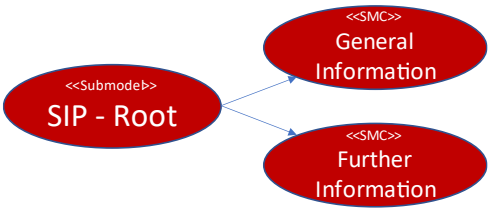
idShort			
AAS-Modelclass			
semanticId			
Parent			
Description@en			
Explanation			
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	

5.1.1 Submodel: Semantic Integration Pattern

Einzelne Semantic Integration Patterns sind in Teilmodellen (Submodel) zusammengefasst. Sie können im Asset Repository anhand ihres eindeutigen Identifier gefunden werden. Um jedoch eine Suchmöglichkeit zu erhalten, welche Semantic Integration Patterns vorhanden sind, muss eine Möglichkeit zur Klassifizierung als Semantic Integration Pattern gefunden werden.

Dies wird mit einem Submodel-Template erreicht, welches als Parent-Element für alle weiteren Semantic Integration Patterns dient. Das Root-Template erhält auch Teilmodell-Elemente in denen weiterführende, statische Informationen angesiedelt sind.

[14] Muster für Template: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2020/Dezember/Submodel_Templates_of_the_Asset_Administration_Shell/201117_I40_ZVEI_SG2_Submodel_Spec_ZVEI_Technical_Data_Version_1_1.pdf



Das Root-Template wird mit einer `semanticId` klassifiziert. Diese `semanticId` ist wesentlich, um im Asset Repository nach Submodel-Instanzen suchen zu können, die von diesem Root-Template abgeleitet sind.

identifier	https://iasset.salzburgresearch.at/SemanticIntegrationPattern		
idShort	SemanticIntegrationPattern		
AAS-Modelclass	Submodel		
semanticId	https://admin-shell.io/SRFG/SIP/1/1		
Parent	-		
Description@en			
Explanation	Single root entry point for retrieving all Submodel belonging to Semantic Integration Patterns		
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[SMC] GeneralInformation	https://admin-shell.io/SRFG/SIP/GeneralInformation/1/1 General Information, manufacturer Information	n/a	1
[SMC] FurtherInformation	https://admin-shell.io/SRFG/SIP/FurtherInformation/1/1 Further Information of the SIP. Validity, Usage	n/a	0..1
[SMC] Operations	https://admin-shell.io/SRFG/SIP/Operations/1/1 Container for Operations	n/a	0..1
[SMC] Events	https://admin-shell.io/SRFG/SIP/Events/1/1 Container for Event Settings	n/a	0..1

5.1.2 SubmodelElementCollection: General Information

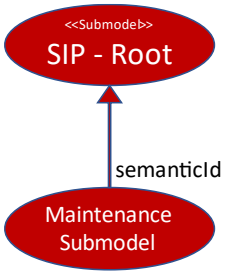
Jedes Semantic Integration Pattern erhält Teilmodell-Elemente, um die Informationen zu organisieren.“ Die „GeneralInformation“ ist hier beispielhaft angeführt und legt den Ersteller des Patterns bzw. ein Logo fest.

idShort	GeneralInformation
AAS-Modelclass	SubmodelElementCollection
semanticId	https://admin-shell.io/SRFG/SIP/GeneralInformation/1/1
Parent	Submodel with semanticId https://admin-shell.io/SRFG/SIP/Submodel/1/1
Description@en	
Explanation	

[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[Property] Creator	https://admin-shell.io/SRFG/SIP/Creator/1/1 Name of the Creator/Maintainer	[string] Example Company	1
[Blob] CreatorLogo	https://admin-shell.io/SRFG/SIP/CreatorLogo/1/1 Image-File for logo of manufacturer	MimeType=image/png	0..1

5.1.3 Submodel: Maintenance

Stellt einen ersten Einstiegspunkt als Semantic Integration Pattern dar. Anhand der `semanticId` wird die Vererbung abgebildet.

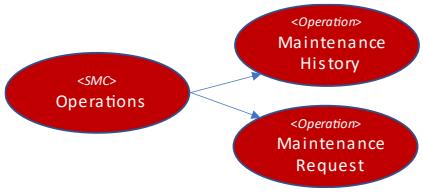


Das Maintenance Submodel erbt somit die Struktur-Information aus dem Root-System (GeneralInformation, FurtherInformation) und fügt weitere Informationen hinzu.

identifier	https://iasset.salzburgresearch.at/SIP/Maintenance		
idShort	Maintenance		
AAS-Modelclass	Submodel		
semanticId	https://iasset.salzburgresearch.at/SemanticIntegrationPattern Die semanticID verweist auf das Root-Teilmodell, dieses wiederum ist mit dem wichtigen Bezeichner https://admin-shell.io/SRFG/SIP/1/1 klassifiziert.		
Parent	-		
Description@en	Container for Maintenance related Integration Patterns		
Explanation			
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[SMC] GeneralInformation	https://admin-shell.io/SRFG/SIP/GeneralInformation/1/1 Name of the Creator/Maintainer	[string] H+H Systems	1
[SMC] Operations	https://admin-shell.io/SRFG/SIP/Operations/1/1 Methods	-	0..1

5.1.4 SubmodelElementCollection: Operations

Das Maintenance Submodel fügt im Teilmodell-Element „Operations“ die entsprechenden Operationen hinzu.



