Final System Design

Deliverable 2.4

des Forschungsprojekts i-Twin

Dietmar Glachs, Georg Güntner, u.a.
Salzburg Research

mit Beiträgen von
COPA-DATA, H&H Systems, IcoSense

Jänner 2024

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

Inhaltsverzeichnis

Inhaltsverzeichnis 2
1 Zusammenfassung 4
2 i-Twin Plattform – Übersicht 5
2.1 Data Integration Layer 7
2.1.1 Asset Repository 9
2.1.2 Directory Service 20
2.1.3 Semantic Lookup (IEC 61360) 22
2.1.4 Distribution Network (Subscriptions) 27
2.1.5 Security & Identity Management 31
2.2 Application / Edge Layer 32
2.2.1 Asset-/Application-Connector 32
3 i-Twin Plattform – Technologie 39
3.1 Platform Setup 41
3.1.1 Plattform-Management 41
3.1.2 Beispiel-Instanz für funktionale Tests und Demonstration 42
3.1.3 Plattform-Konfiguration 42
3.2 Plattform Security 43
3.2.1 OAuth 2.0 und OpenID Connect (OIDC) 44
3.2.2 Keycloak 46
3.3 Streaming-Applikationen 47
3.3.1 Typen von Streaming-Applikationen 47
3.3.2 Ausdruckssprache der Streaming Applikationen 48
3.3.3 Implementierung der Streaming Applikation 50
3.3.4 API der Stream-Apps 51
3.4 Semantic Integration Patterns 53
3.4.1 Typ- vs. Instanz-Elemente 53
3.4.2 Anwendungs-Typen 54
3.4.3 Anwendungs-Instanzen 58
3.4.4 Exemplarische Darstellung am Beispiel von CMMS-Anwendungen 60
4 Anwendungsszenarien 66
4.1 Basis-Szenarien für Plattform-Nutzung 66
4.1.1 Asset-Typ-Information verwalten 66
4.1.2 Verwalten von Asset-Instanzen 68
4.1.3 Suche von Assets (Typen, Instanzen) 70
4.2 Semantic Integration Patterns 71
4.2.1 Applikation in i-Twin einrichten 71
4.3 Use Cases IcoSense 73
4.3.1 Upgrade Brown-Field-Asset zur I4.0 Komponente 73
4.3.2 OEE Werte übermitteln 75
4.3.3 Störmeldung absetzen 76
4.3.4 Auftragsdaten abrufen 79
4.4 Use Cases isproNG 80
4.4.1 Wartungs- und Störmeldungshistorie 80
4.4.2 Assets instanziieren 82
4.4.3 Sensor-Daten erhalten 84
4.4.4 Störungsursachentabelle 86
4.5 Use Cases SRFG 88

© Projektkonsortium i-Twin (2024)

4.5.1 Semantic Repository Integration 88
4.6 Templates 90
4.6.1 Use Case Template 90
4.6.2 Requirements Template 91
5 Ergänzende Projektberichte 93
6 Referenzen 94
Impressum 95

© Projektkonsortium i-Twin (2024)

1 Zusammenfassung

Publizierbare Version

Der vorliegende Bericht beschreibt das System Design für die im Projekt i-Twin entwickelten Plattform für die Interoperabilitätskonzepte und den Datenaustausch zwischen Herstellern, Betreibern und Instandhaltern in Fertigungsnetzwerken. Das vorliegende System-Design basiert auf der im Projekt i-Asset erarbeiteten System-Architektur und erweitert diese Architektur um den Begriff der Semantic Integration Patterns. Das Ziel des System Designs zur Umsetzung in i-Twin ist ein Software-Stack, der auf standardisierten Schnittstellen und Architekturen, Sicherheit, Quelloffenheit und nicht-invasiver Integration von IT-Systemen für das Asset-Management beruht. Das konzeptionelle System Design verwendet das Informationsmodell der Asset Administration Shell (AAS) als zentrales Konzept für die Modellierung von Assets und Anwendungen (auf der Basis einer Typ-Instanz-Relation) und schafft damit neuartige Interoperabilitätskonzepte für Digital Twins.

Der Bericht gibt einen Überblick über die Bausteine der i-Twin Plattform (Abschnitt 2) als zentralen Datendrehscheibe für Digital Twins sowie der erforderlichen Client Connectivity zur Anbindung von Industrie 4.0 (I4.0) Komponenten auf der Basis der AAS.

Abschnitt 3 erörtert die erforderlichen Schritte zum Aufbau und Betrieb der Plattform sowie auch erste Methoden für die semantische Definition Anbindung von höherwertigen Funktionen der angeschlossenen Anwendungen. Mit Hilfe von Semantic Integration Patterns werden funktionale Aspekte definiert, die von konkreten Anwendungen (ERP Systeme, CMMS, Analytics Tools etc.) abgedeckt werden.

In Abschnitt 4 sind jene Anwendungsszenaren aufgeführt, welche die Anforderungen für das in diesem Bericht erarbeitete System-Design definieren.

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.

© Projektkonsortium i-Twin (2024)

2 i-Twin Plattform – Übersicht

Das Projektkonsortium propagiert ein datenzentriertes, integriertes Asset Management durch die Integration bestehender Software-Systeme wie Dokumentenmanagementsysteme (DMS), Instandhaltungsmanagementsysteme (CMMS), Analytik-Systeme z.B. für Predictive Maintenance sowie weiterer (spezialisierte) Planungs- und Kontrollsysteme für die Produktion (ERP).

In mittelständischen Unternehmen finden sich hierfür oftmals verschiedene Softwaresysteme, meist unterschiedlicher Anbieter. Dabei wird eine Vielzahl von Daten erhoben, jedoch kann das Potential dieser Daten 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[1] Plattform** als Bindeglied zwischen allen spezialisierten Systemen. 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 eine generelle Architektur der i-Asset Plattform, welche die Asset-Management-Anwendungen, eine **i-Twin Data Integration Layer** sowie eingebunden Assets, Devices umfasst. Diese Komponenten manifestieren gemeinsam den **Digital Twin** von realen Anlagen für die Zwecke des digitalen Asset Managements.

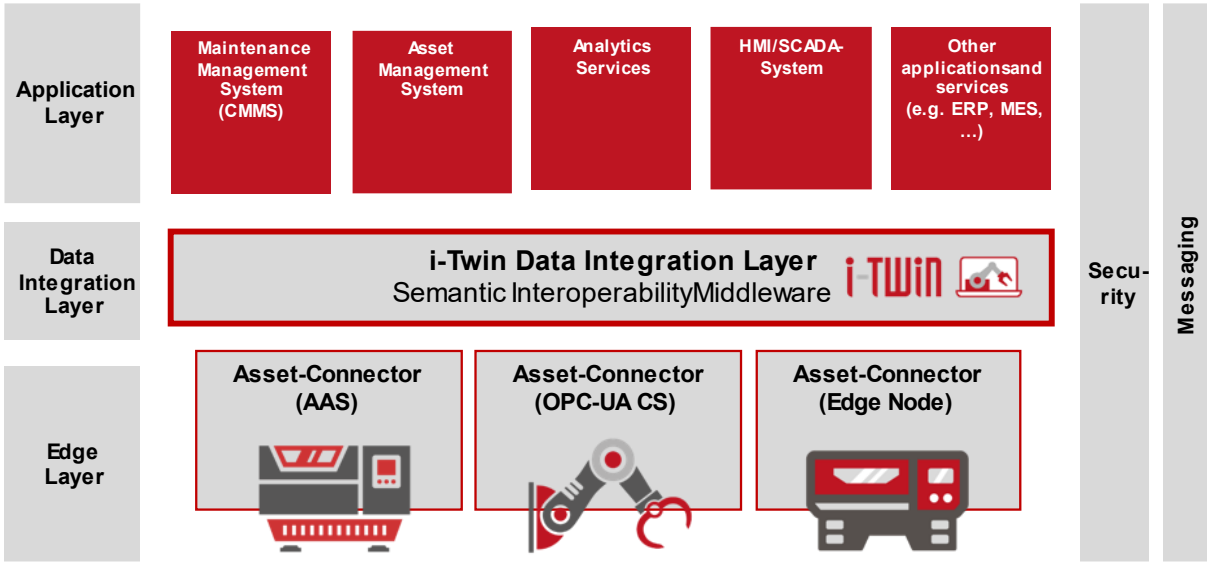


Abbildung 1: i-Asset Übersicht mit angeschlossenen Systemen

Das Hauptaugenmerk in diesem Bericht liegt auf dem i-Twin Data Integration Layer, welcher die Anwendungen (Office-Floor) mit den produzierenden Maschinen (Shop-Floor) verbindet. Eine erste Detaillierung findet sich in Abbildung 2.

[1] Die Namensgebung der Plattform stammt aus dem Projekt *i-Asset*, in dem die Plattform initiiert wurde!

© Projektkonsortium i-Twin (2024)

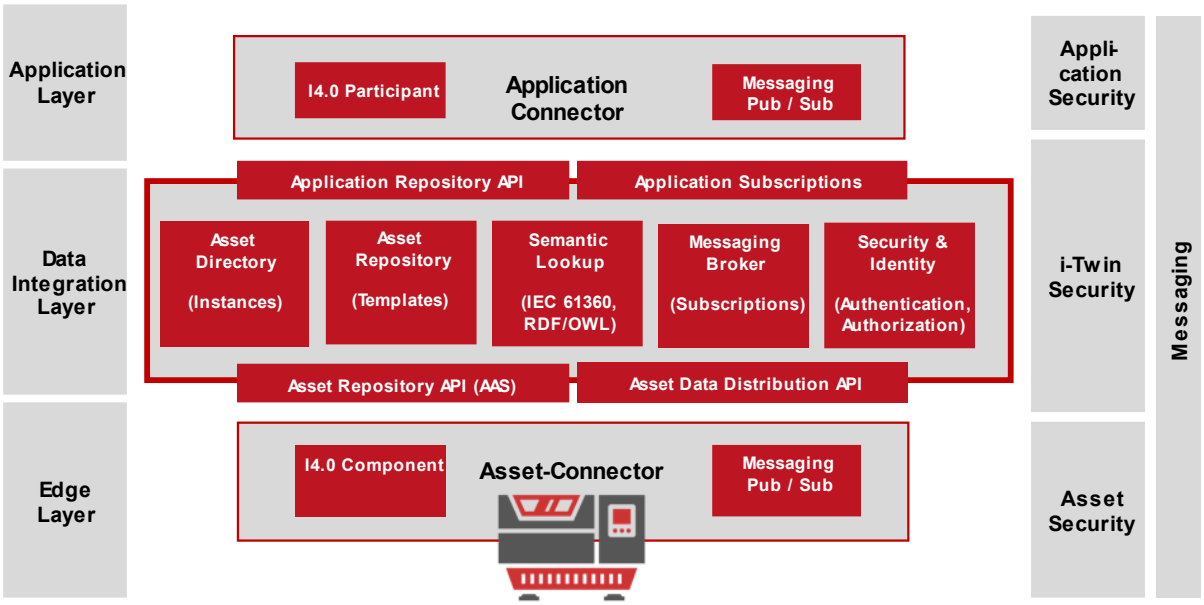


Abbildung 2: i-Twin Data Integration Layer

Für den Data Integration Layer wurden folgende Bausteine als wesentlich erkannt:

- 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.
- Asset Directory: Verwaltet die aktiven Instanzen von Asset Administration Shells. Sobald ein Asset bzw. Application Connector aktiviert wird, meldet sich dieser im Asset Directory an und gibt seine Verbindungsinformationen bekannt. Es entsteht somit ein Verzeichnis von aktiven Asset Administration Shells und ihren Verbindungsinformationen.
- 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.
- Messaging Broker: Mit Hilfe des Data Integration Layer sollen Assets in die Lage versetzt werden, ihre mittels Sensorik erhobenen Daten an beliebige Empfänger zu versenden. Mit Messaging Brokern 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: 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:

© Projektkonsortium i-Twin (2024)

- 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 der 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.

2.1 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 umfassende Datenmanagement und diese Vermittlerrolle ausfüllen zu können. Für dieses Datenmanagement dient das Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0) wie in Abbildung 3 dargestellt.

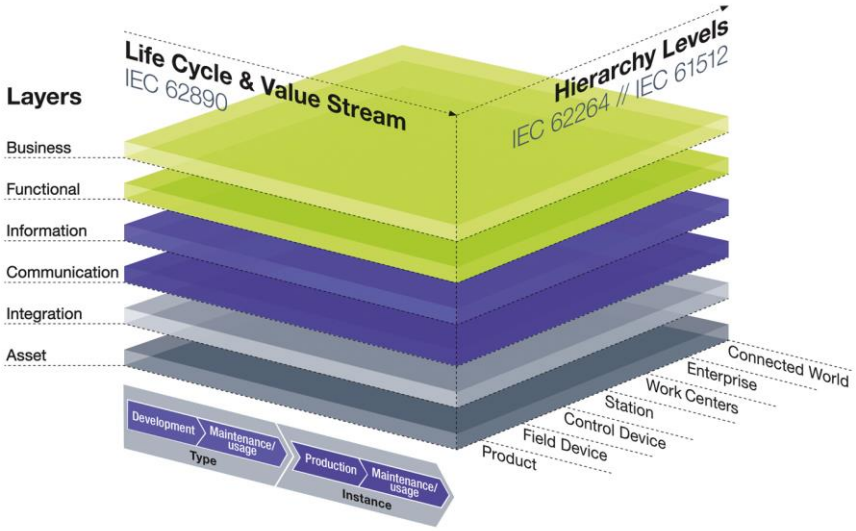


Abbildung 3: RAMI 4.0 Modell gem. Plattform Industrie 4.0

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

© Projektkonsortium i-Twin (2024)

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 2.1.4) 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 verständigt, welches in Abbildung 4 vereinfacht dargestellt ist.

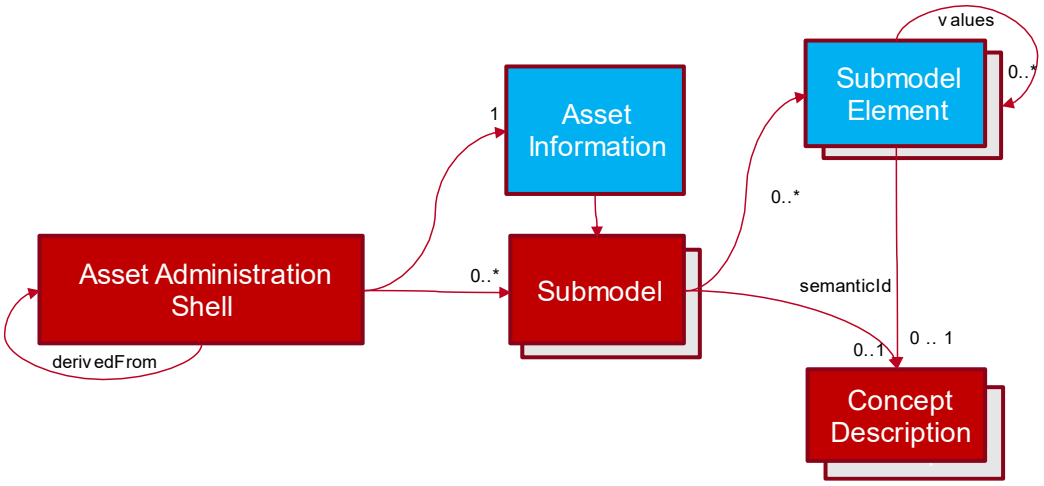


Abbildung 4: Asset Administration Shell - Informationsbausteine

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 noch auf ein oder mehrere Teilmodelle (`Submodel`) des Assets. 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 unterschiedlichen Zwecke abdecken zu können. Teilmodell-Elemente können durch Konzeptbeschreibungen (`ConceptDescription`) zusätzliche Informationen aus

© Projektkonsortium i-Twin (2024)

allgemein gültigen Taxonomien wie z.B. ECLASS[2] oder dem Common Data Dictionary[3] erhalten. 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.

2.1.1 Asset Repository

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 [idta2023-1] 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 Typen und konkreten Asset Instanzen unterschieden, auch Konzepte der Vererbung im Sinne von „ist abgeleitet von“ werden unterstützt. Asset Instanzen können auf diese Weise viele Informationen und vor allem die (Teilmodell)-Struktur vom jeweiligen Asset-Typen erhalten. Asset Typen 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 entsprechend der Spezifikation in [idta2023-2] zur Verfügung stellen kann und vor allem aktiv, via Data Integration Layer, mit der Außenwelt kommunizieren kann. Dieser Baustein ist in Abschnitt 2.2.1 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 5 dargestellte Bild.

[2] https://www.eclass.eu/index.html

[3] https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet

© Projektkonsortium i-Twin (2024)

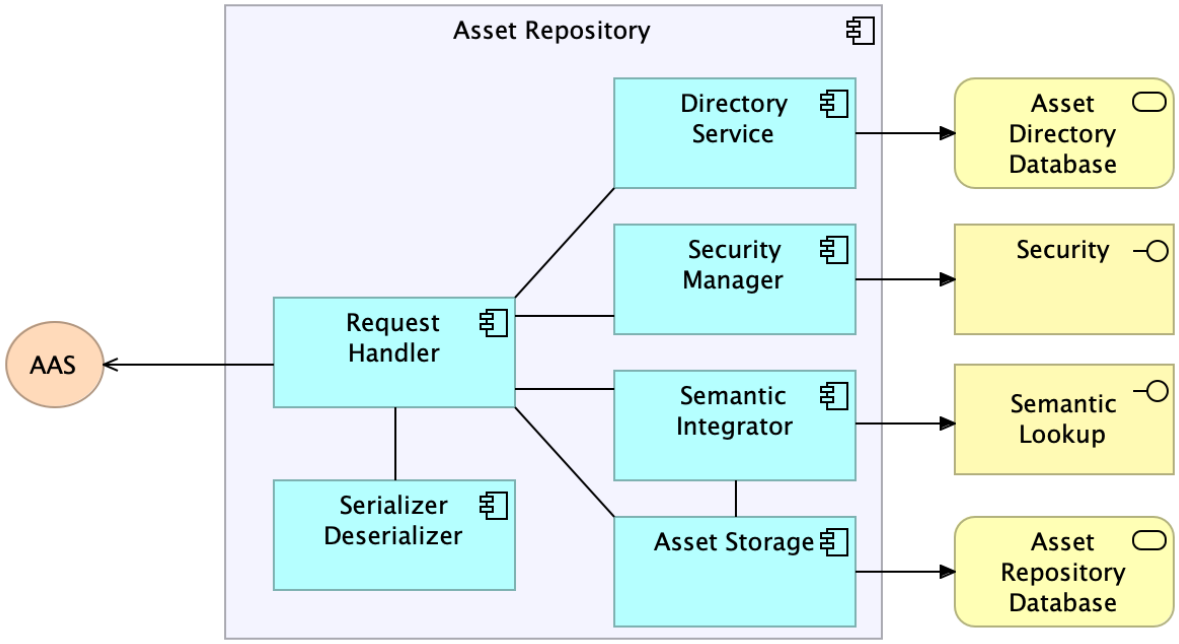


Abbildung 5: Asset Repository (Components)

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

- Request Handler: Dieser Baustein stellt die API gemäß [idta2023-2] zur Verfügung. Alle Anfragen werden zunächst per Security-Manager geprüft und bei POST/PUT Befehlen in das interne, speicherbare Datenmodell des Asset Storage transformiert ebenda gespeichert. Bei GET Befehlen hat dieser Baustein die Aufgabe, die gesuchten Elemente zu identifizieren und gemäß der Spezifikation in [idta2023-1] zu exportieren.
- 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.

© Projektkonsortium i-Twin (2024)

Datenmodell zur Speicherung von Asset und Asset Typen

Als Grundlage für die Speicherung von Asset-Informationen dient das Meta-Modell für die Verwaltungsschale der Plattform Industrie 4.0[4] ([idta2023-1]). Auf dieser Basis wurde im Projekt i-Asset bereits eine erste Version für das Asset Repository definiert (siehe [i-Asset-D2.9.2]). Für die weiteren Erklärungen in diesem Dokument sind die einzelnen Stereotype der einzelnen Elemente nochmals aufgeführt:

- *Referable*: Elemente dieses Stereotyps sind innerhalb einer `AssetAdministrationShell` anhand ihrer `idShort` referenzierbar. Eine `idShort` ist kein global verwendbarer Zugriffsschlüssel, sondern nur innerhalb des jeweils übergeordneten Parent-Elements eindeutig. Jedes `Referable` kennt auch sein übergeordnetes Element (Parent). Dadurch kann die Hierarchie innerhalb der Verwaltungsschale abgebildet werden. Die Verkettung der `idShort` wird auch als Zugriffspfad verwendet.
- *Identifiable*: Diese Elemente erben die Eigenschaften von `Referable` und besitzen zusätzlich einen global verwendbaren eindeutigen Bezeichner. Dieser Bezeichner kann eine URI, z.B. eine Web-Adresse, ein International Registration Data Identifier (IRDI) oder eine UUID sein. Da alle identifizierbaren Elemente auch referenzierbar sind, ist auch eine `idShort` als Bezeichner zusätzlich möglich. So sind z.B. Teilmodelle „identifizierbar“ können sowohl direkt über ihren eigenen Identifier als auch via Identifier der `AssetAdministrationShell` und `idShort` angesprochen werden. `Identifiable` Elemente besitzen neben einem Identifier auch noch administrative Informationen die ein Element weiter auszeichnen.
- *Qualifiable*: Elemente dieses Stereotyps können zusätzliche qualifizierende Elemente, so genannte `Constraint`-Objekte aufweisen. Constraints definieren Bedingungen die für die Verwendung einzelner Elemente maßgeblich sind.
- *HasKind*: Bestimmt, ob ein Element sowohl als Typ als auch als Instanz vorliegen kann. Entsprechend RAMI 4.0 können Elemente zu einem Asset-Typ oder einer real existierenden Instanz eines Assets darstellen (d.h. Kind kann entweder *Type* oder *Instance* sein). Teilmodelle mit HasKind=Type können als Template für Instanzen verwendet werden.
- *HasSemantics*: Elemente können zusätzlich semantisch ausgezeichnet werden. Elemente mit dieser Eigenschaft können mit externen, allgemein gültigen Klassifikationssystemen (wie z.B. ECLASS) annotiert werden. Ist eine Semantik vorhanden, wird diese über eine `semanticId`[5] verlinkt. Im Data Integration Layer wird diese Möglichkeit der weiteren semantischen Auszeichung vom Semantic Lookup (2.1.3) bereitgestellt.
- *HasDataSpecification*: Dieser Stereotyp erlaubt die Benennung von Daten-Templates. Ein Template listet jene zusätzlichen Attribute auf, welche einem Element mittels `semanticId` hinzugefügt wurden.

---

[4] Spezifikation der Verwaltungsschale bzw. Asset Administration Shell:
https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.pdf
Begleitdokument für den Einsatz in der Praxis:
https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2019-verwaltungsschale-in-der-praxis.pdf

[5] Z.B. ist der *Herstellername* in eClass mit der semanticId *0173-1#02-AAO677#002* verlinkt, welcher dann über die eClass Services aufgelöst werden kann: https://www.eclasscontent.com/index.php?action=cc2prdet&language=de&version=10.1&id=&pridatt=0173-1%2302-AAO677%23002

© Projektkonsortium i-Twin (2024)

Um dieses Verhalten im Daten-Modell im Asset Repository möglichst genau abbilden zu können wurden entsprechende Interfaces definiert, welche die erforderlichen Methoden definieren und gleichzeitig die erforderlichen Mapping Informationen zur Speicherung in einer Datenbank aufweisen:

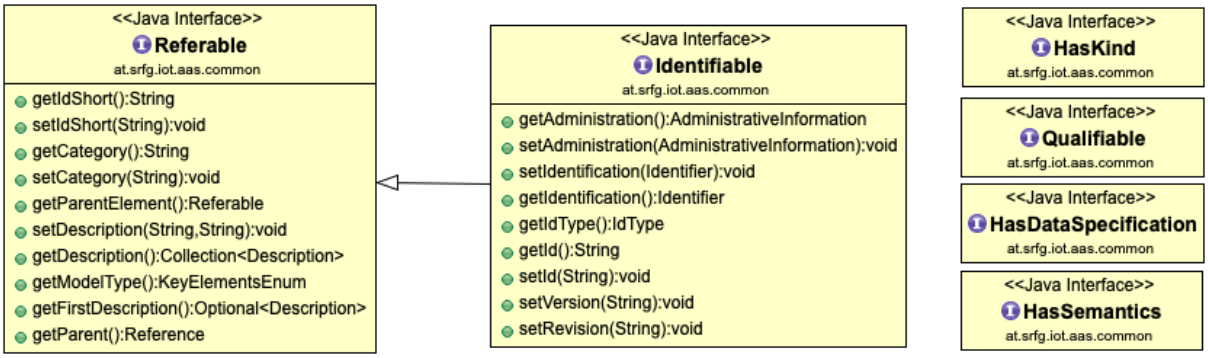


Abbildung 6: „Abstrakte“ Klassen für Elemente der AssetAdministrationShell

Abbildung 6 zeigt die abstrakten Klassen der Asset Administration Shell. Die konkreten Ausprägungen der AAS-Elemente erben die Eigenschaften der abstrakten Klassen. Im Modell wird Mehrfachvererbung unterstützt.

- Jedes `Referable`-Element definiert eine idShort. Diese ist innerhalb seine jeweiligen Kontexts (übergeordnetes Element) eindeutig. Weiter kann für jedes Element ein Name sowie eine Beschreibung in mehreren Sprachen hinterlegt werden.
- `Identifiable`-Elemente enthalten administrative Informationen (Version, Revision) und einen global gültigen `Identifier`. Diese Elemente sind via API direkt ansprechbar. `Identifiable`-Elemente sind Unterklassen von `Referable`, besitzen demnach auch dessen Attribute.
- `Qualifiable` Elemente können eine Liste von `Qualifier`-Objekten enthalten. Ein `Qualifier` kannn als Key-Value Paar gesehen werden, mit dem AAS-Element ausgezeichnet werden können.
- Die `HasSemantics` Eigenschaft erlaubt die Annotierung von AAS Elementen mittels semantischen Referenzen.
- `HasDataSpecification`-Elemente können weitere Definitionen aus externen Datenbanken, z.B. aus IEC61360 kompatiblen Taxonomien einbetten.
- `HasKind` erlaubt die Auszeichnung eines Elements als Template oder als Instanz.

Entsprechend dem Meta-Modell der Plattform Industrie 4.0 sind die Entitäten `AssetAdministrationShell` bzw. `Submodel` wichtige Elemente für den Datenaustausch. Wie in Abbildung 7 dargestellt, sind sowohl `Submodel` wie auch `AssetAdministrationShell` vom Type `Identifiable`, d. h. sie besitzen administrative Informationen und einen global verwendbaren `Identifier`. Gemäß der API-Spezifikation [idta2023-2] können diese Elemente direkt adressiert werden. Dies gilt auch für das `ConceptDescription` Element, welches abstrakte Konzepte aus externen Datenbanken, in Kombination mit der Eigenschaft `HasSemantics` realisiert.

© Projektkonsortium i-Twin (2024)

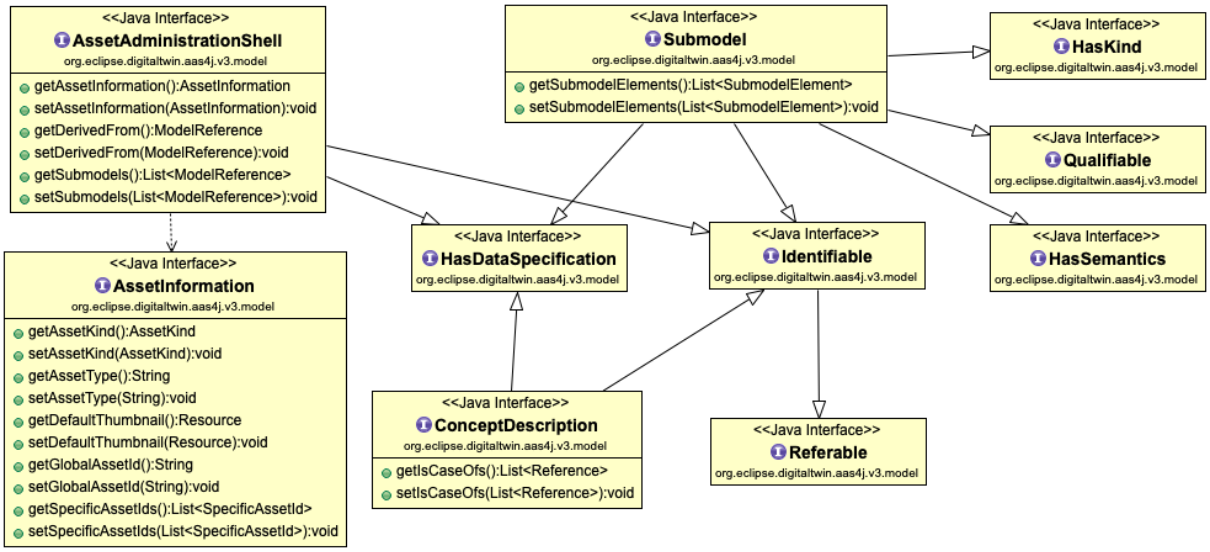


Abbildung 7: AssetAdministrationShell und Asset

Eine `AssetAdministrationShell` verwaltet seine `AssetInformation` in der auch der global genutzte Asset Identifier, z.B. aus einem ERP System, sowie weitere `SpecificAssetId` abgelegt sind. Die AssetInformation gibt auch Auskunft, ob es sich um ein Template oder eine Instanz handelt und, im Falle einer Instanz, auf welchem Typ/Template diese `AssetAdministrationShell` basiert. Jede AAS verwaltet seine zugewiesenen Teilmodelle in einer Liste von Model-Referenzen die jeweils auf ein `Submodel` zeigen.

