Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Evaluation von Datenarchitekturen für die Connected-Cars-Domäne

Tim Schneider

Studiengang:	Software Engineering
Prüfer/in:	Prof. Dr.-Ing. Bernhard Mitschang
Betreuer/in:	Dr. rer. nat. Christoph Stach, Yunxuan Li, M.Sc.
Beginn am:	27. November 2023
Beendet am:	27. Mai 2024

Kurzfassung

Connected Cars sind Fahrzeuge, die mit Sensoren und Kommunikationsschnittstellen ausgestattet sind, um mit anderen Fahrzeugen, der Verkehrsinfrastruktur und dem Internet zu kommunizieren. Durch sie eröffnen sich neuartige digitale Anwendungen, die die Effizienz, Sicherheit und den Komfort von Fahrzeugen steigern können. Für die Ermöglichung dieser Anwendungen ist eine geeignete Datenarchitektur entscheidend. Sie gibt vor, wie die Fahrzeugdaten zu verwalten und zu verarbeiten sind, um die jeweiligen übergeordneten Anwendungsziele zu erreichen. Die Connected-Cars-Domäne stellt an die Datenarchitektur hohe Anforderungen. Dazu gehört beispielsweise der Umgang mit großen Datenvolumen, erzeugt von einer Vielzahl an geografisch verteilten Datenproduzenten. In dieser Arbeit werden bestehende Architekturansätze aus der wissenschaftlichen Literatur hinsichtlich ihrer Eignung als ganzheitliche Datenarchitektur für die Connected-Cars-Domäne evaluiert. Grundlage dafür ist ein Anforderungskatalog, der im Rahmen einer Anforderungsanalyse erstellt wird. Die Evaluation zeigt, dass es unter den betrachteten Veröffentlichungen keinen Vorschlag gibt, der alle definierten Anforderungen erfüllt. Im weiteren Verlauf der Arbeit wird deshalb untersucht, wie eine Datenarchitektur gestaltet werden kann, die alle Anforderungen erfüllt und damit den identifizierten Schwächen bestehender Ansätze begegnet. Das vorgeschlagene Konzept kombiniert Lösungsansätze der Literatur, allgemeine Datenarchitekturen aus dem Fachgebiet des Data Engineerings und eigene Ideen. Im Fokus steht dabei der Data Mesh, ein Ansatz zur dezentralen Organisation analytischer Daten. Es wird gezeigt, wie die Prinzipien des Data Mesh gewinnbringend auf die Connected-Cars-Domäne angewandt werden können und welche Besonderheiten sich dabei ergeben. Die größte Besonderheit ist, dass geografisch verteilte Datenprodukte unterstützt werden, die sich somit die Vorteile des Fog Computings zu Nutze machen können. Eine konzeptionelle Evaluation der Architektur am Ende der Arbeit zeigt, dass sie bis auf eine Einschränkung alle Anforderungskriterien berücksichtigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

CAN Controller Area Network. 36

CDN Content Distribution Network. 32

CoAP Constrained Application Protocol. 37

DoS Denial of Service. 27

ECU Electronic Control Unit. 15

ELT Extract Load Transform. 43

ETL Extract Transform Load. 40

FAAS Function as a Service. 60

HDFS Hadoop Distributed File System. 32

HTTP Hypertext Transfer Protocol. 31

IAAS Infrastructure as a Service. 59

IoT Internet of Things. 18

IoV Internet of Vehicles. 17

LR-WPAN Low-rate Wireless Personal Area Network. 33

MEC Multi-Access Edge Computing. 18

MQTT Message Queuing Telemetry Transport. 32

OBD On-Board Diagnostic System. 36

OEM Original Equipment Manufacturer. 18

OLAP Online Analytical Processing. 22

OLTP Online Transaction Processing. 21

OTA Over-the-Air Update. 22

PAAS Platform as a Service. 59

ProMoTe Data Product Model Template. 68

QoS Quality of Service. 33

RDBMS Relational Database Management System. 25

RDF Resource Description Framework. 36

RLE Run-length Encoding. 35

RSD Roadside Device. 18

RSU Roadside Unit. 17

SAAS Software as a Service. 60

SAPP4C Situation-aware Privacy-preserving Framework for Connected Vehicles. 7

SLA Service Level Agreement. 65

SLO Service Level Objectives. 65

SOAP Simple Object Access Protocol. 36

SQL Structured Query Language. 34

TCP Transmission Control Protocol. 31

V2C Vehicle-to-Cloud. 17

V2I Vehicle-to-Infrastructure. 17

V2V Vehicle-to-Vehicle. 17

V2X Vehicle-to-Everything. 17

YARN Yet Another Resource Negotiator. 37

1 Einleitung

Die fortschreitende Digitalisierung von Industriezweigen macht vor der Automobilindustrie nicht halt. Durch die stark anwachsende Anzahl an Electronic Control Units (ECUs)[1], eingebauten Sensoren und Aktuatoren in Fahrzeugen konnten sie in den vergangenen Jahren mit immer intelligenteren Systemen ausgestattet werden [Gre15]. Diese Systeme sind in der Lage, die Sicherheit und Effizienz im Straßenverkehr zu steigern und erhöhen auch den Fahrkomfort. Besonders weit geht hierbei die Vision des selbstfahrenden Autos, das unter Herstellern ein aktives Forschungsgebiet ist, aber in Teilen auch schon in der Praxis umgesetzt wird [LLZ+20].

Im Kontext dieses technologischen Fortschritts gewinnt das Konzept von Connected Cars rasant an Bedeutung. Connected Cars sind Fahrzeuge, die mit einer Vielzahl von Sensoren und Kommunikationsschnittstellen ausgestattet sind und somit in der Lage sind, Daten in Echtzeit zu sammeln, zu verarbeiten und auszutauschen. Durch ihren Einsatz erweitert sich das Spektrum möglicher digitaler Anwendungen für Fahrzeuge auf alle Anwendungsfälle, die eine aktive Interaktion von Fahrzeugen mit ihrer Umgebung oder räumlich entfernten Internetservices erfordern. Diese Anwendungsfälle erstrecken sich über Anwendungen im Bereich der Echtzeitkommunikation und -datenverarbeitung, aber auch auf Anwendungen, bei denen Daten für langfristige Analysen genutzt werden können. Konkrete Beispiele sind beispielsweise die Realisierung intelligenter Fahrmanöver oder Datenanalysen, die Hersteller zur langfristigen Produktverbesserung einsetzen können. Die Nutzung der Daten eröffnet dabei auch neue Geschäftszweige, die zuvor im Mobilitätssektor nicht möglich waren. Werkstätten, Fahrzeugflottenbetreiber, Verkehrsbehörden sind nur eine kleine Auswahl an Parteien, die Interessen an Fahrzeug- und Verkehrsdaten haben können [CM16].

Die Funktionalität von Connected-Cars-Systemen setzt eine leistungsfähige Datenarchitektur voraus. Sie legt fest, wie Daten verwaltet, transformiert und schließlich den Datenkonsumenten zugänglich gemacht werden. In der Connected-Cars-Domäne ergeben sich für eine solche Datenarchitektur besondere Herausforderungen. Aufgrund der großen Anzahl an Fahrzeugen und Infrastrukturgeräten, die alle als Datenproduzenten zur Verfügung stehen, ergibt sich insgesamt eine große und stetig steigende Datenmenge, die ausgehend von vielen geografisch verteilten Datenquellen verarbeitet und gegebenenfalls gespeichert werden muss. Neben der Realisierung dieser Skalierbarkeit sind Latenz, Zuverlässigkeit und Sicherheit weitere Anforderungen, die mit zunehmender Verwendung von Connected Cars an Bedeutung gewinnen [LWJZ21].

Um diesen Eigenschaften und Anforderungen gerecht zu werden, stehen verschiedene Lösungsansätze zur Verfügung. Zum einen gibt es bestehende Vorschläge für Datenarchitekturen für die Connected-Cars-Domäne in der wissenschaftlichen Literatur [SES17]. Darüber hinaus bietet das

[1] ECUs bezeichnen die elektronischen Steuergeräte in Fahrzeugen, von denen je Fahrzeug derzeit bis zu 100 Stück verbaut sind. Sie steuern jeweils dedizierte Fahrzeugfunktionen wie Brems- oder Infotainmentsysteme [Gre15].

Feld des Data Engineerings allgemeine Vorschläge für Datenarchitekturen mit unterschiedlichen Eigenschaften, die jeweils auf die Erfüllung verschiedener Aufgaben abzielen. Dazu gehören die Konzepte Data Warehouse, Data Lake, Data Lakehouse und der Data Mesh [RH22].

Das Ziel dieser Arbeit ist es, eine Datenarchitektur zu finden, die in der Lage ist, die Anforderungen der Connected-Cars-Domäne zu erfüllen. Dazu muss in einem ersten Schritt identifiziert werden, welche konkreten Anforderungen an eine solche Datenarchitektur gestellt werden. Zu diesem Zweck führe ich eine Anforderungsanalyse basierend auf Anwendungsfällen durch, die für die Connected-Cars-Domäne in der wissenschaftlichen Literatur geäußert werden. Anschließend evaluiere ich die bestehenden Lösungsansätze aus der Literatur auf die Erfüllung der gefundenen Anforderungen und bewerte auch die Eignung der allgemeinen, domänenunabhängigen Datenarchitekturen. Mithilfe der aus der Evaluation gewonnenen Erkenntnisse über Stärken und Schwächen bestehender Lösungsansätze wird nachfolgend untersucht, wie eine Datenarchitektur gestaltet werden kann, die alle Anforderungen erfüllt. Meine vorgeschlagene Architektur macht dabei Gebrauch von den Prinzipien des Data Mesh, einem Ansatz zur Organisation analytischer Daten. Ich zeige auf, wie die Prinzipien auf die Connected-Cars-Domäne gewinnbringend eingesetzt werden können und stelle für zwei Teilaspekte des entwickelten Konzepts konkrete Implementierungskonzepte sowie entwickelte Prototypen vor.

Der Aufbau der Arbeit folgt der beschriebenen Methodologie. Nach einer kurzen Vorstellung begrifflicher Grundlagen in Kapitel 2, analysiere ich in Kapitel 3 die Anforderungen an eine Datenarchitektur für die Connected-Cars-Domäne. Auf Basis des dort aufgestellten Anforderungskatalogs wird im Anschluss in Kapitel 4 die Evaluation der bestehenden Datenarchitekturen unternommen. Kapitel 5 stellt anschließend das auf dem Data-Mesh beruhende Konzept vor, um den Schwächen der bestehenden Lösungsansätze zu begegnen. Kapitel 6 befasst sich nachfolgend mit den Implementierungskonzepten für die zwei ausgewählten Teilaspekte des Konzepts. Die dazugehörigen Prototypen stelle ich in Kapitel 7 vor. Schließlich evaluiere ich in Kapitel 8, ob das beschriebene Gesamtkonzept von Kapitel 5 bis 7 in der Lage ist, alle Kriterien des Anforderungskatalogs zu erfüllen. Die Arbeit endet in Kapitel 9 mit einer Zusammenfassung, einem Fazit und einem Ausblick auf mögliche fortführende Arbeiten.