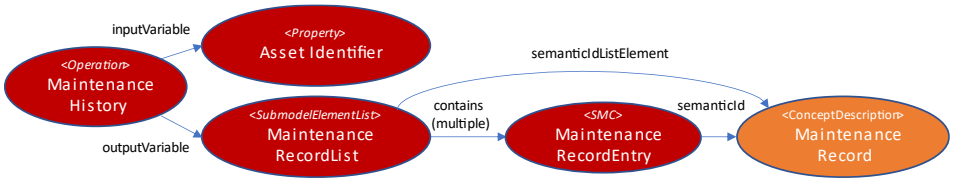
Hier sind exemplarisch zwei Funktionen dargestellt:

- Abfrage einer Wartungshistorie
- Wartungsanforderung bzw. Störmeldung erstellen

idShort	Operations		
AAS-Modelclass	SubmodelElementCollection		
semanticId	https://admin-shell.io/SRFG/SIP/Operations/1/1		
Parent	Submodel mit semanticId https://iasset.salzburgresearch.at/SemanticIntegration-Pattern		
Description@en	Container for Maintenance Related Operations		
Explanation			
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[Operation] MaintenanceHistory	https://admin-shell.io/SRFG/SIP/MaintenanceHis-tory/1/1 Method to query the maintenance history of an asset		1
[Operation] MaintenanceRequest	https://admin-shell.io/SRFG/SIP/Mainte-nanceRequest/1/1 Methods		1

5.1.5 Operation: Maintenance History

Die Abfrage einer Wartungshistorie für ein Asset erfordert als Input-Parameter den eindeutigen Bezeichner des Assets. Als Ergebnis wird eine Liste von Wartungseinträgen erwartet.



idShort	MaintenanceHistory
AAS-Modelclass	Operation

semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceHistory/1/1		
Parent	Submodel with semanticId https://admin-shell.io/SRFG/SIP/Maintenance/1/1		
Description@en	Operation defining the functionality for Maintenance History Requests		
Explanation			
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
inputVariables			
[Property] assetIdentifier	https://admin-shell.io/SRFG/SIP/AssetIdentifier/1/1 Asset Identifier	[string|	1
outputVariables			
[SML] maintenanceHistory	https://admin-shell.io/SRFG/SIP/MaintenanceHistory/ResultList/1/1 SubmodelElementList holding the result of the method execution [semanticIdListElement] https://admin-shell.io/SRFG/SIP/MaintenanceHistory/MaintenanceRecord/1/1 Each of the contained elements in the list must match the definition provided in the `semanticIdListElement`.		1

5.1.6 ConceptDescription: MaintenanceRecord

Als Antwort wird eine Liste von Maintenance Records erwartet. Diese Liste besteht aus Einträgen, deren Semantik mit `semanticIdListElement` definiert ist. Mit Hilfe einer ConceptDescription wird die Struktur des List-Elements definiert. Die `isCaseOf`-Beziehung zeigt auf eine Konzept-Klasse aus IEC 61360. Dieses hat die entsprechenden Attribute zugewiesen.

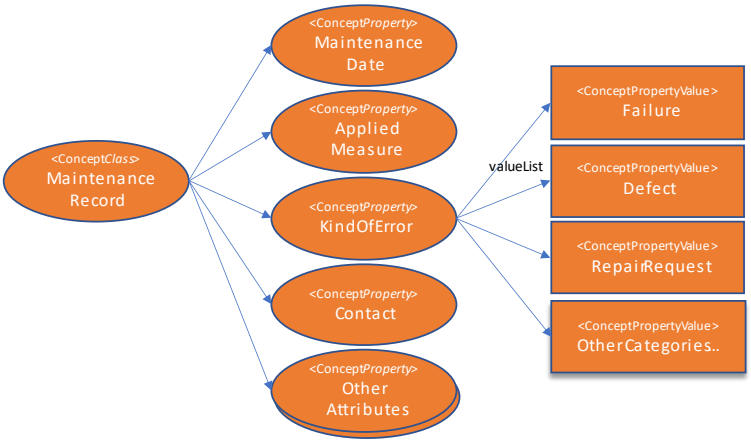


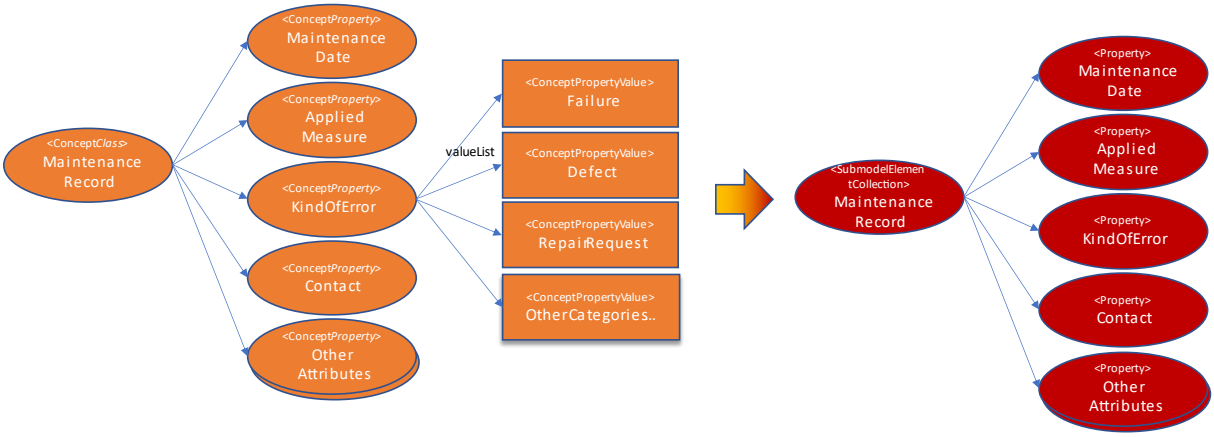
Abbildung 31: Struktur des MaintenanceRecords

idShort	MaintenanceRecord
AAS-Modelclass	ConceptDescription
semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceHistory/MaintenanceRecord/1/1
Parent	-

