Autowerkstatt 4.0 - Hub

Technische Dokumentation

Hochschule Osnabrück, Technische Hochschule Georg Agricola, LMIS AG

None

Inhaltsverzeichnis

1. Hintergrund

Im Projekt gibt es drei Hauptthemen:

- Innovative Messtechnik
- KI gestützte Fehlerdiagnose
- Gaia-X

Im Rahmen der 2022 durchgeführten Arbeiten und Gespräche wurden verschiedene Anforderungen in diesen Bereichen identifiziert, also Aufgaben / Probleme, die durch das **System Autowerkstatt 4.0** gelöst werden sollen.

Um die Anforderungen Ableiten zu können wurden in Abschnitt Szenarien vier Szenarien formuliert. Die daraus resultierenden Aufgaben/Probleme werden im Abschnitt Anforderungsanalyse genauer beschrieben. Darauf aufbauend wird im Abschnitt Lösungsansatz eine Übersicht des AW4.0 Hub präsentiert.

1.1 Szenarien

Ein Szenario ist eine Situation, die mögliche Ereignisse, Bedingungen und Folgen beschreibt.

1.1.1 Szenario 1: Kauf von Assets

Die Werkstatt hat bisher nur ältere Modelle zur Diagnose von Fahrzeugen, die nicht mehr alle Funktionen abdecken können. Sie möchte daher ein neues Modell erwerben, das auf dem neuesten Stand der Technik ist und alle relevanten Daten auswerten kann. Um ein solches Modell zu finden, nutzt sie die Datenraum-Oberfläche des Werkstatt-Hubs, der einen Markt für den Austausch von Daten und Dienstleistungen zwischen Werkstätten und anderen Akteuren bietet. Dort kann sie nach verschiedenen Kriterien nach passenden Modellen suchen, wie z.B. Preis, Leistung, Bewertung oder Verfügbarkeit.

Nach einer kurzen Suche findet sie ein Modell, das ihren Anforderungen entspricht. Sie entscheidet sich, das Modell zu kaufen und schließt einen Vertrag mit dem Anbieter ab. Der Vertrag regelt die Bedingungen für die Nutzung des Modells, wie z.B. die Laufzeit, den Umfang, die Kosten oder die Haftung. Nachdem der Vertrag abgeschlossen ist, wird das Modell entweder auf den Werkstatt-Hub geladen oder der Zugriff auf das Modell wird über den Datenraum gewährt. Die Werkstatt kann nun das Modell zur Diagnose von Fahrzeugen verwenden.

1.2 Szenario 2: Verkauf von Assets

Um ihre Wettbewerbsfähigkeit zu erhöhen, plant die Werkstatt, ihre gesammelten Fahrzeugdaten als Asset zu vermarkten. Die Werkstatt hat bereits mehrere Fahrzeuge über den Werkstatt-Hub diagnostiziert und dabei Daten erzeugt. Die Werkstatt möchte diese Daten nutzen, um in Zukunft bessere Diagnosemodelle anwenden zu können. Mit Hilfe der Datentraum Oberfläche kann die Werkstatt, die von ihr gewünschten Daten auswählen und zu einem Asset bündeln. Die Werkstatt legt die Vertragsbedingungen fest und erstellt eine Asset-Beschreibung. Die Werkstatt-Daten werden anonymisiert und als Asset-Angebot auf den Markt gebracht.

1.3 Szenario 3: Diagnose

Die Werkstatt führt eine Diagnose am Fahrzeug mittels des AW 4.0 Werkstatt-Hubs durch. In diesem Szenario beschreiben wir die Diagnose anhand von Oszilloskop Daten.

Eine Werkstatt erhält ein Fahrzeug zur Diagnose, das einen Fehler aufweist. Um die Ursache des Fehlers zu ermitteln, nutzt die Werkstatt den AW 4.0 Werkstatt-Hub und legt einen Fall für die Diagnose an. Der Werkstatt-Hub liest die OBD-Daten des Fahrzeugs aus und vergleicht sie mit dem Wissensgraphen, um das wahrscheinlichste defekte Bauteil zu bestimmen. Anschließend ruft der Werkstatt-Hub die entsprechende Diagnose für das Bauteil auf und gibt der Werkstatt die Anforderungen und Anweisungen für eine OszilloskopMessung. Die Werkstatt führt die Messung gemäß den Vorgaben durch und leitet die Messdaten an den Werkstatt-Hub weiter. Der Werkstatt-Hub interpretiert die Messdaten und identifiziert das defekte Bauteil. Nachdem die Werkstatt das Bauteil ausgetauscht hat, markiert sie den Fall im Werkstatt-Hub als abgeschlossen.

1.4 Szenario 4: Onboarding

Um die Sicherheit und Integrität der Daten im Datenraum zu gewährleisten, müssen neue Teilnehmer einen standardisierten Onboarding-Prozess durchlaufen. Dieser Prozess besteht aus folgenden Schritten:

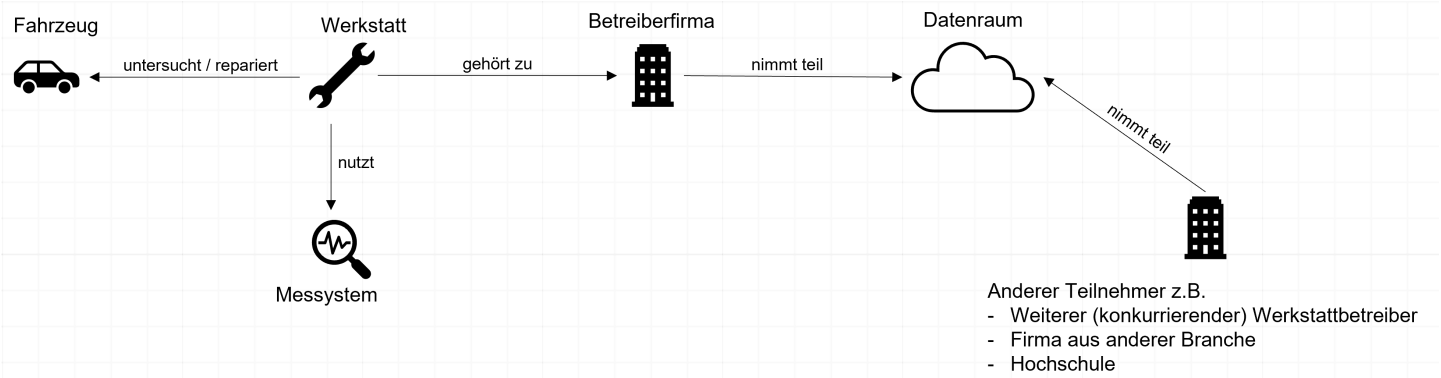
1. Der neue Teilnehmer bekundet sein Interesse an dem Datenraum und erklärt seine Bereitschaft, die Regeln und Anforderungen des Datenraums einzuhalten.
2. Der neue Teilnehmer füllt ein Formular aus, in dem er seine Identität, seine Ziele und seine Datenbedarfe angibt. Dieses Formular wird an den Föderator des Datenraums gesendet, der die Rolle eines Vermittlers und Vertrauenswürdigen Dritten hat.
3. Der Föderator prüft die Angaben des neuen Teilnehmers und vergleicht sie mit den Kriterien und Erwartungen des Datenraums. Der Föderator kann auch zusätzliche Informationen oder Nachweise vom neuen Teilnehmer anfordern, um seine Eignung zu bestätigen.
4. Wenn der Föderator den neuen Teilnehmer für geeignet hält, stellt er ihm ein digitales Zertifikat aus, das seine Identität und seine Berechtigung zur Teilnahme am Datenraum bestätigt. Das Zertifikat enthält auch die Zugangsdaten und die technischen Spezifikationen für den Datenraum.
5. Der neue Teilnehmer wird in den Datenraum aufgenommen und kann auf die verfügbaren Daten zugreifen oder eigene Daten bereitstellen. Die anderen Mitglieder des Datenraums werden über den neuen Teilnehmer informiert und können mit ihm kommunizieren oder kooperieren.