2 Grundlagen

In diesem Kapitel erkläre ich ausgewählte Begriffe, die für das Verständnis der Arbeit relevant sind. Diese sind Connected Cars und das Internet of Vehicles (Abschnitt 2.1), Edge- und Fog-Computing (Abschnitt 2.2) sowie der Begriff der Datenarchitektur (Abschnitt 2.3).

2.1 Connected Cars und das Internet of Vehicles

Connected Cars bezeichnen Fahrzeuge, die mit Sensoren und Kommunikationsschnittstellen ausgestattet sind, über die sie mit anderen Fahrzeugen und dem Internet verbunden sind. Im Rahmen dieser Arbeit wird die folgende Definition verwendet, die an eine Definition des Begriffs von Coppola und Morisio [CM16] angelehnt ist:

Definition 2.1.1 (Connected Cars)
Connected Cars sind Fahrzeuge, die ...

- *... in der Lage sind, jederzeit mit dem Internet über eingebaute oder von Fahrzeughaltern mitgeführte Geräte zu kommunizieren.*
- *... mit digitalen Anwendungen ausgestattet sind, die unter Einbezug kontextueller Informationen der Fahrzeuge, den Fahrern und Passagieren fortschrittliche Infotainment-Funktionalitäten anbieten.*
- *... mit anderen Geräten entlang der Straßeninfrastruktur kommunizieren können.*
- *... fähig sind, mit anderen Fahrzeugen über Netzwerke zu kommunizieren und zu interagieren.*

Aus dieser Definition ergeben sich verschiedene Arten der Netzwerkkommunikation, denen eigene Bezeichnungen in der Literatur zu Connected Cars zugeordnet sind. So bezeichnet Vehicle-to-Vehicle (V2V) die Kommunikation zwischen Fahrzeugen, Vehicle-to-Infrastructure (V2I) zwischen Fahrzeugen und Geräten der Verkehrsinfrastruktur und Vehicle-to-Cloud (V2C) die Kommunikation mit Internetdiensten in der Cloud [ZXCW20]. Das Ziel der Vernetzung von Fahrzeugen mit möglichst allen Geräten, die in Beziehung mit ihnen stehen und genutzt werden können, um gewinnbringende Funktionalitäten zu erhalten, wird auch als Vehicle-to-Everything (V2X) oder als das Internet of Vehicles (IoV) bezeichnet [JZM+20].

Für die Umsetzung der Kommunikation der Fahrzeuge mit ihrer Umgebung werden in der Literatur beispielsweise Ad-Hoc-Netzwerke und zelluläre Netzwerktechnologien diskutiert [TK22]. Roadside Units (RSUs) bezeichnen im Kontext zellulärer Netzwerke Infrastrukturgeräte in der Nähe des Straßennetzwerkes, die den Mobilfunkbasisstationen unterstützend zur Verfügung stehen. Sie sollen die Netzwerkabdeckung und Bandweiten erhöhen sowie Latenzen verringern [FDA+18]. Darüber hinaus können sie Internetzugang bereitstellen oder als Host für Anwendungen intelligenter

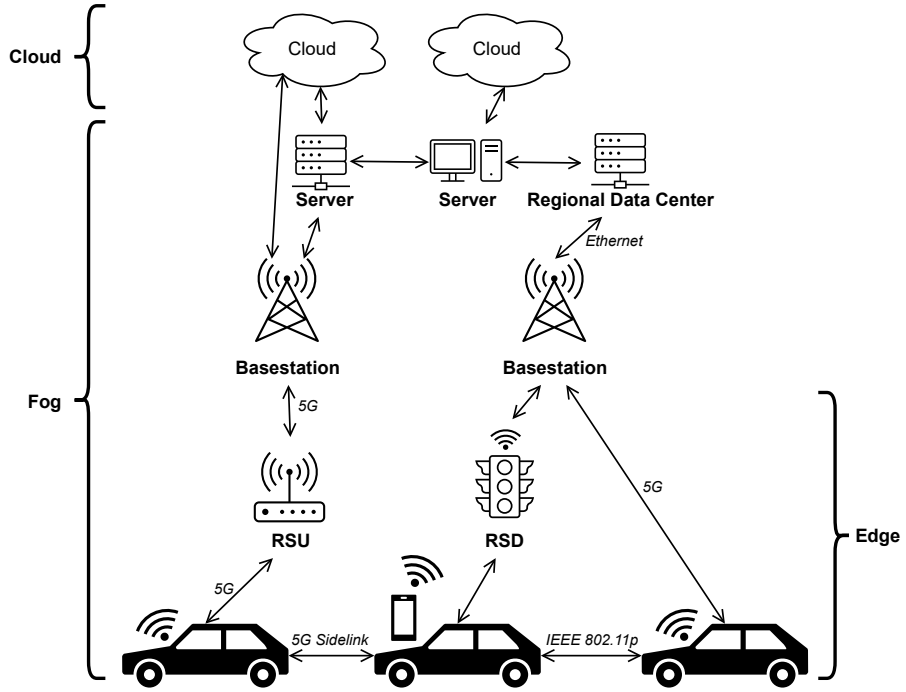


Abbildung 2.1: Beispielhafte Übersicht über Geräten und eingesetzte Technologien im IoV.

Verkehrssysteme dienen [GBC22]. In dieser Arbeit spreche ich allgemein von Roadside Devices (RSDs), wenn ich alle Arten von Geräten entlang der Straßeninfrastruktur meine, die mit Fahrzeugen kommunizieren. Dazu gehören neben RSUs zum Beispiel auch verbundene Ampelanlagen, die selbst nicht zwingend eine Anbindung an zelluläre Netzwerke für andere Geräte ermöglichen müssen.

Unternehmen, die Geräte für das IoV herstellen und daher für die Umsetzung von Connected-Cars-Diensten eine entscheidende Rolle spielen, werden als Original Equipment Manufacturer (OEM) bezeichnet [LWJZ21]. Dies können zum Beispiel Autohersteller sein, aber auch Netzwerkbetreiber, die sich mit der Realisierung von Mobilfunk- und RSU-Infrastruktur für Connected-Cars-Systeme auseinandersetzen.

2.2 Edge- und Fog-Computing

Edge Computing bezeichnet Technologien, die es erlauben, Berechnungen am Rande eines Internet of Things (IoT)-Netzwerkes durchzuführen. Dies kann entweder auf Initiative der IoT-Geräte oder auf Initiative von Cloud-Services geschehen. Im ersten Fall verarbeiten IoT-Geräte lokale Daten und stellen die Ergebnisse dann der Cloud zur Verfügung. Im zweiten Fall verwendet die Cloud die Edge-Geräte, um ihre Daten zu verarbeiten und gegebenenfalls lokalen Datenkonsumenten zur Verfügung zu stellen [DTD19; SCZ+16].

In meiner Auslegung für das IoV zähle ich Fahrzeuge und RSDs zu den Geräten am Rand des Netzwerks. Diese Einsortierung ist auch in Abbildung 2.1 zu sehen. Im Rahmen des sogenannten *Multi-Access Edge Computing (MEC)* werden große Basisstationen für den Mobilfunk jedoch auch noch zur Edge gezählt und als Verarbeitungsort verwendet [DTD19].

Beim *Fog Computing* beschränkt sich die lokale Berechnung nicht nur auf Edge-Geräte. Rechenkomponenten zwischen Edge und Cloud stellen hier ebenso ihre Rechenressourcen zur Verfügung. Es handelt sich somit um eine Erweiterung des Konzepts des Edge Computings [DTD19].

In Abbildung 2.1 führe ich Basistationen des Mobilfunks und sonstige regionale Datenzentren und Server als Beispiele für Fog-Komponenten im IoV auf, die nicht schon von der Edge abgedeckt werden.

2.3 Definition: Datenarchitektur

In der Literatur finden sich verschiedene und zum Teil stark abweichende Definitionen des Begriffs Datenarchitektur [RH22]. Für diese Arbeit wird die folgende Definition verwendet, die sich an Definitionen von Reis und Housley [RH22] sowie IBM [IBM24] anlehnt:

Definition 2.3.1 (Datenarchitektur)
Eine Datenarchitektur ist das Design von Systemen zur Unterstützung der sich fortlaufend entwickelnden Datenbedürfnisse eines Unternehmens. Sie legt fest, wie Daten erfasst, verwaltet, transformiert und schließlich Datenkonsumenten zugänglich gemacht werden. Dabei liefert sie einen Entwurf, wie verschiedene Datenverwaltungs und -verarbeitungssysteme zusammenarbeiten.

Um die Definition auf die Connected-Cars-Domäne anzuwenden, muss zunächst klar sein, wer konkret das Unternehmen darstellt und welche Datenbedürfnisse es besitzt. Im Rahmen dieser Arbeit soll der Fokus auf Fahrzeugherstellern liegen. Als OEM der Fahrzeuge haben sie die umfangreichsten Möglichkeiten, Connected-Cars-Services in ihre Produkte zu integrieren. Der Grund hierfür ist, dass Fahrzeuge im IoV die Hauptdatenerzeuger darstellen. Somit fallen aufseiten der Fahrzeughersteller die größten Herausforderungen an, wie die Daten der Fahrzeuge zu verwalten und zu verarbeiten sind, um Connected-Cars-Funktionen zu realisieren. In dieser Arbeit wird zusätzlich die Möglichkeit offengelassen, dass die OEMs, abseits von Fahrzeugen, auch Zugriffe auf statische IoV-Infrastruktur wie RSUs oder allgemein Rechenkomponenten in der Fog haben. Entweder, weil sie diese Geräte selbst zur Verfügung stellen oder weil ihnen ein anderes Unternehmen operativen Zugriff auf diese gewährt. Fog-Geräte können somit ebenso Teil der technischen Infrastruktur zur Realisierung der Datenarchitektur sein.

Die Datenerhebung und -verwaltung zur Realisierung der eigenen Connected-Cars-Services ist eines der Datenbedürfnisse der Fahrzeughersteller. Diese Services werden jedoch nicht nur von Entscheidungsträgern der Autoherstellerfirma, sondern auch von ihren Kunden verwendet. Somit lassen sich auch aus der Sicht von Fahrzeughaltern oder externen Firmen, die ein Interesse an den Daten haben, Datenbedürfnisse formulieren. Welche konkreten Datenbedürfnisse, beziehungsweise Anforderungen an eine Datenarchitektur, für die Interessensparteien der Daten von Bedeutung sind, wird im nächsten Kapitel 3 im Rahmen einer Anforderungsanalyse behandelt.

3 Anforderungsanalyse

Eine Datenarchitektur in der Connected-Cars-Domäne muss in der Lage sein, alle Datenbedürfnisse der beteiligten Systemnutzer zu erfüllen. Dazu gehören Fahrzeughalter, der OEM der Fahrzeuge selbst, aber auch externe Unternehmen, die Geschäftsmodelle auf der Basis von Connected-Cars-Daten realisieren wollen.