Ein `Submodel` besitzt die Eigenschaft `HasKind`. Dies bedeutet, dass ein Teilmodell entweder als Template zu sehen ist oder eben als eine konkrete Instanz. Alle `Identifiable`-Element besitzen noch die Eigenschaft `HasDataSpecification`. Dadurch können mittels Daten-Templates die erforderlichen Attribute, die für das jeweilige Element erforderlich sind, definiert werden. Die Eigenschaften von Assets werden in der AAS in Teilmodellen (`Submodel`) organisiert wobei jedes Teilmodell wie auch die `AssetAdministrationShell` *identifizierbar*, also vom Typ `Identifiable` ist und demnach auch die Basis-Funktionalität aus `Identifiable` bzw. `Referable` Elementen erbt.

Ein Teilmodell besitzt damit einen global gültigen Identifier, kann also direkt mittels API aufgefunden werden. Gemäß Abbildung 7 sind dem `Submodel` zudem die Stereotype `HasSemantics`, `HasKind`, `Qualifiable` und `HasDataSpecification` zugeordnet.

Das `Submodel` ist jedoch lediglich der Einstiegspunkt zu den wesentlichen Daten, die ein Teilmodell transportieren soll. Dieses besteht daher aus (hierarchisch) angeordneten Submodel-Elementen wie dies in Abbildung 8 aufgeführt ist. Der wesentliche Unterschied zwischen `Submodel` und `SubmodelElement` ist, dass das `SubmodelElement` nicht *identifzierbar*, sondern lediglich *referenzierbar* ist. Ein `SubmodelElement` ist somit nur im Kontext seines übergeordneten AAS-Elements adressierbar.

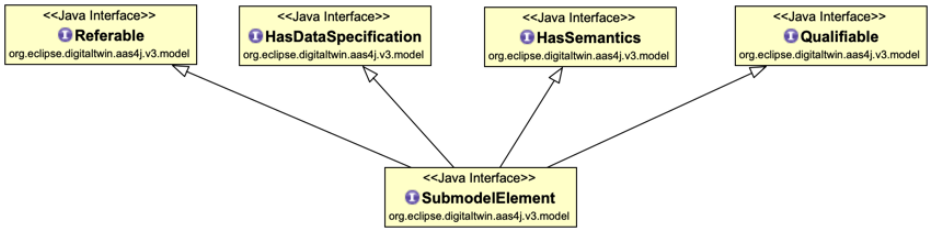


Abbildung 8: Submodel und SubmodelElement

© Projektkonsortium i-Twin (2024)

Das `SubmodelElement` ist das abstrakte Basiselement bzw. Träger aller Informationen innerhalb einer `AssetAdministrationShell`. Für die Unterscheidung unterschiedlicher Informationen (Eigenschaften, Operationen, Ereignisse etc.), sieht das Meta-Modell der Plattform Industrie 4.0 unterschiedliche, konkrete Ausprägungen für `SubmodelElement` vor.

Abbildung 9 zeigt eine Übersicht über die einzelnen Unterklassen von `SubmodelElement` zur Speicherung der unterschiedlichen Asset-Informationen.

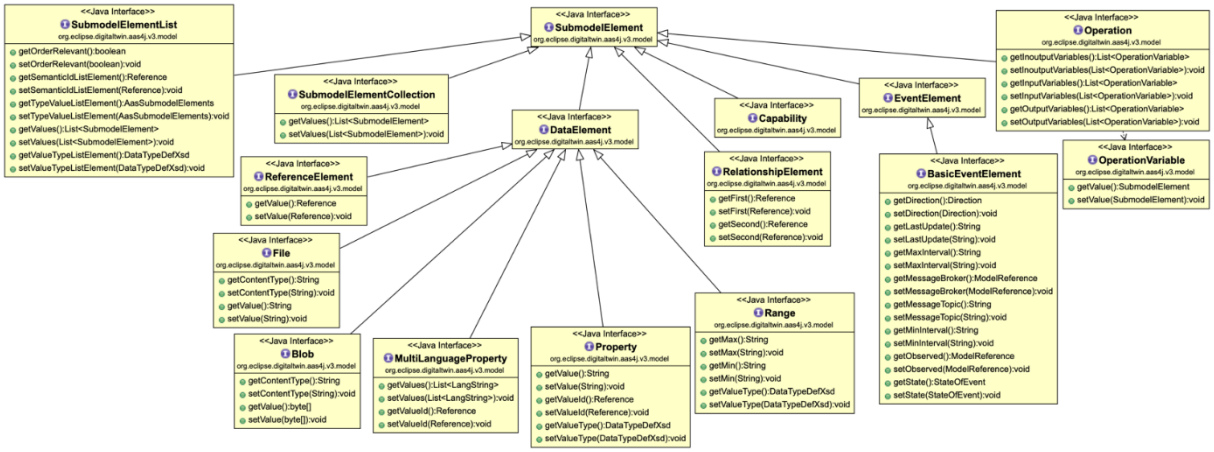


Abbildung 9: SubmodelElement und konkrete Ausprägungen

Die unterschiedlichen Ausprägungen von `SubmodelElement` sind in der folgenden Tabelle detailliert.

SubmodelElement	**Zweck**
RelationshipElement	Definiert eine semantische Beziehung zwischen referenzierbaren Elementen.
Datenelement	Unter `DataElement` werden jene Elemente subsumiert, die jeweils einen bestimmten Wert transportieren: • `Property`: Definiert einen einzelnen Wert. Für jedes Property muss auch der Datentyp definiert werden. • `MultiLanguageProperty`: Definiert einen mehrsprachigen Text. • `Range`: Definiert einen Wertbereich mit • `File`: Verweist auf eine außerhalb der AAS abgelegte Datei per File-Namen und Typ des Datei-Inhalts. Die Datei sollte anhand des Dateinamens im System auffindbar sein. • `Blob`: Speichert den Inhalt einer Datei sowie den Typ des Datei-Inhalts innerhalb einer AAS. • `ReferenceElement`: Speichert einen Verweis auf ein Model-Element oder auf ein externes, abstraktes Konzept.
Capability	Bezeichnet Fertigkeiten, die einem Asset zugrunde liegen.
Operation(-variable)	Definiert, welche Operationen vom Asset unterstützt werden. Zu jeder `Operation` können auch `OperationVariable` definiert

© Projektkonsortium i-Twin (2024)

	werden, wobei jede `OperationVariable` wieder ein `SubmodelElement` definiert.
SubmodelElement-Collection	Damit können Submodel-Elemente zusammengefasst, gruppiert werden. Es ist so auch möglich, beliebige Hierarchie-Stufen von `SubmodelElement` zu definieren. Ein `SubmodelElementCollection` stellt eine Struktur dar, wo jedes Unterelement anhand seiner `idShort` eindeutig adressierbar ist.
SubmodelElement-List	Eine `SubmodelElementList` besteht aus einer nach unten offenen Liste von gleichartigen Unterelementen. Jedes Unterelement ist wiederum ein `SubmodelElement`, kann also auch eine Struktur oder eine Liste sein.
EventEvent bzw. BasicEventElement	Enthält die Informationen zur Integration mit Messaging Systemen.
RelationshipElement	Erlaubt eine benannte Beziehung zwischen zwei `Referable`-Elementen.

Tabelle 1: Konkrete SubmodelElement-Klassen

Jede dieser Klassen erbt mit `HasSemantics`, `HasDataSpecification` und `Qualifiable` die Basis-Eigenschaften von `SubmodelElement`. Die Eigenschaft `HasSemantics` erlaubt die Annotierung einer Eigenschaft mit externen Klassifikationssystemen wie z.B. eCl@ss bzw. dem Common Data Dictionary (CDD). Mittels `HasDataSpecification` wiederum kann für jedes `SubmodelElement` mittels Templates festgelegt werden, welche Attribute zwingend vorzusehen sind. Diese zusätzliche Spezifikation wird für Typ-Elemente festgelegt da diese dann auf alle Instanzen dieses Typs angewendet werden können. Mit `Qualifiable` wird schließlich festgelegt unter welchen Voraussetzungen ein Element relevant ist.

Interaktion mit dem Asset Repository API

Die Spezifikation der Plattform Industrie 4.0 umfasst neben dem Meta-Modell für die Asset Administration Shell auch die Interaktion zur Laufzeit. In [idta2023-2] sind eine Reihe von Zugriffsmethoden definiert wie sie vom Asset Repository und auch von aktiven I4.0 Komponenten implementiert werden müssen um an einem I4.0 kompatiblen System teilnehmen zu können. Diese einzelnen Interfaces sind nachfolgend weiter detailliert.

Asset Administration Repository Interface

Ein zentrales Asset Repository verwaltet ein oder mehrere Asset Administration Shells

Definiert Methoden zur Abfrage und Manipulation von Asset Administration Shell Informationen. Die `AssetAdministrationShell` dient hierbei als Einstiegspunkt zu weiteren Informationen wie eben Asset Information oder die mit der AAS verbundenen `Submodel`-Elemente.

Da Teilmodelle sehr umfangreich sein können, werden diese in einer `AssetAdministrationShell` als Referenz verwaltet. Eine Referenz kann als Verkettung von Bezeichnern gesehen werden, mit dem ein Element innerhalb der AAS aufgefunden werden kann.

© Projektkonsortium i-Twin (2024)

Method	Path
GET	/shells Returns all Asset Administration Shells
POST	/shells Creates a new Asset Administration Shell
GET	/shells/$reference Returns References to all Asset Administration Shells
GET	/shells/{aasIdentifier} Returns a specific Asset Administration Shell
PUT	/shells/{aasIdentifier} Updates an existing Asset Administration Shell
DELETE	/shells/{aasIdentifier} Deletes an Asset Administration Shell
GET	/shells/{aasIdentifier}/$reference Returns a specific Asset Administration Shell as a Reference
GET	/shells/{aasIdentifier}/asset-information Returns the Asset Information
PUT	/shells/{aasIdentifier}/asset-information Updates the Asset Information
GET	/shells/{aasIdentifier}/asset-information/thumbnail Returns the thumbnail image for the asset
PUT	/shells/{aasIdentifier}/asset-information/thumbnail Updates the thumbnail image for the asset
DELETE	/shells/{aasIdentifier}/asset-information/thumbnail Deletes the thumbnail image for the asset
GET	/shells/{aasIdentifier}/submodel-refs Returns all submodel references
POST	/shells/{aasIdentifier}/submodel-refs Creates a submodel reference at the Asset Administration Shell
DELETE	/shells/{aasIdentifier}/submodel-refs/{submodelIdentifier} Deletes the submodel reference from the Asset Administration Shell. Does not delete the submodel itself!
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier} Returns the Submodel
PUT	/shells/{aasIdentifier}/submodels/{submodelIdentifier} Updates the Submodel
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier} Updates the Submodel
DELETE	/shells/{aasIdentifier}/submodels/{submodelIdentifier} Deletes the submodel from the Asset Administration Shell and the Repository.
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$metadata Returns the Submodel's metadata elements
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$metadata Updates the metadata attributes of the Submodel
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$value Returns the Submodel's ValueOnly representation
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$value Updates the values of the Submodel
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$reference Returns the Submodel as a Reference

© Projektkonsortium i-Twin (2024)

Method	Path
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/$path Returns the Submodel's metadata elements paths
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements Returns all submodel elements including their hierarchy
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements Creates a new submodel element
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/$metadata Returns all submodel elements including their hierarchy
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/$value Returns all submodel elements including their hierarchy in the ValueOnly representation
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/$reference Returns all submodel elements as a list of References
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/$path Returns all submodel elements including their hierarchy
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Returns a specific submodel element from the Submodel at a specified path
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Creates a new submodel element at a specified path within submodel elements hierarchy
PUT	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Updates an existing submodel element at a specified path within submodel elements hierarchy
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Updates an existing submodel element value at a specified path within submodel elements hierarchy
DELETE	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Deletes a submodel element at a specified path within the submodel elements hierarchy
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$metadata Returns the metadata attributes if a specific submodel element from the Submodel at a specified path
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$metadata Updates the metadata attributes of an existing submodel element value at a specified path within submodel elements hierarchy
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value Returns a specific submodel element from the Submodel at a specified path in the ValueOnly representation
PATCH	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value Updates the value of an existing submodel element value at a specified path within submodel elements hierarchy
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$reference Returns the Reference of a specific submodel element from the Submodel at a specified path
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$path Returns a specific submodel element from the Submodel at a specified path in the Path notation
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Downloads file content from a specific submodel element from the Submodel at a specified path
PUT	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Uploads file content to an existing submodel element at a specified path within submodel elements hierarchy
DELETE	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Deletes file content of an existing submodel element at a specified path within submodel elements hierarchy

© Projektkonsortium i-Twin (2024)

Method	Path
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke Synchronously invokes an Operation at a specified path
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke/$value Synchronously invokes an Operation at a specified path
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke-async Asynchronously invokes an Operation at a specified path
POST	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/invoke-async/$value Asynchronously invokes an Operation at a specified path
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/operation-status/{handleId} Returns the Operation status of an asynchronous invoked Operation
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/operation-results/{handleId} Returns the Operation result of an asynchronous invoked Operation
GET	/shells/{aasIdentifier}/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/operation-results/{handleId}/$value Returns the ValueOnly notation of the Operation result of an asynchronous invoked Operation

Das Asset Administration Shell Repository Interface erlaubt den Zugriff auf alle Teilmodelle bzw. Teilmodell-Elemente nur anhand des AAS Identifiers. Es wird zunächst geprüft, ob diese Asset Administration Shell im Repository gespeichert ist. Wird die AAS gefunden, so wird geprüft, ob diese auch eine Referenz auf das gewünschte Teilmodell enthält. Nur dann wird der Zugriff auf die Teilmodell-Elemente des gesuchten Teilmodells gewährt.

Submodel Repository Interface

Ein Teilmodell ist identifizierbar, d.h. es besitzt einen global eindeutigen Bezeichner. In einem Repository müssen Teilmodelle auch dann bearbeitet werden, wenn sie keiner Asset Administration Shell zugewiesen sind. Z.B. wenn Teilmodell-Templates bearbeitet werden.

Method	Path
GET	/submodels/{submodelIdentifier} Returns the Submodel
PUT	/submodels/{submodelIdentifier} Updates the Submodel
PATCH	/submodels/{submodelIdentifier} Updates the Submodel
DELETE	/submodels/{submodelIdentifier} Deletes the submodel from the Asset Administration Shell and the Repository.
GET	/submodels/{submodelIdentifier}/$metadata Returns the Submodel's metadata elements
PATCH	/submodels/{submodelIdentifier}/$metadata Updates the metadata attributes of the Submodel

© Projektkonsortium i-Twin (2024)

Method	Path
GET	/submodels/{submodelIdentifier}/$value Returns the Submodel's ValueOnly representation
PATCH	/submodels/{submodelIdentifier}/$value Updates the values of the Submodel
GET	/submodels/{submodelIdentifier}/$reference Returns the Submodel as a Reference
GET	/submodels/{submodelIdentifier}/$path Returns the Submodel's metadata elements paths
GET	/submodels/{submodelIdentifier}/submodel-elements Returns all submodel elements including their hierarchy
POST	/submodels/{submodelIdentifier}/submodel-elements Creates a new submodel element
GET	/submodels/{submodelIdentifier}/submodel-elements/$metadata Returns all submodel elements including their hierarchy
GET	/submodels/{submodelIdentifier}/submodel-elements/$value Returns all submodel elements including their hierarchy in the ValueOnly representation
GET	/submodels/{submodelIdentifier}/submodel-elements/$reference Returns all submodel elements as a list of References
GET	/submodels/{submodelIdentifier}/submodel-elements/$path Returns all submodel elements including their hierarchy
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Returns a specific submodel element from the Submodel at a specified path
POST	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Creates a new submodel element at a specified path within submodel elements hierarchy
PUT	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Updates an existing submodel element at a specified path within submodel elements hierarchy
PATCH	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Updates an existing submodel element value at a specified path within submodel elements hierarchy
DELETE	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath} Deletes a submodel element at a specified path within the submodel elements hierarchy
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$metadata Returns the metadata attributes if a specific submodel element from the Submodel at a specified path
PATCH	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$metadata Updates the metadata attributes of an existing submodel element value at a specified path within submodel elements hierarchy
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value Returns a specific submodel element from the Submodel at a specified path in the ValueOnly representation
PATCH	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$value Updates the value of an existing submodel element value at a specified path within submodel elements hierarchy
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$reference Returns the Reference of a specific submodel element from the Submodel at a specified path
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/$path Returns a specific submodel element from the Submodel at a specified path in the Path notation
GET	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Downloads file content from a specific submodel element from the Submodel at a specified path
PUT	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Uploads file content to an existing submodel element at a specified path within submodel elements hierarchy

© Projektkonsortium i-Twin (2024)

Method	Path
DELETE	/submodels/{submodelIdentifier}/submodel-elements/{idShortPath}/attachment Deletes file content of an existing submodel element at a specified path within submodel elements hierarchy

Concept Description Repository Interface

Eine AAS enthält neben den Teilmodell-Elementen auch weitere semantische Konzept-Beschreibungen. Diese Konzept-Beschreibungen sind ebenfalls identifizierbar und werden im Asset Repository als eigene Entität abgelegt. Sinngemäß bedarf es einer Schnittstelle um diese Konzepte bearbeiten zu können

Method	Path
GET	/concept-descriptions Returns all Concept Descriptions
POST	/concept-description Creates a new Concept Description
GET	/concept-description/{cdIdentifier} Returns a specific Concept Description
PUT	/concept-description/{cdIdentifier} Updates a specific Concept Description
DELETE	/concept-description/{cdIdentifier} Deletes a specific Concept Description

Im Asset Repository werden somit alle identifizierbaren Objekte verwaltet. Alle gespeicherten Identifiable Daten in einem Repository, innerhalb eines Connectors, e.g., Asset Administration Shell, Submodel, Concept Description, werden als *Asset Environment* bezeichnet.

2.1.2 Directory Service

Jede aktive I4.0 Komponente verwaltet potentiell mehrere Asset Administration Shells die über einen definierten Endpunkt erreichbar sind.

Das Directory Service stellt einen Verzeichnisdienst für aktive I4.0 Komponenten zur Verfügung, welcher ein Mapping von Asset Identifiern und den zugehörigen Endpunkten bereitstellt. Das Datenmodell für das Directory-Service ist in Abbildung 10 dargestellt.

© Projektkonsortium i-Twin (2024)

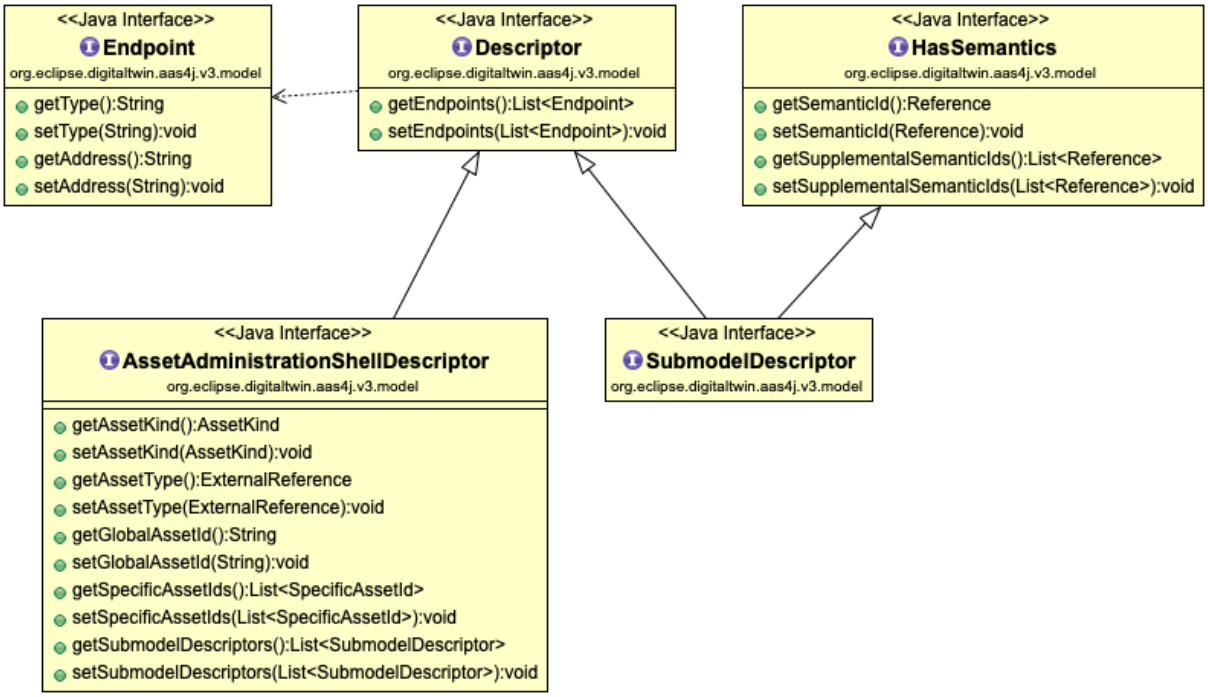


Abbildung 10: Asset Administration Shell & Submodel Descriptor

Der Asset Administraton Shell Descriptor verwaltet die Service-Endpoints, anhand derer die Asset Administration Shell erreichbar ist. Um einen Descriptor suchen zu können, “erbt” dieser die Informationen gemäß `AssetInformation` und kann somit anhand der `GlobalAssetID`, oder einer `SpecificAssetID` gefunden werden.

Analog zur AssetAdministrationShell, diese verwaltet die Liste der Teilmodelle, enthält der Shell Descriptor eine Liste von Submodel Descriptoren. Diese definieren ihrerseits die Service Endpoints, mit denen sie angesprochen werden können. Als Besonderheit erbt der Submodel-Descriptor die HasSemantics-Eigenschaft. Dadurch können die SemanticID’s des Teilmodells, aber auch zusätzliche (supplemental) SemanticID’s im Descriptor „durchsuchbar“ abgelegt werden.

Directory Service API

Die Directory Service API definiert eine Reihe von Methoden, sodass sich aktive I4.0 Komponenten registrieren/anmelden können. Sobald eine I4.0 Komponente abgeschalten wird, muss sie sich auch wieder deregistrieren/abmelden.

Method	Path
GET	/shell-descriptors Returns all Asset Administration Shell Descriptors
POST	/shell-descriptors Creates a new Asset Administration Shell Descriptor, e.g. registers an AAS
GET	/shell-descriptors/{aasIdentifier} Returns a specific Asset Administration Shell Descriptor
PUT	/shell-descriptors/{aasIdentifier} Updates an existing Asset Administration Shell Descriptor
DELETE	/shell-descriptors/{aasIdentifier} Deletes a specific Asset Administration Shell Descriptor, e.g. deregistes an AAS

© Projektkonsortium i-Twin (2024)

Method	Path
GET	/shell-descriptors/{aasIdentifier}/submodel-descriptors Returns all Submodel Descriptors from a specific Asset Administration Shell Descriptor
POST	/shell-descriptors/{aasIdentifier}/submodel-descriptors Creates a new Submodel Descriptor with a specific Asset Administration Shell Descriptor, e.g., registers a new Submodel
GET	/shell-descriptors/{aasIdentifier}/submodel-descriptors/{submodelIdentifier} Returns a specific Submodel Descriptors from a specific Asset Administration Shell Descriptor
PUT	/shell-descriptors/{aasIdentifier}/submodel-descriptors/{submodelIdentifier} Updates a specific Submodel Descriptors from a specific Asset Administration Shell Descriptor
DELETE	/shell-descriptors/{aasIdentifier}/submodel-descriptors/{submodelIdentifier} Deletes a specific Submodel Descriptors from a specific Asset Administration Shell Descriptor, e.g., deregisters a Submodel

2.1.3 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` und `ConceptDictionary`, die dann in einer AAS enthalten sind, wenn diese an anderer Stelle der als `semanticId` referenziert 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.

Gemäß AAS Meta-Model 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 Data Semantics 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 Data Semantics Service orientiert sich am Schema für ECLASS und ist in Abbildung 11 dargestellt.

© Projektkonsortium i-Twin (2024)

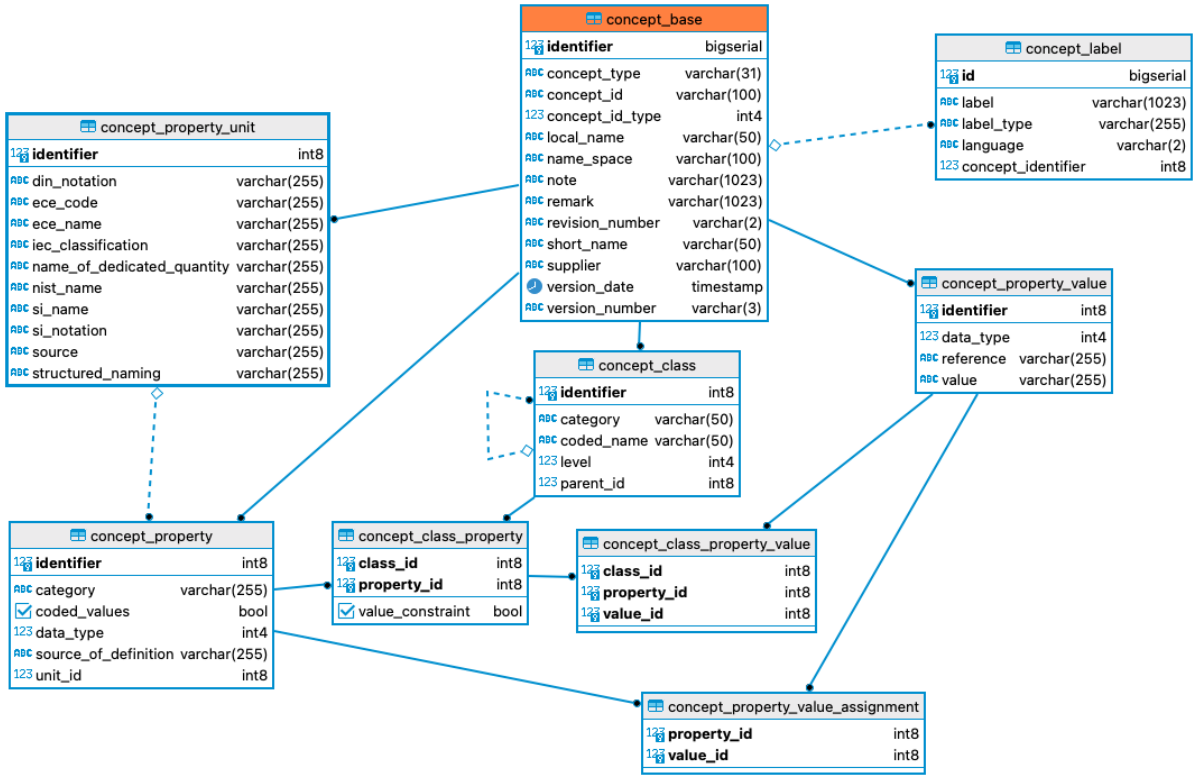


Abbildung 11: Data Specification Service Datenmodell

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.

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.

© Projektkonsortium i-Twin (2024)

Das Data Semantics Service nutzt einen *International Registration Data Identifiier* (IRDI) als eindeutigen Bezeichner. Die Zugriffsmethoden zur Abfrage von Daten aus dem Service sind in der folgenden Tabelle dargestellt.

Methode	Pfad	Beschreibung
GET	/class?irdiCC=<irdiCC>	Abfrage einer einzelnen Klassifikationsklasse anhand des eindeutigen IRDI. Der Parameter `irdiCC` ist zwingend erforderlich.
GET	/property?irdiPR=<irdiPR>	Abfrage einer einzelnen Property anhand des eindeutigen IRDI. Der Parameter `irdiPR` ist zwingend erforderlich.
GET	/properties?irdiCC=<irdiCC>	Abfrage aller Eigenschaften/Properties die mittels der Beziehungsrelation `classification_class_property` zu einer Klasse zugewiesen sind. Der Parameter `irdiCC` ist zwingend erforderlich.
GET	/value?irdiVA=<irdiVA>	Abfrage eines einzelnen Werts anhand des eindeutigen IRDI. Der Parameter `irdiVA` ist zwingend erforderlich.
GET	/values?irdiCC=<irdiCC> &irdiPR=<irdiPR>	Abfrage aller Werte die für die Kombination Klassifikationsklasse und Eigenschaft definiert sind (Beziehungsrelation `classification_class_property_value`). Beide Parameter (`irdiCC`, `irdiPR`) sind zwingend erforderlich.
GET	/unit?irdiUN=<irdiUN>	Abfrage einer Unit anhand des eindeutigen IRDI. Der Parameter `iridUN` ist zwingend erforderlich.

Das Data Semantics Service stellt die Daten als JSON Dokument zur Verfügung. Exemplarische Daten für einzelne Abfragen sind nachfolgend angeführt:

Abfrage einer einzelnen Klasse

```
GET <host:port>/class?irdiCC=0173-1%2301-AFY430%23003
{
      "irdiCC": "0173-1#01-AFY430#003",
      "codedName": "13010301",
      "definition": "Total or partial service for product developments",
      "idCC": "AFY430003",
      "identifier": "AFY430",
      "isoCountryCode": "US",
      "isoLanguageCode": "en",
      "level": 4,
      "keywordPresent": false,
      "subclassPresent": false,
      "note": null,
```


© Projektkonsortium i-Twin (2024)

```
      "preferredName": "Function definition (specification total…
                        product level)",
      "remark": null,
      "revisionNumber": "01",
      "supplier": "0173-1",
      "versionDate": "2017-01-30",
      "versionNumber": "003"
}
```


Bei der Abfrage von Klassen werden eventuell vorhandenen Eigenschaften nicht inline aufgelistet. Diese können mit einem eigenen Request abgefragt werden.