Description@en	ConceptDescription outlining a Maintenance Record entry		
Explanation	The concept description provides the interlinking of the AAS meta model with an external definition of the semantics		
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[isCaseOf] MaintenanceHistory	https://admin-shell.io/SRFG/SIP/MainenanceHistory/1/1 IRDI or IRI of the concept in an external system	n/a	1
[embeddedDataSpec] Or [hasDataSpecification]	Conforms to the ConceptClass Definition out of Semantic Lookup Service! Replicates the structure of an external Semantic Lookup System!	DataSpecificationIEC61360	0..1

5.1.7 SubmodelElementCollection: Maintenance Record

Zur Laufzeit wird das Listen-Element `maintenanceHistory` mit den entsprechenden Einträgen aus einem angeschlossenen System befüllt. Dabei wird darauf geachtet, dass jeder Eintrag der semantischen Definition entspricht.



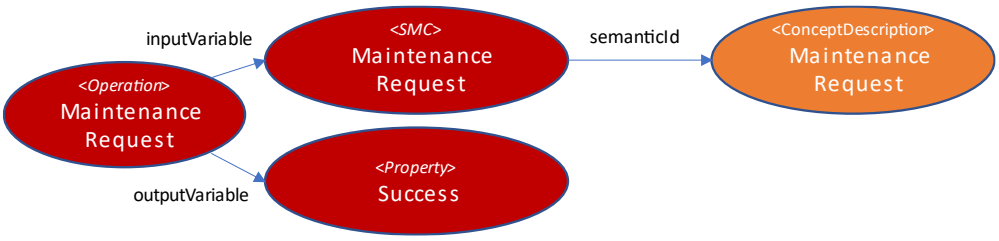
Die Definitionen aus IEC 61360 kompatiblen Systemen werden in die AAS-Struktur überführt.

idShort	MaintenanceRecord		
AAS-Modelclass	SubmodelElementCollection		
semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceHistory/MaintenanceRecord/1/1		
Parent	-		
Description@en	SubmodelElementCollection holding the distinct attributes of a Maintenance Record Entry		
Explanation	The SubmodelElementCollection holds the details of the Maintenance Record.		
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[Property]	https://admin-shell.io/SRFG/SIP/MaintenanceRecord/MaintenanceDate/1/1	DATE	1

MaintenanceDate	Date/time when the maintenance activity took place		
[Property] AppliedMeasure	https://admin-shell.io/SRFG/SIP/MaintenanceRecord/AppliedMeasure/1/1 Measure applied to the asset	STRING	1
[Property] KindOfError	https://admin-shell.io/SRFG/SIP/MaintenanceRecord/KindOfError/1/1 Classification of the Error - Coded value	STRING Failure	1
[Property] Contact	https://admin-shell.io/SRFG/SIP/MaintenanceRecord/Contact/1/1 Executor of the maintenance activity	STRING	1

5.1.8 Operation: Maintenance Reqzest

CMMS-Systeme können Wartungsmeldungen entgegennehmen. Diese sieht als Input-Parameter eine Wartungsanforderung vor. Das Prinzip ist dabei gleich wie bei der Abfrage der Wartungshistorie:



idShort	MaintenanceRequest		
AAS-Modelclass	Operation		
semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1		
Parent	Submodel with semanticId https://admin-shell.io/SRFG/SIP/Maintenance/1/1		
Description@en	Operation defining the functionality for Maintenance Requests		
Explanation			
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
inputVariables			
[SMC] MaintenanceRequest	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1 SubmodelElementCollection conforming to the definition of Maintenance Requests The semanticId points to a ConceptDescription	[string|	1
outputVariables			
[Property] success		[Boolean]	1

5.1.9 ConceptDescription: Maintenance Request

Dieses Element stellt die Verbindung zum externen Semantic Lookup System dar. Anhand der Beziehung `isCaseOf` wird auf das entsprechende Element im Semantic Lookup verwiesen.

idShort	MaintenanceRequest		
AAS-Modelclass	ConceptDescription		
semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1		
Parent	-		
Description@en	ConceptDescription outlining a Maintenance Request		
Explanation	The concept description provides the interlinking of the AAS meta modell with an external definition of the semantics		
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[isCaseOf] MaintenanceHistory	https://admin-shell.io/SRFG/SIP/Mainenan-ceRequest/1/1 IRDI or IRI of the concept in an external system	n/a	1
[embeddedDataSpec] Or [hasDataSpecification]	Conforms to the ConceptClass Definition out of Seman-tic Lookup Service! Replicates the structure of an external Semantic Lookup System!	DataSpecifica-tionIEC61360	0..1

5.1.10 SubmodelElementCollection: Maintenance Request

Entspricht dem AAS-Abbild der `ConceptDescription`. Die einzelnen Elemente werden durch die `isCaseOf`-Beziehung definiert.

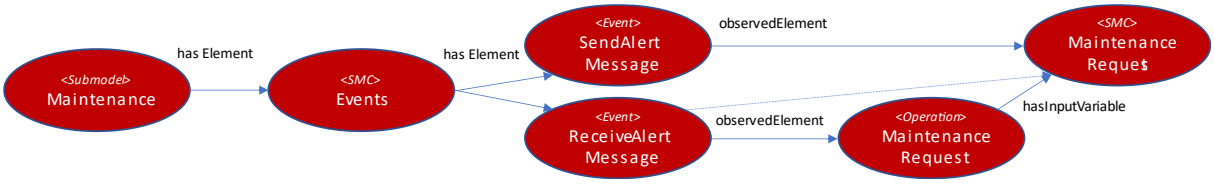
idShort	MaintenanceRequest		
AAS-Modelclass	SubmodelElementCollection		
semanticId	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1		
Parent	-		
Description@en	SubmodelElementCollection holding the distinct attributes of a Maintenance Re-quest Entry		
Explanation	The concept description provides the interlinking of the AAS meta modell with an external definition of the semantics		
[SME type]	semanticId	[valueType]	card.
idShort	Description@en	example	
[Property] assetId	https://admin-shell.io/SRFG/SIP/Mainte-nanceRequest/AssetId/1/1 Identifier of the Asset the Record belongs to	STRING n/a	1
[Property] subElementPath	https://admin-shell.io/SRFG/SIP/Mainte-nanceRequest/SubElementPath/1/1	STRING	1