1.5 Anforderungsanalyse

Im AW4.0 Universum gibt es verschiedene *Rollen*, also Benutzer- / Anwenderebenen die mit dem System interagieren. Diese sind:

- Werkstatt
- Betreiberfirma
- Datenraum
- Fahrzeug
- Messsystem

Diese Rollen sind mit ihren jeweiligen Beziehungen im untenstehenden Diagramm dargestellt.



Aufbauend auf diesen Überlegungen zur Struktur des AW4.0 Universums lassen sich vielfältige Anforderungen identifizieren, also konkrete Funktionalitäten, die das System haben soll. Diese werden im Folgenden beschrieben.

Neben der Durchführung von Messungen muss im Umfeld der Werkstatt ein einfaches *Datenmanagement* vorausgesetzt werden. Es sollte dabei möglich sein, neue "Fälle" anzulegen, wenn ein Auto in die Werkstatt kommt. Diese Fälle sollten z.B. bearbeitet, durchsucht und gelöscht werden können.

Mitarbeiter in Werkstätten sollten außerdem die Möglichkeit haben, KI unterstützte Fehlerdiagnose durchzuführen, also die am DFKI entwickelte State Machine anzuwenden. Dabei soll die Integration gängiger Messsysteme von verschiedenen Herstellern berücksichtigt werden, d.h. es sollte möglich sein, die Daten unterschiedlicher Messsysteme zu einem Fall hinzuzufügen, damit diese von der KI verwertet werden können.

Autowerkstatt 4.0 soll außerdem Anforderungen auf der höher gelagerten Ebene der Betreiberfirma erfüllen. Hier sollte es möglich sein, die eigenen Geschäftsdaten sicher zu verwalten, wobei insbesondere der Schutz personenbezogener Informationen von Mitarbeitern und Kunden sicherzustellen ist.

Eine weitere Anforderung im Umfeld der Betreiberfirma ist die *Teilnahme am AW4.0 Datenraum*.

1.5.1 Teilnahme am AW 4.0 Datenraum

Der Werkstatt-Hub muss einen nach der Gaia-X 10.23 Architektur konformen Zugriff auf den Datenraum gewährleisten. Dabei fungiert der Hub jeweils in der Rolle des Consumers (Kauf von Assets, wie KI-Modellen)) und Providers (Verkauf von Assets wie Daten). Daraus ergeben sich folgende Anforderungen:

- Erwerb/Integration eines Gaia-X Credentials, mit welchem man sich als self sovereign identity ausweisen kann.
- Identifikation im Datenraum mit einem (erweiterten) Gaia-X Credential
- Vertragsabwicklung für den Kauf von Assets, sowie das Aufsetzen eigener Verträge (Policies) für den Verkauf von Assets.
- Selbstbestimmter Erwerb von Assets aus dem Datenraum für eine effizientere Diagnose.
- Selbstbestimmter Verkauf von Assets im Datenraum, hierbei muss sichergestellt werden, dass keine personenbeziehbaren Daten veräußert werden können.
- Empfang von Assets als Consumer, Weitergabe von Assets als Provider

1.6 Lösungsansatz

Das auf dieser Website beschriebene System ist ein Prototyp für den *Autowerkstatt 4.0 Hub*. Dieser ist ein IT-System, mit dem die im vorherigen Abschnitt beschriebenen Anforderungen erfüllt werden sollen.

Die wichtigsten Ideen sind:

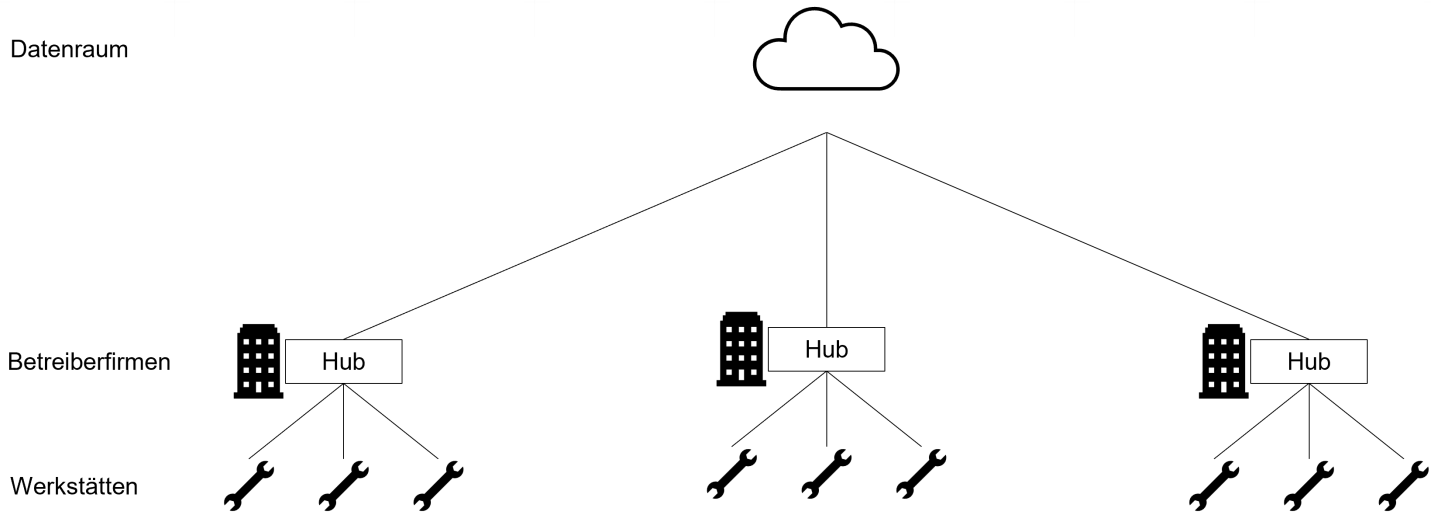
Eine Betreiberfirma hat einen AW4.0 Hub.

- Dieser ist ein Firmen-eigenes IT-System, auf dem nur die eigenen Geschäftsdaten verwaltet werden.
- Nur die eigenen Mitarbeiter (in Werkstätten oder Zentrale) interagieren mit diesem System.

Ein Hub besteht aus verschiedenen Services / Komponenten.

- Diese stellen für die Mitarbeiter in den Werkstätten die benötigten Funktionalitäten zur Verwaltung von Fällen und zur KI gestützten Fehlerdiagnose bereit.
- Weitere Services ermöglichen der Betreiberfirma die selbstbestimmte Verwaltung und Weitergabe der eigenen Geschäftsdaten.

Damit ist der Hub wie in der folgenden Abbildung dargestellt in das AW4.0 Universum einzuordnen:



Im Gegensatz zu früheren Ideen gibt es im AW4.0 Universum also hier nicht *eine Datenpipeline* und *einen Server*. Dieses Grundgerüst erscheint nicht sinnvoll, da die selbstbestimmte Verwaltung und Weitergabe der eigenen Geschäftsdaten eine essenzielle Anforderung an das System ist.