Abfrage aller Properties für eine Klasse


```
GET <host:port>/properties?irdiCC=0173-1%2301-AFY430%23003
[
      {
            "irdiPR": "0173-1#02-AAE203#003",
            "attributeType": "DIRECT",
            "category": "T03",
            "currencyAlphaCode": null,
            "dataType": "REAL_MEASURE",
            "definition": "value of the bending radius of the fixed …
                           internal connecting plate",
            "definitionClass": "0173-1#01-RAA001#001",
            "identifier": "AAE203",
            "idPR": "AAE203003",
            "isoCountryCode": "US",
            "isoLanguageCode": "en",
            "note": null,
            "preferredName": "bending radius of the fixed internal …
                              connecting plate",
            "preferredSymbol": null,
            "remark": null,
            "revisionNumber": "01",
            "shortName": "bendingRadiusOfTheFixedInternalConnectingPlate”,
            "sourceOfDefinition": null,
            "supplier": "0173-1",
            "versionDate": "2012-11-27",
            "versionNumber": "003",
            "unit": {
                  "irdiUN": "0173-1#05-AAA480#002",
                  "comment": null,
                  "definition": null,
                  "dinNotation": "mm",
                  "eceCode": "MMT",
                  "eceName": "millimetre",
                  "iecClassification": null,
                  "nameOfDedicatedQuantity": "distance",
                  "nistName": "millimeter",
                  "shortName": "mm",
                  "siName": "millimetre",
                  "siNotation": "mm",
                  "source": null,
                  "structuredNaming": "millimetre"
            },
            "values": []
      } ,     ...
]
```


© Projektkonsortium i-Twin (2024)

Sind für eine Eigenschaft die zugehörige Unit bzw. eine vordefinierte Werteliste vorhanden, so werden diese gemeinsam mit der Eigenschaft inline aufgelistet. Eine exemplarische Darstellung eines (zulässigen) Werts ist nachfolgend dargestellt.

Abfrage eines einzelnen Werts

```
GET <host:port>/value?irdiVA=0173-1%2307-AAA575%23003
{
      "supplier": "0173-1",
      "idVA": "AAA575003",
      "identifier": "AAA575",
      "versionNumber": "003",
      "revisionNumber": "01",
      "versionDate": "2013-11-28",
      "preferredName": "angular",
      "shortName": null,
      "definition": "Movement are possible in angular direction",
      "reference": null,
      "isoLanguage": "en",
      "isoCountryCode": "US",
      "irdiVA": "0173-1#07-AAA575#003",
      "dataType": "STRING"
}
```

Abfrage einer Unit

```
GET <host:port>/unit?irdiUN=0173-1%2305-AAA480%23002
{
      "irdiUN": "0173-1#05-AAA480#002",
      "comment": null,
      "definition": null,
      "dinNotation": "mm",
      "eceCode": "MMT",
      "eceName": "millimetre",
      "iecClassification": null,
      "nameOfDedicatedQuantity": "distance",
      "nistName": "millimeter",
      "shortName": "mm",
      "siName": "millimetre",
      "siNotation": "mm",
      "source": null,
      "structuredNaming": "millimetre"
}
```

Das Semantic Lookup Service wird im Rahmen des Projekts mit den Daten von ECLASS, v10.0 in der Sprache „en_US“ bereitgestellt. Eine Nutzung des Service mit ECLASS Daten außerhalb des Projekts ist an die Lizenzregelungen von ECLASS gebunden.

© Projektkonsortium i-Twin (2024)

2.1.4 Distribution Network (Subscriptions)

Ein Ziel für das Architektur-Design ist, die Kommunikation zwischen Assets und Anwendungen die via Data Integration Layer miteinander verbunden sind, möglichst einfach zu gestalten. 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. 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 2.2.1 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.

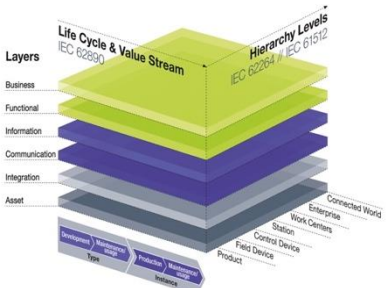


Abbildung 12: RAMI 4.0 und „Systeme“

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 13 zeigt das logische Daten- bzw. Objektmodell.

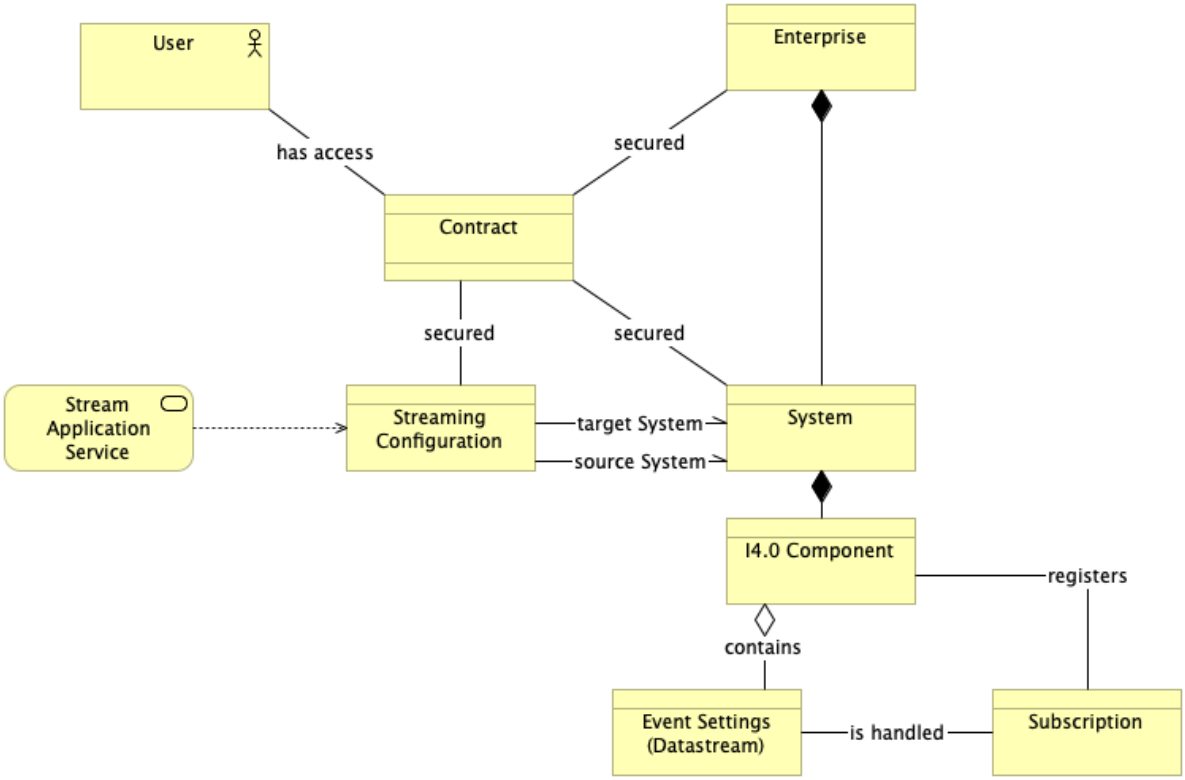


Abbildung 13: Logische Ansicht des Objektmodells des Distribution Network

Bei diesem Modell kommen folgende Überlegungen zum Tragen:

© Projektkonsortium i-Twin (2024)

- 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.

Definition Event-Nachrichten

Das Distribution Network liefet die Grundlage für die Etablierung von Kommunikationskanälen. Als wichtige Kriterien für die Kommunikation zwischen einzelnen I4.0 Komponenten wurden Sicherheit im Sinne von Privacy, Security und Safety erkannt. Zudem sollte eine Kommunikation nur über speziell strukturierte Kanäle erfolgen. In Abschnitt 2.2.1 wurde die Verwendung der Asset Administration Shell zur „Aktivierung“ einer I4.0 Komponente erläutert. Ein wesentlicher Bestandteil dieser Aktivierung sind die `Event`-Einstellungen der Asset Administration Shell wie sie in Abbildung 14 weiter unten dargestellt sind und welche auch für das Distribution Network wesentlich sind. Eine I4.0 Komponente sendet dabei seine Nachrichten an einen Messaging Broker (Apache Kafka, MQTT), der Message Broker übernimmt dann die Verteilung der Nachricht an ein oder mehrere Empfänger. Bevor Nachrichten „zielgerichtet“ zugestellt werden können, bedarf es jedoch eine semantische Definition der Message-Meta-Informationen und auch des wesentlichen Nachrichten-Inhalts. Hierfür sieht die Asset Administration Shell folgende Elemente vor:

- `EventElement`: Definiert den Kontext für einen Datenstrom.
- `EventPayload`: Definiert die Struktur einer Nachricht im i-Asset Distribution Network.

Während das `EventElement` als Teil der Asset Administration Shell die „ausführbaren“ Informationen für die I4.0 Komponente liefert, definiert die `EventPayload` die wichtigen Meta-Informationen der zu versendenden bzw. zu verarbeitenden Nachrichten.

© Projektkonsortium i-Twin (2024)

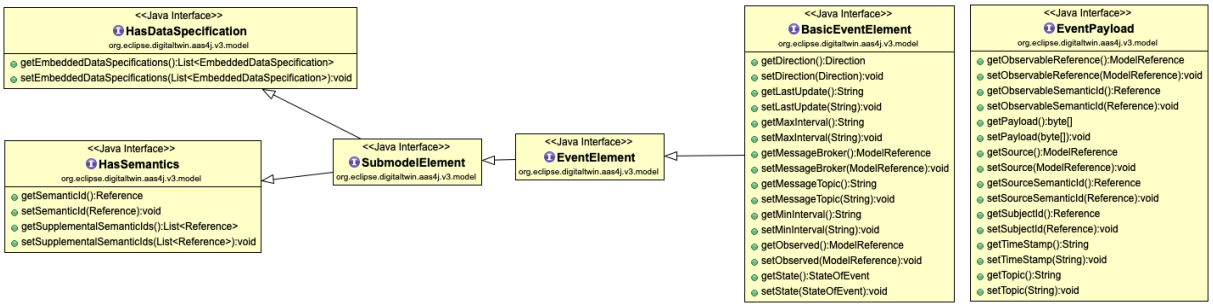


Abbildung 14: Event Settings

Die definierten Attribute sind wie folgt zu interpretieren:

- `source`: Referenz auf das `EventElement`, welches diese Nachricht erzeugt hat. Der Empfänger einer Nachricht kann diese Referenz mit Hilfe der i-Asset Platform auflösen und so alle Informationen erhalten, wie die Nachricht zu verarbeiten ist.
- `sourceSemanticId`: Referenz auf die semantische Auszeichnung des Event Elements.
- `observableReference`: Referenz auf das zu überwachende Element. Das zu überwachende Element definiert letztlich die Struktur des `payload`-Elements.
- `observableSemanticId`: Referenz auf die semantische Auszeichnung des zu überwachenden Elements. Hier können wiederum gemeinsam genutzte Taxonomien, z.B. ECLASS hinterlegt sein, welche z.B. eine Validierung einer Nachricht auf Vollständigkeit erlauben.
- `topic`: Name des zu benutzenden Data Channels
- `subject`: Bezeichner, Label für eine Nachricht
- `timestamp`: Zeitstempel wann diese Nachricht erzeugt wurde
- `payload`: Der aktuelle Wert des zu überwachenden Elements. Der Payload wird durch `observableSemanticId` mit Hilfe des Semantic Lookup vollständig definiert.

Das `EventElement` definiert zunächst den Kontext des Datenstroms innerhalb der Asset Administration Shell indem es das „überwachte Element“, das zu verwendende Messaging System (Message Broker), die Richtung der Kommunikation wie auch das zu verwendende Topic definiert. Das `EventElement` ist als Teilmodell-Element zunächst einem Teilmodell und in weiterer Folge einer Asset Administration Shell zugewiesen und wird innerhalb einer I4.0 Komponente „aktiviert“. Für eine I4.0 Komponente bedeutet dies, dass zunächst das zu überwachende Element innerhalb der Verwaltungsschale gesucht wird und für dieses die Messaging-Funktionalität „aktiviert“ wird. Dies hat bei Message Producern das periodische Auslesen und Versenden des überwachten Elements zur Folge. Bei Message Subscribern wird zunächst das entsprechende Topic des im `EventElement` definierten Message Brokers subskribiert und nutzenden Applikationen die Möglichkeit gegeben, sich über den Erhalt von Nachrichten informieren zu lassen. Das `EventElement` als Teil der Verwaltungsschale definiert somit die Anknüpfungspunkte der Client-Applikation zum Distribution Network. Dies betrifft nicht nur welcher Message Broker, welches Topic für das Senden bzw. Empfangen von Nachrichten herangezogen wird – im `EventElement` wird durch das „überwachte Element“ bzw. dessen `semanticId` auch der Nachrichteninhalt in der `EventMessage` definiert.

Entsprechend der Struktur der Klasse `EventMessage` kann eine Nachricht im JSON-Format gemäß folgendem Listing dargestellt werden:

```
{
   "source":                "<ref-to-event-element>",        // Event-Kontext
```


© Projektkonsortium i-Twin (2024)

```
   "sourceSemanticId":      "<ref-to-source-semantic>",       // Event-Semantics
   "observableReference":   "<ref-to-observable-element>",    // Überwachtes Element
   "observableSemanticId": "<ref-to-event-element>",          // Semantic Überw. Elem
   "topic":                 "at.srfg.iasset.lab.panda",       // Topic
   "payload":               <Any>                             // Payload
   "timestamp":             "2021-07-20T15:00:00.000Z",       // Zeitstempel
   "subject":               "Status-Info Panda",              // Label für die Message
}
```

Nachrichten-Empfänger müssen zunächst die Kontext-Informationen auswerten. Dieses definiert das zu überwachende Element (`observableReference`) sowie das zu beschickende Topic und den genutzten Messaging Broker. Eine Nachricht sollte möglichst auf das Wesentliche reduziert bleiben, aus diesem Grund sind lediglich die Referenzen auf die wichtigen Elemente enthalten. Mit Hilfe der i-Asset Platform können die Referenzen nun aufgelöst werden und die notwendigen Informationen, z.B. wie der wichtigste Teil der Nachricht – der Payload – zu verarbeiten ist. Der Payload kann dabei ein einzelner Wert sein oder eine semantisch definierte Objekt-Klasse sein. Zeigt die `observableReference` auf ein einzelnes `Property`, so definiert die referenzierte Eigenschaft den Datentyp der als `payload` in der Nachricht erwartet wird. Wir mittels der `observableReference` auf eine `SubmodelElementCollection`, also eine Sammlung von Teilmodell-Elementen verwiesen, so werden alle Elemente „unterhalb“ der `SubmodelElementCollection` als Key-Value Paare in der Nachricht aufbereitet, wobei die `idShort` des Elements als Key verwendet wird. Eine (optionale) `observableSemanticId` verweist zusätzlich auf die Definition einer Konzept-Klasse aus dem Semantic Lookup Service, welche in Folge die weiteren Attribute der semantischen Objekt-Klasse spezifiziert. Diese Attribute definieren gemeinsam den `payload` der Nachricht. Dieser kann nun auch auf Vollständigkeit bzw. Korrektheit überprüft werden.

Wie in Abbildung 14 ersichtlich, ist das `EventElement` ein Teilmodel-Element und somit `Referable`, besitzt zudem noch die Eigenschaften `HasSemantics` und auch `HasDataSpecification`. Diese Eigenschaften werden zur weiteren Definition des `payload` herangezogen. Nachrichtenempfänger erhalten so die erforderlichen Informationen wie die Nachricht zu verarbeiten bzw. auch zu validieren ist. Die übermittelten Nachrichten werden von der i-Asset Plattform nicht gespeichert und sind somit auch nicht Teil des Datenmodells.

Hier ist auch zu berücksichtigen, dass eine Asset Administration Shell viele `Event`-Elemente enthalten kann. Jedes dieser Event-Elemente führt zu einem Message Producer welcher Daten auf das definierte Topic publiziert bzw. einem Message Consumer, welcher sich auf die eingehenden Nachrichten des definierten Topics subskribiert. Die Messaging-Funktionalität einer aktiven I4.0 Komponente ist damit vollständig definiert und jede I4.0 stellt einen Client im Sinne des Distribution Networks dar.

Definition von Data Channels

Ein Message Broker stellt die Topics zur Verfügung, welche letztlich von den I4.0 Komponenten genutzt werden können. Jedes `EventElement` verweist dabei auf genau ein Topic wobei jede I4.0 Komponente eine Vielzahl an `EventElement`en aufweisen kann. Jede I4.0 Komponente wird nun in einem (abstrakten) logischen System (innerhalb eines Unternehmens, eines Werks oder Arbeitsstation) verortet, in dem es Nachrichten senden und empfangen kann.

Nachrichten die über ein logisches System hinausgehen, unterliegen einem so genannten Channel Contract welcher die Zustellung, bzw. Weiterleitung von Nachrichten zwischen logischen Systemen kontrolliert. Diese Anforderung bedingt, dass alle Nachrichten die zwischen zwei logischen Systemen ausgetauscht werden, vom Distribution Network kontrolliert werden.

© Projektkonsortium i-Twin (2024)

Für diesen Zweck wird der Begriff der Stream Applikation definiert, welche als Bindeglied zwischen logischen Systemen (Source-System, Target System) fungiert. Diese Überlegungen führen zu den exemplarischen Datenströmen wie in Abbildung 15 dargestellt.

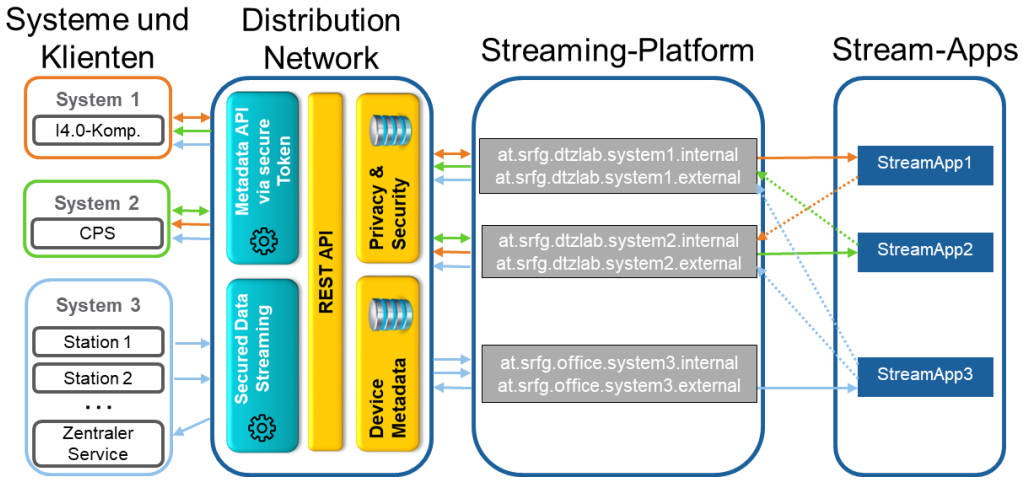


Abbildung 15: Exemplarische Datenströme zwischen unterschiedlichen Systemen

Zur Unterscheidung, welche Nachrichten durch diese Stream Applications überprüft werden müssen, dient ein Suffix im Topic-Namen. Für jeden definierten Datenkanal werden im Message Broker je ein Topic mit dem Suffix `internal` und `external` eingerichtet. I4.0 Komponenten senden Nachrichten mit dem Suffix internal. Bevor diese an system-fremde Empfänger weitergeleitet wird, erfolgt die Prüfung durch die Stream Application und die Nachricht wird an die Empfänger des Topix mit dem Suffix `external` zugestellt.

2.1.5 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[6] Standard. Hierzu wird mit Keycloak[7] 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.

[6] https://openid.net/connect/
[7] https://www.keycloak.org/

© Projektkonsortium i-Twin (2024)

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 ([idta2023-1]). Zusätzlich zum Zugriffschutz steht mit der Eigenschaft `Qualifiable` die Möglichkeit zur Verfügung, ein Teilmodell oder Teilmodell-Element mit eigenen Access-Regeln (`Constraint`) zu versehen.

2.2 Application / Edge Layer

Der Data Integration Layer dient als Vermittler zwischen Anwendungen und Assets. Entsprechend der Abbildung 2 kommunizieren Anwendungen und Assets mittels Data Integration Layer miteinander.

In den Abschnitten2.1.1 (Request/Response) und 2.1.4 (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 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.

2.2.1 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.

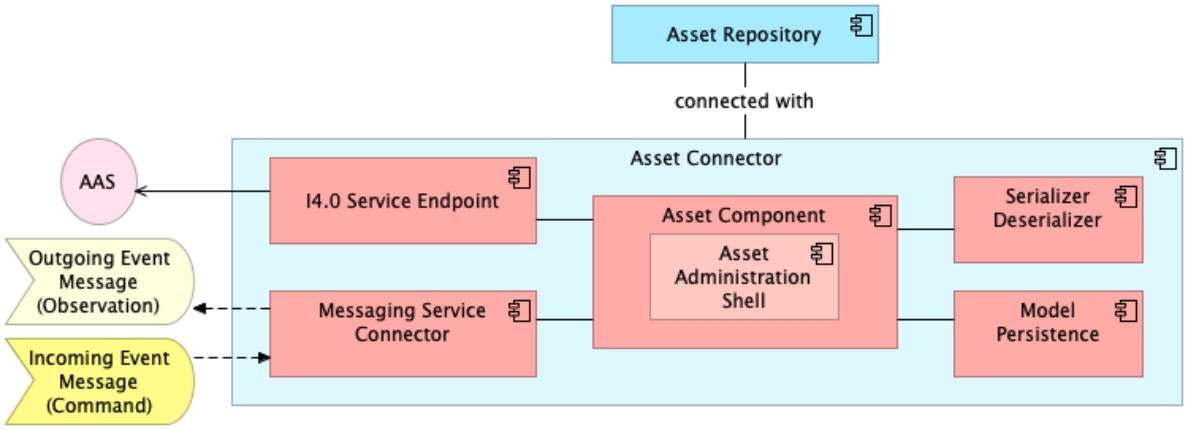


Abbildung 16: Asset Connector Sub-Components

© Projektkonsortium i-Twin (2024)

In Abbildung 16 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 die AAS-für die in der Asset Component geladene AAS entsprechend der in [idta2023-2] definierten Methoden bereitzustellen und die Informationen gemäß dem Meta-Modell ([idta2023-1]) zu repräsentieren bzw. zu konsumieren.
- Serializer/Deserializer: Transformiert die ein- bzw. ausgehenden Informationen vom Meta-Modell der Plattform Industrie 4.0 in das interne Asset Administration Model. Das interne Modell kann persistiert werden bzw. enthält 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.

© Projektkonsortium i-Twin (2024)

Abbildung 17 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.

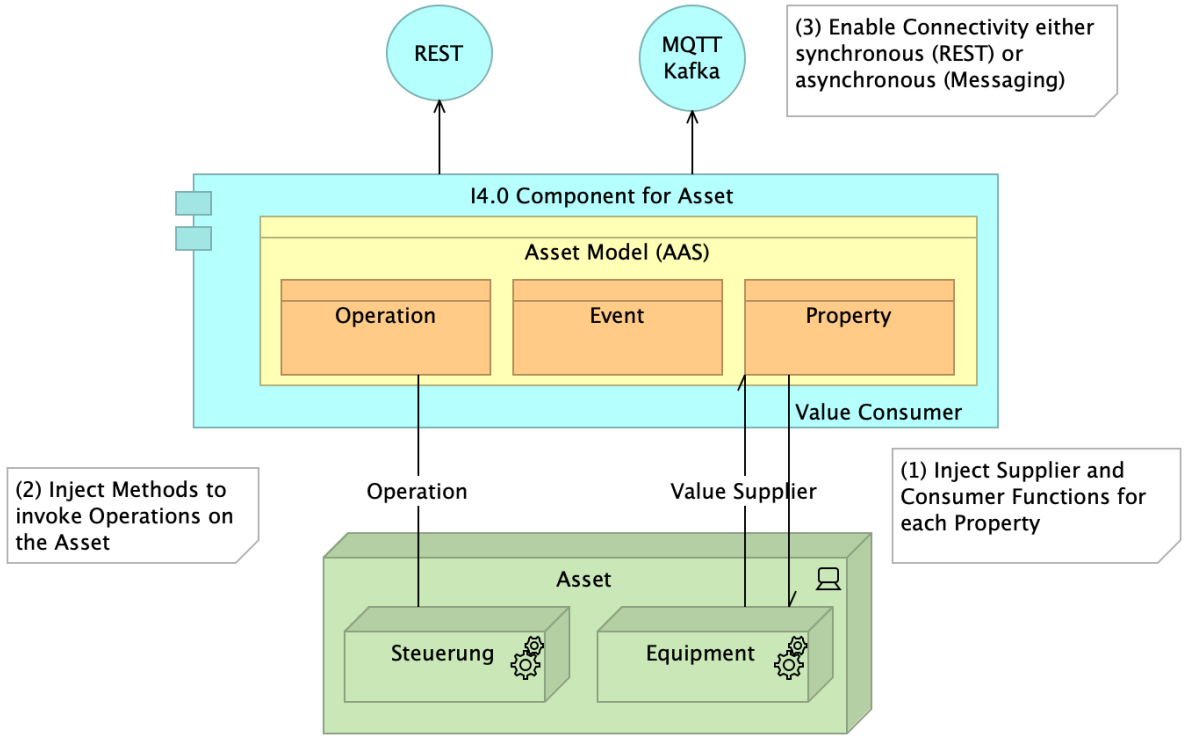
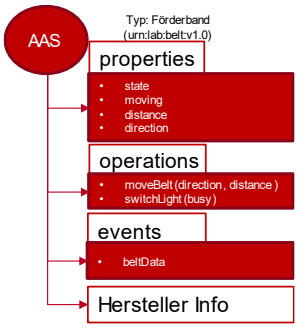


Abbildung 17: I4.0 Komponete – Aktivierung

Die I4.0 Komponente kann nun seine Struktur und auch seine Daten gemäß der Spezifikation in [idta2023-1] zur Verfügung stellen wobei die REST API den Spezifikationen in [idta2023-2] (siehe Abschnitt 2.1.1) entsprechen muss.

Aktivieren einer I4.0 Komponente

Eine I4.0 Komponente wird durch seine AAS-Umgebung bestehend aus Asset Administration Shell und referenzierten Teilmodellen kontrolliert. In den Teilmodellen sind die Detailinformationen in Form von Teilmodell-Elementen enthalten. Manche der Teilmodell-Elemente sollen nun mit den anliegenden Informationen des physikalischen Assets verbunden werden. Dies wird hier anhand des Förderbandes im i-Asset Labor beispielhaft dargestellt.



Das Förderband kann über eine Reihe von Eigenschaften Auskunft geben und auch wenige Aktionen ausführen. Diese Informationen sind in einer Asset Administration Shell mit verschiedenen Teilmodellen für Eigenschaften, Operationen und Ereignisse modelliert. Zusätzlich ist noch die Hersteller-Information im Modell enthalten.

Aktivieren von Properties

Die einzelnen Eigenschaften werden als Property-Elemente dargestellt. Jedes Element muss mit der Steuerung des Assets verbunden werden, so dass bei GET Abfragen von AAS-Informationen der korrekte Wert der jeweiligen Eigenschaft mit einbezogen wird.

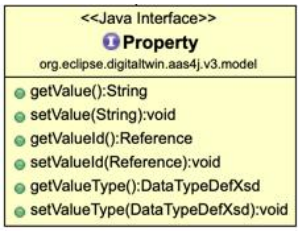
© Projektkonsortium i-Twin (2024)

Hierzu wird der I4.0 Komponente, für ein Teilmodell-Element, eine Methode bereitgestellt, die bei der Anfragebeantwortung aufgerufen wird.

```
i40Component.registerCallback(
                    // name the AAS
                    "urn:lab:belt:v1.0",
                    // name the Submodel
                    "properties",
                    // name the SubmodelElement by it's path!
                    "distance",
                    // define the callback routine
                    new ValueSupplier<Double>() {

                              @Override
                              public Double get() {
                                        // talk to asset and return distance
                                        return Asset.Distance();

                              }
                    });
```



Listing 1: Aktivieren von Properties – Abfragen von Werten

In Listing 1 wird der i40Component, diese verwaltet das Modell und kümmert sich um auch um jedwede Kommunikation mit weiteren I4.0 Komponenten, eine Methode hinzugefügt. Diese Methode lt. Interface `ValueSupplier` wird mit einem Datentyp versehen, so dass die umgebende Anwendung keine Datentransformation mehr machen muss. Diese passiert jedoch innerhalb der i40Component, denn diese erhält den numerischen Wert, validiert den Wert hinsichtlich Datenkonformität mit dem definierten Datentyp. Die Anfragebeantwortung erhält somit immer den aktuellen Wert der Eigenschaft.

Analog dazu kann auch ein PUT Request einen neuen Wert für eine Eigenschaft des physikalischen Elements bereitstellen. Die I4.0 Komponente kann diesen gleichermaßen an das Asset weiterreichen.

```
i40Component.registerCallback(
                    // name the AAS
                    "urn:lab:belt:v1.0",
                    // name the Submodel
                    "properties",
                    // name the SubmodelElement by it's path!
                    "direction",
                    // define the callback routine
                    new ValueConsumer<Direction>() {
                              @Override
                              public void accept(Direction newValue) {
                                        Asset.setForward(Direction.FORWARD.equals(newValue));
                              }
                    });
```

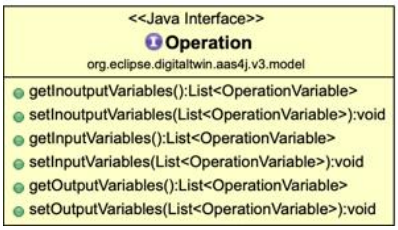
Listing 2: Aktivieren von Properties – Ändern von Werten

Listing 2 demonstriert die Änderung von Eigenschaften am Asset. Wird ein neuer Wert für das Teilmodell-Element an die I4.0 Komponente gesendet, so wird die entsprechende Methode gem. `ValueConsumer` aufgerufen und der bereitgestellte Wert typ-sicher an das Asset weitergereicht.