	Measure applied to the asset		
[Property] faultId	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/FaultIdentifier/1/1 Classification of the Error	STRING	1
[Property] timeStampCreated	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/TimeStampCreated/1/1 Date/Time when the error has been detected	TIMESTAMP	1
[Property] timeStampFinished	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/TimeStampFinished/1/1 Date/Time when the error has been resolved	TIMESTAMP	0..1
[Property] faultCode	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/FaultCode/1/1 Error Classification Code (Coded Value)	STRING	1
[Property] shortText	https://admin-shell.io/SRFG/SIP/ShortText/1/1 Short Description of the Error	STRING	1
[Property] longText	https://admin-shell.io/SRFG/SIP/LongText/1/1 Extended description of the Error	STRING	0..1
[Property] priority	https://admin-shell.io/SRFG/SIP/MaintenanceRequest/Priority/1/1 Priority of the Request	INTEGER	1
[Property] status	https://admiin-shell.io/SRFG/SIP/MaintenanceRequest/Status/1/1 Current Status of the Maintenance Request (Coded Value)	STRING	1

5.1.11 Event: Alert Message

Ein Semantic Integration Pattern ermöglicht auch asynchrone Kommunikation. Diese werden mit Hilfe von Event-Elementen modelliert, welche ebenfalls im Teilmodell integriert und anhand ihrer `semanticId` innerhalb einer AAS identifiziert werden.

Ein Event-Element steht immer nur für eine Kommunikationsrichtung, daher sind je ein Element zum Senden bzw. zum Empfangen einer Alert Message erforderlich



Für das Event-Element ist das Attribut `observedElement` wesentlich. Dieses definiert den Payload jener Daten, die über die Messaging-Infrastruktur versendet werden.

Senden einer Wartungsanforderung

idShort	Send Alert Messsage
AAS-Modelclass	EventElement
semanticId	https://admin-shell.io/SRFG/SIP/SendAlertMessage/1/1
Parent	Submodel with semanticId https://admin-shell.io/SRFG/SIP/Maintenance/1/1
Description@en	EventElement allowing the asynchronous submission of Maintenance Requests
Explanation	
observedElement	AAS-Element with `semanticId` https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1
direction	Out

Empfangen einer Wartungsanforderung

idShort	Receive Alert Messsage
AAS-Modelclass	EventElement
semanticId	https://admin-shell.io/SRFG/SIP/ReceiveAlertMessage/1/1
Parent	Submodel with semanticId https://admin-shell.io/SRFG/SIP/Maintenance/1/1
Description@en	EventElement allowing the asynchronous submission of Maintenance Requests
Explanation	
observedElement	1. Operation Element mit semanticId https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1: Der Payload ist durch die Input-Variable des Operation-Elements definiert. Dieser wird Aufruf-Parameter an das Operation-Element übergeben! Das Operation-Element muss „aktiviert“ sein! 2. SMC-Element mit semanticId https://admin-shell.io/SRFG/SIP/MaintenanceRequest/1/1: Die Anwendung erhält den Payload und kann selbst den Maintenance Request erzeugen.
direction	In

5.2 Exemplarische Darstellung am Beispiel von CMMS-Anwendungen

Im Abschnitt 4.4 und wurden die Kommunikations-Mechanismen für Anwendungen, bereitgestellt durch einen Application Connector beschrieben. Dabei wurden die Elemente für Methoden-Aufrufe (`Operation`) und Ereignisse (`Event`) genauer betrachtet. Auch die semantische Beschreibung der jeweils genutzten Datenstrukturen (`OperationVariable` bzw. `observed`-Element) ist durchgängig gegeben. Dies betrifft jedoch nur die Anwendung selbst, da diese selbst diese Strukturen vorgeben. Jedoch müssen auch andere Anwendungen und vor allem Assets in der Lage sein, die auszutauschenden Datenstrukturen eindeutig zu identifizieren, damit sie mit den Empfängern (der Anwendung) auch Daten austauschen können. Dieser Abschnitt versucht, verschiedene Kommunikations-Formen am Beispiel von CMMS-Anwendungen zu verdeutlichen.

5.2.1 Abrufen einer Wartungshistorie (Maintenance History)

Anwendungen stellen Methoden bereit, um Anfragen zu beantworten bzw. um Aktionen innerhalb einer Anwendung zu initiieren. Methoden erwarten in der Regel ein oder mehrere Input-Parameter und führen dann ihre Aktionen aus. Methoden können aber auch dazu genutzt werden um Informationen von den Anwendungen zu erhalten, d. h. die Methode gibt Daten an die aufrufende Anwendung zurück. Beispielhaft kann hier die Abfrage der Wartungshistorie für ein Asset genannt werden. Als Ergebnis wird hier die Liste aller durchgeführten Wartungen, z.B. mit Datum der Wartung, Durchgeführte Tätigkeit, Art der Wartung etc. erwartet.

Hier gilt es zunächst, im Semantic Lookup Repository die Datenstruktur für einen Wartungseintrag zu hinterlegen. Hierzu wird eine Konzept-Klasse für Wartungen definiert und mit einem global gültigen Bezeichner versehen. Zusätzlich zur Konzept-Klasse werden Attribute (ebenfalls mit global gültigen Bezeichnern) definiert und der Klasse zugewiesen.