Weitere Kernaspekte des AW4.0 Hubs sind:

Separation of Concerns

- Messsysteme mit denen Daten am Fahrzeug generiert werden, sind nicht Teil des Hubs. Die Interfaces des Hubs sind jedoch so gestaltet, dass die Integration verschiedener Messsysteme gewährleistet ist.
- KI (e.g. die State Machine) wird auf dem Hub ausgeführt und nicht auf einem Messgerät, da dies die Integration verschiedener propriäterer Messsysteme deutlich erschweren würde.

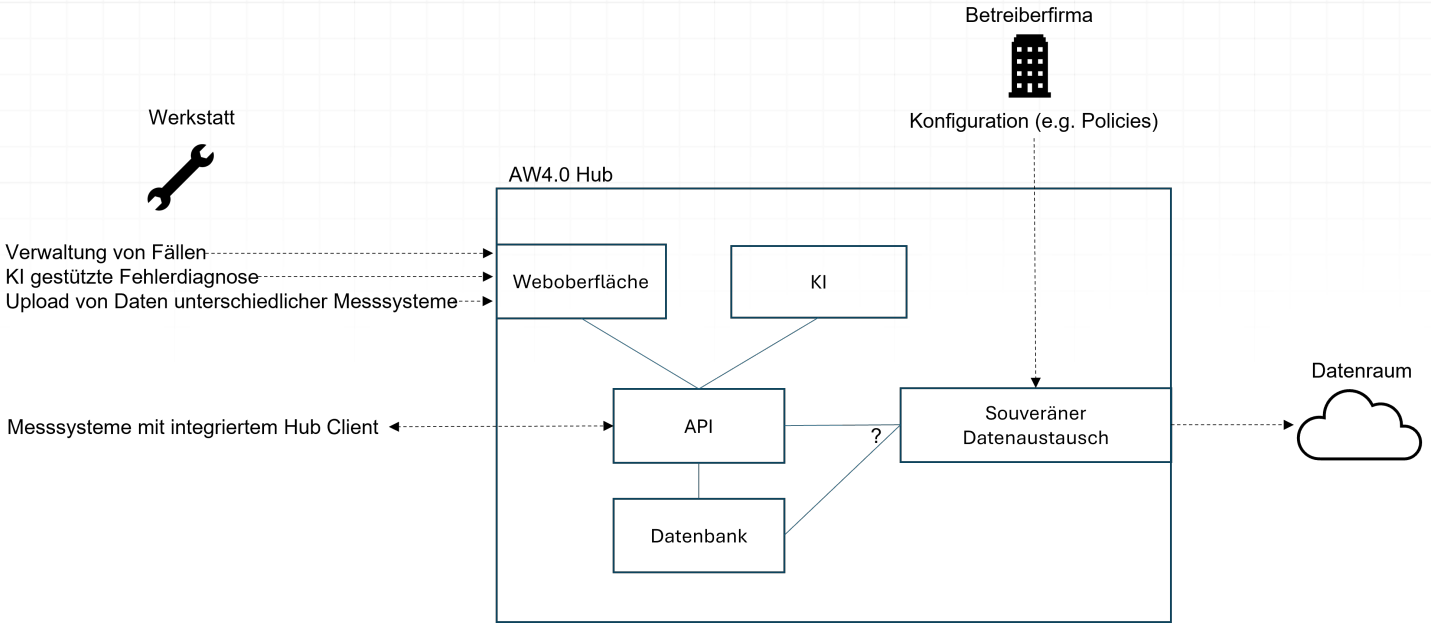
Flexible Interaktionsmöglichkeiten

- Über eine Weboberfläche können die Fälle der eigenen Werkstatt eingesehen und verwaltet werden sowie Daten unterschiedlicher Messsysteme hochgeladen werden.
- Alternativ können spezialisierter Messsysteme direkt über eine API mit dem Hub kommunizieren.

Flexible physische Ausgestaltung

- On-premise vs. Cloud
- Single-host vs. Verteiltes System

Die folgende Abbildung zeigt eine Architektur Skizze des Hub Prototypen:



Anmerkungen

Es gibt noch weitere Anforderungen und daraus abgeleitete Services, die hier nicht dargestellt wurden, aber berücksichtigt werden müssen. Dazu zählt beispielsweise die Anforderung *Authentifizierung und Autorisierung*.

Die Architekturskizze soll nur eine Grundidee basierend auf den vorgestellten Rollen und Anforderungen illustrieren.

Alle Komponenten in der Skizze erfordern weitere spezifische Anforderungsanalysen und daraus abgeleitete Designs. Insbesondere die Komponenten *KI* und *souveräner Datenaustausch* setzen sich wiederum aus mehreren Services zusammensetzen.

2. Technische Umsetzung

2.1 Datenbank

Die Datenbank speichert die zur KI geführten Fehlerdiagnose relevanten Daten einer Betreiberfirma in einem standardisierten Schema.