Das erste Ziel der Anforderungsanalyse dieses Kapitels ist es deshalb, zu identifizieren, welche Anwendungsfälle von der Datenarchitektur ermöglicht werden müssen. In Abschnitt 3.1 stelle ich dazu Anwendungsfälle vor, die in der wissenschaftlichen Literatur für die Connected-Cars-Vision beschrieben werden. Im Anschluss diskutiere ich in Abschnitt 3.2 die technischen Anforderungen an die Datenarchitektur, die zur Realisierung dieser Anwendungsfälle notwendig sind. Mittels eines Anforderungskatalogs in Abschnitt 3.3 werden diese Anforderungen schließlich zu konkreten Kriterien zusammengefasst. Sie dienen als Grundlage der Evaluation bestehender Datenarchitekturvorschläge für die Connected-Cars-Domäne in Kapitel 4.

3.1 Anwendungsfälle

Für die Beschreibung der Anwendungsfälle wird eine datenorientierte Sicht eingenommen. Dabei steht im Vordergrund zu erfassen, welchen Nutzungszweck die Daten innerhalb ihrer jeweiligen Anwendungsfälle erfüllen sollen. Unterschieden wird hierbei zwischen operationalen und analytischen Daten. Diese Unterteilung basiert auf einer Idee von Dehghani [Deh22] und wird im Folgenden an die Connected-Cars-Domäne angepasst.

Alle beschriebenen Anwendungsfälle stammen aus insgesamt 13 wissenschaftlichen Veröffentlichungen und einem Whitepaper ([5GP15]). Die Auflistung erhebt keinen Anspruch auf Vollständigkeit und führt lediglich die Bandbreite möglicher Anwendungen vor Augen.

3.1.1 Verwendungszwecke operationaler Daten

Operationale Daten sind Daten, die in einem System benötigt werden, um den operationalen Betrieb der bereitgestellten Kernfunktionalitäten am Laufen zu halten. Dabei repräsentieren sie meist den aktuellen Zustand, in dem sich eine Systementität befindet, und haben daher eine zeitlich begrenzte Relevanz. Die Daten werden im Rahmen von einfachen Datenbankabfragen oder schnell ausführbaren Programmkontrollstrukturen verwendet. Online Transaction Processing (OLTP) ist ein Beispiel für eine Verwendungsart operationaler Daten [Deh22].

Typische Anwendungsfälle innerhalb der Connected-Cars-Domäne, in denen operationale Daten zum Einsatz kommen, finden sich im Bereich der V2V- oder V2I-Kommunikation. Hier werden Daten in Echtzeit zwischen Verkehrsteilnehmern oder sonstigen Geräten der Verkehrsinfrastruktur,

wie RSUs, ausgetauscht, um gemeinsame Ziele zu erreichen. Dazu gehören autonome Fahrmanöver wie an intelligenten Kreuzungen, Platooning [SES17], kooperative Fahrbahnwechsel [GSS+18; LWJZ21; WWG+19] und Mechanismen zur Kollisionsvermeidung [5GP15; SES17]. Remote Sensing und Bird's-Eyes-View beschreiben darüber hinaus die Möglichkeit, Sensordaten eines Fahrzeugs oder eines RSD, einem anderen Fahrzeug als ergänzende Informationsquelle bereitzustellen. Das kann beispielsweise bei eingeschränkten Sichtverhältnissen des Fahrers hilfreich sein, der dann auf Echtzeit-Videostreams der Umgebung zugreifen kann [GSS+18]. Im Rahmen von Fahrerassistenzsystemen werden externe Datenstreams sowie Verkehrswarnungen auch verwendet, um mittels Augmented Reality dem Fahrer Gefahrensituationen zu markieren, die die fahrzeugeigenen Sensoren sonst nicht erkennen können [5GP15].

Daten im operationalen Betrieb der Connected-Cars-Domäne kommen nicht nur in Form von Sensordaten oder Statusmeldungen vor, sondern auch als Kommandos für Aktuatoren. Das ferngesteuerte Auf- und Abschließen von Autos [SES17] und das Sperren von Autofunktionalitäten nach einem Diebstahl [CM16] gehören hier als Anwendungsfälle dazu. Etwas weitreichender ist sogenanntes Tele-Operated-Driving. Bei diesem kann ein Auto über das Mobilfunknetzwerk entweder manuell von einem Menschen oder von einer automatischen Software in schwierigen oder gefährlichen Verkehrssituationen ferngesteuert werden [HMKA19]. Over-the-Air Updates (OTAs) können auch als eine Form von Aktuation gesehen werden. Sie ermöglichen es, die Software von Fahrzeugen oder der Infrastruktur aktuell zu halten, um Fehlfunktionen zu beheben oder den Nutzern neue Funktionalitäten anzubieten [BJH17].

Vielfältige Verwendungsmöglichkeiten operationaler Daten gibt es auch bei Infotainment-Applikationen der Fahrzeuge. Dazu gehören Video- und Musikstreamingdienste sowie erweiterte Social-Media-Angebote, die eine Vernetzung zwischen Fahrern ermöglichen. Fahrer können Nachrichten an andere Fahrer in ihrer Umgebung senden, um sie auf Probleme ihres Fahrzeugs aufmerksam zu machen, oder Voicechat-Räumen beitreten, um sich zu unterhalten [5GP15; CM16]. Fahrzeughalter können außerdem auf hochauflösende geografische Karten der Umgebung zugreifen. Sie werden von lokalen Datenzentren angeboten und fortlaufend mit aktuellen Informationen aktualisiert [GSS+18].

Operationale Daten für Connected Cars können auch genutzt werden, um Geschäftskonzepte dritter Unternehmen zu realisieren. Dazu zählen zum Beispiel Pay-as-You-Drive-Geschäftsideen zur Realisierung von Car-Sharing-Systemen [CM16] oder Parkplatzsuchdiensten [SES17].

3.1.2 Verwendungszwecke analytischer Daten

Analytische Daten dienen dazu, gewinnbringende Erkenntnisse auf der Basis von statistischen Auswertungen oder der kontinuierlichen Überwachung des produktiven Systems zu erlangen. Sie kommen häufig in der Form größerer Datensätze vor und stammen oft aus der Historisierung oder Aggregation operationaler Daten. Im Gegensatz zu operationalen Daten besteht bei ihnen ein vermindertes Interesse zur Erfüllung von Echtzeitfunktionalitäten. Stattdessen steht der Gewinn von Erkenntnissen im Fokus, die Situationen vorhersagen oder rückblickend bewerten können. Analytische Daten werden innerhalb von Online Analytical Processing (OLAP)-Systemen verarbeitet [Deh22]. Reis und Housley unterscheiden zwischen drei grundlegenden Arten von Analytics-Anwendungen: Business Analytics, Operational Analytics und Embedded Analytics [RH22].

Business Analytics beschäftigt sich mit der Verbesserung von internen Geschäftsprozessen [RH22]. Im Falle der Connected-Cars-Domäne sind das die Systeme und Produkte der OEMs. Historische Leistungsmetriken von einzelnen Fahrzeugen können hier erfasst werden, um beispielsweise Produktverbesserungen für zukünftige Fahrzeugmodelle zu planen. Konkrete Anwendungsfälle aus der Literatur sind hierfür die Verbesserung von Autobatterien [FHS+23] oder die Evaluation der Funktionstüchtigkeit von Emissionskontrollsystemen bei Dieselfahrzeugen [Saa22]. Produktverbesserungen können nicht nur in den Bereich der Hardware fallen, sondern auch Softwareprodukte betreffen, die mittels Nutzungsstatistiken evaluiert werden können.

Das Ziel von Operational Analytics ist die Überwachung des produktiven Systems, um bestenfalls vorausschauend auf Probleme reagieren zu können [RH22]. Diagnostisches Datenmonitoring der Autos eines OEM kann dazu dienen, Autos auf Fehler wie Motorschäden zu überprüfen und die Fahrer zu benachrichtigen, sollte ein technischer Defekt erkannt oder vorhergesagt werden [5GP15]. Weiter kann das Fahrverhalten einzelner Fahrzeughalter analysiert werden, um beispielsweise automatische Falschfahrermeldungen herauszugeben oder vor ermüdeten Fahrern zu warnen [SES17]. Es kann auch dafür gesorgt werden, dass die Langlebigkeit der Autos oder ihr Fahrtkomfort erhöht wird. Dies wird zum Beispiel mit einer automatischen Anpassung der Federungshärte an die vorausliegende Fahrbahnbeschaffenheit möglich [Saa22]. Operational Analytics ist wie Business Analytics auf die Optimierung interner Vorgänge eines Unternehmens fokussiert. Dazu gehört auch die Überwachung der Connected-Cars-Infrastruktur, um beispielsweise Netzwerkprobleme frühzeitig zu erkennen und auf sie reagieren zu können.

Embedded oder External Analytics bezeichnet Analytics-Dienste, die in Form von Datenapplikationen direkt den Systemnutzern oder anderen Parteien angeboten werden [RH22]. Ein Großteil der analytischen Anwendungsfälle der Connected-Cars-Vision ist in diese Kategorie verortbar. Mittels APIs oder dedizierten Applikationen sollen Fahrzeughalter in der Lage sein, auf Metriken zu ihrem Auto zugreifen zu können. Dabei sind sowohl Live-Dashboards [BJH17], als auch Langzeitberichte relevant. Letztere können über alle historischen Fahrten berichten oder dem Fahrer Tipps für ein energieschonenderes Fahrverhalten geben [LWJZ21]. Möchte ein Fahrzeughalter sein Auto verkaufen, so kann ein auf der Basis von historischen Diagnosedaten berechneter Autowert helfen [CM16]. Die Navigationskarten der Autos können nicht nur mit Live-Daten aktuell gehalten werden, sondern auch historische Analyseergebnisse bereitstellen, die Fahrbahnzustände, übliche Rush-Hours oder Risikogebiete für Unfallereignisse in die Empfehlungen von Navigationsrouten einfließen lassen. [CM16; FHS+23; Saa22].

Neben den OEMs sind viele weitere externe Unternehmen denkbar, die ein Interesse an der Verarbeitung der analytischen Autodaten haben. Musik-Streamingdienste können mittels Nutzerdaten Musik der landschaftlichen Umgebung und dem persönlichen Geschmack des Fahrzeughalters anpassen. Auch kontextuelle Werbung ist hier einsetzbar [CM16]. Wetterdienste werden mit detaillierten Wetterdaten befähigt, hochauflösende Vorhersagen für einzelne Straßenabschnitte zu liefern und auch Aussagen über die konkrete Fahrbahnbeschaffenheit zu treffen [MLKS18]. Behörden können Daten der Connected Cars für Verkehrsüberwachung und Stauvorhersagen nutzen oder mittels eines Remote-Zugriffs auf Dashboard-Kameras nach bestimmten Fahrzeugkennzeichen fahnden [LWJZ21; VVE+17]. Logistikunternehmen können mit bereitgestellten Daten ohne zusätzliche Hardware ihre Fahrzeugflotte verwalten sowie ihre Prozesse und Routen optimieren. Ähnliches gilt für Dienstleister intelligenter Transportsysteme [5GP15; SES17].