Im Asset Repository wird im Teilmodell für Wartungen ein `Operation`-Element eingefügt, welches als Input-Parameter (`OperationVariable`) das abgefragte Asset erwartet. Als Rückgabewert wird die Liste der durchgeführten Wartungen erwartet. Auch hierfür muss in der Asset Administration Shell ein entsprechendes Element definiert werden, welches schließlich mittels `semanticId` mit der Konzept-Klasse für Wartungen im Semantic Lookup Repository verbunden ist. Die erforderlichen Elemente für diese Anforderung sind in Abbildung 32 dargestellt.

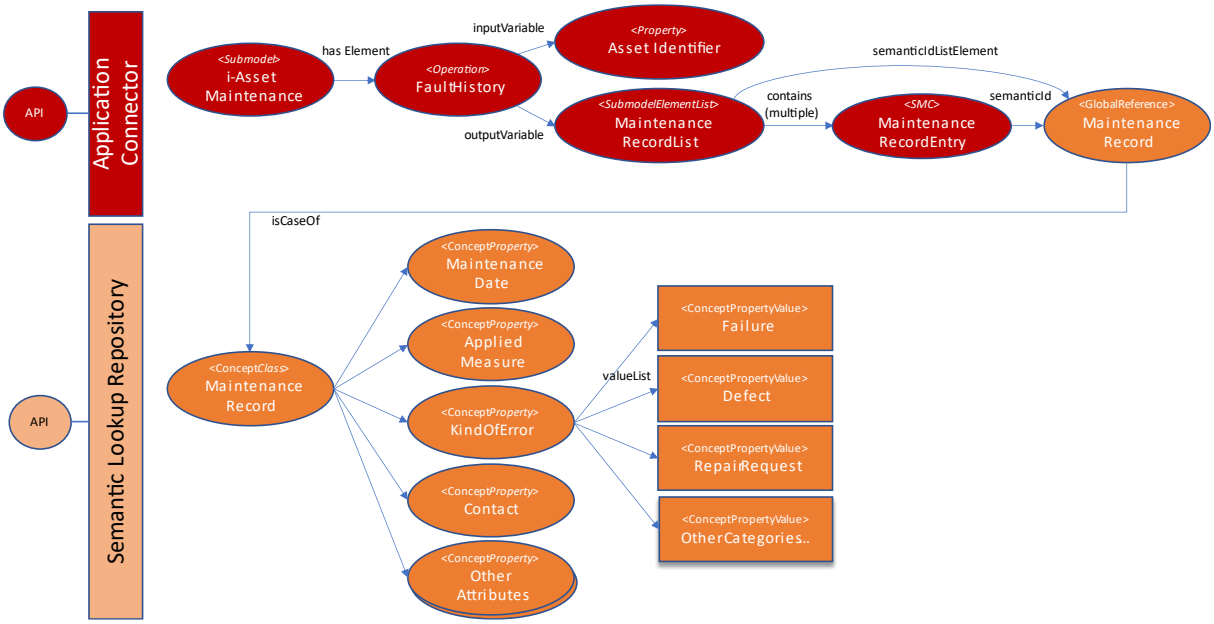


Abbildung 32: Anwendungs-Typ CMMS - Methoden

Wieder erfolgt der Zugriff auf die Elemente über die (vordefinierte bzw. konfigurierte) `semanticId` des Anwendungs-Typs-Elements *Fault History*. Dieses Teilmodell-Element vom Typ `Operation` definiert sowohl `inputVariable` (Asset Identifier) wie auch `outputVariable` (*Maintenance Record List*). Der Rückgabewert ist eine Liste, welche gleichartige Elemente enthält wobei die semantische Auszeichnung für das Listen-Element vom Typ `SubmodelElementCollection` wiederum auf die allgemein gültige Konzept-Klasse für *Maintenance Record* (hier exemplarisch dargestellt) im Semantic Lookup Repository zeigt. Im Semantic Lookup Repository können individuelle Ausprägungen von Wartungen, je nach eingesetzter Anwendung definiert werden. Diese Möglichkeit von hierarchischen Strukturen für Konzept-

Klassen (`ConceptClass`) und der individuellen Zuweisung von Attributen (`ConceptProperty`) ist im Semantic Lookup Repository vorgesehen. Für die vollständige semantische Beschreibung einer Wartung muss innerhalb der Asset Administration Shell einer konkreten Anwendungs-Instanz lediglich die korrekte `semanticId` in der Anwendungs-Instanz gesetzt sein.

Assets und andere Anwendungen müssen letztlich in die Lage versetzt werden, diese Funktion „Abruf der Wartungs-Historie“ aufrufen zu können. Dazu reicht es, im Asset Repository anhand der `semanticId` des Anwendungs-Typs CMMS das relevante `Operation`-Element zu finden. Anhand des `Operation`-Elements erhalten aufrufende Assets/Anwendungen auch die erforderliche Information über die erforderlichen Input-Parameter, die erzeugte Antwort und die dabei verwendeten Datenstrukturen (via `semanticId`). Für die Ausführung der Operation kann nun das Asset Repository beauftragt werden. Dieses dient als Proxy und reicht den Request an die aktive Anwendung weiter. Der „Application Connector“ für die aktive Anwendung erhält die Anfrage und kann die übermittelten Daten verarbeiten und ggf. für die Anwendung aufbereiten. Die angesprochene Anwendung führt schließlich die Operation aus und retourniert das proprietäre Ergebnis zunächst an den Application Connector. Dieser transformiert das Ergebnis zurück in das Format gemäß `Operation`-Element und retourniert seine Antwort an den Aufrufer.