© Projektkonsortium i-Twin (2024)

Aktivieren von Operationen

Assets können in der Regel auch Aktionen ausführen. In unserem Beispiel kann die Kontroll-Lampe des Förderbands ein- bzw. ausgeschaltet werden. Im AAS-Modell ist hierfür ein Teilmodell mit einer Operation `switchLight` vorgesehen welche als Input-Parameter einen Boolean-Wert (true/false) erwartet.




```
i40Component.registerCallback(
                    // name the AAS
                    "urn:lab:belt:v1.0",
                    // name the Submodel
                    "operations",
                    // name the SubmodelElement by it's path!
                    "switchLight",
                    // define the operation as callback
                    new OperationCallback() {
                            @Override
                            public boolean execute(OperationInvocation invocation) {
                                    Boolean busy = invocation.getInput(Boolean.class);
                                    Asset.switchLight(busy);
                                    // success
                                    return true;
                            }
                    });
```

Listing 3: Aktivieren von Operations

In Listing 3 wird der i40Component für die Methode `switchLight` ein `OperationCallback`-Objekt übergeben und mit dem `Operation`-Element im AAS-Modell abgelegt. Bei einem Request zur Ausführung einer `Operation` wird zunächst ein `OperationInvocation`-Objekt erstellt, welches alle konfigurierten Ein-und Ausgabeparameter enthält. Die übergebenen Parameter werden typ-sicher dem `OperationInvocation`-Objekt übergeben und letztlich die bereitgestellte `execute`-Methode aufgerufen. In dieser Methode wird die Kommunikation mit dem Asset durchgeführt und ggf. noch die Ausgabeparameter im `OperationInvocation`-Objekt aktualisiert.

Aktivieren von Events

Events stellen eine Möglichkeit dar, auf Ereignisse zu reagieren. Eine I4.0 Komponente muss alle verwalteten Teilmodelle prüfen und für jedes einzelne Event-Elemente die Verbindungseinstellungen (Messaging Service Connector gem. Abbildung 16) vornehmen. Event Elemente definieren hierbei, ob es um eine ein- oder ausgehende Kommunikation handelt, welches Messaging Topic benutzt werden soll und (optional), welcher Messaging Broker genutzt werden soll.

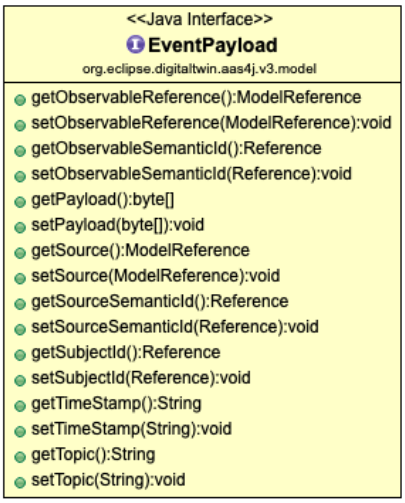
Die wichtigste Einstellung im Event Element ist jedoch das `observed`-Element. Diese Referenz zeigt auf das Teilmodell bzw. Teilmodell-Element, welches die Nachricht bzw. den wesentlichen Payload der Nachricht festlegt.

Bei eingehenden Events kann ein `EventHandler` registriert werden, dessen `onEventMessage` Methode aufgerufen wird, wenn eine Nachricht vom Messaging Broker empfangen wird.

© Projektkonsortium i-Twin (2024)

```
i40Component.registerCallback(
        // name the AAS
        "urn:lab:belt:v1.0",
        // name the Submodel
        "event",
        // name the SubmodelElement by it's path!
        "beltData",
        // define the e as callback
        new EventHandler<BeltData>() {
                @Override
                public void onEventMessage(
                        EventPayload eventPayload,
                        BeltData payload) {
                        // process the message provided in BeltData
                }
        });
```

Listing 4: Subscription von Events



In Listing 4 wird ein `EventHandler` mit einem typisierten Payload-Objekt (`BeltData`) registriert, welches in seiner Struktur dem `observed`-Element des Event Elements entsprechen muss. Bei einer eingehenden Nachricht wird die `onEventMessage`-Methode des `EventHandler's` aufgerufen. Diese Methode erhält das `EventPayload`- und das eigentliche Payload-Objekt als Parameter.

Das `EventPayload`-Objekt enthält Informationen Event Element, welches für diese Kommunikation verantwortlich ist. Hierzu gehören eine Model-Referenzen zum `observed`-Element aber auch Informationen zum `source`-Element.

Das generische Payload-Argument liefert typ-sicheren Zugriff auf die übertragenen Daten. Das Payload-Objekt entspricht in seiner Struktur dem Schema des im Event Element definierten `observed`-Element wobei hier die *ValueOnly*-Darstellung der AAS Elemente herangezogen wird.

Das Senden von Ereignissen erfolgt auf analoge Art und Weise. Dazu wird von der I4.0 Komponente ein `EventProducer` für das gewünschte EventElement angefordert. Mit dem `EventProducer`-Objekt kann nun ein Event versendet werden.

```
EventProducer<BeltData> dataProducer = i40Component.getEventProducer(
        // name the AAS
        "urn:lab:belt:v1.0",
        // name the Submodel
        "event",
        // name the SubmodelElement by it's path!
        "beltData",
        // Type of the Event Payload
        BeltData.class);
// send an event to the outer messaging infrastructure
dataProducer.sendEvent(beltData);
```

Listing 5: Senden von Events

Die Messaging Infrastruktur für dieses Event Element nimmt das Payload-Objekt entgegen und validiert dieses auf Übereinstimmung mit dem `observed`-Element gemäß den Event

© Projektkonsortium i-Twin (2024)

Einstellungen. Das validierte Objekt wird nun in ein `EventPayload`-Objekt eingebettet und an den Message Broker übergeben.

© Projektkonsortium i-Twin (2024)

3 i-Twin Plattform – Technologie

Das Projektkonsortium propagiert ein datenzentriertes, integriertes Asset Management durch die Integration bestehender Software Systeme wie Dokumentenmanagementsysteme (DMS), Instandhaltungsmanagementsysteme (CMMS), Analytik-Systeme z.B. für Predictive Maintenance sowie weiterer (spezialisierte) Planungs- und Kontrollsysteme für die Produktion (ERP). Der Architektur-Ansatz mit dem dieses Ziel erreicht werden soll ist in Abbildung 1 weiter oben dargestellt. Hierbei orientiert sich die i-Asset Plattform an den neuesten Technologien zur Entwicklung von Server- bzw. Cloud-basierten Plattformen.

Als technologische Basis bieten sich Micro-Services an. Hier sind auch verschiedene Open-Source-Komponenten verfügbar, die für die i-Asset Plattform erforderlich sind und wiederverwendet werden können.

Weitere Vorteile einer Micro-Service-Architektur[8] bestehen in der Modularisierung mittels wohldefinierter, funktional separierter Bausteine. Hier sind zu erwähnen:

- Bausteine kommunizieren über offene Schnittstellen
- Jedes Micro-Service kann unterschiedliche Basis-Technologien (e.g., Development-Stack) verwenden
- die Komplexität eines einzelnen Micro-Service ist eher gering
- notwendige Änderungen können leichter implementiert werden
- Fehler können schneller identifiziert und behoben werden
- Third-Party-Developer können leichter in das Projekt integriert werden
- Saubere Trennung von Verantwortlichkeiten im System auf einzelne Service-Projekte

Diesen Vorteilen stehen jedoch folgende Nachteile gegenüber:

- Integrationstests sind aufwendiger
- Systemkomplexität nimmt insgesamt zu
- Ressourcen-Overhead

Da die Vorteile überwiegen, wird diese Plattform-Architektur weiterverfolgt. Bei der Umsetzung kann zudem auf bestehende Erfahrungen und Micro-Services aus erprobten Plattformen zurückgegriffen werden. Hinzu kommen weitere Infrastruktur-Bausteine, welche zur Laufzeit zur Verfügung gestellt werden müssen, damit die Funktion der zu entwickelnden funktionalen Komponenten gewährleistet ist.

[8] Siehe z.B. https://microservices.io/patterns/microservices.html

© Projektkonsortium i-Twin (2024)

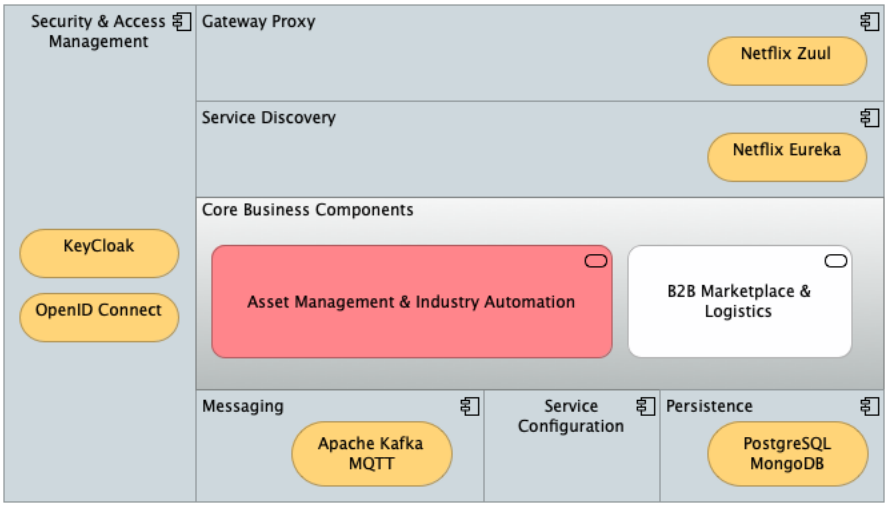


Abbildung 18: i-Asset Plattform – Infrastruktur

Abbildung 18 gibt einen Überblick über erforderliche Infrastruktur-Bausteine und benennt auch jene Open-Source-Module, welche zur Abdeckung der Funktionalität herangezogen werden. Die eigentliche Funktionalität, der Data Integration Layer mit seinen Bausteinen wie in Abschnitt 2 beschrieben liefert die gewünschte inhaltliche Ausrichtung der Plattform.

Die einzelnen Bausteine sind nachfolgend erklärt:

- **Security & Access Management**: Dieser Baustein ermöglicht die Verwaltung von Benutzern, so genannten Realms und übernimmt das Login-Management für die zentralen *Core-Business* Komponenten mit seinen Services. Zusätzlich stellt dieser Baustein die erforderlichen Security (OAuth)-Tokens für die Zugriffskontrolle bereit, d.h. jede Nutzung der Core-Services wird mit einem Security-Token abgesichert.
- **Gateway-Proxy & Service Discovery:** Die funktionalen Core-Services müssen möglichst skalierbar sein, es können zur Laufzeit einzelne Bausteine mehrfach instanziiert werden um möglichst viele gleichzeitige Anfragen abarbeiten zu können. Hierfür übernimmt der Gateway-Proxy das Load-Balancing, verteilt somit die Anfragen gleichmäßig an die verfügbaren Service-Instanzen. Dazu wird jedoch das Service-Discovery benötigt, um die vorhandenen *Core-Business* Services zu identifizieren. Das Service-Discovery wird auch von den *Core-Business* Services (implizit) genutzt, um Abhängigkeiten aufzulösen – so benötigt z.B. das Asset Repository das Semantic Lookup um die Asset Administration Shell vollständig beschreiben zu können. Die Nutzung dieser erforderlichen Services wird durch das Service Discovery zur Verfügung gestellt.
- **Messaging**: Stellt die Infrastruktur für das Versenden und Empfangen von asynchronen Nachrichten zur Verfügung. Die Core-Services müssen sich auf die Verfügbarkeit eines Message-Brokers verlassen können.
- **Persistence**: Daten müssen im Sinne einer Anwendung persistiert werden, hierfür kommen SQL-basierte Systeme wie PostgreSQL[9] aber auch dokument-basierte Systeme wie MongoDB[10] in Frage. Core-Services fordern lediglich eine Datenbank-Verbindung an, die sie nutzen können.

[9] https://www.postgresql.org/

[10] https://www.mongodb.com/

© Projektkonsortium i-Twin (2024)

- **Service-Configuration**: Alle Dienste und Services müssen zentral konfigurierbar sein. Die Service-Configuration erlaubt Plattform-Administratoren die Einstellungen einmal vorzunehmen und an alle instanziierten Dienste weiterzugeben.

Erst durch die Bereitstellung dieser Infrastruktur-Bausteine können sich die Micro-Services auf ihre funktionalen Aufgaben konzentrieren. Die Core Business Components für Asset Management und Industrie Automation wie in Abschnitt 2.1 beschrieben finden so ihre Laufzeitumgebung vor.

3.1 Platform Setup

3.1.1 Plattform-Management

Für das Management der einzelnen Plattform-Komponenten in einer Laufzeitumgebung werden Docker Container zur Verfügung gestellt. Die Anpassung der Services auf die konkrete Instanz erfolgt über Konfigurationsparameter (siehe Abschnitt 3.1.2).

Basierend auf den Konzepten der NIMBLE-Plattform[11] werden für unterschiedliche Setups (Development, Staging, Production Instanzen) separate Verzeichnisstrukturen mit Shell-Skripten, Docker-Compose Files (docker-compose.yml) und und Files mit (default) Umgebungsvariablen bereitgestellt (env_vars*). Für konkrete Instanzen müssen diese entsprechend angepasst werden.

Je Instanz benötigt Komponenten aus drei Bereichen:

- **Microservice Infrastruktur Kompenenten (“`infra`”)**, wie Load Balancer (Gateway-Proxy), Konfigurations-Server, Service Discovery oder Circuit Breaker
- **i-Asset Business Service Komponenten (“`services`”)**, wie Asset Repository, Semantic Lookup, Distribution Network, etc.
- **generelle, z.T. optionale Infrastrukturkomponenten** (z.B. Datenbanken[12], Identity Management[13], Reverse Proxy[14], Continuous Deployment[15], etc.), welche auch von extern bereitgestellten Servern genützt werden können.

Separate Docker Compose Files für “infra” und “services” sind verfügbar. Die einzelnen Komponenten müssen untereinander erreichbar sein, z.B. über ein gemeinsames „docker network“.

Der Zusammenhang zwischen den einzelnen Komponenten und Kategorien ist in Abbildung 19 schematisch dargestellt. Das Git-Repository „docker_setup“ ist öffentlich auf GitHub verfügbar: https://github.com/i-asset/docker-setup.

[11] Technologiebasis für i-Asset ist die NIMBLE-Plattform (https://github.com/nimble-platform/), vereinzelte Docker-Container wie z.B. der config-server https://hub.docker.com/r/nimbleplatform/config-server können unverändert von dieser übernommen werden.

[12] SQL z.B. Postgres: https://www.postgresql.org/ und No-SQL, z.B. SOLR: https://solr.apache.org/

[13] Open-ID Connect-kompatibel, z.B. Keycloak: https://keycloak.org/

[14] HTTP(s)- und Websocket-fähiger Reverse Proxy, z.B. NGINX: https://nginx.com/

[15] Für automatisches Build, Deliver & Deploy, z.B.Jenkins: https://www.jenkins.io/

© Projektkonsortium i-Twin (2024)

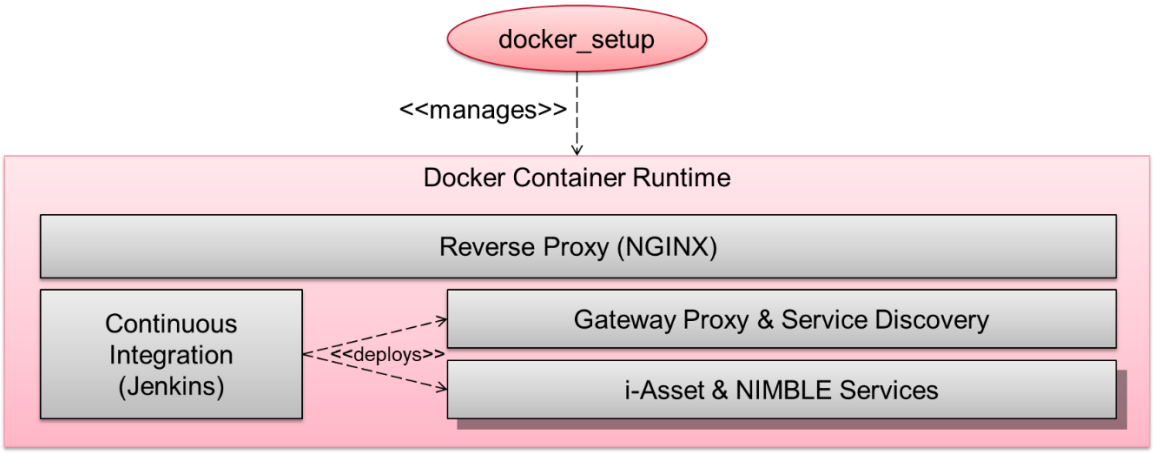


Abbildung 19 i-Plattform Setup Übersicht

3.1.2 Beispiel-Instanz für funktionale Tests und Demonstration

Die i-Asset Cloud Plattform läuft derzeit in einer virtuellen Serverumgebung im Labor von Salzburg Research, wie in Abbildung 20 schematisch dargestellt.

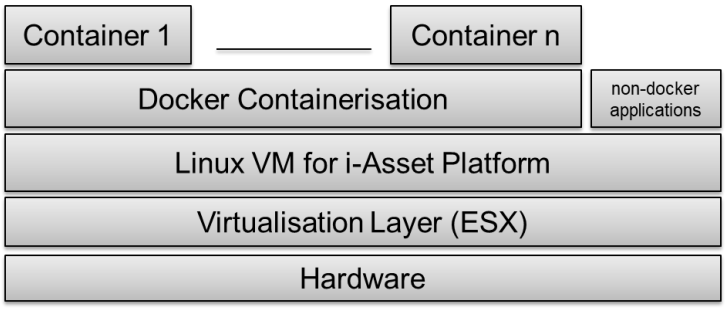


Abbildung 20 i-Asset Beispiel Deployment

Die Plattform läuft auf einem Ubuntu Linux Server in einer Docker Laufzeitumgebung. Diese Struktur bietet Flexibilität, Plattformunabhängigkeit und trotz Isolierung die Möglichkeiten, Ressourcen miteinander zu teilen (CPU, RAM und Storage). Die Instanz verwendet die „staging“-Variante der Plattform und dient als Referenz zum Testen der aktuellen Implementierung.

3.1.3 Plattform-Konfiguration

Die Konfiguration aller Plattformkomponenten erfolgt über Umgebungsvariablen, die erst beim Starten der Container in der Deployment-Phase gesetzt werden müssen. Die ausgelieferten Container kommen nur mit einer Basis-Konfiguration, die für das Betreiben der Plattform nicht ausreicht. Es wird zwischen der Backend-Konfiguration und der Frontend-Konfiguration unterschieden. Der Großteil der Businesslogik findet im Backend statt, die Frontend-Konfiguration beinhaltet größtenteils nur die verwendeten Service-Endpunkt-Adressen der Backend-Services.

© Projektkonsortium i-Twin (2024)

Backend-Konfiguration

Das Spring Framework für Java, das für die Implementierung der Service-Komponenten mehrheitlich genutzt wird, stellt unterschiedliche Konfigurationsmöglichkeiten zur Verfügung[16]. Überschreiben von Konfiguration einzelner Variablen zur Laufzeit ist möglich, wenn z.B. Umgebungsvariablen beim Start einzelner Container gesetzt werden. Wie in Abschnitt 3.1.1 beschrieben, ist der Konfigurationsserver für die Spring-Cloud[17] basierten Microservices Teil der Microservice-Infrastruktur. Wenn dessen Konfiguration nicht geändert wird, bezieht die minimale Default-Konfiguration von einem öffentlich verfügbaren GitHub Repository der NIMBLE Plattform[18].

Für jedes Deployment wird empfohlen, eine gültige Konfiguration über Umgebungsvariablen beim Start der Docker-Container zu injecten. Die vorgefertigten Docker-Compose Files verweisen dabei in den jeweiligen Sektionen (`env_file:`) auf die Files, die für die Konfiguration berücksichtigt werden. Alternativ, bzw. zusätzlich können Variablen direkt im Docker-Compose File in die Umgebungssektion (`environment:`) gesetzt werden.

Die komplette Liste der aktuell verwendeten Konfigurationsparameter befindet sich im projekteigenen Confluence-Wiki („Platform Secrets“) und wird laufend aktualisiert.

Frontend-Konfiguration

Für das iAsset-Frontend[19] wird das JavaScript-Framework Angular[20] auf Basis von TypeScript zum Einsatz. Für diese Komponente wird daher eine separate Konfiguration bereitgestellt. Neben einer Default-Konfiguration in den Files „globals.ts“ und „global-styles.css“ werden hier für unterschiedliche Deployment-Varianten (dev, staging, production) werden hier jeweils separate Konfigurationsfiles im Docker-Container mitgeliefert, die beim Container-Startup mit einer Konfigurationsvariable „TARGET_ENVIRONMENT“ aktiviert werden. Ist z.B. TARGET_ENVIRONMENT=*staging*, dann werden während des Containerstarts die default-Files mit den Inhalten aus „globals.*staging*.ts“ und „global-styles.*staging*.css“ überschrieben. Zu beachten ist daher, dass die Konfiguration während der Container-Build-Time[21] beim Container-Start durch das setzen der Variable „TARGET_ENVIRONMENT“ außer Kraft gesetzt wird.

3.2 Plattform Security

Der Absicherung jeglicher Kommunikation zwischen Assets und Anwendungen kommt eine hohe Bedeutung zu. Wie in der Architektur-Übersicht in Abbildung 1 angedeutet umspannt die Security sowohl den Application Layer, den Data Integration Layer und auch den Asset Layer.

Innerhalb des Data Integration Layers ist der Baustein Security & Identity Management (siehe Abbildung 2) angesiedelt. Dieser integriert mit KeyCloak ein Identity-Management Werkzeug

[16] https://spring.io/ , https://spring.io/blog/2020/04/23/spring-tips-configuration

[17] https://cloud.spring.io/spring-cloud-config/reference/html/

[18] https://github.com/nimble-platform/cloud-config

[19] https://github.com/i-asset/frontend-service

[20] https://angular.io/

[21] z.B. im “Continuous Build and Deployment”-Prozess von Jenkins

© Projektkonsortium i-Twin (2024)

welches zur Authentifizierung der Benutzer und zur Autorisierung derselben bei der Nutzung der Plattform genutzt wird. Hierfür nutzt die Plattform das OAuth 2.0[22] Protokoll. Dieses stellt den Industrie-Standard für Autorisierung dar und bietet eine umfassende Single-Sign-On (SSO) Funktionalität für Web- und Desktop-Anwendungen, Micro-Services und natürlich auch sonstige Devices (aka I4.0 Komponenten), im Folgenden nur kurz „Anwendungen“ genannt.

SSO erspart einerseits Benutzern Zeit erhöht aber gleichzeitig auch die Sicherheit. Es ist nur eine Anmeldung für eine oft sehr verteilte Infrastruktur notwendig, auch die Anzahl der verschiedenen Anmeldeinformationen (Benutzername/Passwort) reduziert sich für Benutzer erheblich. Aktuelle übliche Portal-Lösungen erlauben es, Anmeldeinformationen in einem einzigen Portal einzugeben – die zu Authentifizierenden/Autorisierenden Anwendung selbst erhält keinerlei Kenntnis über die Benutzername/Passwort Kombination, da Anwendungen nur zeitlich beschränkt geltende Schlüssel (Token) bereitgestellt werden. Durch eine einfach zu validierende Portallösung werden Phishing-Attacken schwieriger, aber es auch wird Bewusstsein für Benutzer geschaffen, wo Anmeldeinformationen mit wenig Risiko eingeben werden können.

Der Datenaustausch zwischen Anwendungen und Portal kann heute mit den Protokollen SAML2 oder OAuth 2.0 mit OpenID Connect (OIDC) durchgeführt werden, wobei der OIDC wesentlich neuer und verbreiteter ist. SAML2 basiert auf XML, wobei OIDC auf JSON basiert. Der Datenfluss der beiden Protokolle ist vergleichbar, haben jedoch teilweise andere Begriffe in Verwendung.

3.2.1 OAuth 2.0 und OpenID Connect (OIDC)

Bei OAuth 2.0 handelt es sich um einen Standard, das die Autorisierung für den Zugriff auf geschützte Ressourcen (Anwendungen, Daten) steuert. OIDC basiert auf OAuth 2.0 und erlaubt föderalisierte Authentifizierung, also über mehrere Systeme hinweg. Dabei wird einer Anwendung die Berechtigung erteilt, auf Daten in einer anderen Anwendung zugreifen zu dürfen. Vereinfacht sind in diesem Prozess die folgenden Begrifflichkeiten in OAuth 2.0 zu beachten:

- *Resource Owner*: Der Besitzer von Daten, dieser erlaubt anderen Teilnehmern (Clients) den Zugriff auf seine Daten.
- *Resource Server*: Das Service welches zum Zugriff auf die geschützten Daten genutzt wird. Der *Resource Server* stellt dafür eine API zur Verfügung die mittels OAuth 2.0 geschützt wird.
- *Authorization Server*: Das Service welches das (User-)Interface zur Verfügung stellt, in dem der *Resource Owner* die Daten-Anfrage bestätigt oder ablehnt. In einfanchen Anwendungen kann diese Aufgabe auch vom *Resource Server* wahrgenommen werden, zumeist wird diese Aufgabe als separate Komponente definiert. Der *Resource Owner* muss dem *Authorization Server* bekannt sein.
- *Client*: Die Anwendung, welche den *Resource Server* kontaktiert um Daten zu erhalten oder Aktionen auszuführen. Dazu muss der Client vom *Resource Owner* autorisiert werden.

[22] https://oauth.net/2/

© Projektkonsortium i-Twin (2024)

- *Authorization Code*: Ein temporärer Zugriffsschlüssel, den der *Client* benutzt um einen (länger lebenden) *Access Token* zu erhalten.
- *Access Token:* Ein Schlüssel, den der Client zur Kommunikation mit dem *Resource Server* benötigt. Der *Resource* Server prüft bei jedem Zugriff das *Access Token* um den Zugriff zu gewähren bzw. zu verweigern.
- *Client ID:* Anhand dieser ID wird der *Client* im *Authorization Server* identifiziert.
- *Client Secret:* Ein geheimes Passwort welches nur der *Client* und der *Authorization Server* kennen.

Vereinfacht dargestellt stellt sich der Ablauf des Autorisierungsprozesses wie folgt dar:

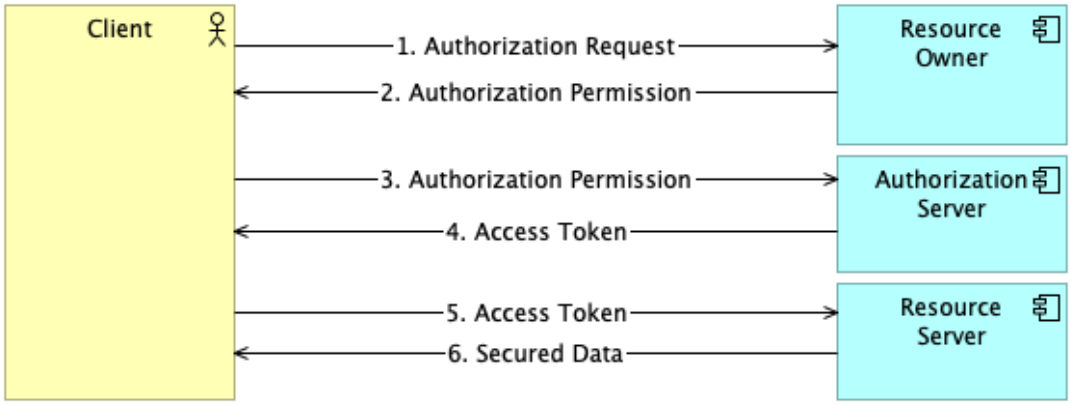


Abbildung 21: Authoriztation Flow

Die einzelnen Schritte sind dabei wie folgt:

1. Der Client stellt eine Autorisierungsanfrage an den Resource Owner. Dieser muss die Erlaubnis erteilen. In der Regel wird diese Anfrage indirekt vom Authorization Server abgehandelt.
2. Der Client erhält eine Autorisierungsgenehmigung vom Resource Owner.
3. Der Client fordert vom Authorization Server mit seiner Autorisierungsgenehmigung einen Access Token an.
4. Der Authorization Server authentisiert den Client und prüft die Autorisierungsgenehmigung. Nach erfolgreicher Prüfung wird ein Access Token erstellt.
5. Der Client greift auf geschützte Daten auf dem Resource Server zu. Zur Überprüfung der Autorisierung muss der Client bei jeder Anfrage den Access Token übermitteln.
6. Der Resource Server prüft das Access Token und beantwortet die Anfrage mit den geschützten Daten bzw. führt die geschützte Aktion aus.

Eine i-Asset Plattform besteht aus einer Vielzahl von einzelnen Devices (I4.0 Components), Anwendungen und Micro-Services, was die föderalisierte Authentifizierung notwendig macht. So müssen sich Devices und Anwendungen mit dem Asset Repository verbinden. Innerhalb des Data Integration Layer benötigen einzelne Bausteine ebenfalls Zugriff auf weitere Dienste, z.B. das Asset Repository muss ggf. auf das Semantic Lookup Repository zugreifen. Das Distribution Network nutzt ebenfalls das Asset Repository um die Event-Settings der einzelnen Assets zu erhalten. Alle diese Bausteine agieren somit als Client und die jeweils angefragten Services als Resource Server.

© Projektkonsortium i-Twin (2024)

3.2.2 Keycloak

Keycloak wird von Read Hat unter der Apache License 2.0 entwickelt und ist eine SSO Lösung auf Basis von Java. Die Lösung bietet viele Funktionen bereits fertig, aber anpassbar, an. Dazu gehören beispielsweise:

1) Benutzerregistrierung
2) Single Sign-On und Single Sign-Out
3) Protokolle wie OAuth 2.0, OIDC, SAML2
4) Authentifizierung mit externen OIDC/SAML Anbietern.
5) Social Log-In: Es können bei Bedarf existierende Dienste von Facebook, Twitter, Google, etc. verwendet werden.
6) Multi-Faktor-Authentisierung: Die Zugangsberechtigung wird durch mehrere unabhängigen Merkmale (Faktoren) überprüft.
7) User Federation: Synchronisierung mit LDAP und Active Directory Servern.
8) Oberflächen für Administration, Registrierung, Log-In, Log-Out, Profil bearbeiten, Zurücksetzen von Passwörtern, Validieren von E-Mail-Adressen usw.
9) Mandantenfähigkeit

Dabei setzt Keycloak auf einige Konzepte, die in Abbildung 22 dargestellt sind. Ein Realm erlauben es beispielsweise, verschiedene Organisationen abzubilden, zu einem Realm gehören Benutzer, Anmeldeinformationen, Rollen und Gruppen. Ein Benutzer muss sich bei einem Realm anmelden.