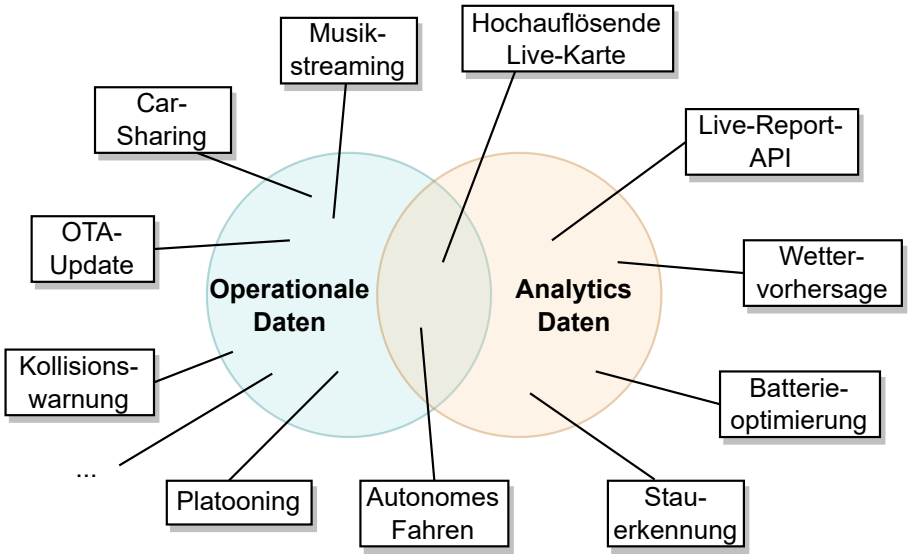


Abbildung 3.1: Gruppierung von Anwendungsfällen der Connected-Cars-Vision nach der Art ihrer Datennutzung. Anwendungsfälle, die sowohl von analytischen, als auch operationalen Daten Gebrauch machen, befinden sich in der dargestellten Schnittmenge.

Zuletzt sind analytische Daten auch notwendig, um statistische Modelle, wie solche im Bereich des Machine Learnings, zu erstellen und zu verbessern. Viele operationale Funktionalitäten zukünftiger Autos basieren auf solchen Modellen. Dabei müssen diese nicht einmal zwingend Gebrauch von der Connected-Cars-Technologie machen. Zu nennen sind hier beispielsweise autonomes Fahren [5GP15] und die Sprachsteuerung von Fahrzeugfunktionen [FHS+23].

3.1.3 Hybride Anwendungsfälle

Die Einordnung einzelner Anwendungsfälle in operationale und analytische Datennutzung ist nicht immer trennscharf. Je nachdem, welche Granularität man für die Beschreibung eines Anwendungsfalls wählt, können für diesen sowohl operationale als auch analytische Daten von Relevanz sein. Sieht man beispielsweise das Erstellen von Machine-Learning-Modellen als einzelnen Anwendungsfall an, ist dieser klar in den Bereich der Nutzung analytischer Daten zu verordnen. Bezieht man in den Anwendungsfall jedoch auch die produktive Nutzung des ausgebrachten Modells ein, werden operationale Daten benötigt, die eine in Echtzeit bereitgestellte Vorhersage durch das Modell anstoßen. Ein weiteres Beispiel sind die im obigen Abschnitt beschriebenen hochauflösenden Karten. Diese bieten Funktionalitäten, die rein über operationale Daten umgesetzt werden können, wie die Beantwortung der Frage, wo sich aktuell ein Fahrzeug befindet und welche Navigationsanweisungen als nächstes gegeben werden müssen. Gleichzeitig werden aber auch Funktionalitäten bereitgestellt, die eine zuvorige Auswertung historischer oder großflächig aggregierter Daten erfordern, die dann in den Bereich von analytischen Daten fallen. Dazu gehört beispielsweise die Berechnung von Risikoindizes für Straßenabschnitte, um eine möglichst sichere Navigationsroute vorzuschlagen. Anwendungsfälle, bei denen aufgrund ihrer Grobkörnigkeit eine klare Einordnung nicht sinnvoll ist, werden in dieser Arbeit als hybrid bezeichnet. In Abbildung 3.1 befinden sich diese Anwendungsfälle genau in der Schnittmenge von operationaler und analytischer Datennutzung.

3.2 Anforderungen an die Datenarchitektur

In diesem Kapitel stelle ich Anforderungen auf, die von einer Datenarchitektur für Connected-Cars-Domäne erfüllt werden müssen, um die im vorherigen Abschnitt 3.1 vorgestellten Anwendungsfälle umsetzen zu können. Die vier identifizierten Kernanforderungen sind Big-Data-Fähigkeit, die Berücksichtigung von Data Governance, verteiltes Infrastruktur- und Datenmanagement sowie Interoperabilität und Mandantenfähigkeit.

3.2.1 Big-Data-Fähigkeit

In der Connected-Cars-Domäne agiert jedes an das System angebundene Fahrzeug als potenzielle Datenquelle. Dadurch wächst die zu verarbeitende Datenmenge bei steigender Verbreitung von Connected Cars schnell an. Die Datenarchitektur muss folglich gewährleisten, dass mit sehr großen, potenziell stetig ansteigenden Datenmengen umgegangen werden kann. Technologien, die solche Eigenschaften besitzen, werden häufig als Big-Data-Technologien bezeichnet. Für sie stellen sich Anforderungen an die Dateninfrastruktur, die weit über die Verwendung traditioneller Datenmanagementsysteme wie Relational Database Management System (RDBMS) hinausgehen [DDM14]. Es gibt verschiedene Ansätze zur genauen Definition von Big Data [DGDM13]. Diese Arbeit stützt sich auf einen Ansatz der Gartner Inc. [Gar], die eine Definition mittels dreier Eigenschaften vorsehen (auch bekannt als die *3Vs*).

Volume

Connected Cars erzeugen in ihrem Betrieb fortlaufend Sensor-Daten, die neben der Erfüllung operationaler Anwendungen auch einen Wert für eine weitergehende Datenanalyse haben können. Im Jahr 2015 erzeugten laut einem Rechenbeispiel von Wollschläger [Wol16] 60 Millionen Connected Cars eine jährliche Datenmenge von 438 Exabyte. Die Datenarchitektur der Connected-Cars-Systeme, verteilt über alle Fahrzeuge, RSDs und Datenzentren, muss in der Lage sein, mit Datenmengen in dieser Größenordnung umzugehen, ohne Einbußen in der Funktion und Verfügbarkeit der Softwareanwendungen zu erhalten. Dazu muss sichergestellt werden, dass die Netzwerke zur Datenübertragung ausreichende Bandbreiten zur Verfügung stellen. Darüber hinaus müssen geeignete Speichermechanismen und -ressourcen bereitgestellt werden, die die Persistierung der operationalen und analytischen Daten über kurze und lange Zeiträume ermöglichen. Zuletzt müssen die Datenmengen auch verarbeitet werden können. Besonders bei den umfangreichen Datensätzen analytischer Daten benötigt es hierfür geeignete Konzepte und Tools.

Velocity

Die Geschwindigkeiten, mit denen die Daten verarbeitet werden müssen, reichen von nahezu Echtzeit bis hin zu Zeithorizonten von Stunden, Tagen oder Monaten. Vor allem die Anwendungsfälle mit operationalen Daten benötigen niedrige Nachrichtenlatenzen, um ihre zeitkritischen Ziele zu erreichen. Eine Übertragung von Echtzeitstreams der erzeugten Sensordaten ist aber beispielsweise auch für die Live-Reports der analytischen Datenerfassung von Relevanz. Darüber hinaus ist es für die Übertragung analytischer Daten oft ausreichend, wenn diese in Form von Batches in größeren

Zeitintervallen zu ihrem Verarbeitungs- oder Speicherort übertragen werden. Die Sensordaten, die für solche Zwecke erhoben werden, werden dennoch in hoher Geschwindigkeit erzeugt und müssen deshalb in jedem Fall schnell vorverarbeitet oder zwischengespeichert werden können. Des Weiteren gibt es Anwendungsfälle, die eine hybride Form der Datenverarbeitung benötigen, indem sie Datenstreams mit den Ergebnissen zeitunkritischer Berechnungen kombinieren. Ein Beispiel hierfür ist der Anwendungsfall der hochauflösenden Navigationskarten.

Variety

Daten in der Connected-Cars-Domäne sind heterogen bezüglich ihrer Struktur. Neben Sensorwerten, die mittles simplen Key-Value-Paaren von Fließkommazahlen repräsentiert werden können (zum Beispiel Geschwindigkeiten), haben geografische oder graphenbasierte Datenstrukturen eine erhöhte Bedeutung. Das ist insbesondere bei Kartenanwendungen der Fall. Darüber hinaus werden unstrukturierte Daten in Form von Bild-, Video-, Audio- oder LIDAR-Dateien erzeugt. Die Datenarchitektur muss eine Speicherung und Verarbeitung dieser heterogenen Daten in geeigneten Datenformaten ermöglichen.

3.2.2 Berücksichtigung von Data Governance

Es ist nicht ausreichend, eine Datenarchitektur zu entwerfen, die einzig die technischen Aspekte der Datenverarbeitung berücksichtigt. Damit die Daten überhaupt einen Wert für die Erfüllung ihrer jeweiligen Anwendungsfälle haben, müssen auch Qualitätseigenschaften erfüllt werden. Gleichzeitig besteht ein Interesse seitens des OEMs, sowie ein persönliches Interesse der Datenerzeuger, dass die Daten Sicherheits- und Datenschutzbestimmungen erfüllen [LHSM22]. Solche Eigenschaften von Datenarchitekturen, die das Ziel verfolgen, den Wert der Daten unter Einhaltung von adäquaten Sicherheitsmechanismen zu maximieren, werden nach Reis und Housley [RH22] unter dem Begriff Data Governance zusammengefasst. Bei der Berücksichtigung von Data Governance sollen durch die Datenarchitektur insbesondere folgende Teilanforderungen erfüllt werden, die von Reis und Housley [RH22] als Eigenschaften guter Data Governance aufgeführt werden:

Discoverability

Daten können nur verwendet werden, wenn sie innerhalb einer Organisation auffindbar sind. Datenkonsumenten, wie Entwickler einzelner Connected-Car-Anwendungsfälle, benötigen Mechanismen, die es erlauben, alle für sie relevanten Datenpunkte zu finden und Zugriff auf diese zu erlangen. Darüber hinaus soll es nachvollziehbar sein, woher die Daten kommen, wie sie mit anderen Daten in Beziehung stehen und welche realen Gegebenheiten sie abbilden. Metadatenmanagement ist nach Reis und Housley [RH22] eine Möglichkeit, Discoverability zu ermöglichen.

Data Accountability

Es muss ersichtlich sein, welche Personengruppe oder Organisation zuständig für die Bereitstellung von bestimmten Daten ist. Klare Verantwortlichkeiten sind wichtig, um die Datenqualität und die Erfüllung der Governance-Eigenschaften sicherzustellen [RH22].

Data Quality

Genauigkeit, Vollständigkeit und Aktualität sind nach Eryurek et al. [EGL+21] die drei Hauptmerkmale guter Datenqualität. Genauigkeit stellt sicher, dass die Daten korrekt die Wirklichkeit abbilden, aus der sie stammen [EGL+21]. Fehler bei Messungen, Dateninkonsistenzen sowie Datenduplikate sollen möglichst schon bei der Datenerzeugung vermieden werden. Die Datenarchitektur soll es daher ermöglichen, bei Bedarf Datenbereinigungsschritte in den Prozess der Datenerfassung zu integrieren, um akkurate Daten für die Verarbeitung zu erhalten. Fehler und vermehrt auftretende Ungenauigkeiten müssen den Datenkonsumenten transparent kommuniziert werden.