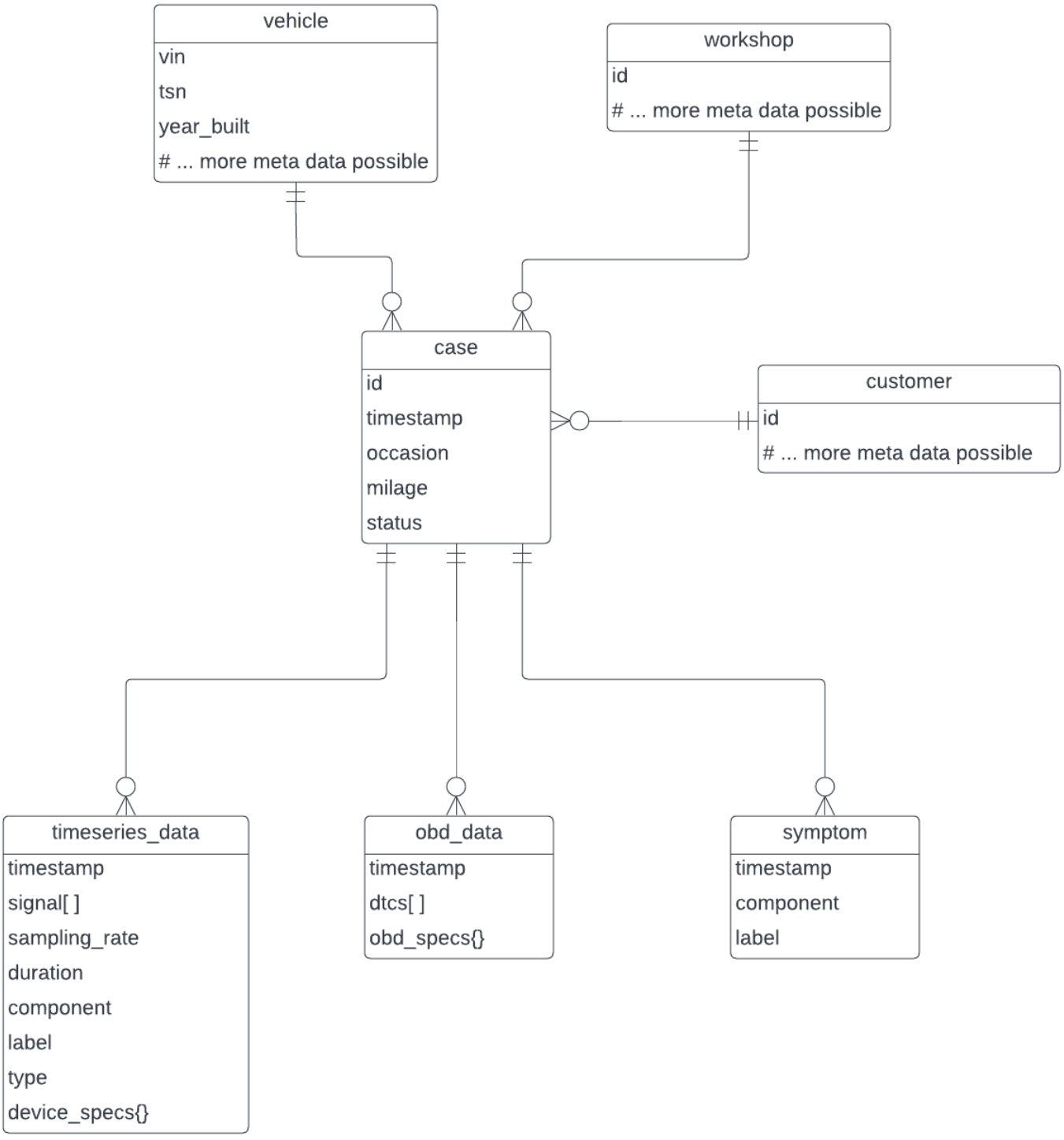
2.1.1 Konzeptionelles Datenschema

Ein Fall (`case`) repräsentiert genau einen Aufenthalt genau eines Fahrzeugs (`vehicle`) in genauer einer Werkstatt (`workshop`). Der Fall ist dabei mit genau einem Kunden (`customer`) assoziiert, also beispielsweise der Person oder Firma, die das Fahrzeug in die Werkstatt bringt.

Während des Werkstattaufenthalts des Fahrzeugs werden diagnostische Daten zu dem Fall hinzugefügt. Es werden drei verschiedene Typen von diagnostischen Daten in Betracht gezogen:

- Zeitreihen Daten (`timeseries_data`)
- OBD Daten (`obd_data`)
- Symptome (`symptom`)

Die untenstehenden Abbildung zeigt ein auf diesen Überlegungen basierendes ER Diagramm:



Der nächste Abschnitt enthält eine detailiertere Beschreibung der verschiedenen Entitäten.

Details zu den Entitäten

`case`

Ein Fall repräsentiert genau einen Aufenthalt genau eines Fahrzeugs in genauer einer Werkstatt und ist das zentrale organisierende Objekt im Datenmodell. Ein Fall beinhaltet die folgenden atomaren Attribute:

Attribut	Beschreibung	Beispiele
`id`	Eindeutiger Identifikator	
`timestamp`	Zeitpunkt der Erstellung	
`occasion`	Anlass für den Werkstattaufenthalt	"Service / Routine", "Problem / Defekt"
`milage`	Aktueller Kilometerstand	
`status`	Status des Falls	"open", "closed"

`vehicle`

Repräsentiert das Fahrzeug als technisches Objekt. Jede Instanz, identifiziert durch seine `vin`, existiert höchstens einmal in der Datenbank. Aktuell werden außerdem die Attribute `tsn` und Baujahr (`year_built`) verwendet. Theoretisch sind auch weitere technische Attribute denkbar, solange diese sich nicht mit der Zeit ändern, also beispielsweise *Hersteller*, *Fahrzeugtyp* etc.
Attribute die den Zustand des Fahrzeugs *zu einem bestimmten Zeitpunkt* beschreiben werden nicht der Fahrzeug Entität, sondern dem Fall zugeordnet (Beispiel: Kilometerstand).

`workshop`

Repräsentiert eine spezifische Werkstatt. Diese Informationen wird eigentlich nicht für diagnostische oder KI Anwendungen benötigt. Da ein Hub aber zu einer *Betreiberfirma* gehört die wiederum mehrere Werkstätten betreiben könnte, ist es innerhalb eines Hubs nötig nachzuhalten, welche Fälle zu welcher Werkstatt gehören. Theoretisch könnten dieser Entität noch weitere Meta Daten zur Werkstatt zugeordnet werden wie z.B. Kontakt Informationen. Der Hub Prototyp beinhaltet diese Informationen noch nicht.

`customer`

Repräsentiert diejenige Person / Firma, die das Fahrzeug in die Werkstatt bringt, üblicherweise also den/die Besitzer/in des Fahrzeugs zum Zeitpunkt des Falls. Auch diese Information wird eigentlich nicht zu Diagnosezwecken oder für KI Anwendungen benötigt. Die Kundenentität ist im aktuellen Prototypen dennoch enthalten, da so z.B. das Durchsuchen von Fällen nach Kunde möglich ist. Es liegt nahe, dass dies in einem realistischen Anwendungsszenario eine nützliche Funktionalität ist.

Mit Blick auf die Gaia-X Storyline im Projekt kann die Berücksichtigung von (möglicherweise künstlichen) Kundendaten auzßerdem nützlich sein um verschiedene Konzepte zu illustrieren:

- Für die Betreiberfirma ist das Handling von Kundendaten ein normaler Bestandteil des eigenen Geschäftsprozesses. Diese Informationen sind zu entfernen, bevor Daten mit Partnern im AW4.0 Datenraum geteilt werden
- Transparenz: Der Datensatz ermöglich die Beantwortung von Kundenanfragen wie "Welche Daten liegen über mich vor?"

`timeseries_data`

Eine Instanz repräsentiert eine einzelne Zeitreihen Messung, also z.B. ein Oszilloskop Signal. Attribute sind in der folgenden Tabelle genauer beschrieben.

Attribut	Beschreibung	Beispiele
`timestamp`	Zeitpunkt der Erstellung	
`signal[]`	Das eigentliche Signal, e.g Array mit Floats	
`sampling_rate`	Abtastrate der Messung in Hz	
`duration`	Dauer der Messung in Sekunden	
`component`	Das gemessene Fahrzeugbauteil	"Batterie"
`label`	Label des Datensatzes	"Regelfall / Unauffällig", "Anomalie / Auffälligkeit"
`type`	Typ des Datensatzes	"oscillogram", "engine load"
`device_specs{}`	Technische Spezifikationen des Messgeräts. "`{}`" bedeutet hier, das dies ein nicht-standardisiertes Objekt mit verschiedenen key-value Paaren sein kann, abhängig vom benutzten Messgerät	

`obd_data`

Eine Instanz repräsentiert das einmalige Auslesen des Fahrzeug Fehlerspeichers. Attribute sind in der folgenden Tabelle genauer beschrieben.

Attribut	Beschreibung	Beispiele
`timestamp`	Zeitpunkt der Erstellung	
`dtcs[]`	Array mit DTCs	["P0101", "P0202", "P0303"]
`obd_specs{}`	Technische Spezifikationen des Messgeräts. "`{}`" bedeutet hier, das dies ein nicht-standardisiertes Objekt mit verschiedenen key-value Paaren sein kann, abhängig vom benutzten Messgerät	

`symptom`

Eine Instanz repräsentiert das Ergebnis einer (manuellen / ausführlichen / nicht durch die anderen Datentypen abgedeckten) Überprüfung eines spezifischen Bauteils. Attribute sind in der folgenden Tabelle genauer beschrieben.