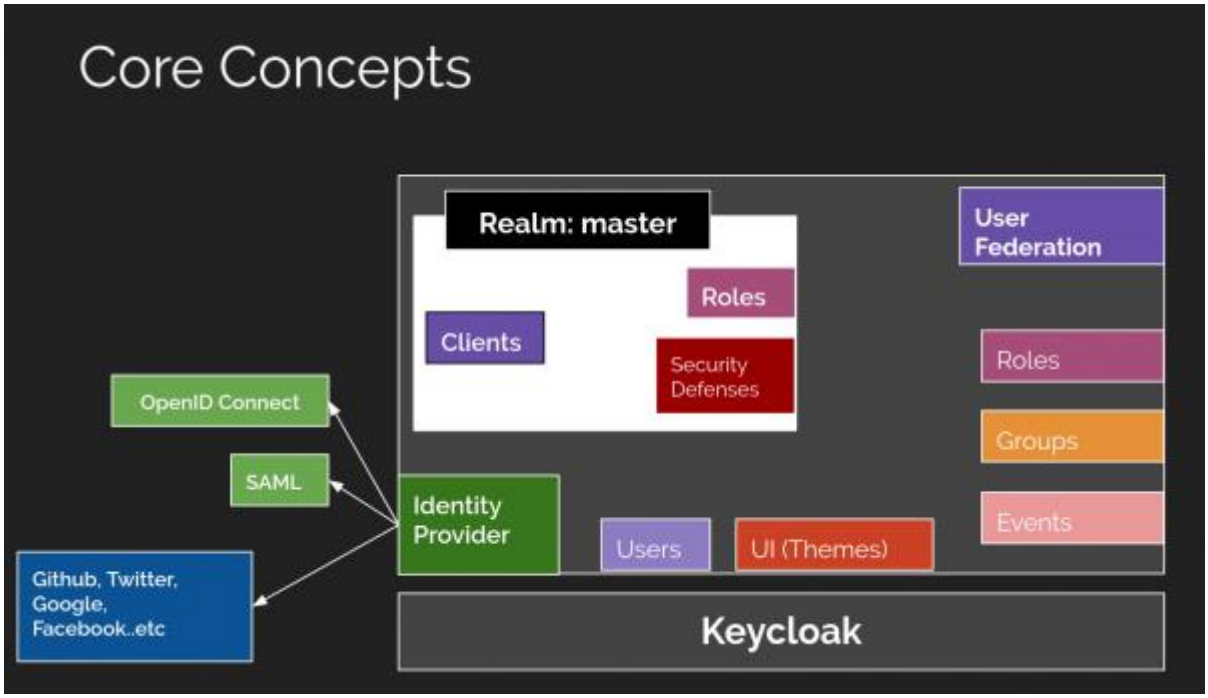


Abbildung 22: Kernkonzepte von Keycloak

Identity Provider sind Adapter, die Schnittstellen zu externen Diensten anbieten. Dabei werden einige Adapter mit Keycloak geliefert, es können aber selbst Identity Provider entwickelt werden. Das UI ist auf z. B. Corporate Design Richtlinien adaptierbar. Keycloak bietet ein Audit Logs Events für Administratoren an.

© Projektkonsortium i-Twin (2024)

Das Security & Identity Management (siehe Abbildung 1) deckt die Rolle des Authorizaton Servers ab.

3.3 Streaming-Applikationen

Die bisher vorgestellten Komponenten ermöglichen die Kommunikation zwischen Client-Anwendungen und dem ihnen zugeordneten System und somit auch die Kommunikation zwischen verschiedenen Anwendungen und Geräten innerhalb desselben Systems. Es gibt jedoch zahllose Beispiele für Fälle, in denen eine strukturierte Art der gemeinsamen Nutzung einer Teilmenge von Streaming-Daten von einer oder mehreren Quellen zu einem einzigen Zielsystem erforderlich ist. Um die Gesamtkomplexität zu reduzieren, unterscheiden wir zwischen zwei Grundtypen von Streaming-Applikationen. Einerseits gibt es Streaming-Applikationen, die nur Daten aus einer einzigen Quelle abrufen und optional eine Filterung anwenden, um den Datenstrom für das Zielsystem zu erzeugen. Andererseits können komplexere Anwendungen zwei Quelldatenströme als Eingabe haben und eine Verknüpfungsfunktion auf bestimmte Paare von Datenpunkten anwenden, um den Ausgabestrom für das Zielsystem zu erzeugen. Für beide Fälle wurden separate Prototypen implementiert, die über eine einzige Benutzeroberfläche in den Stream Hub Service der i-Asset Plattform integriert sind. Beide Arten von Streaming-Applikationen und ihre Implementierungen werden in den folgenden Unterabschnitten ausführlicher beschrieben.

3.3.1 Typen von Streaming-Applikationen

Wie bereits erwähnt, stellen wir zwei verschiedene Arten von Streaming-Applikationen für den Datenaustausch vor, von denen die eine Daten aus einer einzigen Quelle (d. h. einer Client-Anwendung) und die andere aus mehreren Quellen bezieht. Der trivialere Fall ist die *Single-Source Stream App*, bei der nur eine Teilmenge der Streaming-Daten konsumiert wird, eine benutzerdefinierte Filterung angewendet wird und die resultierenden Daten an das Zielsystem weitergeleitet werden.

Die *Single-Source Stream App* scheint flexibel zu sein, reicht aber nicht für komplexe Datenströme aus, die auf der Zusammenführung von zwei Streams auf der Basis von nahen Zeitstempeln basieren. Hierfür ist eine *Multi-Source Stream App* erforderlich. In einem Industrie 4.0 Szenario betrachten wir eine Produktionsmaschine, welche die Produktionsdaten nur an jenen Kunden senden, für welchen gerade produziert wird. Stellen Sie sich vor, dass die Datenströme von einer Maschine und einem ERP-System stammen. Wenn der Hersteller die Maschinendaten für Qualitätszertifizierungen nur dann an einen anderen Tenant (z. B. den Kunden) weiterleiten möchte, wenn eine bestimmte Produkt-ID, die vom ERP-System vergeben wird, auf der Maschine verarbeitet wird, und wenn eines der Systeme (z. B. das ERP-System) viel höhere Latenzen aufweist als das andere, d. h. das ERP-System liefert Datensätze, die die Produkt-ID enthalten, deutlich nachdem die Maschine mit der Arbeit an dem spezifischen Produkt begonnen hat. Damit der Kunde die Daten noch zeitnah zum Produktionsstart erhält, sollte das Datenstreaming beginnen, sobald alle Daten aus den Datenströmen aller angeschlossenen Systeme verfügbar sind. Im schlimmsten Fall würden unsynchronisierte Datenströme in einem solchen Szenario zur Übertragung von Maschinendaten anderer Kunden führen, nur, weil die das ERP-System mit der Produkt-ID eine erhöhte Latenzzeit hat! Daher ist der Determinismus eines solchen Zeitreihen-Joins sehr wichtig.

Diese Klasse an Probleme tritt auch in vernetzten Lieferketten oder Mobilität auf. Betrachten wir zum Beispiel den Fall, in dem Daten nur dann von einem Fahrzeug zum anderen

© Projektkonsortium i-Twin (2024)

übertragen werden, wenn der relative Abstand zwischen diesen unter einem bestimmten Schwellenwert liegt. Da sich die Positionen der Fahrzeuge dynamisch ändern, müssen die Koordinaten beider Fahrzeuge häufig abgefragt werden, um ihren relativen Abstand zu berechnen. Außerdem werden die Koordinaten der Fahrzeuge nicht synchron gesendet, so dass sich die Zeitstempel der einzelnen Datenpunkte im Allgemeinen unterscheiden. Obwohl wir davon ausgehen können, dass die Uhren in den angeschlossenen Fahrzeugen ausreichend synchronisiert sind, können Verzögerungen bei der Datenübertragung aufgrund von Netzwerkausfällen auftreten. Um dieses Problem zu lösen, ist eine Zeitreihenverbindung der Daten beider Fahrzeuge erforderlich, bei der jeder Datenpunkt - ein resultierender raumzeitlicher Sensorwert - mit dem vorhergehenden und nachfolgenden Datenpunkt des anderen Fahrzeugs verbunden wird. Anschließend wird der relative Abstand zwischen jedem Datenpunktpaar berechnet und nach einem vorgegebenen Schwellenwert gefiltert. Schließlich wird der Ergebniswert mit dem ursprünglichen Zeitstempel und den Koordinaten an das Zielsystem übermittelt.

Zusammenfassend lässt sich sagen, dass es sowohl in der Logistik als auch in der Industrie 4.0 Fälle gibt, die deterministische Zeitreihen-Joins und eine komplexere Stream-Sharing-Semantik erfordern. Um dies zu erreichen, wurde ein effizienter Algorithmus für Zeitreihen-Joins entwickelt, der deterministisch ist, minimale Latenzzeiten hat und einen hohen Durchsatz ermöglicht. In unserem Testaufbau waren bis zu 100.00 Joins pro Sekunde nur in Python und bis zu 15.000 Joins pro Sekunde mit *Exactly-once*-Verarbeitung unter Verwendung von Apache Kafka als Streaming-Plattform auf einem üblichen Desktop-Computer möglich.

3.3.2 Ausdruckssprache der Streaming Applikationen

Da es sich bei der i-Asset Plattform um einen Prototyp handelt, ist die Ausdruckssprache noch nicht für beide Typen von Streaming-Applikationen vereinheitlicht. Diese Entscheidung gewährleistet jedoch die Flexibilität des zweiten Typs, während die Einfachheit des ersten Typs erhalten bleibt.

Ausdruckssprache für die Single-Source StreamApp:

Das Hauptziel für die Ausdruckssprache für *Single-Source StreamApp* ist die Einfachheit. Daher haben wir einen SQL-ähnlichen Ausdruck gewählt, da SQL eine Standardsprache ist, die leicht zu erlernen und dennoch kompakt ist.

Wenn zum Beispiel ein bestimmter Temperaturwert, der an einer Wetterstation gemessen wurde, 30°C übersteigt oder unter 4°C liegt und an ein Zielsystem übertragen werden soll, könnte man diesen Ausdruck anwenden, um das Verhalten der StreamApp zu definieren:

```
SELECT * FROM * WHERE name = at.srfg.itwin-lab.Stations.Station_1.Temperature' AND (result < 4 OR result > 30);
```

Dieser Ausdruck leitet jeden Datenpunkt (gekennzeichnet durch das Asterisk-Symbol '*') weiter, der dem nach dem "WHERE"-Schlüsselwort definierten Filtermechanismus genügt. Der Ausdruck nach dem "FROM"-Schlüsselwort wird noch nicht benötigt und ist daher ebenfalls mit dem Asterisk-Symbol '*' gekennzeichnet, da der Eingangsstrom aus dem Quellsystem bereits durch das "TARGET_SYSTEM" innerhalb der UI des Stream Hub Service definiert ist.

© Projektkonsortium i-Twin (2024)

Ausdruckssprache für die Multi-Source StreamApp:

Um die Komplexität einer *Multi-Source StreamApp* und die Forderung nach hoher Flexibilität zu bewältigen, werden die anpassbaren Teile der Applikation in einer Datei namens *'custom_fct.py'* definiert, die einer vordefinierten, einfachen Struktur folgt. Sie besteht lediglich aus einigen Konstanten und zwei Funktionen, die *'ingest_fct'* und *'on_join'* heißen. Die erforderlichen Konstanten (in Großbuchstaben) sind die folgenden:

KAFKA_BOOTSTRAP_SERVERS: Kafka-Knoten der Form 'mybroker1,mybroker2'
SYSTEM_IN: Liste der Quell-Systeme, durch Komma getrennt
SYSTEM_OUT: Zielsystem, an das die resultierenden Daten ausgegeben werden sollen
TIME_DELTA: Maximaler Zeitunterschied zwischen zwei zu verbindenden Datensätzen
ADDITIONAL_ATTRIBUTES: optionale Attribute in den Datenpunkten, zB.: "att1,att2,..."
USE_ISO_TIMESTAMPS: boolean: Zeitstempelformat der resultierenden Datensätze, ISO 8601 oder Unix-Zeitstempel bei False
MAX_BATCH_SIZE: konsumiere bis zu dieser Anzahl von Nachrichten auf einmal
TRANSACTION_TIME: Zeitintervall für das Kommiten (Kakfa-spezifisch) von Transaktionen
VERBOSE: boolesch, gibt mehr Logs aus

Zusätzlich zu diesen Konstanten müssen zwei Funktionen definiert werden:

- *ingest_fct*: Diese Methode erhält den empfangenen Datensatz und die Stream-Buffer-Instanz als Argumente und gibt an, unter welchen Bedingungen der Datensatz in den linken oder rechten Puffer (oder gar nicht) der Stream-Buffer-Instanz eingefügt wird. Beachten Sie, dass jeder binäre Join zwei einzelne Datenpunkte von zwei verschiedenen Eingängen benötigt, wobei einer davon als linker, der andere als rechter Join-Partner bezeichnet wird.
- *on_join*: Diese Funktion erhält zwei Datensätze, einen linken und einen rechten Join-Kandidaten. Innerhalb der Funktion können benutzerdefinierte Join- und Filtermechanismen definiert werden. Anschließend wird der resultierende Datensatz oder ein *NULL*-Wert zurückgegeben, falls kein Join gewünscht ist.

In Abbildung 32 ist die Verknüpfung von ereignisbasierten Zeitreihendaten dargestellt. Oben ist die Latenz aller Datenpunkte gleich, somit entspricht die Ereigniszeit der Ankunftszeit (*ingestion time*). Im unteren Bild wird eine hohe Latenz von r angenommen, das heißt es werden zuerst alle Datenpunkte von s empfangen, und danach erst jene von r. Für beide Fälle sind die ermittelten Vereinigungen ident. Der Algorithmus ist somit deterministisch hinsichtlich der Latenz der empfangenen Datenpunkte, solange die Reihenfolge der Datenpunkte innerhalb einer Quelle richtig ist. Die unterschiedlichen Join-Fälle (*cases*) deuten die Arbeitsweise des Algorithmus an. Details zum Algorithmus werden in [schranz2020] beschrieben.

© Projektkonsortium i-Twin (2024)

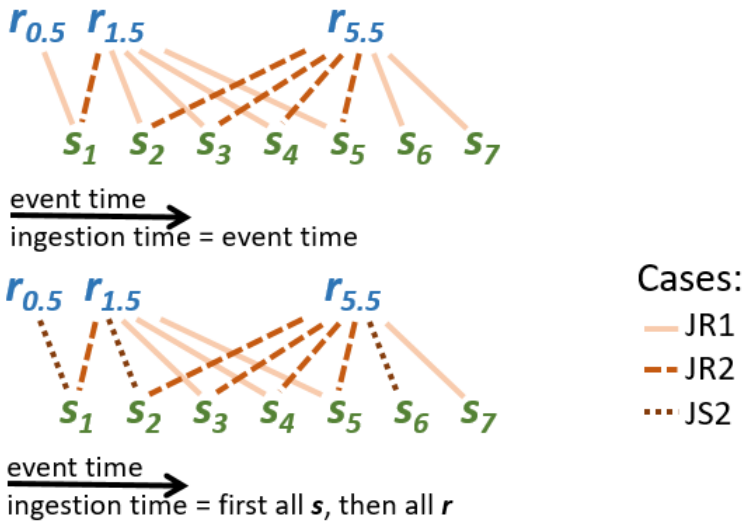


Abbildung 32: Der Join zweier asynchroner Zeitreihen ist trotz unterschiedlicher Sendezeitstempel deterministisch.

3.3.3 Implementierung der Streaming Applikation

Für beide vorgestellten Fälle wird eine Beispielimplementierung vorgestellt, die in die i-Asset Plattform integriert ist. Ziel jeder Implementierung ist es, eine eigenständige Streaming-Applikation zu haben, die von außen gesteuert werden kann, d. h. sie kann eingesetzt und angehalten werden, und es können Logging-Ausgaben abgerufen werden. Eine praktische Möglichkeit, dies zu erreichen, besteht darin, jede StreamApp in einem separaten Container bereitzustellen, wie es durch Docker ermöglicht wird. Docker ist eine Orchestrierungssoftware, die Applikationen in isolierten Containern ausführt. Dies hat den Vorteil, dass jede StreamApp in einem eigenen Docker-Container mit verschiedenen Programmiersprachen (Java, Python) laufen kann, während sie über die Benutzerschnittstelle der i-Asset Plattform gesteuert werden kann.

Implementierung der Single-Source Stream App:

Der Quellcode für die *Single-Source StreamApps* wurde in Java unter Verwendung des *Kafka Streams*[23] Frameworks implementiert. Kafka Streams wird als Bibliothek verwendet, um Daten aus einem Kafka Cluster zu konsumieren und zu produzieren, wobei es eine flexible Anwendung von Filtermechanismen ermöglicht. Die Haupt-Java-Anwendung erhält gegebene Variablen wie den Stream-Namen, das Quell- und Zielsystem als Umgebungsvariablen und parst die Ausdruckssprache in eine spezialisierte *StreamQuery*-Instanz. Sobald diese rekursive *StreamQuery*-Instanz basierend auf einem Filterausdruck initialisiert ist, können damit neu empfangene Daten sehr schnell gefiltert werden.

Die *Single-Source StreamApp* verarbeitet die Daten genau einmal (*exactly-once*), da die Zustellungsgarantien von der Kafka-Streams-Bibliothek übernommen werden. Die Java-

[23] Kafka Streams: https://kafka.apache.org/documentation/streams/

© Projektkonsortium i-Twin (2024)

Anwendung wird in Docker implementiert, um eine vollständige Kontrolle über die Benutzeroberfläche zu ermöglichen.

Implementierung der Multi-Source StreamApp:

Im Gegensatz zur *Single-Source-StreamApp* basiert die Multi-Source-Variante auf einem effizienten Zeitreihen-Join von Daten aus zwei Datenströmen. Die vorgeschlagene In-Memory-Implementierung ermöglicht die *Exactly-once*-Verarbeitung eines Joins für Apache Kafka als Streaming-Plattform. Sie ist selbst bei beliebig hohen, jedoch endlichen, Latenzen der empfangenen Daten deterministisch.

Die *Multi-Source StreamApp* ist in Python mit der *confluent_kafka_python* Bibliothek und einfachem Apache Kafka geschrieben und wird mit Docker orchestriert. Sie erhält die erforderlichen Konstanten und Funktionen und initialisiert eine neue Stream-Buffer-Instanz. Basierend auf der angegebenen Methode „*ingest_fct*“ werden die empfangenen Daten in den linken oder rechten Puffer (oder gar nicht) der Stream Buffer-Instanz aufgenommen. Sobald ein neuer Join-Kandidat verfügbar ist, werden beide Join-Partner an die Funktion „*on_join*“ übergeben, die das beliebige Verknüpfungs- und Filterverhalten der Join-Partner definiert. Der resultierende Datensatz wird dann an das Zielsystem ausgegeben.

3.3.4 API der Stream-Apps

In der Tabellen 3 wird die API für das Verwalten von Datenströmen und Subskriptionen von systemübergreifenden (externen) Datenströmen beschrieben.

Methode	Pfad	Beschreibung
GET	/distributionnetwork/datastreams/{personId}/{system}	Return all datastreams that belong to a system.
POST	/distributionnetwork/datastreams/{personId}/{system}	Create a new datastreams for a system that belongs to a person.
PUT	/distributionnetwork/datastreams/{personId}/{system}	Create or update datastreams for a system that belongs to a person.
GET	/distributionnetwork/datastreams_per_client/{personId}/{system}/{client_name}	Return all datastreams that belong to a client application of a system.
GET	/distributionnetwork/datastreams_per_thing/{personId}/{system}/{thing_name}	Return all datastreams that belong to a thing of a system.
DELETE	/distributionnetwork/delete_datastreams/{personId}/{system}/{thing_name}	Delete datastreams from from a system.
GET	/distributionnetwork/subscriptions/{personId}/{system}	Return all datastream subscriptions that belong to a system.
GET	/distributionnetwork/subscriptions_per_client/{personId}/{system}/{client_name}	Return all datastream subscriptions that belong to a client application of a system.

© Projektkonsortium i-Twin (2024)

POST	/distributionnetwork/subscriptions_per_client/{personId}/{system}/{client_name}	Create new datastream subscription for a client application that belongs to a person.
PUT	/distributionnetwork/subscriptions_per_client/{personId}/{system}/{client_name}	Create or update datastream subscriptions for a client application that belongs to a person.
DELETE	/distributionnetwork/delete_subscriptions/{personId}/{system}/{client_name}	Delete datastream subscription from from a client application.

Tabelle 3: API Methoden für Datastreams und Subskriptionen

In der Tabelle 4 werden die Methoden für das Erstellen und Verwalten von StreamApps dargestellt.

Methode	**Pfad**	**Beschreibung**
GET	/distributionnetwork/stream_apps/{personId}/{system}	Return all stream apps that belong to a system.
POST	/distributionnetwork/stream_apps/{personId}/{system}	Create a new stream app for a system that belongs to a person.
PUT	/distributionnetwork/stream_apps/{personId}/{system}	Create or update a stream app for a system that belongs to a person.
GET	/distributionnetwork/stream_apps/{personId}/{system}/{stream_name}	Return a specific stream app that belongs to a system.
DELETE	/distributionnetwork/delete_stream_app/{personId}/{system}/{stream_name}	Delete a stream app from a system.
GET	/distributionnetwork/stream_app_statistic/{personId}/{system}/{stream_name}	Return the statistic of a specific stream app that belongs to a system.
POST	/distributionnetwork/stream_app_deploy/{personId}/{system}/{stream_name}	Deploy an existing stream app of a system.
PUT	/distributionnetwork/stream_app_deploy/{personId}/{system}/{stream_name}	Deploy an existing stream app of a system, stop and redeploy if it already runs.
POST	/distributionnetwork/stream_app_stop/{personId}/{system}/{stream_name}	Stop an existing stream app of a system.

Tabelle 4: API Methoden für StreamApps

Die hier vorgestellte API basiert auf existierende PersonIDs sowie eindeutigen System-Namen. Wie bereits näher beschrieben, erlaubt diese API die Lösung einer großen Klasse an praktischen Problemen, die sogar auf die Verknüpfung von zwei Datenströmen angewiesen sind.

© Projektkonsortium i-Twin (2024)

3.4 Semantic Integration Patterns

Mit zunehmender Vernetzung von Assets mit Anwendungen bzw. auch von Anwendungen untereinander steigt die Notwendigkeit hier mittels vordefinierter Templates und Kommunikations-Patterns den Konfigurationsaufwand zu minimieren. Vorrangiges Ziel ist es hierbei, innerhalb der Anwendungstypen jene Teilmodelle zu definieren, die für die Kommunikation mit den einzelnen Anwendungen erforderlich sind. Sinngemäß soll sich eine Anwendung als „Instanz eines Typen“ in der Plattform andocken können. Mit der Registrierung einer Anwendungs-Instanz werden die im Typ definierten Kommunikationspfade auch tatsächlich instanziiert. In den Anwendungs-Typen werden das asynchrone (`EventElement`) und auch das synchrone (`Operation`) Verhalten von Anwendungen definiert. Dabei werden durch die Elemente der AAS auch jene Daten-Objekt-Strukturen definiert die bei der Kommunikation maßgeblich sind.

Die nachfolgende Beschreibung dient dazu, dass Prinzip der Verwendung der Asset Administration Shell zur Modellierung von Anwendungs-Typen und -Instanzen soweit vorzustellen, wie es das Verständnis der Systemarchitektur erfordert. Eine erweiterte und detaillierte Darstellung der Semantic Integration Patterns erfolgt in zwei dedizierten Deliverable (D2.2 „Semantic Integration Patterns for Manufacturing“ und D2.3 „Semantic Integration Patterns for AI“).

3.4.1 Typ- vs. Instanz-Elemente

Eine Asset Administration Shell setzt sich aus vielen einzelnen Teilmodellen, Teilmodell-Elementen und Konzept-Beschreibungen zusammen wobei `Identifiable`-Elemente (`AssetAdministrationShell`, `Submodel`, `Asset`) mit global eindeutigen Identifiern ausgestattet sind und `Referable`-Elemente innerhalb ihres Kontexts anhand ihrer `idShort` referenziert werden können. Je nach Ausprägung der `Identifiable`- bzw. `Referable`-Elemente werden innerhalb einer AAS jene Struktur-Informationen abgelegt, welche den digitalen Zwilling eines Assets darstellen. Um z.B. für viele Maschinen gleicher Bauart nicht jede einzelne Information immer wieder redundant ablegen zu müssen, wird zwischen Typ- und Instanz-Informationen unterschieden. Diese Unterscheidung wird durch den Stereotyp *HasKind* definiert, siehe dazu auch Abschnitt 2.1.1:

- Typ-Informationen: Datenobjekte mit `HasKind=TEMPLATE` repräsentieren abstrakte oder statische Informationen die letztlich an konkrete Anwendungs-Instanzen weitervererbt werden können. Typ-Informationen können analog der objekt-orientierten Datenmodellierung keine aktive Rolle in einer i-Asset Plattform einnehmen. Sie können sinngemäß nicht als I4.0 Komponente aktiviert werden und somit auch keine dynamischen Informationen von Assets repräsentieren.
- Instanz-Informationen: Datenobjekte mit `HasKind=INSTANCE` repräsentieren konkrete digitale Abbilder von Anwendungs-Eigenschaften. Diese können, müssen aber nicht von abstrakten Typ-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

© Projektkonsortium i-Twin (2024)

Die Typ-Instanz-Beziehung kann nicht nur für die Vererbung von Struktur und statischen Informationen genutzt werden. Neben den dynamischen Eigenschaften kann ein Asset-Typ auch Ereignisse und Methoden definieren, die letztlich von den Instanzen „ausgestaltet“ werden müssen. Ein Typ kann somit generell als Klassifikationsmerkmal für Instanzen betrachtet werden.

Diese Typ-Instanz-Unterscheidung gilt sowohl für Assets als auch insbesondere für Anwendungen, da Anwendungstypen die möglichen funktionalen Aspekte (ERP, CMMS, Dashboards, Analytics-Anwendungen) innerhalb einer i-Asset Plattform definieren wie sie in Abbildung 1 dargestellt sind. Mit Hilfe von Anwendungs-Typen können jene konkreten Anwendungs-Instanzen identifiziert werden, welche die funktionalen Aspekte konkret umsetzen.

Vereinfacht dargestellt, stellt ein Anwendungs-Typ eine funktionale Vorlage für konkrete Anwendungen dar. Mit Hilfe von Teilmodellen, ebenfalls als Typ markiert (`HasKind.TEMPLATE`), werden die einzelnen Operationen und sonstige Kommunikations-Einstellungen für Anwendungen definiert.

3.4.2 Anwendungs-Typen

Jedes Teilmodell erhält den Anforderungen der Asset Administration Shell entsprechend einen global gültigen Bezeichner. Somit kann das Teilmodell direkt angesprochen werden. Jedes Teilmodell-Element (Operationen, Ereignisse, etc.) erhalten die im jeweiligen Kontext gültige `idShort` zur eindeutigen Identifikation innerhalb ihres übergeordneten Elements. Die einzelnen Elemente können somit ausgehend vom Teilmodell referenziert werden.

Anwendungs-Typen und Datenstrukturen

Anwendungen sollen miteinander kommunizieren können, also Daten in strukturierter Form austauschen. Mit Hilfe der Event- und Operation-Elemente werden zunächst die Kommunikations-Wege definiert. Der wesentliche Mehrwert bei jeder Form der Kommunikation liegt jedoch in den ausgetauschten Daten. Mit diesen Daten sollen Methoden aufgerufen werden bzw. sollen ein oder mehrere Empfänger in der Lage sein, den Payload einer asynchronen Nachricht auch korrekt verarbeiten zu können.

Ziel ist es somit, die erforderlichen Parameter für die Ausführung einer Funktion bzw. den Payload einer asynchronen Nachricht vollständig zu beschreiben. Das Meta-Modell der Asset Administration definiert hierfür den Stereotyp *HasSemanics*, um interne Konzeptbeschreibungen oder auch externe Taxonomie-Systeme zu integrieren.

Abbildung 23 verdeutlicht das Prinzip. Ein Anwendungs-Typ definiert (zumindest) ein `Submodel`, dieses wiederum enthält Teilmodell-Elemente vom Typ `Operation` und/oder `EventElement`. Die weitere Ausgestaltung dieser Elemente ist für deren Verwendung wesentlich. Das `Operation` Element verweist mittels Input- bzw. Output-Variablen auf die Beschreibung der erforderlichen Eingabe-Daten bzw. auf die zu erzeugende Antwort. Ein `EventElement` besitzt mit dem Attribut `observed` eine Referenz auf das zu „überwachende Element“. Dieses Element definiert sinngemäß die Struktur des Payloads in einer Event-Message. Je nach Ausprägung des `observed`-Elements kann der Payload ein primitiver Wert oder eine komplexe Datenstruktur darstellen, je nachdem ob das `observed`-Element gem. Meta-Modell Unterelemente enthält oder nicht.

© Projektkonsortium i-Twin (2024)

Abbildung 23 verdeutlicht die Definition von Datenstrukturen mit Hilfe des Semantic Lookup Repository. Das exemplarische Teilmodell enthält ein `EventElement` („Alert Message“) um die asynchrone Kommunikation zu beschreiben. Das `EventElement` verweist (`observed`-Attribut) auf ein `Entity`-Element („Fault Indication“). `Entity`-Elemente können so wie `Submodel` oder `SubmodelElementCollection`-Elemente auch Unterelemente aufweisen, sie stellen somit eine komplexe Datenstruktur dar. Vor allem aber sind sie vom Stereotyp *HasSemantics* und verweisen mittels `semanticId` auf eine Datenstruktur im Semantic Lookup Repository.