Unvollständige Daten sind insbesondere bei dem Trainieren von statistischen Modellen ein Problem. Hier besteht die Gefahr, dass aufgrund nicht ausreichend repräsentativer Daten Vorurteile erlernt werden [SPV22]. Die Datenarchitektur soll Mechanismen vorsehen, die es ermöglichen, unvollständige Daten zu erkennen und Strategien anzubieten, mit diesen Unvollständigkeiten umzugehen. So könnten zum Beispiel Ersatzdaten anderer Fahrzeuge des gleichen Modells für analytische Zwecke angeboten werden, wenn ein bisher verwendetes Fahrzeug als Datenquelle ausfällt.

In der Connected-Cars-Domäne sind sowohl zeitliche, als auch räumliche Aktualität der Daten entscheidend für ihren Wert. Die Datenarchitektur soll in der Lage sein, den Datenlebenszyklus an die unterschiedlichen Anwendungsfälle anpassen zu können. Daten, bei denen der berechtigte Verdacht besteht, dass sie in Zukunft wieder Relevanz erlangen könnten, sollen über längere Zeiträume in ihrem Rohformat gespeichert werden können. Andere Daten, wie im operationalen Betrieb der V2V-Kommunikation sind nur für eine zeitlich begrenzte Verkehrssituation von Interesse. Auch geografisch können sie beispielsweise nur für umliegende Fahrzeuge relevant sein. Solche Daten sollen nicht unnötig persistiert oder durch das Netzwerk transportiert werden, wenn sie dadurch ohnehin an Wert verlieren.

Security und Privacy

Die Vielzahl an beteiligten Geräten, Technologien und Netzwerken in der Connected-Cars-Domäne stellt eine große Angriffsfläche dar [DB18]. Datenintegrität, Vertraulichkeit, Authentifikation und Autorisierung für Ressourcenzugriffe sind Eigenschaften, die für alle Komponenten der Datenarchitektur gewährleistet werden müssen [GDO+17]. Dies gilt für Software wie auch für die Hardwareinfrastruktur. Auch die gezielte Beeinträchtigung von Kommunikation, etwa durch Denial of Service (DoS)-Attacken, ist eine zu verhindernde Gefahr [SS13].

Zusätzlich zu allgemeinen Sicherheitsmaßnahmen muss die Datenarchitektur Regularien des Datenschutzes umsetzen. Fahrzeugdaten sind persönliche Daten der Fahrzeugführer, die nur unter bestimmten Umständen zur Datenverarbeitung genutzt werden dürfen [FHS+23]. Fahrzeughalter sollen einwilligen können, welche Daten sie mit dem OEM oder anderen Parteien teilen wollen und unter welchen Voraussetzungen. Zu solchen Voraussetzungen können spezifische geografische Gebiete der Datenverarbeitung gehören sowie Anonymisierungsanweisungen.

3.2.3 Verteiltes Daten- und Infrastrukturmanagement

Die Geräteinfrastruktur der Connected-Cars-Domäne zeichnet sich durch eine starke geografische Verteilung aus. Einzelne Systemkomponenten müssen und können nicht ausschließlich auf zentralen Datenzentren installiert werden. Stattdessen sind Fahrzeuge, RSDs und lokale Datenzentren zusätzliche Rechenknoten, die als Teil der Architektur berücksichtigt werden müssen. Anwendungsfälle der V2V- und V2I-Kommunikation bedingen beispielsweise Datenverarbeitungsaufgaben, die nur auf den Fahrzeugen und RSDs selbst zu lösen sind [DB18]. Aber auch für analytische Anwendungsfälle werden zum Beispiel lokale Datenzentren vorgesehen, die hochauflösende Kartendaten ihrer geografischen Region bereitstellen [WWG+19].

Eine Datenarchitektur muss deshalb Möglichkeiten vorsehen, ihre zugrundeliegende Infrastruktur zu verwalten und Softwareservices auf einzelne Rechenknoten auszubringen. Um die Erweiterbarkeit der Architektur für zukünftige Anwendungsfälle offen zu halten und um Wartungen zu ermöglichen, ist es nicht ausreichend, wenn die Infrastruktur lediglich bei initialer Inbetriebnahme mit Softwarekonfigurationen verstehen wird. Stattdessen wird eine Form des kontinuierlichen Deployments (Continous Deployment) benötigt. Im Falle der Fahrzeuge können dies beispielsweise OTA-Updates sein. Entwickler der Connected-Cars-Anwendungsfälle benötigen Werkzeuge und Prozesse, die es ihnen ermöglichen, solche Updates für ihre Applikationen anzustoßen.

Neben der allgemeinen Verwaltung der technischen Infrastruktur und ihrer Software müssen auch Zugriffsmechanismen auf die verteilten Daten dieser Infrastruktur geschaffen werden. Die Architektur muss festlegen, welche Schnittstellen zur Datenkommunikation vorhanden sein sollen und wie Entwickler auf diese zugreifen. Beispielsweise muss es möglich sein, gezielt auf Daten bestimmter Infrastrukturgeräte zugreifen zu können, wie etwa die aktuellen Ampelphasen von verbundenen Ampelanlagen. Es muss außerdem transparent sein, welche Rechenknoten welche Ressourcen zur Verarbeitung oder Speicherung dieser Daten besitzen und wie diese Ressourcen im Zusammenspiel verwendet werden können.

3.2.4 Interoperabilität und Mandantenfähigkeit

Interoperabilität bezieht sich auf das Zusammenspiel aller Komponenten und organisationsübergreifende Nutzbarkeit der Daten der Datenarchitektur. Die große unabgeschlossene Menge an möglichen Anwendungsfällen für die Connected-Cars-Domäne macht es notwendig, dass verschiedenste Parteien und Interessensgruppen Connected-Cars-Anwendungen entwickeln oder die erhobenen Daten nutzen wollen. Zunächst sind hierbei die verschiedenen Unterabteilungen der OEMs zu nennen. Es ist unrealistisch anzunehmen, dass die gesamte Datenarchitektur mit ihren Applikationen von nur einem einzigen Entwicklerteam entwickelt werden kann. Wahrscheinlicher ist es, anzunehmen, dass viele verschiedene Teams an der Entwicklung des Systems beteiligt sind. Es ist daher wichtig, dass die Datenarchitektur geteilte Komponenten, Werkzeuge und Richtlinien vorgibt, mit deren Hilfe die Entwicklerteams die Applikationen ihrer Subdomäne bauen können. Durch diese Vorgaben der Architektur soll gewährleistet werden, dass die Komponenten untereinander interoperabel sind, also gemeinsam produktiv funktionieren können. Dies ist insbesondere für die Erweiterbarkeit des Systems relevant. Das Ziel dieser organisationsinternen Interoperabilität ist die Möglichkeit, erhobene oder verarbeitete Daten zwischen den Zuständigkeiten verschiedener Entwicklerteams austauschen zu können, ohne zunächst grundlegende neue Systemanpassungen zur Kompatibilitätsherstellung durchführen zu müssen.

Neben interner Interoperabilität müssen auch externe Unternehmen auf die Daten der Infrastruktur zugreifen können, um ihre jeweiligen Applikationen zu entwickeln. Hierfür sind Mechanismen notwendig, die es diesen Parteien erlauben, Zugriff auf Daten zu erlangen oder ihre eigenen Applikationen der Dateninfrastruktur wieder zugänglich zu machen. Vor allem für diesen Aspekt kann es erforderlich sein, dass auch die Connected-Cars-Infrastruktur von mehreren Parteien geteilt werden muss. So könnten Musikstreaming-Anbieter ebenso von OTA-Updates Gebrauch machen, um ihre Streamingsoftware auf den Fahrzeugen zu aktualisieren. Als weiteres Beispiel benötigen verschiedene Autohersteller möglicherweise operativen Zugriff auf die gleichen RSDs eines bestimmten OEM und wollen ihren Kunden einzigartige Services über diese anbieten. Das ist insbesondere bei Anwendungen des MEC der Fall [WWG+19]. Die Notwendigkeit, den Zugriff auf technische Infrastruktur und deren Ressourcen mit verschiedenen Parteien zu teilen, wird durch die Anforderung der Mandantenfähigkeit ausgedrückt.

3.3 Anforderungskatalog

Damit der Erfüllungsgrad der Anforderungen aus Abschnitt 3.2 für Architekturvorschläge einfacher und nachvollziehbar bewertet werden kann, konkretisiere ich sie in Form von Kriterien eines Anforderungskatalogs. Jeder der vier Kernanforderungen ordne ich dabei eine Liste von Unteranforderungen zu. Diese Unteranforderungen müssen von einer Datenarchitektur erfüllt werden, damit die gesamte Kernanforderung als erfüllt gilt.

A1: Big-Data-Fähigkeit

- **A1.1, Volume:** Die Speicherung und Verarbeitung großer Datenmengen wird mindestens durch die horizontale Skalierbarkeit der dafür verwendeten Technologien unterstützt.
- **A1.2, Velocity:** Sowohl Stream-, als auch die Batchverarbeitung von Daten wird ermöglicht.[1]
- **A1.3, Variety:** Das System kann strukturierte und unstrukturierte Daten in geeigneten Datenformaten verarbeiten und speichern.

A2: Berücksichtigung von Data Governance

- **A2.1, Disoverability:** Die Architektur sieht eine Komponente vor, die Metadaten über alle im System verfügbaren Daten verwaltet und Entwicklern zugänglich macht.
- **A2.2, Accountability:** Es wird vermerkt, welche Entwickler oder externe Organisationen für die Bereitstellung welcher Daten im System verantwortlich sind.
- **A2.3, Quality:** Die Architektur unterstützt das Bauen von Datenpipelines, die für Datentransformationen zur Qualitätssicherung eingesetzt werden können. Die Daten werden nur auf Anfrage und nicht standardmäßig ohne konkreten Anwendungsfall persistiert oder in zentrale Verarbeitungssysteme geleitet (Sicherstellung der Aktualität).

[1] Die Berücksichtigung von Anwendungsfällen mit operationalen Daten, die höhere Zeitanforderungen haben, wird durch die Möglichkeit der Ausbringung von Software auf lokale Infrastruktur (wie Fahrzeugen) mittels Anforderung **A3.1** unterstützt. Dies ermöglicht die Realisierung von V2V- und V2I-Anwendungsfällen auf der Edge des Systems.

- **A2.4, Security und Privacy:** Die Architektur setzt sich konzeptionell mit Fragen der Security und Privacy auseinander. Ein Aufzeigen von zu berücksichtigen Problemen oder Verweise auf die Literatur genügt.