Attribute	Description	Examples
`timestamp`	Zeitpunkt der Erstellung	
`component`	Das untersuchte Fahrzeugbauteil	"Lichtmaschine"
`label`	Das Ergebnis der Untersuchung	"defekt", "nicht defekt"

Anmerkungen zum Datenmodell

Anmerkung 1: `customer` - `vehicle` Relation

Die *real-world* Relation "besitzt" zwischen Kunde und Fahrzeug wird im hier vorgestellten Datenmodell nicht durch eine direkte Verbindung zwischen `customer` und `vehicle` berücksichtigt. Die Begründung ist wie folgt:
Die Relation ist eine *many-to-many* Beziehung. Ein/Eine Kunde/Kundin besitzt im Laufe der Zeit mehrere Fahrzeuge

und ein Fahrzeug hat über seine Nutzungsdauer mehrere Besitzer. Sowohl für Kunden, als auch für Fahrzeuge sollte jede *real-world* Instanz höchstens einmal in der Hub Datenbank sein, um Inkonsistenzen zu vermeiden, z.B. wenn Meta Daten aktualisiert werden. Desweiteren ist ein spezifisches (`customer`, `vehicle`) Paar nur über eine bestimmte Zeit Teil der "besitzt" Relation. Die *many-to-many* Relation müsste also ohnehin durch eine dedizierte Tabelle repräsentiert werden und nicht mittels Fremdschlüsseln die direkt von `vehicle` zu `customer` (oder andersherum) verweisen. Die Aufgabe dieser dedizierten Tabelle wird bereits durch die Liste der Fälle abgedeckt.

2.1.2 Technische Umsetzung

Die Umsetzung des oben beschriebenen Datenmodells ist mit verschiedenen relationalen und nicht-relationalen Datenbanksystemen denkbar. Für den Hub Prototypen ist die Entscheidung zunächst auf MongoDB gefallen, da ... - ... das flexible Dokumenten-basierte Datenmodell eine einfachere Anpassung im Laufe des Projekts ermöglicht - ... mit GridFS die Möglichkeit besteht auch größere Oszilloskop Signale im Binärformat zu speichern

2.2 API

HTTP Schnittstelle für das standardisierte Management der gespeicherten Daten.

Erstellt mit FastAPI.

2.2.1 Übersicht

Die API hat mehrere Teilbereiche (Router)

Bereich	Pfad	Beschreibung	Authentifizierung
Diagnostics	`/diagnostics`	Zugriffspunkte für das Diagnosebackend.	API Key
Health	`/health`	Funktionskontrolle	keine
MinIO	`/minio`	*WIP*	API Key
Shared	`/shared`	Lesezugriff auf geteilte Ressourcen innerhalb einer Betreiberfirma.	Keycloak
Workshops	`/{workshop_id}`	Verwaltung eigener Daten und Diagnosen durch Endnutzer Anwendungen in einzelnen Werkstätten.	Keycloak
Knowledge	`/knowledge`	Vereinfachter Zugriff auf Informationen, die im Wissensgraph gespeichert sind.	Keycloak

2.2.2 Details

Die oben beschriebenen Router können grob in zwei Gruppen unterteilt werden.

Die erste Gruppe bilden die Bereiche Diagnostics, Health und MinIO. Alle in diesen Routern bereitgestellten Endpunkte sind für die M2M Kommunikation zwischen verschiedenen Services "im Hintergrund" vorgesehen.

Die zweite Gruppen bilden die Bereiche Shared, Workshops und Knowledge welche für die Kommunikation mit (menschlichen) Endnutzern vorgesehen sind.

Workshop Router

Wie in Hintergrund beschrieben, ist der Hub als Plattform für mehrere, zu einer *Betreiberfirma* gehörenden *Werkstätten* vorgesehen.

Im Hub Prototypen ist es daher vorgesehen, dass jede Werkstatt einen eigenen Nutzeraccount hat. Diese Accounts werden mittels Keycloak verwaltet. Dabei ist jedem Werkstattaccount die Rolle `workshop` zuzuweisen, die für den Zugriff auf Ressourcen unter `/{workshop_id}` vorausgesetzt wird.

Jede Client Applikation für Endanwender (e.g. das im Hub integrierte Web Frontend oder Messgeräte mit integrierter Verbindung zur Hub API) muss bei Zugriff auf die Endpunkte unter `/{workshop_id}` nachweisen, dass die Anfrage durch die Werkstatt mit dieser `{workshop_id}` berechtigt ist.

Shared Router

Die `/shared` Endpunkte der Hub API ermöglichen Lesezugriff auf die in der Hub Datenbank gespeicherten Daten zu Fällen, Fahrzeugen etc., beispielsweise zu Analysezwecken.

Auch für die zu diesem Bereich gehörenden Endpunkte wird eine Authentifizierung mittels eines von Keycloak ausgestellten Tokens vorausgesetzt. Dem Nutzeraccount muss dabei die Rolle `shared` zugewiesen sein.

Knowledge Router

Die unter `/knowledge` bereitgestellten Endpunkte ermöglichen den Zugriff auf ausgewählte Fakten zu im Wissensgraph beinhalteten Informationen, wie z.B. die Namen der gespeicherten Fahrzeugbauteile.
Der Zugriff ist mit einem von Keycloak ausgestellten Token möglich, wobei sowohl die Rollen `workshop` als auch `shared` autorisiert sind.

2.3 Gaia-X

Wie ist die Teilnahme am AW4.0 Datenraum technisch umgesetzt?

2.4 KI

In diesem Abschnitt wird beschrieben, wie die Funktionalität zur KI unterstützten Fehlerdiagnose umgesetzt ist.

2.4.1 Vorüberlegungen

Kernstück der KI unterstützten Fehlerdiagnose ist die am DFKI entwickelte **State Machine**.
Die State Machine nutzt - Am Fahrzeug generierte Messdaten, also insbesondere OBD Daten und Oszillogramme - Einen Wissensgraph zur Beurteilung der OBD Daten - ML Modelle zur Bewertung von Oszillogrammen

Die State Machine ist ein Agent, der viele komplexe Schritte der Fehlerdiagnose weitgehend autonom durchführt. Insbesondere die Anwendung von Wissensgraph und ML Modellen wird eigenständig gehandhabt. Der Nutzer hat lediglich die benötigten Eingabedaten zur Verfügung zu stellen. Diese werden zum Teil erst während der Fehlerdiagnose ermittelt. Beispielsweise entscheidet die State Machine erst nach Auswertung der Fehlercodes, für welche Bauteile Oszillogramme zu erstellen sind. Entsprechende Anweisungen müssen dann an den Nutzer übermittelt werden. Die Diagnose kann fortgesetzt werden, sobald der Nutzer die Daten zur Verfügung gestellt hat.

Aus Nutzersicht werden die folgenden Anforderungen an das KI gestützte Diagnosesystem festgelegt:

- Diagnose Management: Starten, Abbrechen etc.
- Informationen über Zustand und Ergebnisse der Diagnose einsehen
- Benachrichtigt werden, wenn eine eigene Handlung erforderlich ist (z.B. Daten hochladen)
- Es müssen mehrere „offene“ Diagnosen gleichzeitig vorliegen können, e.g. wenn eine Diagnose auf ein bestimmtes Oszillogramm „wartet“, dann sollte es möglich sein, weiter an anderen Fällen zu arbeiten
- Die Diagnose sollte automatisch weitergehen, sobald benötigte Nutzerhandlungen (z.B. Daten hochladen) ausgeführt wurden