5.2.2 Senden einer Störmeldung (Maintenance Request)

Ein Asset (bzw. die Steuerung eines Assets) stellt fest, dass eine Fehlfunktion vorliegt. Es soll daher eine Störmeldung an das zuständige CMMS gesendet werden. Diese Störmeldung kann aber nur vom CMMS verarbeitet werden, wenn diese auch korrekt aufgebaut ist. Die einzelnen Schritte für das Asset, um eine Störmeldung absetzen zu können sind nachfolgend aufgelistet:

1. Es wird zunächst das Teilmodell für die Kommunikation mit CMMS gesucht. Als Suchkriterium wird die `semanticId` für CMMS-Interaktion herangezogen.
2. Innerhalb des Teilmodells für CMMS wird das `EventElement` für Störmeldung gesucht. Auch hier wird als Suchkriterium die `semanticId` für CMMS/Störmeldung verwendet.
3. Das identifizierte `EventElement` definiert den zu verwendenden Message-Broker und auch das definierte Message-Topic. Zudem verweist es mit seinem `observable`-Element auf ein `Entity`-Element (bzw. `SubmodelElementCollection`) welche die Störmeldung des aktiven CMMS Systems definiert.
4. Die `semanticId` des `observable`-Element verweist auf die Struktur der Störmeldung und liefert somit die einzelnen Elemente, die ausgefüllt werden müssen. Dabei steht die komplette Meta-Information der Störmeldung zur Verfügung. Dies umfasst Kurzname, Bezeichnung, Datentyp, ggf. Einheit oder eine vordefinierte Werteliste (z.B. Störursachen-Codes).
5. Das Asset befüllt die einzelnen Werte und ermittelt den Inhalt der Störmeldung indem letztlich nur die einzelnen Elemente als Key-Value Paare aufgelistet werden.
6. Das Asset verwendet das eingangs ermittelte `EventElement` für Störmeldungen, um die Nachricht zu publizieren.

Die für diesen Ablauf erforderlichen Elemente sind in Abbildung 33 dargestellt. Das Asset erhält im Asset Connector jenen Ausschnitt der Anwendungs-Instanz, um die Kommunikation mit der Anwendung aufbauen zu können. Die `semanticId` für das `Submodel` bzw. für das

`EventElement` wird vom Anwendungs-Typ auf die Anwendungs-Instanz übertragen. Individuell angepasst wird jedoch `semanticId` des `observed`-Elements wie in Abbildung 33 dargestellt. Das `observed`-Element des `EventElement` ist für den Payload der Nachricht verantwortlich.

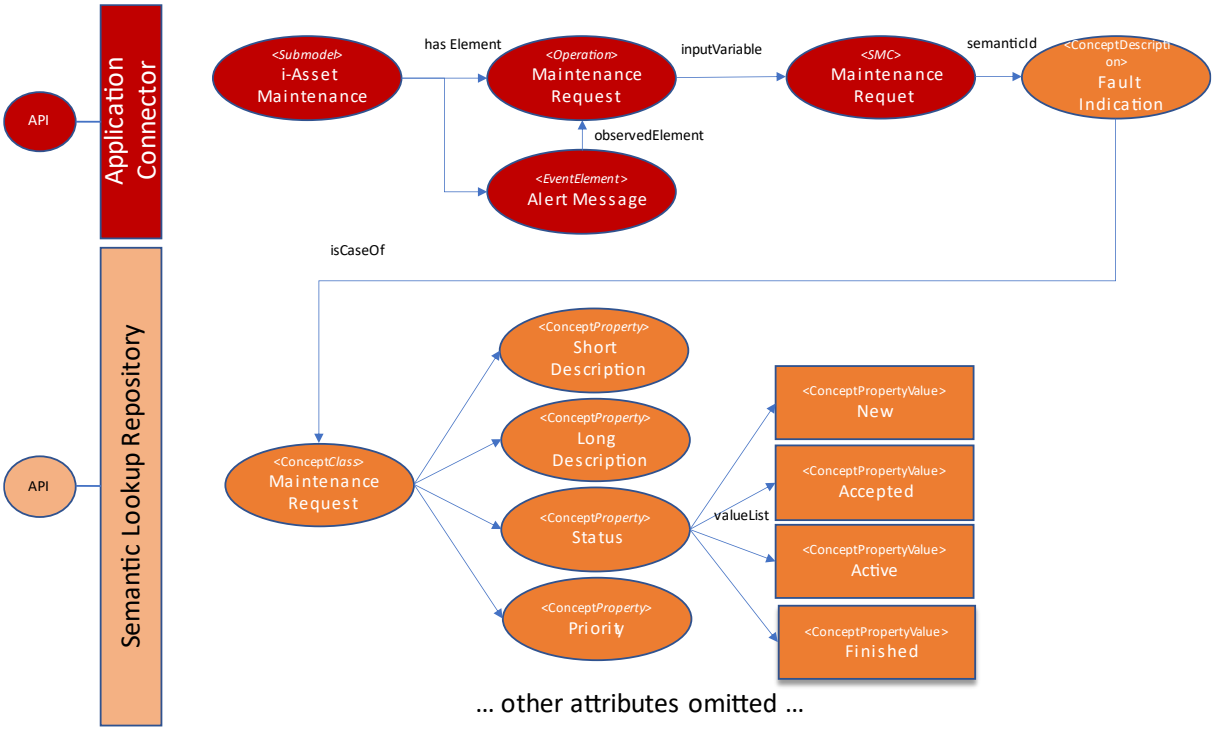


Abbildung 33: Asset-Instanz - Nachricht an CMMS erstellen

Während die Anwendungs-Instanz für CMMS „eingehende“ Nachrichten erwartet, muss die Asset-Instanz diese Nachrichten erzeugen/absenden. Die Einstellungen im Teilmodell für Maintenance in der Asset-Instanz müssen entsprechend angepasst werden. So muss z.B. die Richtung der Nachrichten-Kommunikation angepasst werden.