A3: Verteiltes Infrastruktur- und Datenmanagement

- **A3.1, Infrastrukturmanagement** Es gibt eine Komponente, die technische Daten der Infrastrukturknoten verwaltet. Auch werden Komponenten vorgesehen, die OTA-Updates in der verwalteten Infrastruktur anstoßen können.
- **A3.2, Datenmanagement:** Ergänzend zu Anforderung **A2.1** gibt es eine Komponente, die Informationen verwaltet, auf welchen Infrastrukturknoten welche Daten zur Verfügung stehen und wie man Zugriff auf sie erhält.

A4: Interoperabilität und Mandantenfähigkeit

- **A4.1, Interne Interoperabilität:** Entwicklern werden Werkzeuge und klare Mechanismen bereitgestellt, die es ermöglichen, ihre verarbeiteten Daten untereinander auszutauschen.
- **A4.2, Externe Interoperabilität:** Es gibt einen Mechanismus oder eine dedizierte Komponente, die es dritten Parteien erlaubt, Zugriff auf die erhobenen und verarbeiteten Daten zu erlangen.
- **A4.3, Mandantenfähigkeit:** Interne und dritte Parteien können sich die verteilte Infrastruktur teilen, um gleichzeitig ihre jeweilige Software in der Fog zu betreiben. Die Architektur unterstützt diese Mandantenfähigkeit durch Mechanismen oder Komponenten, die einen geteilten Zugriff auf die Infrastruktur verwalten.

4 Evaluation von Datenarchitekturen

Das Ziel dieses Kapitels ist es, die Eignung bestehender Ansätze für Datenarchitekturen für die Connected-Cars-Domäne zu evaluieren. In Abschnitt 4.1 stelle ich dazu zunächst Architekturen für die Connected-Cars-Domäne vor, die sich bereits in der wissenschaftlichen Literatur finden lassen. Inwiefern diese Architekturen in der Lage sind, meine definierten Anforderungen von Kapitel 3 zu erfüllen, ist anschließend Thema von Abschnitt 4.2.

Im zweiten Teil des Kapitels setze ich dann einen Fokus auf domänenunabhängige Datenarchitekturen, beziehungsweise Datenplattformen, und diskutiere auch ihre Eignung für den Einsatz im Kontext der Connected Cars. Konkret behandle ich dazu in Abschnitt 4.3 die Konzepte Data Warehouse, Data Lake, Lakehouse und Data Mesh. Am Ende fasse ich in Abschnitt 4.4 alle gewonnenen Erkenntnisse zusammen.

4.1 Connected-Cars-Datenarchitekturen in der wissenschaftlichen Literatur

Die in diesem Abschnitt vorgestellten Architekturansätze stammen aus wissenschaftlichen Veröffentlichungen, die ich mittels der Suchmaschinenabfragen `data [platform|architecture] connected cars` bei Google Scholar gefunden habe. Berücksichtigt habe ich dabei für jede Abfrage die fünfzig ersten Treffer von Anfang Februar 2024. Insgesamt gab es mit dieser Auswahl 17 Publikationen, die eine Datenarchitektur vorschlagen. Die Vorschläge reichen dabei von ganzheitlichen Datenarchitekturen, mit dem Ziel, verschiedene Anwendungsfälle der Connected-Cars-Vision umzusetzen, bis hin zu Ansätzen, die sich ganz speziell auf die Realisierung eines konkreten Anwendungsfalls fokussieren. Zusätzlich haben manche Architekturen einen Fokus auf die Nutzung von zentralen Clouddiensten, während andere Vorschläge auch die Nutzung von Edge- beziehungsweise Fog-Komponenten miteinbeziehen.

4.1.1 Cloud-zentrierte Ansätze

Marosi et al. [MLKS18] schlagen eine IoT-Plattform für Connected Cars vor. Es handelt sich um eine Cloud-Architektur, bestehend aus drei Tiers: Availability, Application- und Database-Tier. In der ersten Schicht sorgen Load Balancer und Reverse Proxies für die Skalierbarkeit der Applikationen. Zur Datenspeicherung in der Database Tier werden horizontal skalierbare Datenbanken wie Cassandra und MongoDB vorgeschlagen. Ergänzt werden sie durch einen Objektspeicher, wie beispielsweise Ceph, für unstrukturierte Daten. Die Daten der Connected Cars werden über Smartphones der Fahrzeughalter an die Cloud via Hypertext Transfer Protocol (HTTP) oder Transmission Control Protocol (TCP) gesendet. Der sogenannte CAN Data Collector ist die Komponente, die Daten in der Cloud dazu entgegennimmt und optionale Vorverarbeitungsschritte

an ihnen durchführen kann. Genauso wie die Fahrzeugdaten, werden auch bei Fahrzeugupdates die Smartphones der Fahrzeugführer als Vermittler zwischen Fahrzeug und Cloud verwendet. Für die Einspielung dieser Updates soll ein Content Distribution Network (CDN) zum Einsatz kommen, um ausreichende Bandbreiten zu gewährleisten. Eine von den Autoren vorgeschlagene Anbindung von Car-Sharing- und Parkplatzdiensten an die Architektur, wird durch eine direkte und statische Einbettung der Applikationen in die Application Tier umgesetzt. Die Skalierbarkeit des Systems wurde im Rahmen eines kleinen Versuchsaufbaus von den Autoren getestet. Im Kontext von Data Governance werden die Eigenschaften Security und Privacy thematisiert. Security wird mittels verschlüsselter Kommunikation, digitalen Signaturen und Zugangskontrollen für die Datenbanken berücksichtigt. Um Privacy-Bedenken zu adressieren, können Fahrzeughalter in ihrem Smartphone einstellen, welche Daten sie mit der Plattform teilen wollen.

Daniel et al. [DSP+17] stellen eine Datenarchitektur und ein Workflow-Modell für Analytics-Anwendungsfälle vor. Nach der Erhebung von Fahrzeugdaten werden diese hinsichtlich ihrer Qualität gefiltert. Dazu gehört die Erkennung von unbrauchbarer, unvollständiger oder dupliziert vorliegender Daten. Ein Klassifikationsalgorithmus entscheidet anschließend, ob die Daten für die Echtzeitverarbeitung oder die Verarbeitung in Batches relevant sind. Basierend auf dieser Entscheidung werden die Daten entweder in einer Cloud für die spätere Verarbeitung gespeichert oder in einem verteilten Datenspeicher, wie dem Hadoop Distributed File System (HDFS), für die Echtzeitverarbeitung zwischengespeichert. Anschließend erfolgt die getrennte oder, sofern von den Anwendungsfällen erfordert, auch kombinierte Verarbeitung der Batch- und Streamingdaten. Es wird nicht validiert, ob die Datengeschwindigkeit nach den verschiedenen Vorverarbeitungs- und Zwischenspeicherungsschritten immer noch den Latenzanforderungen von Echtzeitanwendungsfällen gerecht werden kann.

Häberle et al. [HCF+15] entwerfen eine Prototyping-Plattform für Connected-Cars-Applikationen, die in der Cloud verortet ist. Die Plattform stellt Daten der Connected-Cars-Domäne bereit, die dann genutzt werden können, um eigene Connected-Cars-Applikationen prototypisch zu entwickeln und zu erproben. Dabei sollen Templates genutzt werden können, die bewährte Architekturentscheidungen für die Connected-Cars-Domäne in der Cloud repräsentieren. Die vorgeschlagenen Architekturen für konkrete Anwendungsfälle wenden die Cloud-Computing-Entwurfsmuster von Fehling et al. [FLR+14] an. Dazu gehören zum Beispiel Message Queues zur losen Kopplung der Cloud-Applikationen und Load Balancer. Damit sollen elastische und skalierbare Anwendungen ermöglicht werden, die mit den großen Datenmengen der Connected Cars umgehen können. Bei der Wahl von Datenspeichern wird empfohlen, die Daten getrennt nach Konsistenzanforderungen zu speichern. *Strict Consistency* werde beispielsweise bei der Speicherung des Türverriegelungsstatus benötigt und *Eventual Consistency* bei der Erfassung von Fahrzeugpositionen via Streams.

Eine Datenarchitektur, die im realen Betrieb eines Autoherstellers (Stellantis) erprobt wurde, wird von Mostefaoui et al. [MMH+22] und Haroun et al. [HMD17] vorgestellt. Um die Daten von den Fahrzeugen in ihre Cloud-Plattform zu transportieren, kommt ein zweiteiliger Message Broker zum Einsatz. Ein Message Queuing Telemetry Transport (MQTT)-Broker empfängt zunächst die Daten von den Fahrzeugen über den Mobilfunk und reiht sie anschließend in die Warteschlangen von Apache Kafka ein. Die Vorteile bei der Verwendung von MQTT seien die Leichtgewichtigkeit des Protokolls, der Umgang mit schlechten Netzwerkverbindungen und die Möglichkeit, eine große Anzahl an Clients zu bedienen. Kafka ist dann in der Lage, die Datenstreams als Buffer für verschiedene Anwendungsfälle bereitzustellen, ohne die Daten endgültig zu konsumieren. Alle an die Plattform angebundenen Infrastrukturgeräte, ihre Software und Einstellungen einzelner Services

werden in einer eigenen Schicht der Plattform verwaltet. Unter anderem werden Geräteupdates, Zugriffsrichtlinien für Services und Daten, sowie Datenkontrakte gespeichert. Fahrzeughalter, Partner der Herstellerfirma und Administratoren haben Zugriff auf diese Schicht, um ihre Einstellungen zu verwalten. Für die Datenverarbeitung wird eine Speed-Processing-Schicht angeboten. Sie wird mittels Kafka, IBM Streams und Apache Flink umgesetzt. Historische Daten können außerdem in einem Data Lake und in einem Data Warehouse gespeichert werden. Der Data Lake basiert auf HDFS, Elasticsearch und einem zusätzlichen Objektspeicher. Der Objektspeicher wurde nachträglich zu der Plattform hinzugefügt, da bei HDFS die Geschwindigkeit bei der Verwaltung sehr kleiner Dateien aus Sicht der Autoren unzureichend sei. Der Grund sei, dass alle Metadaten auf einem Verwaltungsknoten in HDFS gespeichert werden und daher ein Flaschenhals entstehe. Auch seien Sicherheitsaspekte mit HDFS komplizierter und nur durch Konfiguration dritter Software umzusetzen. Um die aufbereitenden Daten den Datenkonsumenten zugänglich zu machen, werden sie in NoSQL-Datenbanken für einen schnelleren Zugriff indiziert. Die Endnutzer der Daten können dann via MQTT oder RESTful APIs auf die Daten zugreifen. Zu den Nutzern können dabei auch externe Unternehmen gehören. Die Datensicherheit der Zugriff-APIs wird mittels des Authentifizierungsprotokolls OAuth hergestellt. Die Autoren evaluieren ihre Plattform hinsichtlich Performanz, Datenqualität und den Quality of Services (QoS) von Apache Kafka.