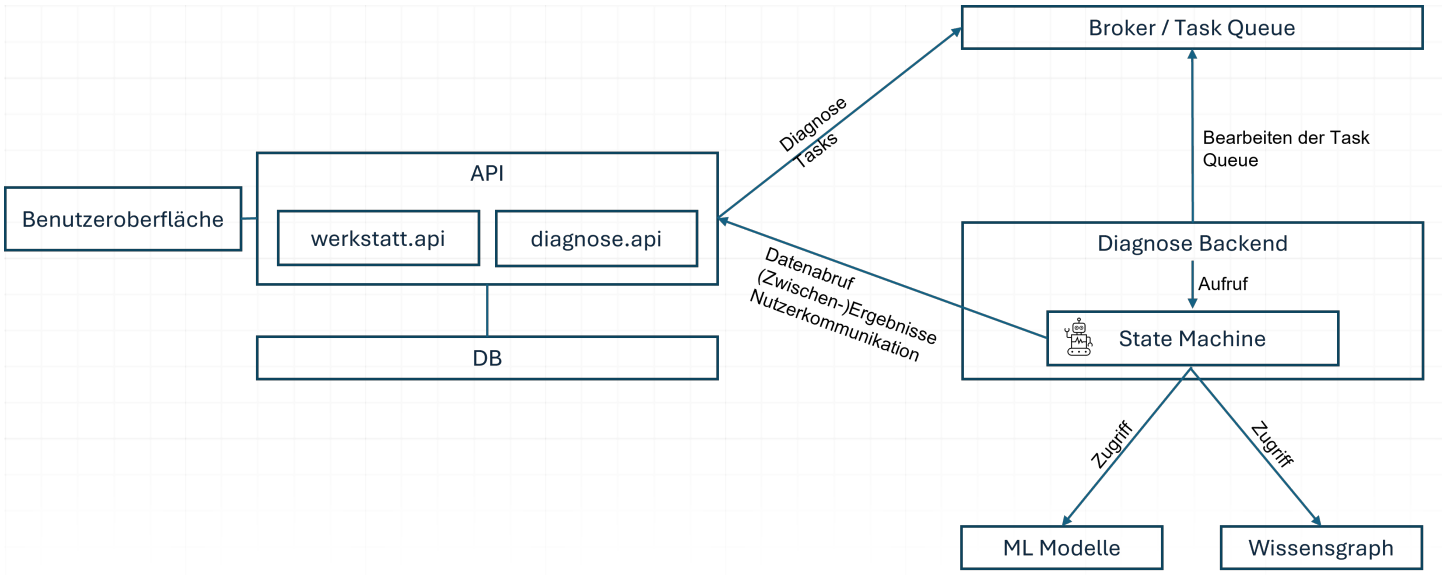
Auch die State Machine als "Rechenkern" des Diagnosesystems stellt eine Reihe von Anforderungen an das System, in das sie eingebettet sind. Diese sind:

- Zugriff auf Messdaten / Informationen zum Fall (Fehlercodes und Oszillogramme)
- Nutzerkommunikation ((Zwischen-)Ergebnisse, benötigte Daten etc.)
- Laden von ML Modellen für die Oszillogramme spezifischer Bauteile
- Zugriff auf (zentralen) Wissensgraph

Es ist auch vorgesehen, dass das Diagnosesystem nicht direkt auf die Datenbank des Hubs zugreift, sondern über eine dedizierte API. Dadurch wird sichergestellt, dass nur die benötigten Messdaten in das Diagnosesystem gelangen und keine anderen Geschäfts- oder Kundendaten.

2.4.2 State Machine Integration

Die unten stehende Abbildung skizziert die Integration der State Machine.



Beschreibung der Abbildung:
AnwenderInnen interagieren über eine Benutzeroberfläche mit dem System. Über diese können die durch die `werkstatt.api` definierten Datenmanagement Operationen durchgeführt werden. Dies sind u.A. *Anlegen* und *Bearbeiten von Fällen*, *Hochladen von Fahrzeugdaten* (Fehlercodes, Oszillogramme), das *Starten der Diagnose* für einen Fall und der *Abruf von Informationen zum aktuellen Diagnosezustand*.

Wird mittels der `werkstatt.api` die Aktionen *Starten der Diagnose* durchgeführt, triggert dies die Erzeugung eines Diagnose Tasks, der vom API Service in einer Task Queue abgelegt wird.

Diese wird kontinuierlich vom Diagnose Backend Service bearbeitet. In diesen ist die State Machine als Softwarebibliothek eingebunden. Die State Machine interagiert mit dem Datenkern des Hubs über die `diagnose.api`. Diese ermöglicht den Abruf der relevanten Daten zum Fall, sowie das Senden von Zwischenergebnissen und Messaufträgen. Auf diese Informationen können Anwender wiederum mittels der `werkstatt.api` zugreifen.

Durch diesen *asynchronen Dialog* zwischen Anwendern und State Machine wird sichergestellt, dass sich auf einem AW4.0 Hub mehrere Diagnosen gleichzeitig "in Bearbeitung" befinden können, ohne dass dies die Interaktionsmöglichkeiten von Anwendern mit dem Datenkern blockiert oder einschränkt.

Zur Umsetzung der Task Queue basierten Kommunikation zwischen API und Diagnose Backend verwendet der Hub Prototyp Celery mit Redis als Broker verwendet.

2.4.3 Skalierbarkeit

Der Prototyp beinhaltet derzeit jeden Service einmal. Im Szenario einer sehr großen Betreiberfirma mit vielen Werkstätten ist es denkbar, dass viele offene Diagnosen gleichzeitig vorliegen. Aufgrund der potenziell rechenintensiven Ausführung der State Machine ist nicht auszuschließen, dass es dabei zu längeren Wartezeiten kommt, falls die Task Queue nur von einem Diagnose Backend bearbeitet wird. Der im vorherigen Abschnitt vorgestellte Integrationsansatz kann horizontal Skaliert werden, indem Replikate des Diagnose Backends die Task Queue parallel bearbeiten.

2.5 Integration heterogener Messgeräte

Wie im Hintergrund beschrieben, sind Messsysteme, mit denen am Fahrzeug Daten generiert werden, kein direkter Teil des AW4.0 Hubs.
Stattdessen sollen Daten auf dem Hub in einheitlichem Format vorliegen, damit die KI basierte Verwertung unabhängig vom Messsystem möglich ist und auch die Datenraumanbindung einfach standardisierbar ist.
Hinzu kommt, dass der hier vorgestellte Prototyp eher auf einen stationären, Server-seitigen Betrieb ausgerichtet ist.

Funktionalität zum hochladen und empfangen von Messungen ist daher essentiell, damit überhaupt Fahrzeugdaten in das System gelangen können.

Um diese Funktionalitäten bereitzustellen sind zwei prinzipielle Ansätze vorgesehen, die als *Hub-seitige Messsystemintegration* bzw. *Messsystem-seitige Hubintegration* bezeichnet werden können.
Diese Ansätze werden im Folgenden anhand von Beispielszenarien beschrieben.