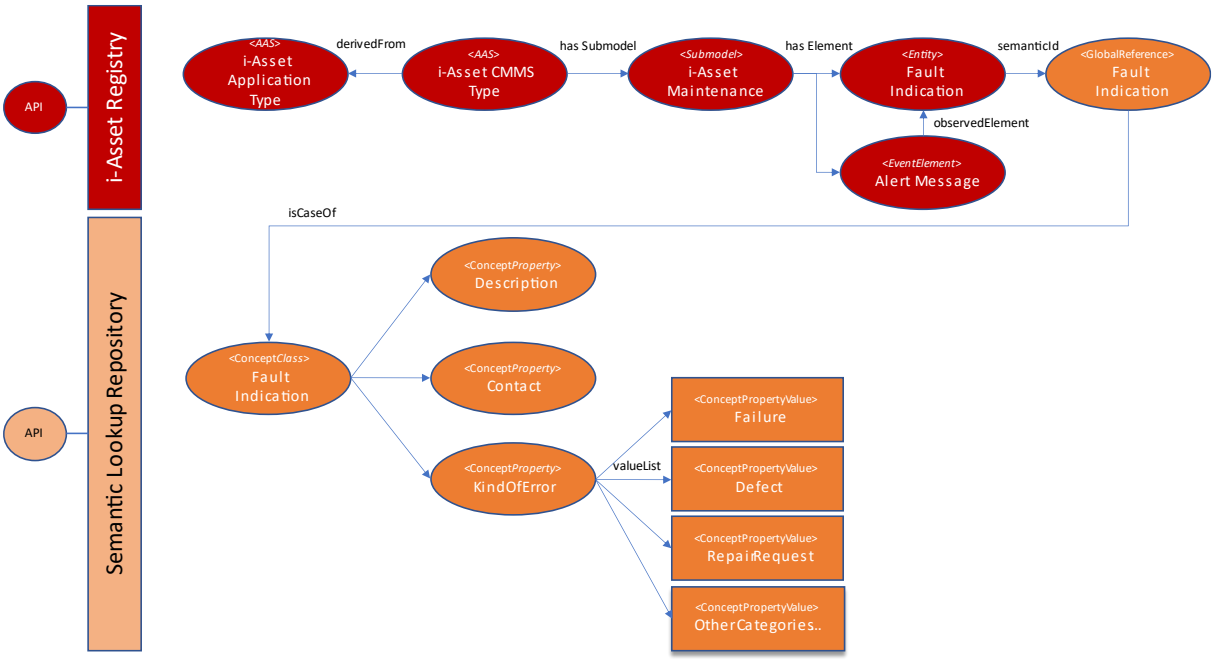


Abbildung 23: Anwendungs-Typen und semantische Datenstrukturen

Diese Datenstruktur in Form von Konzept-Beschreibungen, Eigenschaften bis hin zu vordefinierten Wertelisten kann von allen Teilnehmern an der Plattform abgerufen werden und ermöglicht die vollständige Validierung der ausgetauschten Datenobjekte. Anwendungs-Typen definieren so ihre Funktionalität und geben zusätzlich einen Rahmen für die genutzten Datenstrukturen vor. Konkrete Anwendungs-Instanzen „realisieren“ letztlich diese Funktionalität und verfeinern die vordefinierten Datenstrukturen, wo dies erforderlich ist. Dies ist in Abschnitt 3.4.3 nachfolgend weiter detailliert.

Einzelne Details von Anwendungstypen können anhand ihrer Bezeichner/Referenz bzw. auch anhand der `semanticId` direkt angesprochen werden. Die Liste der Anwendungs-Typen ist jedoch nicht vordefiniert und kann beliebig gestaltet sein. Daher ist es auch erforderlich, die vorhandenen Anwendungs-Typen aufzulisten, um eine Übersicht zu erhalten bzw. um die weiteren Details zu erhalten. Für diese Abfrage wird ein zentraler Einstiegspunkt definiert, im Sinne einer Anwendungshierarchie ein Einstiegspunkt für alle Anwendungen. Ausgehend von diesem Root-Element können alle Anwendungs-Typen, deren Teilmodelle und auch Teilmodell-Elemente durchsucht werden.

© Projektkonsortium i-Twin (2024)

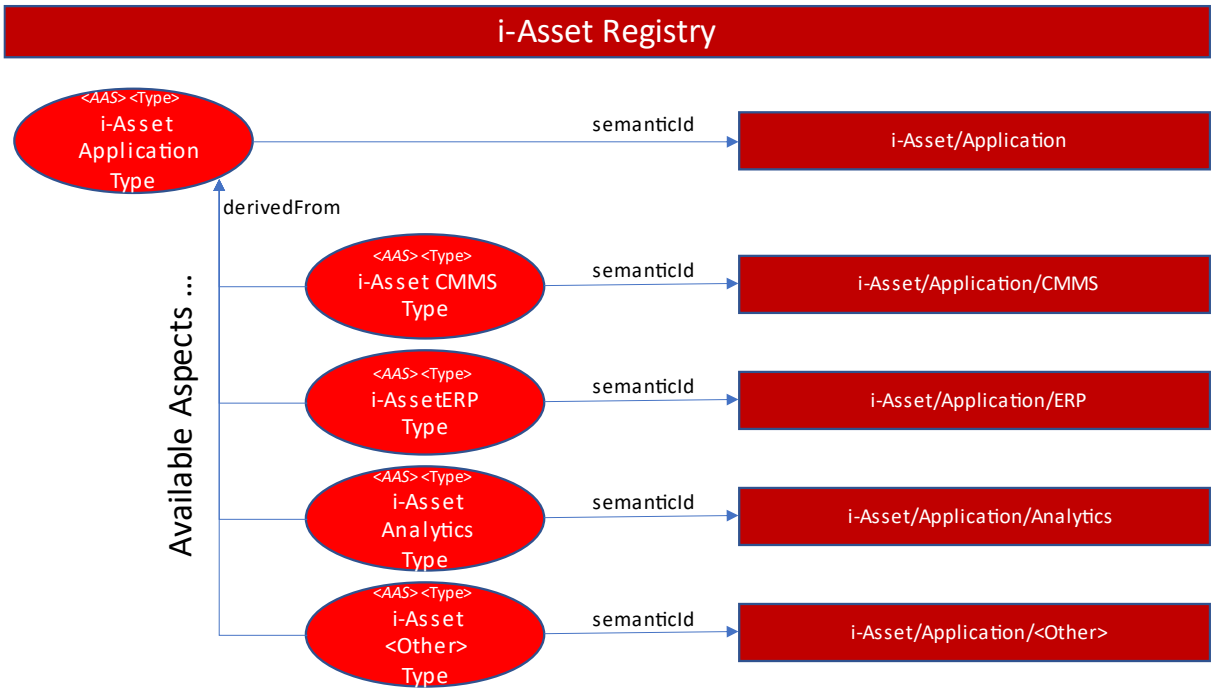


Abbildung 24: Anwendungs-Typen (derivedFrom)

Abbildung 24 zeigt die Definition eines zentralen Anwendungs-Typen. Spezifische Anwendungs-Typen nutzen das Attribut `derivedFrom` (siehe Abbildung 7, Seite 13) um sich als Anwendungs-Typ zu deklarieren. Durch diese Verbindung erhalten Anwendungs-Typen mittels Vererbung die generelle Struktur für Anwendungstypen und deren semantischen Informationen. In weiterer Folge fügen diese letztlich ihre Teilmodelle, Operationen und Event-Einstellungen hinzu, wie dies in Abbildung 25 vereinfacht dargestellt ist.

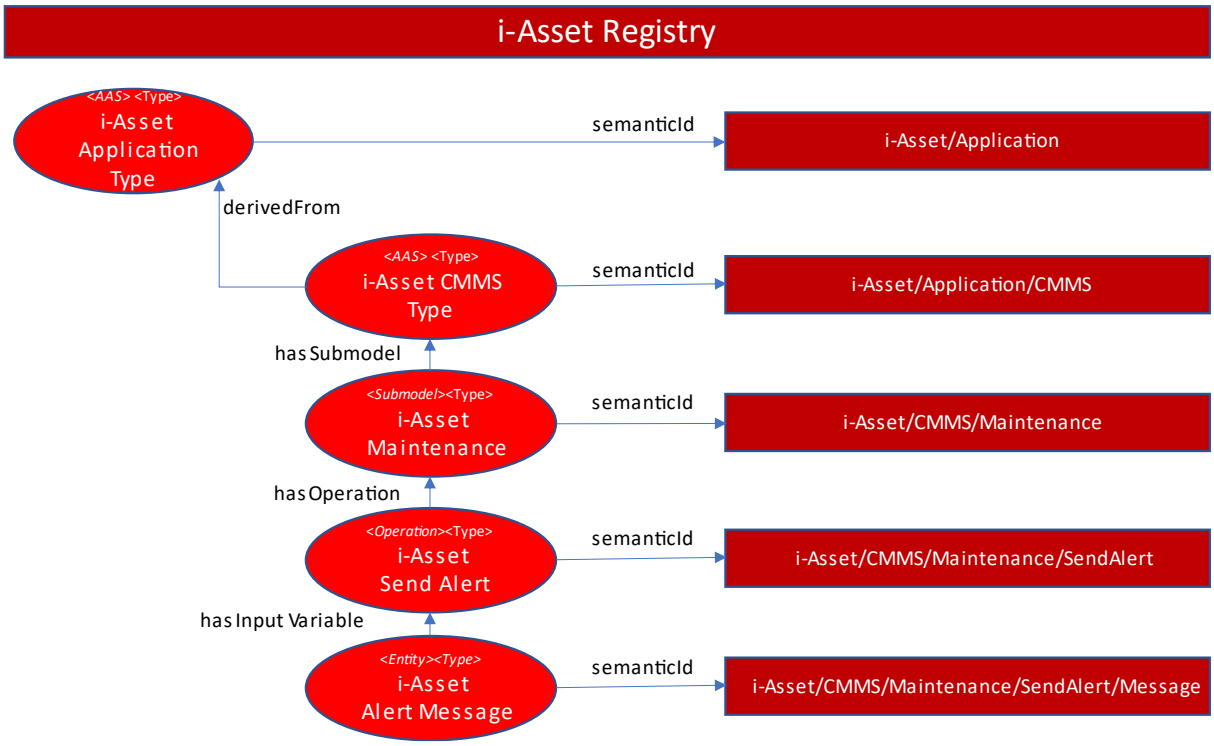


Abbildung 25: Anwendungs-Typ, Teilmodelle und Teilmodell-Elemente

© Projektkonsortium i-Twin (2024)

Sinngemäß wird in Abbildung 25 der funktionale Aspekt „CMMS“ definiert, dieser ist mit dem semantischen Bezeichner „i-Asset/Application/CMMS“ eindeutig definiert. In weiterer Folge finden sich in der Definition ein (oder mehrere) Teilmodell(e) welche die relevanten Teilmodell-Elemente für Event- bzw. Operations-Einstellungen enthalten. Die Teilmodell-Elemente (`Event`, `Operation`) repräsentieren jene Kommunikationsfähigkeiten, die im Anwendungs-Typ vorgesehen sind und letztlich von konkreten Anwendungs-Instanzen bereitgestellt werden.

Abbildung 25 zeigt auch die notwendigen Zugriffe im Asset Repository um die für die Kommunikation relevanten Elemente zu identifizieren:

1. Anhand der Beziehung `derivedFrom` werden alle Asset Administration Shells identifiziert, die sich vom Root-Application-Type ableiten.
2. Für jeden Anwendungs-Typ werden alle konfigurierten Teilmodelle (`Submodel`) abgearbeitet.
3. Innerhalb der Teilmodelle werden Elemente vom Typ `Event`, `Operation` gesucht. Deren `semanticId` ist für die Anbindung von Anwendungs-Instanzen wesentlich.

Mit den Elementen für Asset Administration Shell, Teilmodelle und vor allem Teilmodell-Elemente zur Beschreibung der Kommunikation mit einer Anwendung ist schließlich ein funktionaler Aspekt gem. Abbildung 1 in der Plattform definiert. Diesen Typ-Elementen kommt in weiterer Folge eine besondere Bedeutung zu: Diese Elemente definieren jene `semanticId` die für diese Funktionalität im gesamten System eindeutig ist.

Konkrete Instanzen von Anwendungs-Typen und in Folge auch Teilmodellen weisen eine Beziehung zu ihren definierenden Typen auf und können letztlich anhand der `semanticId` des Typ-Elements identifiziert werden. Sinngemäß können alle im System existierenden (konkreten) Anwendungen anhand der vordefinierten/bekannten `semanticId` ihres Anwendungs-Typs gefunden werden. Abbildung 26 zeigt diesen Zugriffsweg in vereinfachter Form. Dabei werden zunächst alle Anwendungs-Typen ermittelt. Dies sind jene Asset Administration Shells, welche vom generischen Typ-Element für Anwendungen (Asset Administration Shell „i-Asset Application Type“) abgeleitet sind (Attribut: `derivedFrom`). Das Suchkriterium ist hierbei die `semanticId` des generischen Typ-Elements für Anwendungen.

Im Ergebnis dieser Anfrage finden alle „konfigurierten“ funktionalen Aspekte. Diese funktionalen Aspekte beschreiben die Kommunikation in abstrakter Form. Sinngemäß müssen diese funktionalen Aspekte mit Hilfe von konkreten Anwendungen realisiert werden. Diese konkreten Anwendungen für die funktionalen Aspekte wiederum können mit Hilfe der vordefinierten `semanticId` (z.B. für CMMS Systeme, Dashboards, ERP-Systeme) gefunden werden. Ist keine konkrete Anwendung im System vorhanden, so ist der funktionale Aspekt im System nicht nutzbar. Dies gilt auch für einzelne Detail-Aspekte. Wenn eine konkrete Instanz ein Teilmodell bzw. eine Operation nicht „instanziiert“, so ist diese Funktion im System nicht verfügbar.

Die Instanz-Elemente von `Submodel`, `Operation` bzw. `Event` aktivieren letztlich eine Funktion und sorgen auch für eine individuelle Ausprägung der Kommunikationseinstellungen.

© Projektkonsortium i-Twin (2024)

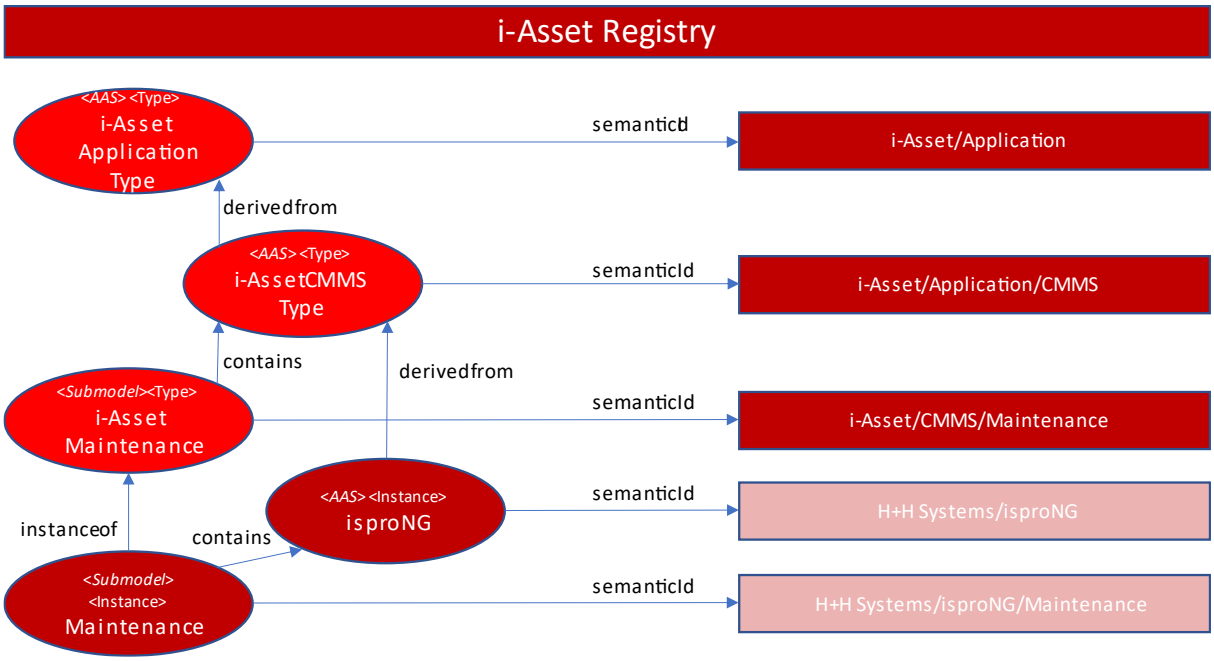


Abbildung 26: Anwendungs-Typen und Anwendungs-Instanzen

Auf diese Weise definieren Anwendungs-Typen in einer generischen Form jene Methoden und Ereignisse, die für die Anwendungsgruppe wesentlich ist, ohne zunächst alle konkreten Ausprägungen zu kennen. Konkrete Anwendungs-Instanzen realisieren diese Funktionen und stellen diese unter der `semanticId` des Anwendungs-Typen bereit. Die i-Asset Plattform kann letztlich feststellen ob einzelne Funktionen tatsächlich verfügbar sind oder nicht.

3.4.3 Anwendungs-Instanzen

In Abschnitt 3.4.2 wurden Anwendungs-Typen vorgestellt. Diese stellen eine abstrakte Vorlage für die konkreten Anwendung-Instanzen dar. Entsprechend den Ausführungen in Abschnitt 2.2 werden Anwendungs-Instanzen auf die gleiche Art und Weise in die Plattform integriert wie auch Asset-Instanzen. Jede Anwendung wird mit Hilfe eines Application Connector in die Plattform integriert wobei sich dieser in seiner Funktionalität nicht vom Asset Connector (siehe dazu auch Abschnitt 2.2.1, Abbildung 16) unterscheidet, sondern lediglich mit der konkreten Anwendung interagiert und so auf Basis einer instanziierten Asset Administration Shell deren Funktionen zur Verfügung stellt.

© Projektkonsortium i-Twin (2024)

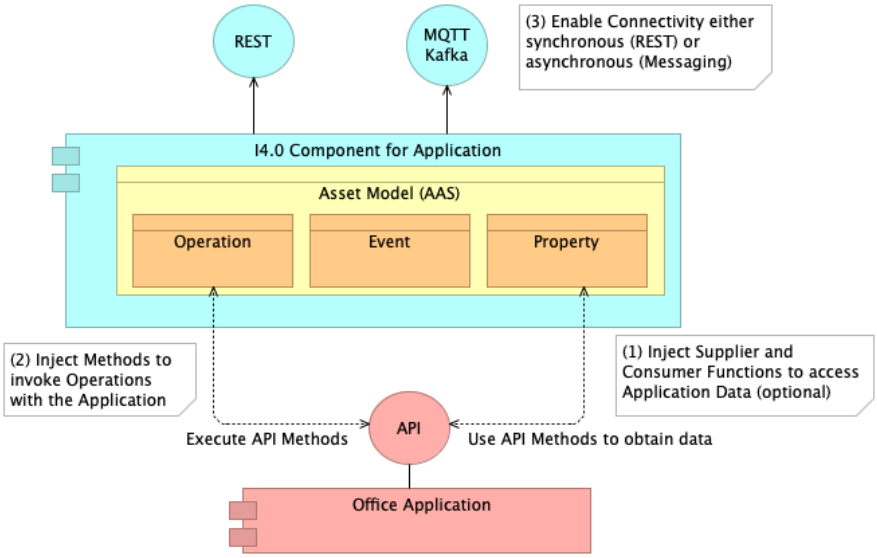


Abbildung 27: Application Connector

Die Aktivierung einer Anwendung im Application Connector wird zunächst die Asset Administration Shell des zugrundeliegenden Applikations-Typen instanziiert. Dabei werden alle Typ-Elemente des zugrundeliegenden Asset-Typs in Instanz-Elemente überführt wobei es aber zu individuellen Ergänzungen kommen kann:

- `Operation`-Element: Die `OperationVariable`-Einstellungen werden individualisiert, es können dabei zusätzliche Elemente hinzukommen bzw. entfallen. Auch mit Hilfe der `semanticId` der `OperationVariable` kann die erwartete/erzeugte Datenstruktur angepasst werden. Die `semanticId` des `Operation`-Elements zeigt auf das `Operation`-Element des Application-Typen.
- `Event`-Einstellungen: Das `observed`-Element kann mit Hilfe der semanticId angepasst werden. Die `semanticId` des `Event`-Elements zeigt auf das `Event`-Element des Application-Typen.

Das Ergebnis ist in Abbildung 28 dargestellt. Die referenzierten (externen) Datenstrukturen von Typ- und Instanz-Element können unterschiedlich sein. Die weitere Ausgestaltung, gegebenenfalls mittels Vererbung und individueller Attributzuordnung erfolgt im Semantic Lookup Repository.

© Projektkonsortium i-Twin (2024)

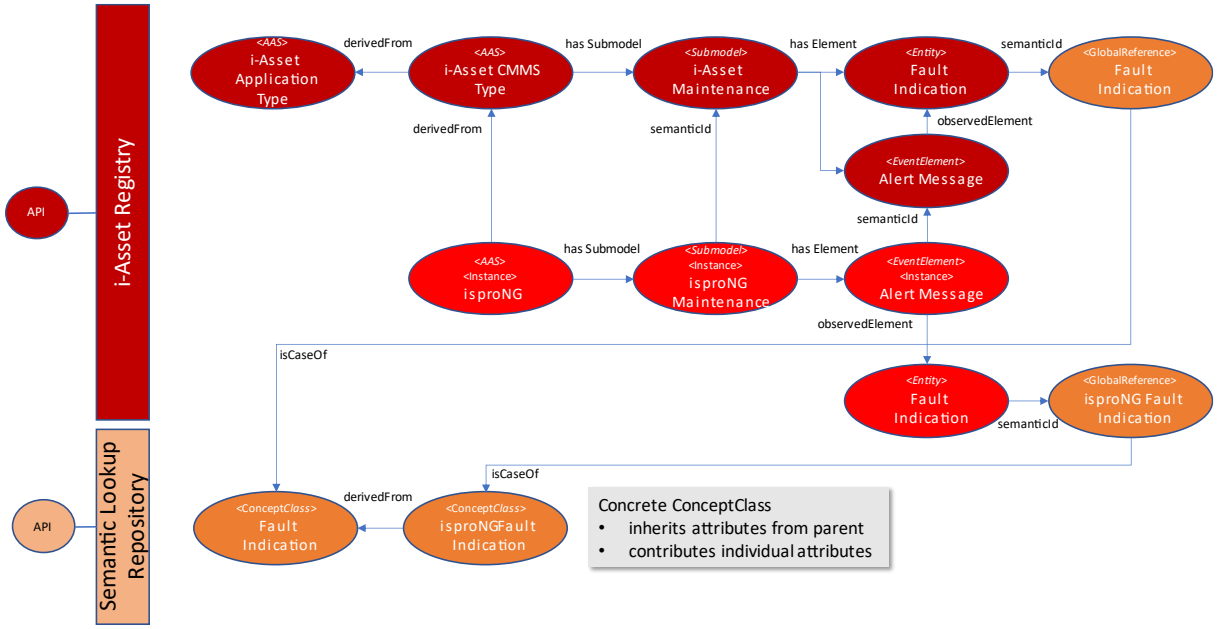


Abbildung 28: Anwendungs-Instanz und abgeleitete Datenstrukturen

Sobald die Asset Administration Shell der Anwendungs-Instanz vollständig vorliegt, erfolgt die Aktivierung der Anwendungs-Instanz im Application Connector. Dies geschieht auf die gleiche Art und Weise wie schon in Abschnitt 2.2.1 beschrieben. `Operation`-Elemente werden mit den Methoden verbunden die via Anwendungs-API zur Verfügung gestellt werden. `Event`-Elemente verbinden sich mit dem Message Broker und erhalten so die definierten Nachrichten vom System. `Property`-Elemente können ebenfalls mit variablen Informationen versehen werden.

Die Anwendung ist somit als I4.0 kompatibler Teilnehmer in der Plattform vorhanden und kann mittels synchroner (REST) und asynchroner (Messaging) Kommunikations-Patterns angesprochen werden. Die dabei ausgetauschten Daten sind vollständig beschrieben.

3.4.4 Exemplarische Darstellung am Beispiel von CMMS-Anwendungen

In den Abschnitten 0 und 3.4.3 wurden die Kommunikations-Mechanismen für Anwendungen 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.

Abrufen einer Wartungshistorie

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

© Projektkonsortium i-Twin (2024)

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 29 dargestellt.

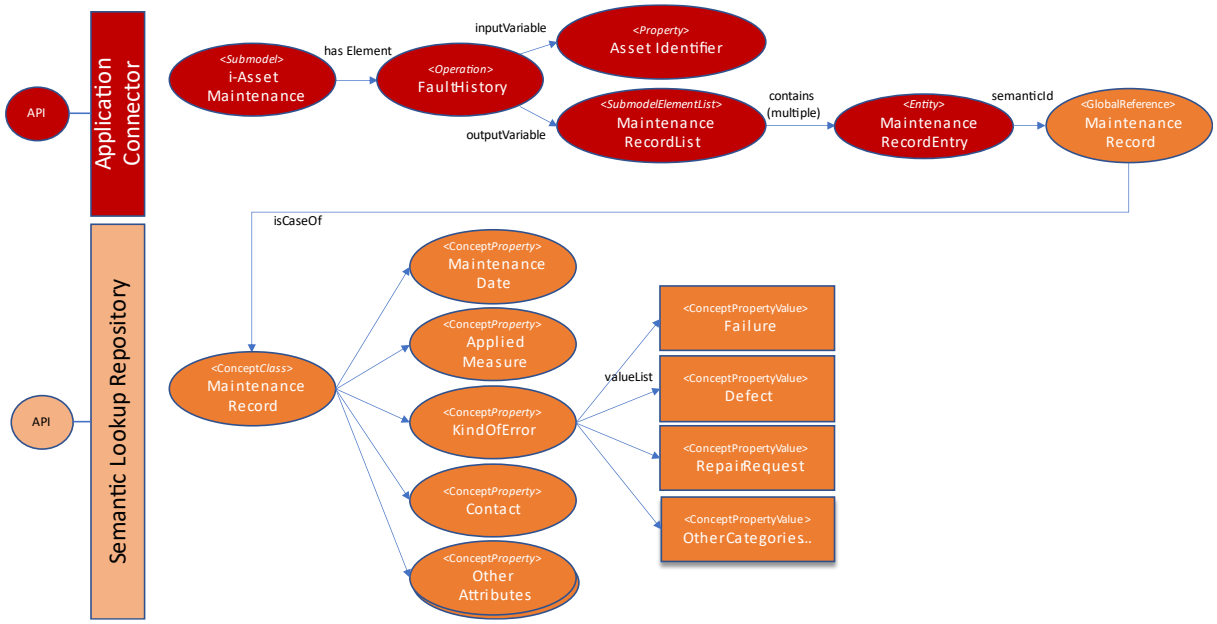


Abbildung 29: Anwendungs-Typ CMMS - Methoden

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 `Entity` 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

© Projektkonsortium i-Twin (2024)

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 Anwort 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.

Senden einer Störmeldung

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 30 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 30 dargestellt. Das `observed`-Element des `EventElement` ist letztlich für den Payload der Nachricht verantwortlich.

© Projektkonsortium i-Twin (2024)

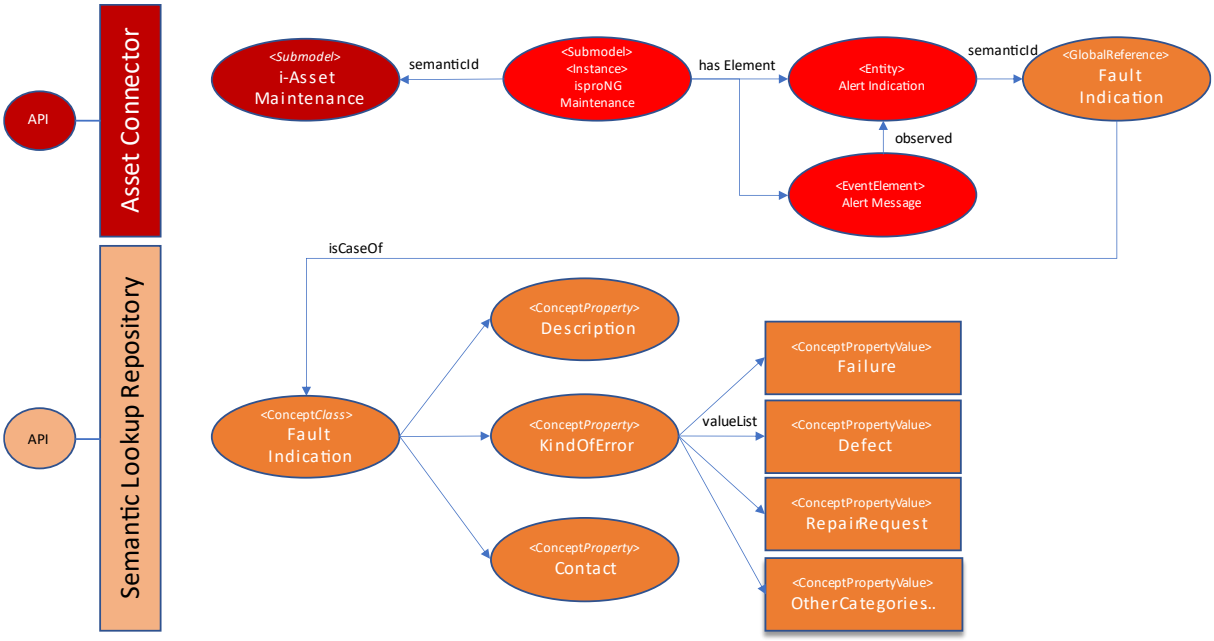


Abbildung 30: Asset-Instanz - Nachricht an CMMS erstellen

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.

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.