Liu [Liu18] befasst sich in seinem Paper mit der Implementierung einer Lambda-Architektur [WM15] für die Connected-Cars-Domäne. Die Anwendungsfälle der Architektur basieren dabei auf V2V- und V2I-Kommunikation über ein Low-rate Wireless Personal Area Network (LR-WPAN). Sobald die Fahrzeuge in gegenseitiger oder in Reichweite von RSUs sind, werden die Daten ausgetauscht. Die weitere Verarbeitung großer Datenmengen findet dann in dedizierten Datenzentren statt, die eine Netzwerkverbindung zu den RSUs haben. Optional werden die Daten von lokalen Datenzentren in Straßennähe zwischengespeichert. Die Lambda-Architektur wird implementiert mittels HDFS und Apache Spark für die Batch Layer und via Apache-Storm für die Speed Layer. Die Serving Layer kombiniert die Berechnungsergebnisse aus der Echtzeit- und Batchdatensicht und speichert sie in Datenbanken (Imapla und HBase). Applikationen, die diese Daten benötigen, sollen dann eine Abfrage-API verwenden, die intern auf diese Datenbanken zugreift.

4.1.2 Ansätze mit dem Einbezug von Edge- und Fog-Computing

Van Raemdonck et al. [VVE+17] stellen eine Streaming-Plattform für die Connected-Cars-Domäne vor. Im Unterschied zu Stream-Processing-Frameworks wie Apache Storm und Apache Spark berücksichtigt ihre Plattform die Eigenschaften geografisch weit verteilter Rechenknoten. Dadurch können mittels Edge- und Fog-Processing Latenzen und Bandweiten optimiert werden. Jede verteilte Rechenkomponente, genannt Processing Site, hat einen Zugriff auf multimodale Verarbeitungskomponenten. Diese ermöglichen zunächst gängige Streaming-Operationen wie `map`, `filter` und `join`, die über eine neue Abfragesprache namens XStream, nutzbar gemacht werden können. Darüber hinaus gibt es Verarbeitungskomponenten speziell für Geo- und Videodaten. Die Kommunikation zwischen den Processing Sites wird über Message Broker oder Media-Servern ermöglicht. Die Datenproduzenten geben ihre Daten dabei nur bei Bedarf an die jeweiligen Message Broker weiter. Dies soll die Bandweiten und Rechenressourcen des Connected-Car-Netzwerks zusätzlich schonen. Auch werden dynamische Datenstreams ermöglicht, die beispielsweise die spontane Anpassung

von Datenquellen an die aktuelle Fahrzeugumgebung realisieren. Bezüglich Data Governance wird bei der Vorstellung ihres Konzepts hervorgehoben, dass die dezentralisierte Verarbeitung es erlaubt, geografische Datenschutzrichtlinien in detailliertem Umfang umzusetzen.

Zhang et al. [ZWZ+18] kommen in einer Analyse zu dem Ergebnis, dass die Bandweitenanforderungen von Connected Cars mit derzeitigen Netzwerktechnologien nicht umsetzbar seien und erst mit der flächendeckenden Einführung von 5G realisiert werden können. Gleichzeitig sei bei steigender Fahrzeuggeschwindigkeit mit hohen Paketverlusten zu rechnen. Deshalb sprechen sie sich gegen zentrale Cloudarchitekturen und für eine lokale Datenplattform aus, die auf den Fahrzeugen selbst verortet ist. Die Plattform, genannt OpenVDAP, basiert auf einem neu entworfenen Betriebssystem, das im Kernel neben Verwaltungsmechanismen für Speicher- und Rechenressourcen auch Kommunikationsfunktionalitäten speziell für Connected Cars bereitstellt. Darüber liegt ein eigens entworfenes Betriebssystem, das Dienste zur Datenfreigabe an OEM-eigene oder dritte Applikationen bietet. Eine Servicemanagement-Komponente entscheidet außerdem, basierend auf aktuellen Netzwerkparametern, welche Berechnungsaufgaben von Services auf dem Fahrzeug, in der Edge oder in der Cloud verarbeitet werden sollen. Auch hybride Ausführungsstrategien sind möglich, um die Ressourcen der Infrastruktur optimal zu nutzen. Damit Entwickler ihre eigenen Applikationen auf Grundlage der Plattform bauen können, wird eine Bibliothek zur Verfügung gestellt. Sie sorgt für eine Isolation der Applikationen, erlaubt Zugriffe auf die lokalen Ressourcen und implementiert Mechanismen zur Datenteilung mit anderen Applikationen unter Berücksichtigung von Sicherheits- und Datenschutzbestimmungen. Da die Plattform vor allem auch zur Entwicklung von Diensten des autonomen Fahrens gedacht ist, werden außerdem gängige Frameworks für maschinelles Lernen angeboten. Die Autoren betonen, dass ihr Konzept noch vor vielen ungelösten Problemen steht. Ein Beispiel hierfür ist die Frage, wie die Plattform an aktuelle Netzwerkparameter für das verteilte Task-Scheduling kommen kann.

Hull et al. [HBZ+06] stellen bereits im Jahr 2006 CarTel als System zur Erfassung und Verarbeitung von Fahrzeugdaten vor. Die Daten werden lokal auf jedem Fahrzeug in einer PostgreSQL-Datenbank gespeichert. Sie werden erst an einen Server übertragen, wenn die Fahrzeuge Netzwerkkonnektivität haben. Diese erlangen sie beispielsweise durch das Parken in Reichweite privater oder öffentlicher Wi-Fi-Netzwerke. Dieses Konzept der physischen Übertragung von Daten durch Fahrzeuge wird Data Muling genannt. Erst wenn Erreichbarkeitszonen des Netzwerks erreicht werden, werden die Daten über die Netzwerkinfrastruktur transportiert. Auf dem Server werden die Daten zur Analyse von Fahrerverhalten und des Verkehrs genutzt. Auch werden den Nutzern Visualisierungen ihrer Fahrten angeboten. Zu diesen erhalten sie Zugriff mittels einer Passwort-Authentifizierung. Die Daten werden jedoch nicht nur ausschließlich auf dem Server selbst verarbeitet. Es wird ein an Structured Query Language (SQL) angelehntes Datenabfragemodell implementiert, das Datenvorverarbeitungen schon auf den Fahrzeugen selbst veranlasst.

Ähnlich wie Zhang et al. [ZWZ+18] argumentieren Darwish und Bakar [DB18] in ihrem Paper zunächst, weshalb aus Latenz- und Ressourcengründen keine zentrale Cloud-Architektur für ihre Big-Data-Analytics-Architektur in Frage kommt. Sie diskutieren anschließend die Vor- und Nachteile von drei Typen des dezentralen Computings. Sogenannte Cloudlets sind kleine Rechenzentren, die sich an der Edge des Netzwerks befinden. Ihre Rechen- und Speicherressourcen können von geografisch nahen Geräten verwendet werden. Ein Nachteil der Cloudlets sei jedoch, dass fahrende Fahrzeuge häufig ihre Bezugscloudlet wechseln müssten, da sich die geografische Position zu den Datenzentren schnell ändern kann. Dies erzeuge eine hohe Komplexität beim Management der Applikationen. Mobile Edge Computing hingegen sieht lokale Server ausschließlich auf den

Basisstationen zellulärer Netzwerke vor. Diese Einschränkung sei gemäß der Autoren auch der Hauptnachtteil dieser Art des dezentralen Computings. Das Paradigma des Fog Computings hingegen bezieht alle Arten verfügbarer Rechenknoten ein und ermögliche dadurch eine höhere Flexibilität für die Anwendungsfälle. Eine Sonderform des Fog Computings ist Vehicular Fog Computing. Hier können, zusätzlich zur Verwendung statischer Rechenknoten der Fog, spontane Rechenverbünde zwischen Fahrzeugen geschlossen werden, um Ressourcen intelligent zu teilen. Dies böte sich insbesondere bei Staus oder parkenden Autos an. Wie solche dynamischen Rechenverbünde realisiert werden können, wird beispielsweise von Shojafar et al. [SCB16] diskutiert. Die Architektur für Data Analytics, die von Darwish und Bakar vorgeschlagen wird, basiert ebenso auf einem Fog-Ansatz. Sie schlagen drei sogenannte Dimensionen für ihre Architektur vor. Die Computing-Dimension besteht aus allen Fog-Geräten, die die technische Realisierung der Applikationen ermöglichen. Sie sind in konzeptionelle Schichten unterteilt hinsichtlich unterschiedlicher Eigenschaften ihrer Ressourcenkapazität. Die unterste Schicht bilden Edge-Geräte wie Fahrzeuge, Smartphones und Ampelanlagen. Die oberste umfasst Datenzentren der Cloud. Die auf der Computing-Dimension aufbauende Analytics-Dimension ist für die Datenverarbeitung zuständig und besteht aus Batch-, Speed- und Serving-Layer. Die Batch Layer wird in der Cloud realisiert, während die Speed-Layer die Echtzeitvorteile der Fog-Komponenten ausnutzen soll. Die dritte Dimension, genannt IoV-Dimension, beinhaltet konkrete aufgabenorientierte Komponenten. Vorgesehen ist Software für die Umgebungswahrnehmung via Sensoren, eine Netzwerk- und Kommunikationsschicht, Komponenten für künstliche Intelligenz, sowie schließlich eine Anwendungs- und Businessschicht. Die letzten beiden befassen sich mit der konkreten Umsetzung von Anwendungsfällen, beziehungsweise dem eigentlichen Geschäftsinteresse der OEMs. Darwish und Bakar [DB18] diskutieren offene Probleme bei der Realisierung ihrer Architektur. So wird speziell auf zu berücksichtigende Security- und Privacy-Bedenken eingegangen. Insgesamt bleiben sie auf einer eher abstrakten konzeptionellen Ebene. Etwa werden keine konkreten Technologien zur technischen Umsetzung genannt.

Benaissa et al. [BBM21] schlagen eine On-Board-Datenplattform für Fahrzeuge vor. Diese Plattform ermöglicht die Speicherung und Verarbeitung von Daten auf dem Fahrzeug selbst. Dadurch sollen niedrige Latenzen sowie eine Schonung der Netzwerkressourcen ermöglicht werden. Die Plattform sieht die Datenspeicherung und -bereinigung in einer Storage-Schicht vor. Eine sogenannte Virtualisierungsschicht kümmert sich anschließend um die weitere Aufbereitung der Daten via Modellierung, Labelling und Datenreduktion. Anschließend können auf diesen Daten die eigentlichen Datenanalysen vollzogen werden. Um die Ergebnisse der Analysen oder allgemein Daten externen Parteien oder Applikationen zukommen lassen zu können, ist eine gesonderte Zugriffsschicht vorgesehen. Die Details zu dieser Schicht gehören jedoch zur Future Work der Arbeit. Anhand eines konkreten Beispieldatensatzes wird die Architektur erprobt. Dabei werden die ankommenden Datenstreams in CSV-Dateien gespeichert. Anschließend werden sie bereinigt und in ein passenderes Datenmodell transformiert. Außerdem wird Run-length Encoding (RLE) angewandt, um die Daten zu komprimieren. Die transformierten Daten werden in einer Postgres-Datenbank gespeichert. Sie stellen die Grundlage für Verkehrsvorhersagen dar, die durch ein vorher trainiertes Machine-Learning-Modell getätigt werden.