2.5.1 Ansatz 1: Hub-seitige Messsystemintegration

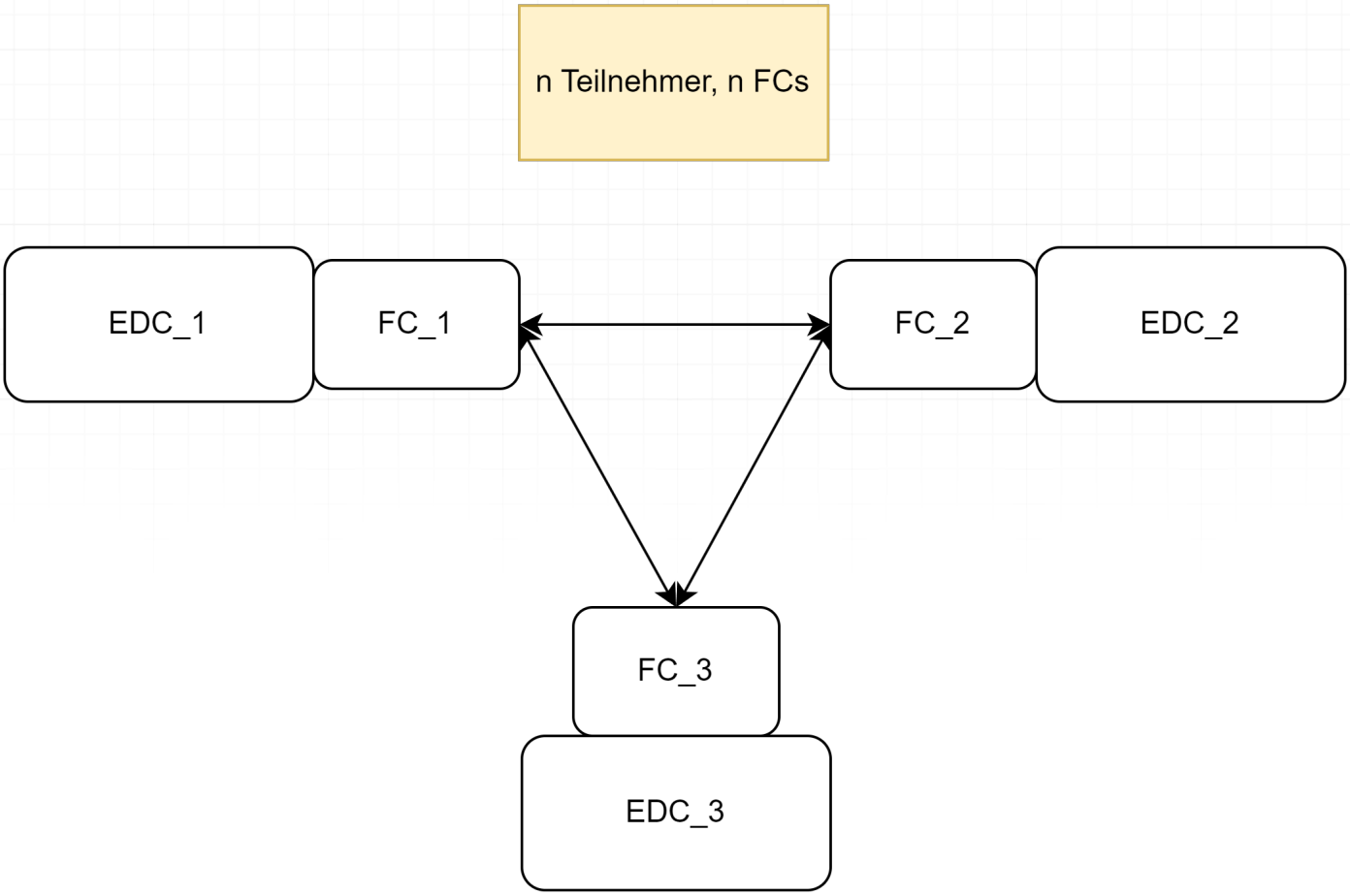
Szenario:
Eine Werkstatt nutzt ein bestimmtes proprietäres Messystem zum Auslesen von Fehlercodes bzw. zum oszilloskopieren. Die Messsystemsoftware ermöglich es, Ergebnisse als Textdatei zu exportieren.
Da das Messsystem weit verbreitet ist, beinhaltet die API des AW4.0 Hubs einen Endpunkt, der die exportierten Ergebnisse dieses Messsystems als Upload akzeptiert, in das Hub-interne Datenformat konvertiert und in der Datenbank speichert.
Mitarbeiter der Werkstatt können also weiterhin die bekannte Software nutzen und die entstehenden Daten über die Weboberfläche des Hubs hochladen, so dass diese beispielsweise zur KI unterstützten Fehlerdiagnose verwertet werden können.

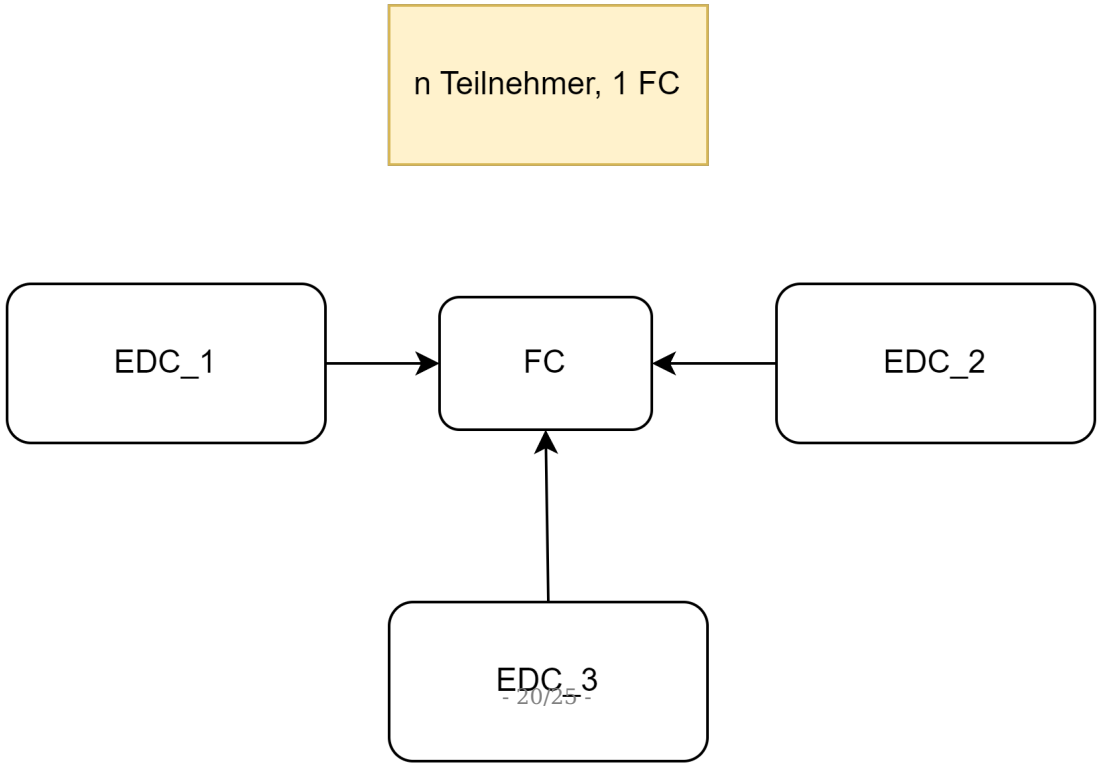
2.5.2 Ansatz 2: Messsystem-seitige Hubintegration

Szenario:
Eine Oszilloskop Hersteller möchte, dass die Kombination des eigenen Messsytems mit der AW4.0 Fehlerdiagnose möglichst einfach und anwenderfreundlich ist. In die eigene Messsoftware wird daher ein Client integriert, um direkt mit der Hub API zu kommunizieren.
In der Benutzeroberfläche des Messsystems können nun die aktuell zu bearbeitenden Fälle angezeigt werden und insbesondere die von der KI erstellten „Messaufträge“. Diese Aufträge können dann direkt mit dem Messsystem erledigt werden. Die entstandenen Daten werden automatisch an die Hub API geschickt, wo sie zur KI unterstützten Fehlerdiagnose verwertet werden können.