© Projektkonsortium i-Twin (2024)

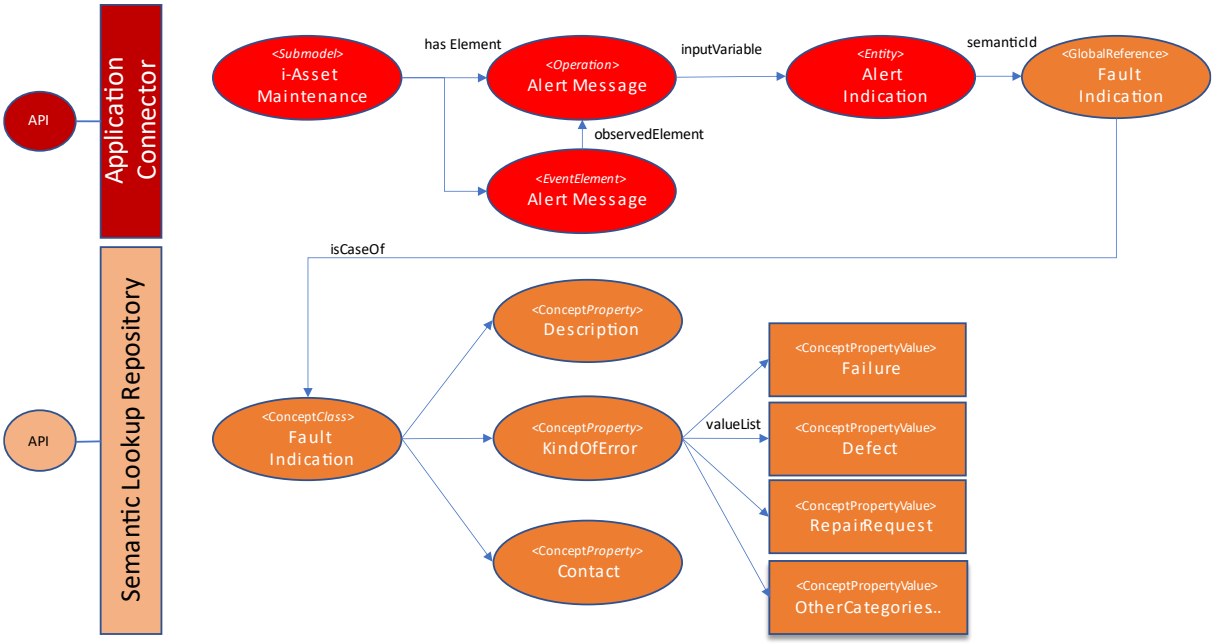


Abbildung 31: Anwendungs-Typ CMMS – Eingehende Nachrichten

Eingehende Nachrichten werden schließlich geprüft, ob diese auch die richtige `observable-SemanticId` 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 17 in Abschnitt 2.2.1) 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 2.2.1 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

© Projektkonsortium i-Twin (2024)

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 2.2.1) aufruft.

© Projektkonsortium i-Twin (2024)

4 Anwendungsszenarien

In diesem Abschnitt werden die Anwendungsszenarien die für die Nutzung der i-Twin Plattform maßgeblich sind, detailliert beschrieben. Diese Szenarien definieren auch wesentliche Anforderungen für die Services der i-Twin Plattform.

Für die Beschreibung der Use Cases und der Requirements wurde Templates bereitgestellt, welche in Abschnitt 4.6 verfügbar sind.

4.1 Basis-Szenarien für Plattform-Nutzung

4.1.1 Asset-Typ-Information verwalten

Die i-Twin Plattform stellt Werkzeuge und Methoden bereit um Asset-Typen strukturiert in der Asset-Registry ablegen zu können. Eine entsprechende Benutzeroberfläche erlaubt das manuelle Erfassen und Bearbeiten von Asset-Typen. Die Integration von Klassifikationssystemen wie eCl@ss bzw. CDD wird durch das Semantic Lookup Service gewährleistet. Die im Semantic Lookup Service definierten Klassifikations-Klassen bringen dabei die erforderlichen Attribute mit die bei der manuellen Anlage des Asset-Typen ausgefüllt werden müssen.

Ein registrierter Benutzer erhält die Möglichkeit einen neuen Asset-Typen zu definieren. Per Menüpunkt bzw. einem Button wird die Neuanlage initiiert. In einem ersten Schritt werden die Basis-Informationen für den Asset-Typen (ID, Bezeichnung) festgelegt. Die auszufüllenden Felder werden zunächst durch die Datenfelder der Verwaltungsschale definiert. Zum Typen können nun verschiedene Teilmodelle hinzugefügt werden wobei für jedes Teilmodell die entsprechenden Informationen eingetragen werden. Wird ein Teilmodell mit einer semantischen Klassifikation verlinkt, so werden die dort verlinkten Eigenschaften als Datenfelder aufbereitet und die entsprechenden Informationen können eingegeben werden. Zusätzlich besteht die Möglichkeit eigene Eigenschaften zum Asset-Typen zu definieren und im Teilmodell zu organisieren.

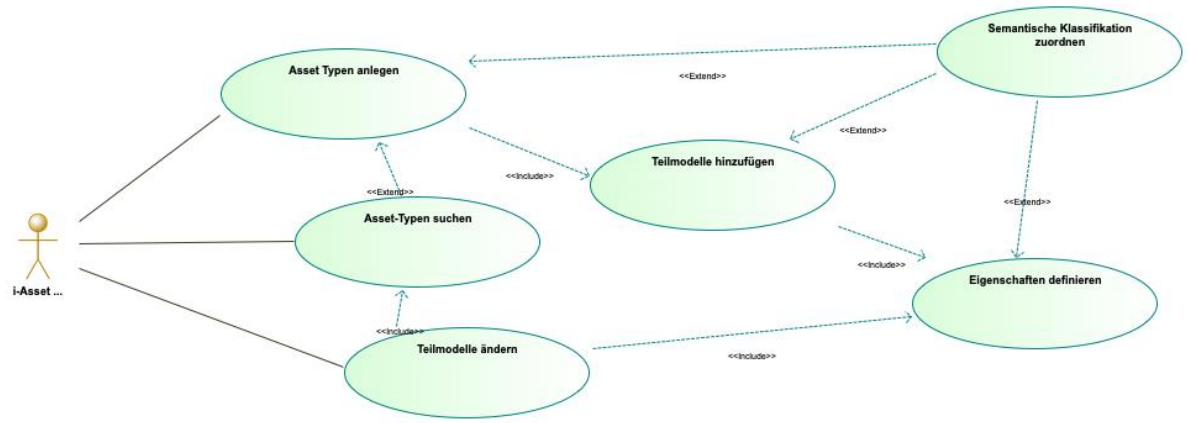


Abbildung 32: Diagramm UC 4.1.1 - Asset-Typ-Information verwalten

© Projektkonsortium i-Twin (2024)

Use Case ID:	UC4.1.1		
Use Case Name:	Asset Typen Verwalten		
Created By:	Dietmar Glachs	Last Updated By:	
Date Created:	15.06.2020	Date Last Updated:	

Actors:	i-Twin User
Description:	Verwalten von Asset-Typen mit der i-Twin Plattform.
Trigger:	Ein Asset-Typ wird als Grundlage für die Instanziierung eines Assets benötigt.
Pre-Conditions:	1. Benutzer ist in der i-Twin Plattform registriert und angemeldet 2. Semantischer Lookup ist integriert
Post-Conditions:	1. Asset-Typ ist vollständig in der Asset-Registry abgelegt 2. Die Suche nach dem Asset anhand von Beschreibung, Eigenschaften bzw. Teilmodellen ist möglich.
Normal Flow:	1. Benutzer wählt „Neuen Asset-Typen anlegen“ 2. Formular mit „Basis-Daten“ für Assets (ID, Administrative Daten, Beschreibung) wird angezeigt, die Daten ausgefüllt und gespeichert. 3. Benutzer wählt Teilmodell hinzufügen 4. Formular mit Grunddaten für Teilmodell (ID, Beschreibung, Administrative Daten, Semantische Klassifikation) wird angezeigt. a. Ein Semantic Lookup „klassifiziert“ das Teilmodell. b. Eine Klassifikation kann auch „Eigenschaften“ vorgeben, dadurch wird der Schritt 5 vordefiniert. 5. Benutzer wählt „Neue Eigenschaft“ wobei unterschiedliche Element-Typen (siehe „Verwaltungsschale im Detail“)[24] zur Auswahl stehen. Abhängig vom ausgewählten Element-Typ werden die entsprechenden Formular-Felder angezeigt (z.B. Datentyp für Property-Elemente) und können eingegeben werden. a. Ein Semantic Lookup „klassifiziert“ die Eigenschaft. Dabei werden Datentyp, Einheit und ggf. auch erlaubte Werte übernommen.
Alternative Flows:	Bei der Anlage von Teilmodellen kann eine semantische Klassifizierung vorgenommen werden. Die in der Klassifikations-Klasse

[24] https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.html

© Projektkonsortium i-Twin (2024)

	zugewiesenen Eigenschaften werden als Basis für den Schritt 5 herangezogen.
Exceptions:	
Includes:	Semantic Lookup – Auswahl einer Klassifikations-Klasse bzw. einer Eigenschaft.
Priority:	Hoch
Frequency of Use:	
Business Rules:	Asset-Instanzen basieren auf Asset-Typen und erben deren Struktur und Eigenschaften.
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.1.2 Verwalten von Asset-Instanzen

Die i-Twin Plattform stellt Werkzeuge und Methoden bereit um Asset-Instanzen auf Basis von Asset-Typen instanziieren zu können. Die Basis-Struktur des Asset-Typen wird dabei übernommen und individuelle Attribute können ergänzt werden.

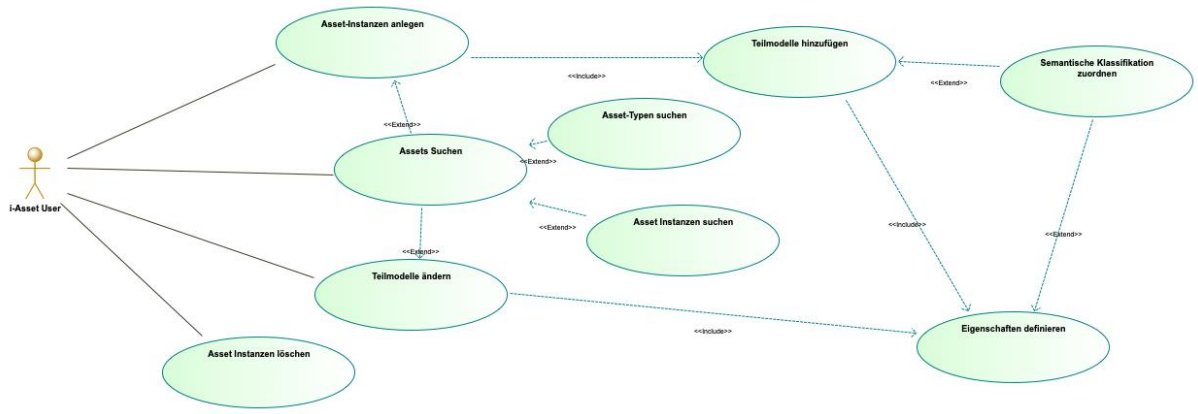


Abbildung 33: Diagramm UC 4.1.2 - Verwalten von Asset-Instanzen

Use Case ID:	UC4.1.2		
Use Case Name:	Asset-Instanzen verwalten		
Created By:	Felix Strohmeier	Last Updated By:	
Date Created:	17.07.2020	Date Last Updated:	

Actors:	i-Twin User

© Projektkonsortium i-Twin (2024)

Description:	i-Twin Instanzen sind die konkreten Ausprägungen der i-Twin Typen. Dieser Use Case ermöglicht das Anlegen, Lesen, Ändern und Löschen (CRUD) von Assets in der i-Twin Plattform
Trigger:	Asset wird in Betrieb genommen und soll an die Plattform angebunden werden
Pre-Conditions:	1. Benutzer ist in der i-Twin Plattform registriert und angemeldet 2. Semantischer Lookup ist integriert 3. Der Asset-Typ muss bereits in der i-Twin Registry vorhanden sein.
Post-Conditions:	1. Asset-Instanz ist vollständig in der Asset-Registry abgelegt 2. Die Suche nach der Instanz anhand von Typ-Eigenschaften (Größe, Form, etc.) bzw. Instanz-Eigenschaften (Standort, etc.) ist möglich
Normal Flow:	1. Benutzer wählt „Neue Asset-Instanz vom Typ X anlegen“ 2. Formular mit Instanz-Eigenschaften des ausgewählten Typs (Standort, Betreiber, Eigentümer, etc.) wird angezeigt und kann ausgefüllt und gespeichert werden.
Alternative Flows:	1. Benutzer wählt „Instanz bearbeiten“ 2. Formular der bestehenden Instanz wird angezeigt und kann geändert und gespeichert werden.
Exceptions:	
Includes:	
Priority:	Hoch
Frequency of Use:	1x pro Asset bzw. -änderung
Business Rules:	Asset-Instanz kann gegen den entsprechenden Asset-Typ auf Vollständigkeit verifiziert werden.
Special Requirements:	
Assumptions:	
Notes and Issues:	

© Projektkonsortium i-Twin (2024)

4.1.3 Suche von Assets (Typen, Instanzen)

Die i-Twin Plattform bietet eine Suche nach Assets. Über Hersteller, Betreiber, Eigenschaften etc.

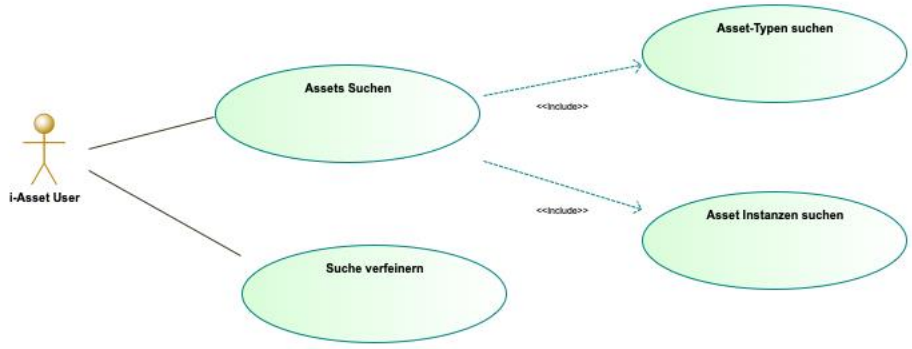


Abbildung 34: Diagramm UC 4.1.3 - Suche von Assets (Typen, Instanzen)

Use Case ID:	UC4.1.3		
Use Case Name:	Assets in der Registry suchen		
Created By:	Felix Strohmeier	Last Updated By:	
Date Created:	17.07.2020	Date Last Updated:	

Actors:	i-Twin User
Description:	Die i-Twin Registry bietet die Möglichkeit Suchanfragen auf Asset-Basis zu beantworten. Der Index wird während der „semantischen Klassifikation“ (UC3.1.1) entsprechend befüllt.
Trigger:	User möchte statische oder dynamische Informationen über die i-Twin Instanz abfragen.
Pre-Conditions:	Registrierte Assets sind im semantischen Suchindex vorhanden.
Post-Conditions:	Suchergebnis ist vollständig und bildet den aktuellen Asset-Bestand wieder.
Normal Flow:	1. Benutzer wählt über ein „Lupen“-Icon die Einfache Suche aus und gibt einen Suchbegriff ein 2. Das System ermittelt das Suchergebnis auf Basis des Suchbegriffs und blendet die passendsten Treffer ein. Bei einer großen Ergebnismenge wird nur die erste Seite mit der Option weiterzublättern eingeblendet. Zudem wird für (alle/ausgewählte) Eigenschaften der Assets eine Facettierte Suche angeboten. 3. Der Benutzer kann a. den Suchbegriff verändern b. im Suchergebnis seitenweise navigieren c. anhand der angebotenen Facetten das Ergebnis weiter eingrenzen/verfeinern

© Projektkonsortium i-Twin (2024)

	4. Benutzer wählt einen Vorschlag aus oder drückt „Enter“
Alternative Flows:	1. Erweiterte Suche ermöglicht die gezielte Suche nach speziellen Asset-Typ-Eigenschaften.
Exceptions:	
Includes:	
Priority:	Hoch
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.2 Semantic Integration Patterns

4.2.1 Applikation in i-Twin einrichten

Innerhalb der i-Twin Plattform werden (Meta-)Daten zu Assets verwaltet. Diese Daten sollen angeschlossenen Applikationen zur Verfügung stehen und mit diesen austauschen. Zugleich versteht sich die i-Twin Plattform als Datendrehscheibe und stellt eine Messaging-Infrastruktur bereit um Nachrichten zwischen

1. Assets und i-Twin Plattform
2. i-Twin Plattform und Anwendung

austauschen zu können. Anwendungen müssen sich zur Teilnahme an der i-Twin Plattform registrieren und die Kommunikations-Anforderungen der i-Twin Plattform erfüllen. Dazu werden im Projekt Semantic Integration Patterns (vgl. Abschnitt 3.4 auf Seite 53) entwickelt, die grundlegende Funktionen der Anwendungen und die von ihnen bereitgestellten und verarbeiteten Informationen semantisch beschreiben.

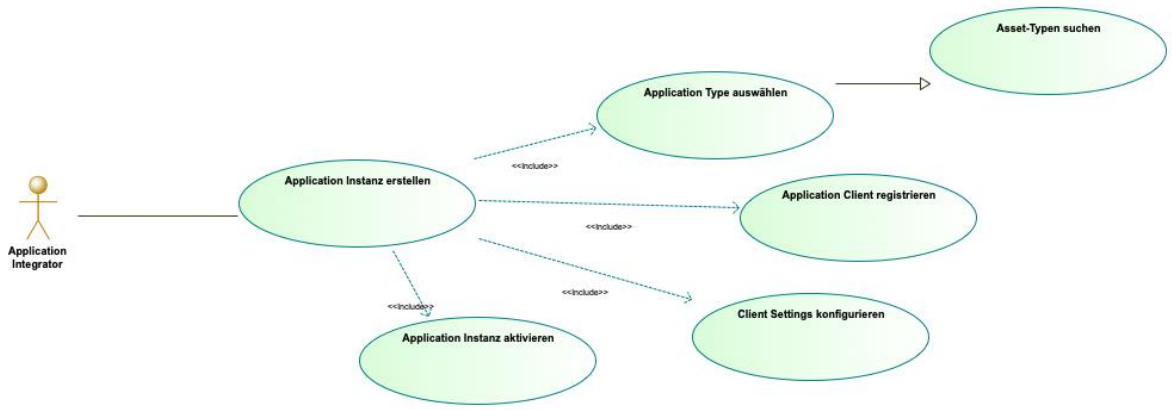


Abbildung 35: Diagramm UC 4.2.1 - Applikation in i-Twin einrichten

© Projektkonsortium i-Twin (2024)

Use Case ID:	UC4.2.1		
Use Case Name:	Applikationen in i-Twin einrichten		
Created By:	Dietmar Glachs	Last Updated By:	
Date Created:	17.07.2020	Date Last Updated:	

Actors:	Application Integrator
Description:	Applikationen werden für die Teilnahme an der i-Twin Plattform registriert, eingerichtet und aktiviert.
Trigger:	Applikationen möchten am i-Twin System teilhaben, Nachrichten senden/empfangen.
Pre-Conditions:	1. Typ für Applikations-Verwaltungsschale ist in i-Twin hinterlegt 2. Benutzer besitzt die Berechtigung zur Anlage von Anwendungen
Post-Conditions:	1. Anwendung ist mit i-Twin verbunden 2. Nachrichten werden zugestellt 3. Direkte Anfragen an die Anwendung werden von dieser beantwortet.
Normal Flow:	1. Applikations-Integrator bereitet den funktionalen Adapter für das einzubindende System vor und nutzt dazu die i-Twin Client Connectivity. 2. Applikations-Integrator sucht nach vorhandenen Applikations-Verwaltungsschalen (CMMS, Analytics, IoT-Plattform u.a) und wählt den passenden Applikations-Typ aus. 3. System übernimmt die Voreinstellungen des Applikations-Typen 4. Applikations-Integrator adaptiert die Voreinstellungen 5. System erstellt einen „Client“ in den Security-Settings und erhält so das „Client-Secret“ & „Client-Id“. 6. Benutzer ergänzt die Security Einstellungen mit dem Client-Secret & Client-Id 7. Benutzer aktiviert die i-Twin Client Connectivity 8. Die Client Connectivity subskribiert sich auf konfigurierte Topics
Alternative Flows:	2. Erweiterte Suche ermöglicht die gezielte Suche nach speziellen Asset-Typ-Eigenschaften.
Exceptions:	
Includes:	
Priority:	Hoch
Frequency of Use:	
Business Rules:	

© Projektkonsortium i-Twin (2024)

Special Requirements:	
Assumptions:	
Notes and Issues:	

4.3 Use Cases IcoSense

4.3.1 Upgrade Brown-Field-Asset zur I4.0 Komponente

Erweiterung einer bestehenden Anlage zu einer I4.0 Komponente.

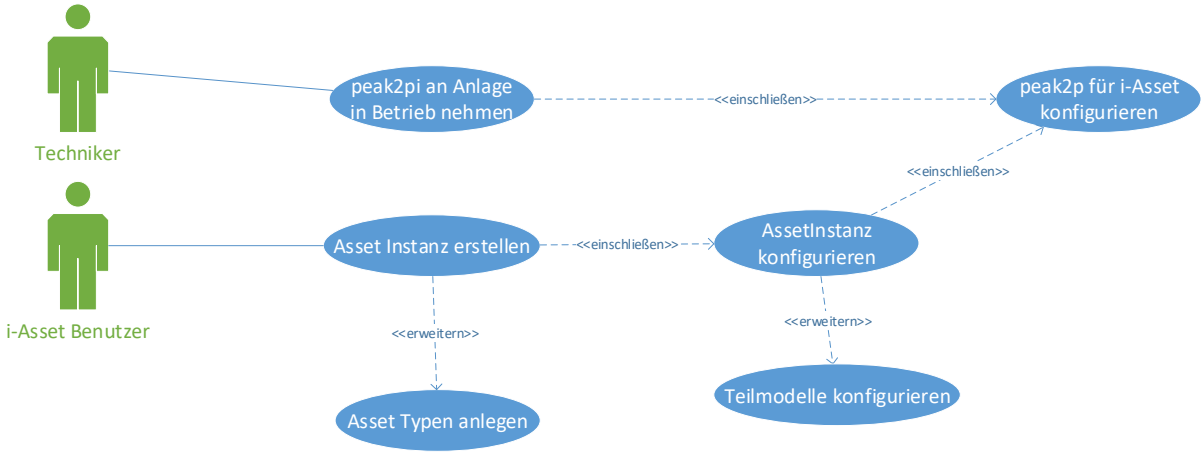


Abbildung 36: Diagramm UC 4.3.1 - Upgrade Brown-Field-Asset zur I4.0 Komponente

Use Case ID:	UC4.3.1		
Use Case Name:	Upgrade Brown-Field zur I4.0 Komponente		
Created By:	Martin Brugger	Last Updated By:	
Date Created:	24.06.2020	Date Last Updated:	

Actors:	i-Twin User, peak2pi User
Description:	Erweiterung einer bestehenden Anlage zu einer I4.0 Komponente
Trigger:	Eine bestehende Anlage soll in eine I4.0 Komponente upgegraded werden
Pre-Conditions:	1. Notwendige Datenpunkte sind verfügbar 2. Benutzer ist in der i-Twin Plattform registriert und angemeldet 3. Asset-Typ ist in der Asset-Registry abgelegt
Post-Conditions:	Die anlage ist ine I4.0 Komponente
Normal Flow:	1. Benutzer nimmt peak2pi an Anlage in Betrieb

© Projektkonsortium i-Twin (2024)

	2. Benutzer erzeugt neue Asset Instanz in i-Twin Plattform 3. Benutzer konfiguriert die notwendigen Teilmodelle (OEE, Sensordaten, Störungen) 4. Benutzer konfiguriert peak2pi für die Kommunikation mit der i-Twin Plattform
Alternative Flows:	Falls noch kein Typ in der Plattform vorhanden ist muss dieser angelegt werden.
Exceptions:	
Includes:	
Priority:	Hoch
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Requirements

Who			
Als Anlagenbetreiber will ich mittels Peak2Pi meine Anlage als Industrie Komponente 4.0 Komponente einbinden			
What			
Konfiguration der Verbindungsdaten an Peak2Pi und i-Twin Konfiguration der Module/Submodule welche verwendet werden sollen. Die Konfiguration soll über eine Oberfläche oder Konfigurationsdatei erfolgen			
Why			
Die Korrekte Konfiguration ist notwendig damit der Datenaustausch mit dem System funktionieren kann			

Acceptance criteria			

© Projektkonsortium i-Twin (2024)

1. Verbindungsdaten konfiguriert 2. Module eingestellt 3. Eine Verbindung zu i-Twin pro Edge-Node 4. Der Verbindungstatus wird dem Nutzer angezeigt

4.3.2 OEE Werte übermitteln

Das Device ermittelt die wesentlichen KPI Kennzahlen und sendet die Sensor-Nachricht über die Event-Mechanismen der i-Twin Plattform.

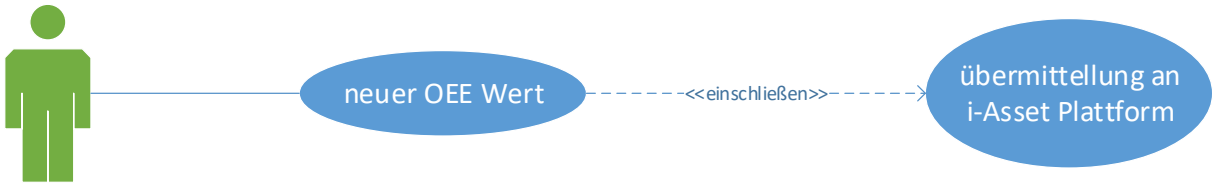


Abbildung 37: Diagramm UC 4.3.2 - OEE Werte übermitteln

Use Case ID:	UC4.3.2		
Use Case Name:	OEE Werte übermitteln		
Created By:	Martin Brugger	Last Updated By:	
Date Created:	24.06.2020	Date Last Updated:	

Actors:	peak2pi
Description:	OEE Werte von peak2pi an i-Twin Plattform übermitteln
Trigger:	OEE Wert Änderung
Pre-Conditions:	1. peak2pi mit Anlage verbunden 2. peak2pi für i-Twin konfiguriert 3. Instanz in i-Twin Plattform erstellt 4. Datastreaming Channel vorhanden
Post-Conditions:	OEE Daten werden an Datastream Channel übermittelt
Normal Flow:	1. Neuer OEE Wert wurde berechnet 2. Übermittelung der Daten an einen Datastream der i-Twin Plattform
Alternative Flows:	
Exceptions:	

© Projektkonsortium i-Twin (2024)

Includes:	
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Requirements

Who			
Als Peak 2 Pi will ich das die OEE Werte jede Minute an i-Twin übertragen werden.			
What			
Übertragung der OEE Information in einem Zyklus von einer Minute			
Why			
Der Digital Twin soll immer über den aktuellsten Status der OEE Verfügen			

Acceptance criteria			

4.3.3 Störmeldung absetzen

Funktioniert ein Device nicht, werden keine Daten mehr übertragen, so soll eine vollständige Störmeldung erstellt werden können. Dazu muss das Device (I4.0 Komponente bzw. Peak2PI)

© Projektkonsortium i-Twin (2024)

die entsprechenden Stör-Codes der Maschine anhand einer Störursachen-Tabelle auflösen können und schließlich eine Störmeldung formulieren & absenden wie sie für die Anlage vorgesehen ist. Dazu sind weitere Aktionen erforderlich:

- Störmeldung formulieren: Die Asset Registry muss in den Event-Einstellungen zum jeweiligen Asset das zu beschickende Messaging Topic sowie auch die Information wie die Störmeldung aufgebaut ist, bereithalten. Das Device erstellt auf dieser Basis die Störmeldung und löst den Event aus. Teil einer Störmeldung ist die Stör-Ursache. Diese ist zumeist vordefiniert und erlaubt nur vordefinierte Werte.
- Stör-Ursachentabelle abholen: In der Asset Registry muss die für die Anlage gültige Stör-Ursachentabelle abgeholt werden. Dabei durchsucht die Asset Registry die Typ-Informationen des Assets gibt diese Stör-Ursachen zurück.

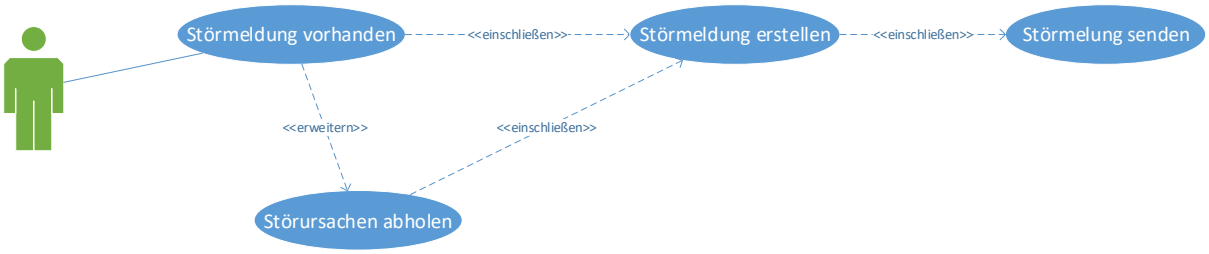


Abbildung 38: Diagramm UC 4.3.3 - Störmeldung absetzen

Use Case ID:	UC 4.3.3		
Use Case Name:	Störmeldung absetzen		
Created By:	Martin Brugger	Last Updated By:	
Date Created:	24.06.2020	Date Last Updated:	

Actors:	Peak2pi
Description:	Absetzen einer Störmeldung an i-Twin
Trigger:	Störmeldung vorhanden
Pre-Conditions:	1. Asset Instanz in I-Twin abgelegt 2. Peak2pi konfiguriert
Post-Conditions:	Störmeldung an i-Twin gemeldet
Normal Flow:	1. Störmeldung vorhanden 2. Störmeldung in peak2pi erstellen 3. Störmeldung an i-Twin übermitteln
Alternative Flows:	Sofern keine Störursachen vorhanden oder diese veraltet sind, müssen diese von i-Twin abgeholt werden
Exceptions:	
Includes:	

© Projektkonsortium i-Twin (2024)

Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Requirements

Who			
Als Benutzer will ich eine Rückmeldung ob die Störmeldung im CMMS System angekommen ist.			
What			
Wenn eine Störmeldung an i-Twin abgesetzt wurde soll eine Rückmeldung vom CMMS System Erfolgen sobald diese Entgegengenommen wurde			
Why			
Damit der Benutzer über den Empfang einer Meldung informiert wird			

Acceptance criteria			
1. Wird eine Störmeldung abgegeben muss nach dem Empfang eine Bestätigung gesendet werden 2. Der Empfang muss dem Benutzer dargestellt werden.			