Eine wichtige Fragestellung für dieses Szenario lautet, wie bzw. wann eine Asset-Instanz das Teilmodell für die Kommunikation mit dem CMMS erhält, so dass dieses im Störungsfall auch verfügbar ist. Hier sind verschiedene Strategien möglich.

- Das Teilmodell wird dem Asset „injiziert“, sobald dieses sich in der i-Asset Plattform anmeldet. Im Anlassfall (Auftreten einer Störung) sind alle Informationen bereits im Asset Connector verfügbar. In der i-Asset Plattform wird zudem festgehalten, welche Assets welche Topics definieren. Diese Information wird im Distribution-Network festgehalten.
- Das Asset konsultiert erst im Problemfall (bei Auftreten einer Störung) die i-Asset Plattform und ermittelt das erforderliche Teilmodell für die Kommunikation zur Laufzeit. Die Kommunikation zwischen der i-Asset Plattform und dem Asset erlaubt es auch, während der Laufzeit einzelne Elemente bzw. komplette Teilmodelle auszutauschen.

5.2.3 Verarbeiten einer Störmeldung

Auf der Empfänger-Seite definiert das CMMS ebenfalls eine Asset Administration Shell. Dieses enthält ebenfalls das `EventElement` Störmeldung, diesmal jedoch als Nachrichten-Empfänger. Es gilt wiederum – das `EventElement` definiert den Message Broker sowie auch das

Message Topic auf dem sich das CMMS als Message Consumer registrieren soll. Das `observedElement` verweist auf ein `Operation`-Element welches als `OperationVariable` das `Entity`-Element Störmeldung erwartet. Mit Hilfe der `semanticId` des `Entity`-Elements ist dabei die Struktur des Nachrichten-Inhalts definiert.

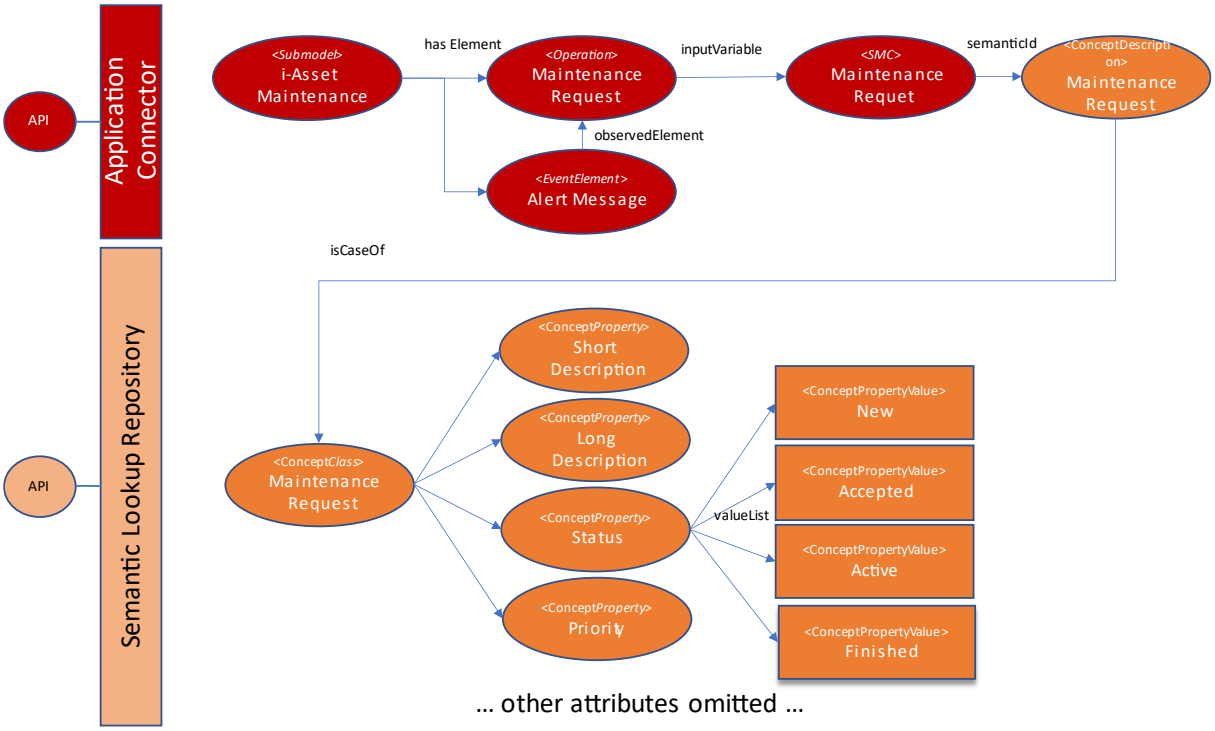


Abbildung 34: Anwendungs-Typ CMMS – eingehende Nachrichten

Eingehende Nachrichten werden schließlich geprüft, ob diese auch die richtige `observableSemanticId` enthält. Diese muss mit der `semanticId` des `Entity`-Element *Alert Indication* übereinstimmen. Ist das der Fall, kann die Nachricht an verschiedenen Stellen weiter auf ihre Korrektheit evaluiert werden:

1. Vollständig, gültig: Der „Application Connector“ kann auf Basis der AAS Struktur-Elemente überprüfen, ob die Nachricht vollständig und gültig ist, sprich ob im Payload alle erforderlichen Attribute enthalten sind und die Werte auch im richtigen Datenformat sind.
2. Im Application Connector werden `Operation`-Element bzw. auch `Property`-Elemente mit individuellen Methoden versehen. Diese Methoden (siehe dazu Abbildung 10 in Abschnitt 3.2.1) interagieren mit der Steuerung des Assets (Asset Connector) bzw. mit der API der „dahinter liegenden“ Anwendung (Application Connector). An dieser Stelle müssen oftmals proprietäre Datenobjekte erzeugt/verarbeitet werden, wie sie in der API verlangt werden. Auch hier findet eine Evaluierung der Nachricht statt. Ist diese Evaluierung erfolgreich, wird auch die API der Anwendung damit befasst. An dieser Stelle erfolgt die Transformation zwischen der I4.0/AAS Welt in die proprietäre Datenstruktur der jeweiligen Anwendung.
3. Die Anwendung selbst prüft eingehende Nachrichten und gibt entsprechende Rückmeldung, ob die Verarbeitung erfolgreich durchgeführt werden konnte.

Zusammengefasst: Erst wenn alle Details korrekt validiert sind, dann wird die Störmeldung erzeugt und an das `Operation`-Element übergeben. Damit diese dann auch tatsächlich im CMMS ankommt muss die `Operation` wie in Abschnitt 3.2.1 beschrieben mit dem CMMS verbunden sein. Dies ist dann der Fall, wenn dem `Operation`-Element eine Methode „injiziert“

wurde, welche als Argument die Störmeldung verarbeiten kann und diese letztlich an das dahinterliegende CMMS weitergibt.

Generell gilt jedoch für Asset/Application Connectoren und der Verarbeitung eingehender Nachrichten:

- Wenn bei eingehenden `EventElement` ein `Operation`-Element als `observed` definiert ist, so muss die eingehende Nachricht die Input-Variablen der Operation als Payload enthalten. Der Asset/Application Connector führt dann die hinterlegte Methode aus.
- Wenn bei eingehenden `EventElement` ein Property-Element als `observed` definiert ist, so muss die eingehende Nachricht einen gültigen Wert für das `Property`-Element erhalten. Der Asset/Application Connector verändert den Wert für das `Property` Element, indem die Value-Setter-Methode für das `Property`-Element (Siehe Abschnitt 3.2.1) aufruft.

6 Literaturverzeichnis

AAS Part 1. (2023, April). *AAS Part 1: Metamodel.* Retrieved 14. September 2023, from IDTA - Der Standard für den Digitalen Zwilling: https://industrialdigitaltwin.org/wp-content/uploads/2023/06/IDTA-01001-3-0_SpecificationAssetAdministrationShell_Part1_Metamodel.pdf

AAS Part 2. (2023, Juni). *AAS Part 2: Application Programming Interfaces.* Retrieved September 2023, from IDTA - Der Standard für den Digitalen Zwilling: https://industrialdigitaltwin.org/wp-content/uploads/2023/06/IDTA-01002-3-0_SpecificationAssetAdministrationShell_Part2_API_.pdf

AAS Part 3a. (2023, April). *AAS Part 3a: Data Specification - IEC 61360.* Retrieved from IDTA - Der Standard für den Digitalen Zwilling: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/04/IDTA-01003-a-3-0_SpecificationAssetAdministrationShell_Part3a_DataSpecification_IEC61360.pdf

Åkerman, M. (2018). *Implementing Shop Floor IT for Industry 4.0, Doctoral Thesis*. Abgerufen am 20. Juli 2022 von https://www.researchgate.net/publication/326224890_Implementing_Shop_Floor_IT_for_Industry_40

CDD. (2017). *IEC61360 Common Data Dictionary (CDD)*. Von https://en.wikipedia.org/wiki/IEC_61360 abgerufen

DIN SPEC 91345. (2016). *Reference Architecture Model Industrie 4.0 (RAMI 4.0)*. Abgerufen am 27. Juli 2022 von https://www.beuth.de/technische-regel/din-spec-91345/250940128

Heidel, R., Hoffmeister, M., Hankel, M., & Döbrich, U. (2017). *Industrie 4.0 - Basiswissen RAMI4.0.* VDE Verlag GMBH.

IEC 62832-1. (27. 10 2020). *Industrial-process measurement, control and automation - Digital factory framework - Part 1: General principles*. Von https://standards.iteh.ai/catalog/standards/iec/cfef54c6-c080-4e1d-980a-a7d53a784409/iec-62832-1-2020 abgerufen

Das vorliegende Konzept der Semantic Integration Patterns beruht auf folgenden Berichten des Projekts i-Twin bzw. wird dort ergänzt:

- D2.3 „Semantic Integration Patterns for Analytics“
- D2.4 „Final System Design“
- D3.2 „Asset Connectors“
- D3.3 „Application Connectors“

Impressum

Titel	Semantic Integration Patterns for Manufacturing (finale Fassung)
Bezeichnung	Deliverable 2.2b (i-Twin)
Autoren	Dietmar Glachs, Georg Güntner (Salzburg Research Forschungsgesellschaft m.b.H.)
Dateiname	D22b_SIPs4Manufacturing_final.docx
Publikationsstatus	Public
Letzte Änderung	22.01.2024 von Georg Güntner
Kontakt	Salzburg Research Forschungsgesellschaft m.b.H. Herr DI Georg Güntner Jakob Haringer Straße 5/3 | 5020 Salzburg | Austria T +43-662-2288-401 | georg.guentner@salzburgresearch.at
Copyright	Projektkonsortium i-Twin, Jänner 2024 p.a. Salzburg Research Forschungsgesellschaft m.b.H. Jakob Haringer Straße 5/3 | 5020 Salzburg | Austria T +43-662-2288-401 | i-twin-office@salzburgresearch.at

Das Projekt i-Twin wird gefördert vom BMK und von der FFG aus Mitteln des Programms IKT der Zukunft.