2.6 Federated Catalog

Der Federated Catalog (FC) entspricht einer Liste sämtlicher Teilnehmer eines Datenraumes mitsamt ihrer angebotenen Services und Assets. Die FC-Implementierung des AW4.0-EDC sieht vor, dass jeder Teilnehmer im Datenraum einen eigenen FC enthält, dieser also dezentral organisiert ist. In einem Datenraum mit `n` Teilnehmern gibt es also `n` FCs. Das Gegenstück zu diesem Konzept wäre ein zentral-organisierter FC, `n` Teilnehmer greifen also auf `1` FC zu.


n Teilnehmer, n FCs
EDC_1
FC_1
FC_2
EDC_2
FC_3
EDC_3


n Teilnehmer, 1 FC
EDC_1
FC
EDC_2
EDC_3

Der FederatedCatalogApiController der FC-Extension liefert drei weitere Endpunkte zu den bereits vorhandenen. Die Beispiele sind aus Sicht eines hypothetischen Datenraumteilnehmers "LMIS" beschrieben:

2.6.1 /federatedcatalog

Dieser Endpunkt gibt die zuvor beschriebene Liste sämtlicher Datenraumteilnehmer mitsamt ihrer angebotenen Services und Assets zurück. Der EDC erwartet hierbei eine explizite Query im JSON-Body.

Beispiel:

```
POST http://<LMIS-Adresse>/management/federatedcatalog/
```

mit Body

```
{
    "@context": {
    "edc": "https://w3id.org/edc/v0.0.1/ns/"
    },
    "querySpec": {
    "offset": 0,
    "limit": 50,
    "sortOrder": "DESC",
    "sortField": "fieldName",
    "filterExpression": []
    }
}
```

Unmittelbar nach dem Start des EDC ist der FC leer, der Post-Request liefert also eine leere Liste zurück.

2.6.2 /insert

Mithilfe dieses Endpunkts könnten erste beziehungsweise weitere Datenraumteilnehmer in dem jeweils eigenen FC registriert werden.

Beispiel:

Ein neuer Datenraumteilnehmer "HSOS" ist im Besitz des EDC_2, der Datenraumteilnehmer "LMIS" mit EDC_1 möchte diesen abfragen.

```
POST http://<LMIS-Adresse>/management/federatedcatalog/insert
```

mit Body

```
{
    "name": "HSOS",
    "url": "http://<HSOS-Adresse>:8282/protocol",
    "supportedProtocols": [
        "dataspace-protocol-http"
    ]
}
```

"LMIS" kann nun seinen eigenen /federatedcatalog-Endpunkt mit der im vorherigen Beispiel gezeigten Query ansprechen und erhält den folgenden Eintrag:


```
[
    {
        "@id": "e3f5436b-ee50-4ea9-ad7d-2cc46f164105",
        "@type": "dcat:Catalog",
        "dcat:dataset": [],
        "dcat:service": {
            "@id": "135543b0-2aa1-49e8-9079-bd307020fea0",
            "@type": "dcat:DataService",
            "dct:terms": "connector",
            "dct:endpointUrl": "http://<HSOS-Adresse>:8282/protocol"
        },
        "edc:originator": "http://<HSOS-Adresse>:8282/protocol",
        "edc:participantId": "provider",
        "@context": {
            "dct": "https://purl.org/dc/terms/",
            "edc": "https://w3id.org/edc/v0.0.1/ns/",
            "dcat": "https://www.w3.org/ns/dcat/",
            "odrl": "http://www.w3.org/ns/odrl/2/",
            "dspace": "https://w3id.org/dspace/v0.8/"
        }
    }
]
```

Falls "HSOS" zusätzlich ein Asset mit ID "Messergebnis" bei sich registriert, erweitert sich der Eintrag zu:

```
[
    {
        "@id": "3d2e3930-7454-4974-82ea-8f8f2e0dc665",
        "@type": "dcat:Catalog",
        "dcat:dataset": {
            "@id": "Messergebnis",
            "@type": "dcat:Dataset",
            "odrl:hasPolicy": {
                "@id": "MQ==:TWVzc2VyZ2Vibmlz:MjBhZGM5MGMtNTdjYi00MzlmLWE3ZWMtZjlmMWQ4NzhkMTQ3",
                "@type": "odrl:Set",
                "odrl:permission": [],
                "odrl:prohibition": [],
                "odrl:obligation": [],
                "odrl:target": "Messergebnis"
            },
            "dcat:distribution": [
                {
                    "@type": "dcat:Distribution",
                    "dct:format": {
                        "@id": "HttpProxy"
                    },
                    "dcat:accessService": "135543b0-2aa1-49e8-9079-bd307020fea0"
                },
                {
                    "@type": "dcat:Distribution",
                    "dct:format": {
                        "@id": "HttpData"
                    },
                    "dcat:accessService": "135543b0-2aa1-49e8-9079-bd307020fea0"
                }
            ],
            "edc:name": "product description",
            "edc:id": "Messergebnis",
            "edc:contenttype": "application/json"
        },
        "dcat:service": {
            "@id": "135543b0-2aa1-49e8-9079-bd307020fea0",
            "@type": "dcat:DataService",
            "dct:terms": "connector",
            "dct:endpointUrl": "http://<HSOS-Adresse>:8282/protocol"
        },
        "edc:originator": "http://<HSOS-Adresse>:8282/protocol",
        "edc:participantId": "provider",
        "@context": {
            "dct": "https://purl.org/dc/terms/",
            "edc": "https://w3id.org/edc/v0.0.1/ns/",
            "dcat": "https://www.w3.org/ns/dcat/",
            "odrl": "http://www.w3.org/ns/odrl/2/",
            "dspace": "https://w3id.org/dspace/v0.8/"
        }
    }
]
```

Beachte, dass mit der Registrierung eines ersten Assets durch "HSOS" auch Informationen über dessen unterstützte Dataplanes bezüglich eben jenes Assets im FC auftauchen. Hat "HSOS" keine Dataplanes bei sich registriert, wirft der "LMIS"-EDC nach Aufruf des Endpunkts eine Fehlermeldung,

2.6.3 /participants

Dieser Endpunkt liefert eine Liste aller Datenraumteilnehmer gemäß der Attribute Name, Connector-Url und Protokoll-Spezifikation zurück, die beim Registrieren gesetzt worden sind.

Beispiel:

```
GET http://<LMIS-Adresse>/management/federatedcatalog/participants
```

mit den Datenraumteilnehmern "HSOS" und "THGA":

```
[
    {
        "name": "HSOS",
        "url": "http://<HSOS-Adresse>:8282/protocol",
        "supportedProtocols": [
            "dataspace-protocol-http"
        ]
    },
    {
        "name": "THGA",
        "url": "http://<THGA-Adresse>:8282/protocol",
        "supportedProtocols": [
            "dataspace-protocol-http"
        ]
    }
]
```

2.6.4 Konfigurationsmöglichkeiten

Es gibt insgesamt drei Eigenschaften, die in der properties-Datei des EDC bezüglich des FC gesetzt werden können:

edc.catalog.cache.execution.delay.seconds

Anzahl an Sekunden, bis der EDC initial den ersten Abfrage-Vorgang startet.

Beispiel:

```
edc.catalog.cache.execution.delay.seconds=10
```

bedeutet, dass der FC das erste Mal 10 Sekunden nach Start des EDC aktualisiert wird.

edc.catalog.cache.execution.period.seconds

Häufigkeit des Abfrage-Prozesses in Sekunden.

Beispiel:

```
edc.catalog.cache.execution.period.seconds=5
```

bedeutet, dass sich der FC alle 5 Sekunden aktualisiert (nach dem initalen Abfrage-Vorgang, s. vorheriger Absatz).

edc.catalog.cache.partition.num.crawlers

Anzahl an `Crawler`-Objekten, die die sogenannten "Work Items" (also das Tripel aus Name, Connector-Url und supportedProtocols-Liste) verarbeiten.

Beispiel:

```
edc.catalog.cache.partition.num.crawlers=2
```

bedeutet, dass im Falle von 10 Work Items jeder `Crawler` 5 Work Items abfragt.