Als Peak2Pi will ich über eine Statusänderung der Störmeldung informiert werden			
What			

© Projektkonsortium i-Twin (2024)

Wenn sich der Status einer Störmeldung geändert hat soll Peak2Pi informiert werden			
Why			
Damit der aktuelle Status abgerufen und angezeigt werden kann			

Acceptance criteria			
1. Wird eine Störmeldung bearbeitet soll Peak2Pi eine Benachrichtigung erhalten 2. Nach dem Empfang kann Peak2Pi weiter Aktionen einleiten.			

4.3.4 Auftragsdaten abrufen

Oft ist es an der Anlage relevant dass Auftragsbezogene Daten zur Verfügung stehen und dargestellt werden. Ziel ist es dass diese Infos über das ERP/MES System durch i-Asset Plattform zur Verfügung gestellt werden können.

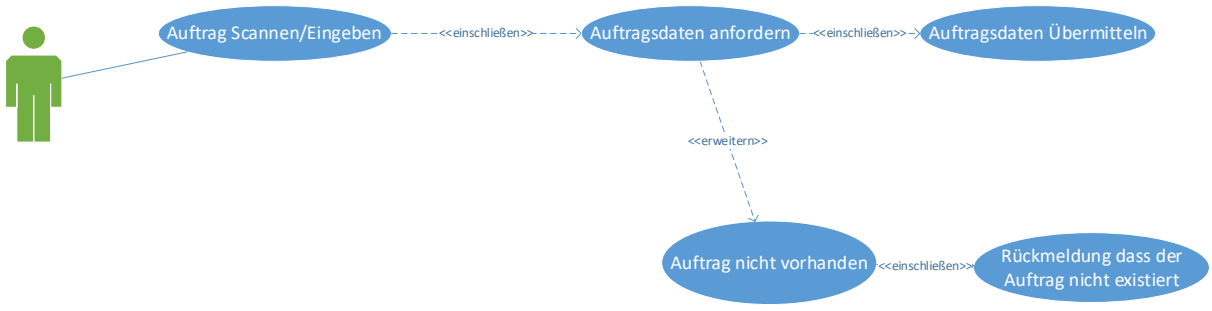


Abbildung 39: Diagramm UC 4.3.4 – Auftragsdaten prüfen

© Projektkonsortium i-Twin (2024)

Use Case ID:	UC 4.3.3		
Use Case Name:	Auftragsdaten abrufen		
Created By:	Martin Brugger	Last Updated By:	
Date Created:	14.03.2022	Date Last Updated:	

Actors:	Peak2pi
Description:	Abrufen von Auftragsdaten über i-Twin
Trigger:	Neuer Auftrag
Pre-Conditions:	1. Asset Instanz in I-Twin abgelegt 2. Peak2pi konfiguriert
Post-Conditions:	Auftrags Daten übermittelt
Normal Flow:	1. Auftragsnummer wird gescannt oder eingegeben 2. Auftragsdaten werden angefordert 3. Auftragsdaten werden übermittelt
Alternative Flows:	Handling wenn Auftrag nicht existiert
Exceptions:	
Includes:	
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.4 Use Cases isproNG

4.4.1 Wartungs- und Störmeldungshistorie

Definition eines Teilmodells für Wartungsinfo – Eigenschaften für Wartung bzw. für Störungsmeldungen.

© Projektkonsortium i-Twin (2024)

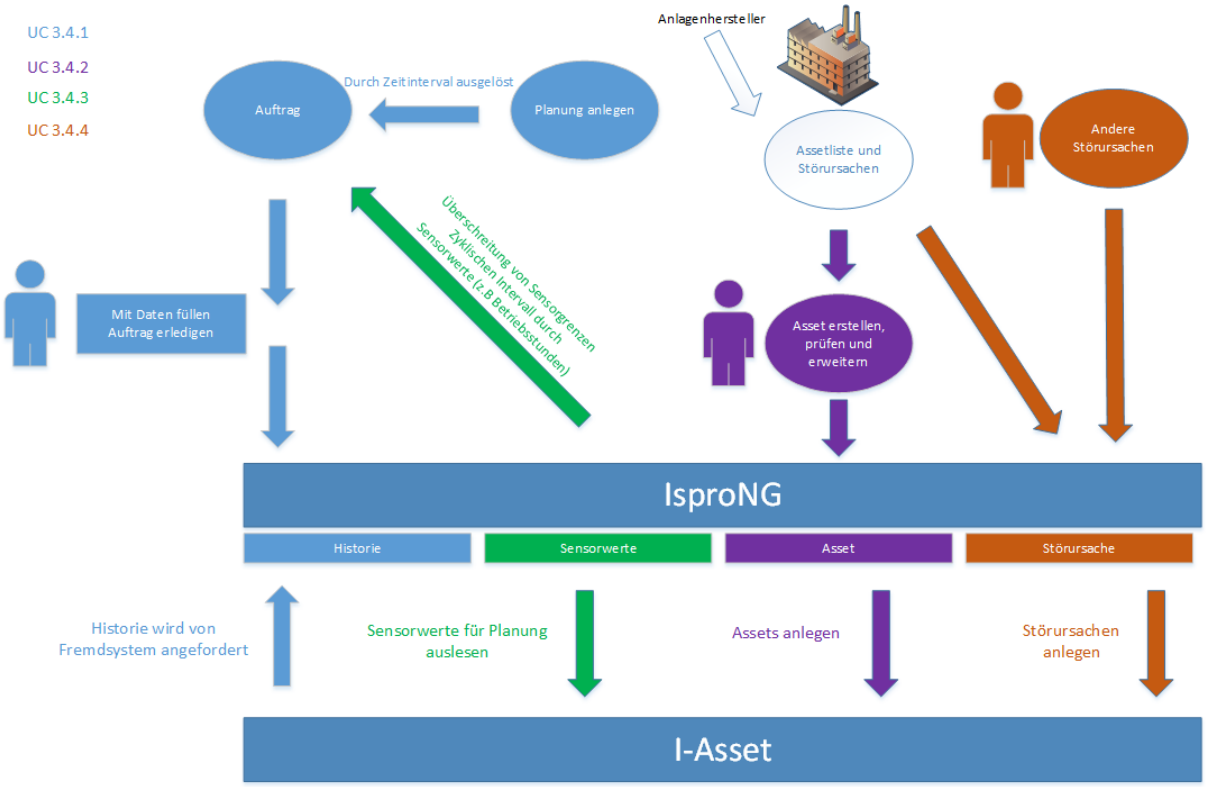


Abbildung 40: Diagramm UC 4.4.1 - Wartungs- und Störmeldungshistorie

Die Abbildung umfasst mehrere Use Cases. Die Aktoren und Komponenten des Use Cases 4.4.1 sind in hellblauer Farbe dargestellt.

Use Case ID:	UC4.4.1		
Use Case Name:	Wartungs und Störmeldungshistorie		
Created By:	Adelsmair	Last Updated By:	Adelsmair
Date Created:	25.06.2020	Date Last Updated:	25.06.2020

Actors:	IsproNG
Description:	Erfolgte Wartungen und Störmeldungen werden anderen Systemen zur Verfügung gestellt. Andere Systeme können dadurch ableiten, dass angeforderte Wartungen bereits erledigt sind bzw. Sensoren wieder zurückgesetzt werden.
Trigger:	Andere Systeme benötigen historische Informationen zu einem Asset
Pre-Conditions:	1. Asset ist in der i-Twin Registry abgelegt. 2. Das Teilmodell für Wartung ist im Asset hinterlegt 3. Der Service-Endpunkt des CMMS ist definiert
Post-Conditions:	Die erfolgten Wartungstätigkeiten sind in der Registry abrufbar
Normal Flow:	1. Wartung wird aufgrund einer Planung, eines Sensorwertes oder Meldung erstellt

© Projektkonsortium i-Twin (2024)

	2. Wartung mit entsprechenden Informationen aufbereitet: Arbeitszeit, Durchführungsdatum, Störursache, Ersatzteile 3. Wartung wird durchgeführt 4. Wartung steht über i-Twin zur Verfügung
Alternative Flows:	
Exceptions:	
Includes:	Asset, Störursache
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.4.2 Assets instanziieren

Verbindung einer Anwendung (CMMS) mit der Asset-Registry. Es werden konkrete Asset-Instanzen sowohl in der Registry als auch in CMMS abgelegt. Die dabei erforderlichen Attribute stammen aus dem Asset-Typ bzw. zusätzlich für Teilmodelle aus dem verlinkten Klassifikations-System (eCl@ss, CDD).

© Projektkonsortium i-Twin (2024)

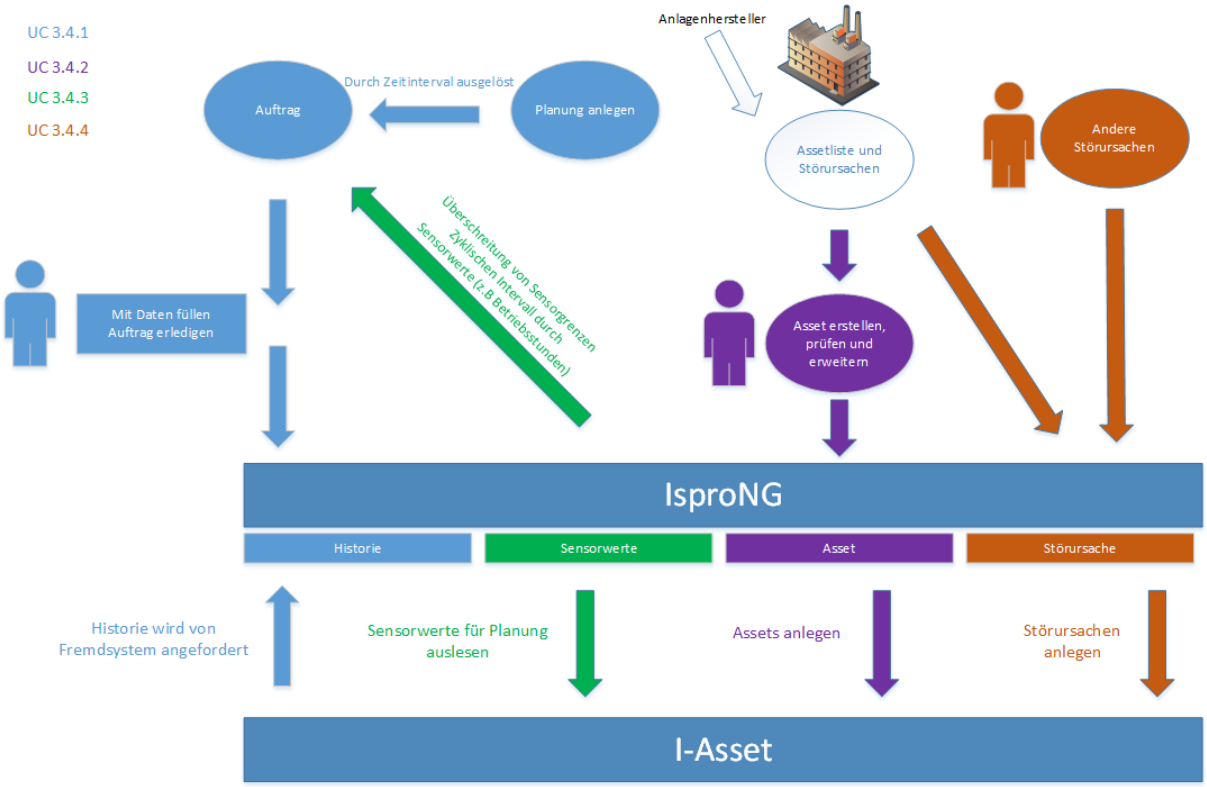


Abbildung 41: Diagramm UC 4.4.2 - Assets instanziieren

Die Abbildung umfasst mehrere Use Cases. Die Aktoren und Komponenten des Use Cases 4.4.2 sind in violetter Farbe dargestellt.

Use Case ID:	UC4.4.2		
Use Case Name:	Assets instanziieren (z.B. reale Instanz auf Basis eines Asset-Typs erstellen)		
Created By:	Adelsmair	Last Updated By:	Adelsmair
Date Created:	25.06.2020	Date Last Updated:	25.06.2020

Actors:	IsproNG
Description:	Hersteller bzw. Lieferant liefert Anlagenliste. Anlagenliste muss in spezifisches Format umgewandelt werden um im Ispro importiert zu werden. Danach wird der Import der Anlage durchgeführt. Es wird hier vom Benutzer eine Doublettenkontrolle durchgeführt. Nach Kontrolle Erweiterung der Daten (z.B genauer Standort) wird die Freigabe im IsproNG gemacht und die Anlage im I-Twin durchgeführt. Durch die Anlage wird eine eindeutige ID von I-Twin vergeben. Diese wird danach im IsproNG als Referenz gespeichert.
Trigger:	Prüfen und Freigabe durch den Benutzer
Pre-Conditions:	Zugriff auf I-Twin durch I-Twin-Benutzer (UC 3.1.1)

© Projektkonsortium i-Twin (2024)

Post-Conditions:	Assets stehen anderen Systemen zu Verfügung
Normal Flow:	1. Anlagenliste von Lieferant oder Hersteller anfordern 2. Liste in geeignetes Format aufbereiten 3. Import der Liste im IsproNG 4. Ergänzung der Daten (z.B. genauer Standort) 5. Freigabe der Assets 6. Übertragen der Assets an i-Twin 7. Abgleich der i-Twin-Identifier im IsproNG als Foreign Key
Alternative Flows:	1. Anlage direkt Im IsproNG 2. Freigabe des Assets 3. Übertragen des Assets an i-Twin 4. Abgleich der i-Twin-Identifier im IsproNG als Foreign Key
Exceptions:	
Includes:	
Priority:	Hoch
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.4.3 Sensor-Daten erhalten

Fortführung der Arbeiten aus i-Maintenance: Befüllung der Sensorik-Schnittstelle. Hinzu kommt nun die Möglichkeit, mit dem i-Twin Distribution Network jene Datenströme zu identifizieren, die für das CMMS relevant sind. Die erhaltenen Daten werden letztlich mit den entsprechenden Informationen an das CMMS weitergereicht.

© Projektkonsortium i-Twin (2024)

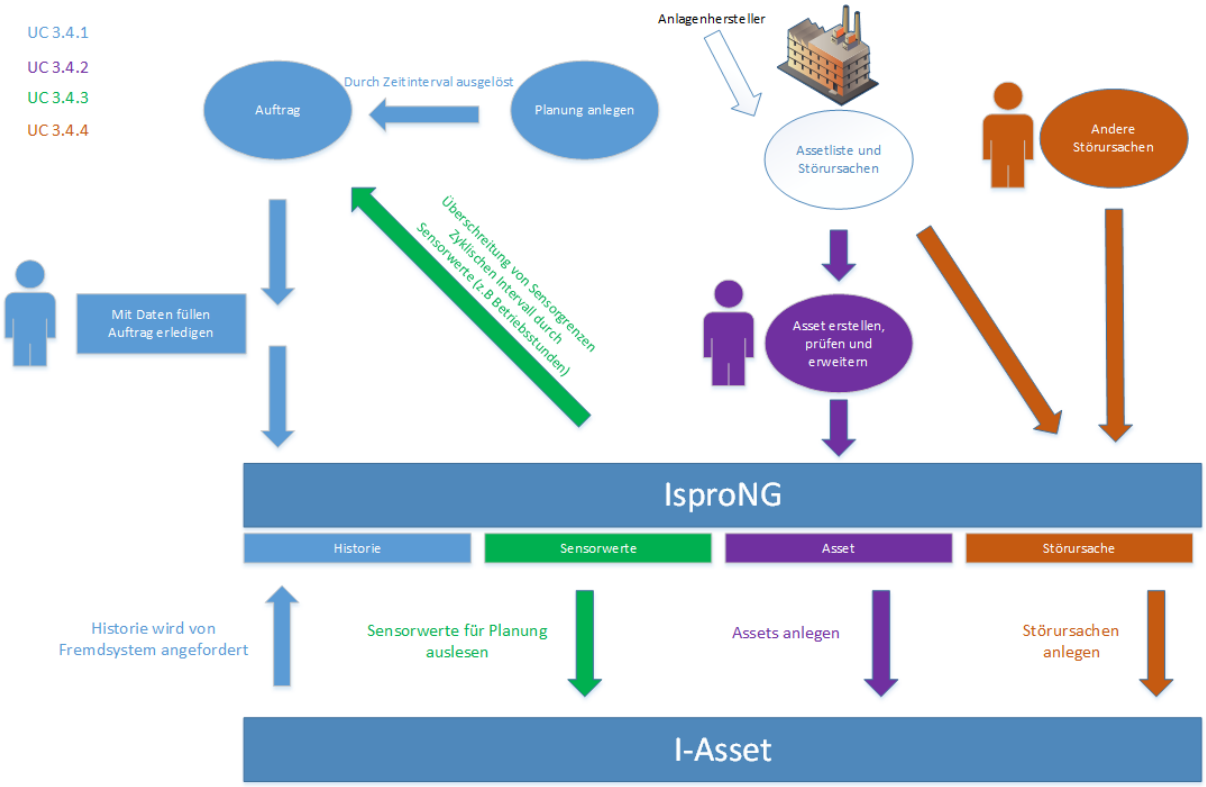


Abbildung 42: Diagramm UC 4.4.3 - Sensor-Daten erhalten

Die Abbildung umfasst mehrere Use Cases. Die Aktoren und Komponenten des Use Cases 4.4.3 sind in grüner Farbe dargestellt.

Use Case ID:	UC4.4.3		
Use Case Name:	Sensordaten auslesen		
Created By:	Adelsmair	Last Updated By:	Adelsmair
Date Created:	25.06.2020	Date Last Updated:	25.06.2020

Actors:	IsproNG
Description:	Im IsproNG werden Planungen und die dazugehörigen Sensoren erstellt. Planungen werden entweder nach Überschreitung eines gewissen Intervalls oder bei festgelegten Grenzüberschreitungen ausgelöst. Sensorwerte werden dazu zyklisch aus I-Twin ausgelesen und IsproNG hinterlegt. Erledigte Aufträge werden I-Twin wieder zur Verfügung gestellt.
Trigger:	Zyklischer oder anlassbasierender Trigger von IsproNG
Pre-Conditions:	Sensorwerte müssen im i-Twin zur Verfügung stehen
Post-Conditions:	Aufträge können aufgrund dieser Infos ausgelöst werden.
Normal Flow:	1. Anlegen einer Planung für ein Asset 2. Wiederkehrender Zyklus definieren 3. Sensorpunkt anlegen und zuweisen

© Projektkonsortium i-Twin (2024)

	4. Abfrage des Sensorwertes 5. Planung löst entsprechenden Auftrag aus, wenn Intervall überschritten 6. Abarbeitung des Auftrags
Alternative Flows:	1. Sensor anlegen 2. Sensorgrenze anlegen 3. Sensor zuweisen 4. Zyklisches auslesen kritischer Werte 5. Sensorgrenze wird überschritten 6. Auftrag wird ausgelöst
Exceptions:	
Includes:	I-Twinid, eindeutige Sensorid
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.4.4 Störungsursachentabelle

Werteliste für Eigenschaft Störungsursache. Im Klassifikationssystem (Semantic Lookup Service) können Eigenschaften mit vordefinierten Werte-Listen versehen werden. Wird eine Eigenschaft in der Verwaltungsschale mittels `semanticId` semantisch mit einem Property mit angehängter Werte-Liste verbunden, so dürfen nur gültige Werte lt. Werteliste eingetragen werden.

Die erforderliche Infrastruktur zur Speicherung der Wertelisten ist bereits in der i-Twin Plattform vorgesehen bzw. vorhanden. Die entsprechende Erfassung und Verwaltung sind zu koordinieren.

© Projektkonsortium i-Twin (2024)

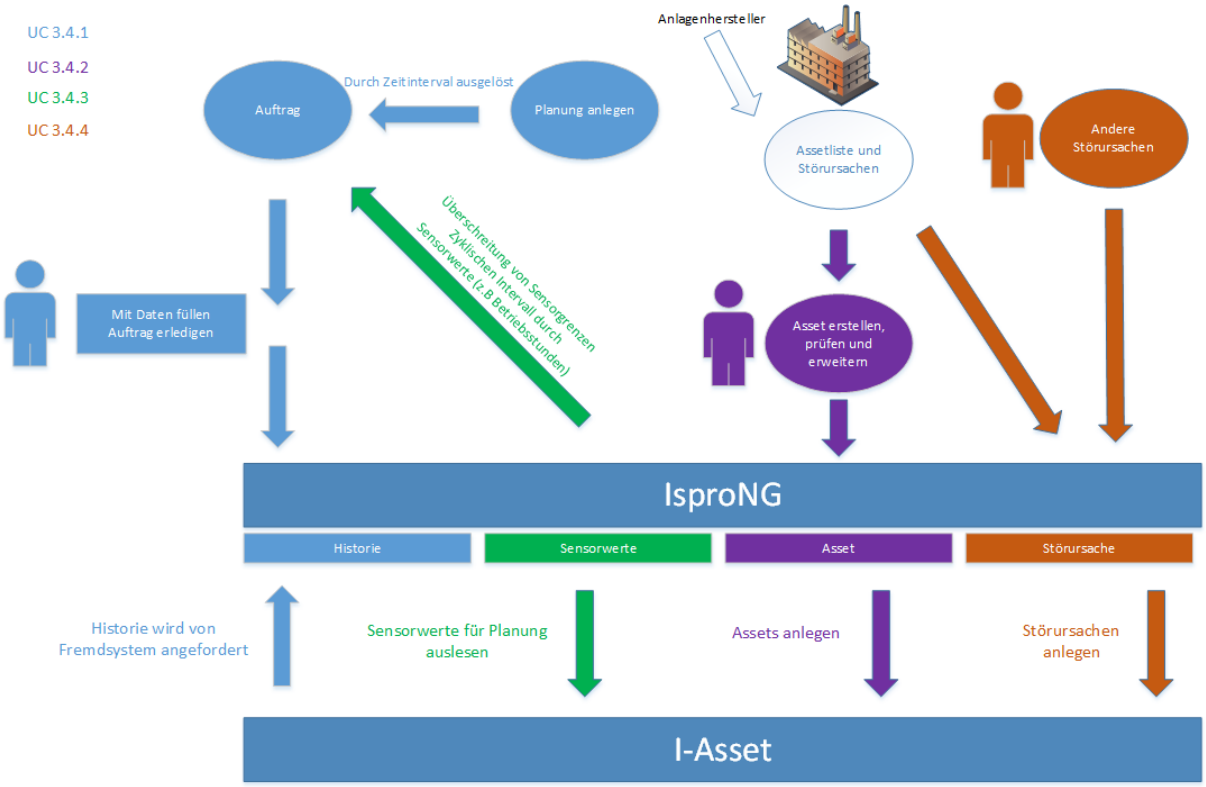


Abbildung 43: Diagramm UC 4.4.4 - Störungsursachentabelle

Die Abbildung umfasst mehrere Use Cases. Die Aktoren und Komponenten des Use Cases 4.4.4 sind in brauner Farbe dargestellt.

Use Case ID:	UC4.4.4		
Use Case Name:	Störursachentabelle		
Created By:	Adelsmair	Last Updated By:	Adelsmair
Date Created:	Adelsmair	Date Last Updated:	Adelsmair

Actors:	IsproNG
Description:	Störursachen werden im IsproNG importiert oder angelegt. Störursachen werden danach im I-Twin angelegt
Trigger:	Anlage einer neuen Störursache
Pre-Conditions:	Zugriff auf I-Twin durch I-Twin-Benutzer (UC 3.1.1)
Post-Conditions:	Störursachentabelle kann von anderen Systemen geladen werden
Normal Flow:	1. Anlegen einer Störursache 2. Übertragen der Störursache in I-Twin
Alternative Flows:	1. Import der Störursachen über Herstellertabelle 2. Übertragen der Störursachen in I-Twin

© Projektkonsortium i-Twin (2024)

Exceptions:	
Includes:	
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.5 Use Cases SRFG

4.5.1 Semantic Repository Integration

…

Use Case ID:	UC 4.5.1		
Use Case Name:	Semantic Repository Integration		
Created By:	Dietmar Glachs	Last Updated By:	
Date Created:	01.04.2022	Date Last Updated:	

Actors:	i-Twin Konsortium
Description:	Das Semantic Repository dient als „Clearing-Stelle“. Hier können gemeinsam genutzte Begrifflichkeiten, Taxonomien hinterlegt werden. Nutzer des Systems sind • das Asset Repository, dieses verweist via **semanticId** auf semantische Beschreibungen, • die I4.0 Komponenten, diese können so eingehende Informationen auf Gültigkeit überprüfen (das kann direkt oder mittelbar über die AAS erfolgen)
Trigger:	Eine AAS (bzw. einzelne Elemente daraus) wird aus dem Semantic Repository exportiert – dazu müssen alle semanticId Beziehungen aufgelöst und als „HasDataSpecification-Extension“ in der exportierten Datei (JSON, XML, AASX) vorhanden sein.
Pre-Conditions:	• Verbindung zwischen Asset Repository und Semantic Lookup ist aktiv • Semantic Repository ist mit den erforderlichen Daten gefüllt
Post-Conditions:	Die „Zusatz“-Information gem. *HasDataSpecification* ist in den AAS-Elementen enthalten

© Projektkonsortium i-Twin (2024)

Normal Flow:	1. Asset Repository exportiert ein AAS-Element mit einer semanticId 2. Die semanticId wird genutzt um die Meta-Daten aus dem Semantic Lookup zu erhalten 3. Die Felder gem. HasDataSpecification im AAS Element werden gefüllt.
Alternative Flows:	(2) Die semanticId verweist auf eine ConceptDescription, diese ist direkt im AssetRepository abgelegt.
Exceptions:	
Includes:	
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Eine Voraussetzung zur Verwendung zusätzlicher semantischer Beschreibungen im Asset Repository ist die Verfügbarkeit des Semantic Lookup Service. Diese Anforderung ist nachfolgend formuliert!

Requirements

Who			
System Integrator, Anbieter von ERP-, CMMS-, Analytics Anwendungen			
What			
Eine semantische Beschreibung der in den jeweiligen Anwendungen verwendeten Datenobjekte zur Verfügung stellen.			
Why			
Aufgrund der semantischen Definition erhalten angeschlossene Systeme die notwendige Information „welche Daten“ erforderlich sind um mit dem angeschlossenen System zu kommunizieren.			

© Projektkonsortium i-Twin (2024)

Acceptance criteria			
• Konzept-Klassen können mittels Services in das Semantic Repository geladen werden. • Konzept-Klassen/Eigenschaften können mittels User-Interface im Semantic Repository definiert werden			

4.6 Templates

4.6.1 Use Case Template

Bezeichnung

…

Use Case ID:	UC i.j.k		
Use Case Name:			
Created By:		Last Updated By:	
Date Created:		Date Last Updated:	

Actors:	
Description:	
Trigger:	
Pre-Conditions:	
Post-Conditions:	
Normal Flow:	1.
Alternative Flows:	
Exceptions:	
Includes:	
Priority:	
Frequency of Use:	
Business Rules:	
Special Requirements:	
Assumptions:	

© Projektkonsortium i-Twin (2024)

Notes and Issues:	

4.6.2 Requirements Template

Requirements

Who			
What			
Why			

Acceptance criteria			

© Projektkonsortium i-Twin (2024)

Beispiel

Who			
As an administrator of the iAsset Platform I want to configure an connection to the zenon Service Gird REST Interface			
What			
Through an configuration Dialog I want to enter the Source address, secrets for the fetching of the data. After Saving the configuration I get an Visual Feedback about the Sucess.			
Why			
between the two system. An Interface extension is needed to set uo the connection paramters. AI			
Acceptance criteria			
1. All secrets must be stored in a secure manner accoring to Industry standards (e.g. BSI) 2. It must be possible to configure more than 1 connection. Each connection gets an unique ID 3. The User gets an optica Feedback about the success of the confguration			

Abbildung 44: Beispiel für Requirements (Quelle: COPA-DATA)

© Projektkonsortium i-Twin (2024)

5 Ergänzende Projektberichte

Der vorliegende Bericht bildet das finale System Design für die i-Twin Plattform und die Semantic Integration Patterns. Da die Plattform auf den Ergebnissen des Vorgängerprojekts i-Asset aufbaut, wurden viele Konzepte und die Use Cases daraus übernommen und angepasst.

Das System Design wird in folgenden Berichten ergänzt und erweitert:

- D2.2 „Semantic Integration Patterns for Manufacturing“
- D2.3 „Semantic Integration Patterns for Artificial Intelligence“
- D3.1 „i-Twin Middleware“
- D3.2 „Asset Connectors“
- D3.3 „Application Connectors“

© Projektkonsortium i-Twin (2024)

6 Referenzen

[idta2023-1] Industrial Digital Twin Association: “Part 1: Metamodel”; Specification of the Asset Administration Shell. IDTA, April 2023. https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/06/IDTA-01001-3-0_SpecificationAssetAdministrationShell_Part1_Metamodel.pdf

[idta2023-2] Industrial Digital Twin Association: “Part 2: Application Programming Interfaces”; Specification of the Asset Administration Shell. IDTA, Juni 2023. https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/06/IDTA-01002-3-0_SpecificationAssetAdministrationShell_Part2_API_.pdf

[i-Asset-D2.9.2] Glachs D.: Design und Architektur der i-Asset Plattform, August 2021

[RAMI40-2017] Heidel R., Hoffmeister M., Hankel M., Döbrich U.; „Basiswissen RAMI4.0 – Referenzarchitekturmodell mit Industrie 4.0 Komponente“; Beuth Verlag Berlin, 2017, ISBN 978-3-410-26482-8

© Projektkonsortium i-Twin (2024)

Impressum

Titel	Final System Design
Bezeichnung	Deliverable 2.4 (i-Twin)
Autoren	Dietmar Glachs, Georg Güntner, Felix Strohmeier (Salzburg Research Forschungsgesellschaft m.b.H.) Thomas Lehrer (Ing. Punzenberger Copa-Data GmbH); Martin Brugger (IcoSense Gmbh); Bernhard Adelsmair (H & H Systems Software GmbH)
Dateiname	D24_Final_System_Design.docx
Publikationsstatus	Public
Letzte Änderung	29.01.2024 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.

© Projektkonsortium i-Twin (2024)