Datta et al. [DHBD17] entwerfen eine IoV-Architektur für die Connected-Cars-Domäne, die einen besonderen Fokus auf datenorientierte Netzwerke, Edge-Computing, Cloud- und Dateninteroperabilität legen will. Dies seien Eigenschaften, die von bisherigen Architekturen wenig beleuchtet wurden. Eine Perception Layer der Architektur besteht aus den verfügbaren Sensoren und Aktuatoren auf einem Fahrzeug. Um zu wissen, welche Sensordaten oder Aktuatorenaktionen jedes Fahrzeug bietet, werden Metadaten über die Verfügbarkeit und Eigenschaften dieser Ressourcen lokal gespeichert.

Diese Metadaten dienen jedoch nicht nur zur Registrierung und der Discovery von Fahrzeugeigenschaften, sondern sollen auch eine generische Methode der Datenverarbeitung ermöglichen. Diese wird realisiert, indem semantische Regeln über die Metadaten ausgeführt werden, konkret mittels des Resource Description Framework (RDF) und der darauf arbeitenden Abfragesprache SPARQL. Auf diese Weise kann die Datenverarbeitung sich interoperabel an die jeweiligen verfügbaren Ressourcen, Umgebungseigenschaften und Aktuatorenaktionen anpassen. Für die Datenkonsumenten werden durch diese Art der Datenverarbeitung Handlungsempfehlungen berechnet, die mittels der Ansprechung von Aktuatoren anschließend umgesetzt werden können. Damit die Berechnung der Handlungsempfehlungen für Echtzeitszenarien geeignet ist, empfehlen Datta et al. Verarbeitungskomponente auch auf Edge-Komponenten zu platzieren. Über eine Applikationsschicht und RESTful APIs können Entwickler Zugang zu den Verarbeitungs- und Speicherressourcen der Edge und der Cloud erlangen. Der anschließende Prozess der Applikationsentwicklung läuft dabei nach einem genormten Mechanismus ab. Zunächst erfolgt die Discovery von verfügbaren und benötigten Geräten. Danach wird für die Entwickler automatisiert ein Applikationstemplate erstellt. Dieses soll die Implementierung der Services auf Basis der verfügbaren Ressourcen vereinfachen. Zu den Entwicklern, die diesen Mechanismus nutzen, können auch Datenkonsumenten externer Parteien wie Versicherungsunternehmen gehören. Security und Privacy werden als allgemeines Problem in kurzem Umfang diskutiert und Lösungen aus verwandten Arbeiten vorgestellt. Diese Lösungen sollen im Rahmen einer nicht weiter spezifizierten Sicherheits- und Datenschutzkomponente in der Architektur berücksichtigt werden können.

4.1.3 Anwendungsfallspezifische Architekturen

Cárdenas-Benítez et al. [CAM+16] schlagen eine Architektur zur Erkennung von Verkehrsstaus mittels Connected-Cars-Daten vor. Fahrzeuge senden jede Sekunde ihre Geschwindigkeit und Position an ein algorithmisch ausgewähltes Cluster-Head-Fahrzeug (LORA-CBF-Algorithmus, siehe Santos et al. [SEA06]). Dieses Fahrzeug aggregiert die Daten seines Clusters und sendet die Informationen zu einer RSU. Die RSU wiederum leitet die Daten an ein Rechencluster weiter. Dort werden die Datenstreams in einer verteilten Cassandra-Datenbank gespeichert und anschließend mittels Hadoop verarbeitet. Die Verarbeitungsergebnisse werden in SQL-Tabellen geschrieben, die dann mittels HTTP-APIs zugänglich gemacht werden. Die Datenkonsumenten, die diese APIs verwenden, sind Fahrzeuge, die für ihren lokalen geografischen Bereich Informationen zu aktuellen Verkehrsmeldungen anfragen. Die Autoren evaluieren und bestätigen die Genauigkeit ihrer Berechnungen mithilfe einer Simulation.

Prehofer und Mehmood [PM20] behandeln als konkreten Analytics-Anwendungsfall die Berechnung der Energieeffizienz von Elektroautos. Ihr Ziel ist es, zu evaluieren, ob Streamprocessing mit Apache Flink oder Batchprocessing mit Apache Spark für diese Aufgabe besser geeignet ist. In ihren Experimenten schneidet der Versuchsaufbau unter Verwendung von Apache Flink besser ab. Darüber hinaus plädieren sie dafür, die Streamverarbeitung zur Verringerung von Latenzen möglichst geografisch nahe an den Fahrzeugen selbst durchzuführen.

Lim et al. [LSB20] stellen ihre Datenplattform REView vor, die Daten von elektrischen Fahrzeugen sowie Ladesäulen sammelt und anschließend analysiert. Fahrende Fahrzeuge senden jede Minute über Smartphones On-Board Diagnostic System (OBD)-II Daten des Controller Area Network (CAN)-Buses an ein Serverbackend. Parkende Fahrzeuge tun dies jede halbe Stunde. Die Kommunikation mit den Ladestationen und den Fahrzeugen läuft über TCP und Simple Object Access Protocol

(SOAP). Die Datenanalysen zur Berechnung der Energienutzung und von Fahrzeugrouten werden in regelmäßigen Abständen via Cronjobs durchgeführt. Zur Datenspeicherung und für einfache Analysen wird auf dem Server die PostgreSQL-Datenbank eingesetzt. Nutzer authentifizieren sich mittels RFID-Tags bei den Ladesäulen, um Datenintegrität zu gewährleisten.

Amini et al. [AGP17] entwerfen eine Datenarchitektur für die Echtzeit-Verkehrsüberwachung. Apache Kafka wird verwendet, um eine bidirektionale Kommunikation zwischen den Verkehrsgeräten (Fahrzeugen, RSDs) und der Analytics-Plattform herzustellen. Die Daten, die von den Geräten in Kafka-Topics veröffentlicht werden, werden gleichzeitig auch im Rohformat in einem Data Warehouse, basierend auf HDFS, für etwaige spätere Analysen persistiert. Innerhalb einer eigens entwickelten Analytics Engine können mittels `Reducer`- und `Evaluator`-Funktionen eigene Verarbeitungstasks implementiert werden. Die Ergebnisse der Analysen werden dann entweder in eine NoSQL-Datenbank geschrieben oder wieder über Kafka-Topics Datenkonsumenten zugänglich gemacht. Das System wurde mit einer Verkehrssimulation für eine drei Kilometer lange Autobahnstrecke prototypisch implementiert.

Vorhersagen für Fahrzeugwartungen ist der berücksichtigte Anwendungsfall für die Architektur von Gerloff und Cleophas [GC17]. Die Fahrzeugdaten werden von eingebetteten Fahrzeugmodems über eine HTTP- oder Constrained Application Protocol (CoAP)-Schnittstelle zu der Verarbeitungsschicht der Architektur weitergeleitet. Diese ist mittels Apache Spark implementiert und dafür zuständig, dass die Daten in geeigneter Qualität für die nachfolgende Analyse bereitgestellt werden. Der Fokus des Papers liegt auf der Implementierung der Verarbeitungsschicht in Form von drei hintereinandergeschalteten Datenpipelines. Das Ziel ist es, zu zeigen, wie mit diesen Pipelines eine gute Datenqualität erreicht werden kann. Unter anderem werden Datenduplikate entfernt und zusammengehörende Fahrzeugdaten aggregiert. Es wird außerdem festgestellt, welche Fahrzeuge Wartungen hatten, für welche eine solche vorgeschlagen wird und ob nach Wartungsevents die aufgetretenen Probleme erfolgreich behoben werden konnten. Im Anschluss an die Verarbeitungsschicht findet eine Analyse der aufbereiteten Daten statt. Dazu wird MLib, die Machine-Learning-Bibliothek von Spark, verwendet. Auch wird der Einsatz einer Speicherschicht auf der Basis von HDFS und Yet Another Resource Negotiator (YARN) vorgesehen, um auch historische Daten verarbeiten zu können.

Nkenyereye und Jang [NJ17] präsentieren eine Architektur, mit der Fahrzeugdiagnosedaten mittels Map-Reduce verarbeitet werden können. Die Daten stammen von simulierten OBD-II-Modulen. In der Praxis sollen diese Daten über die Smartphones der Fahrzeughalter versendet werden. Im Serverbackend werden ankommende Daten mit Echtzeitrelevanz in eine MongoDB-Datenbank geschrieben und Daten ohne zeitliche Einschränkungen in eine MySQL-Datenbank. Anschließend werden die Daten aus beiden Speichersystemen mittels Hadoop und HiveQL analysiert. Die Komponente, die diese Hadoop-Software umfasst, wird von den Autoern als Data Warehouse bezeichnet. Die Ergebnisse der Berechnungen werden über einen Anwendungsserver den Datenkonsumenten bereitgestellt sowie wieder in der MySQL-Datenbank persistiert. Die Evaluation der Autoren mehrerer Map-Reduce-Implementierungen ergibt, dass die Berechnung mittels HiveQL am schnellsten erfolgt. Am zweitschnellsten schneiden händisch implementierte Map-Reduce-Jobs, basierend auf Hadoop, ab. Am wenigsten performant waren gewöhnliche SQL-Abfragen der MySQl-Datenbank.

4.2 Diskussion und Einordnung der Architekturen

Die vorgestellten Architekturen werden nun darauf untersucht, ob und in welchem Ausmaß sie die definierten Anforderungen aus Kapitel 3.3 erfüllen können. Diese Diskussion wird auf konzeptioneller Ebene geführt. Das bedeutet, dass eine Anforderung dann als erfüllt gilt, wenn das Paper ein Konzept beschreibt, das das Anforderungskriterium erfüllt. Es ist nicht möglich basierend auf konkreten Metriken beispielsweise die Big-Data-Eigenschaften zu überprüfen. Dafür fehlen oftmals Implementierungen der Architekturen oder konkretere Architekturdetails. Zwar führen mehrere Paper eigene Evaluationsmessungen durch, jedoch sind diese aufgrund unterschiedlicher Versuchsaufbauten untereinander nicht vergleichbar.

Es ist daher nicht abschließend zu beantworten, ob die vorgestellten Architekturen tatsächlich in der Lage sind, ihre beworbenen Eigenschaften zu erfüllen. Der Mehrwert meiner Bewertung ist stattdessen die angestrebte Erkenntnis, ob es bereits Architekturen gibt, die versuchen, ganzheitlich alle identifizierten Anforderungen umzusetzen.

Tabelle 4.1 fasst meine Bewertungen der Architekturen hinsichtlich ihrer Anforderungsüberdeckung zusammen. Grundlage dieser Einordnung sind die Beschreibungen der Kriterien aus dem Anforderungskatalog in Kapitel 3.3.

Tabelle 4.1: Übersicht über die bewerteten Erfüllungsgrade der Anforderungen aller betrachteten Architekturen. Ein ✓ steht für eine erfüllte Anforderung, ein ✗ für eine unerfüllte. Ein ● symbolisiert eine zum Teil erfüllte Anforderung.

Paper-referenz	A1			A2				A3		A4		
	A1.1	A1.2	A1.3	A2.1	A2.2	A2.3	A2.4	A3.1	A3.2	A4.1	A4.2	A4.3
Cloud-zentrierte Ansätze												
[MLKS18]	✓	✓	✓	✗	✗	●	✓	●	✗	✓	✗	✗
[DSP+17]	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
[HCF+15]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[MMH+22]	✓	✓	✓	✓	✗	●	✓	✓	●	✓	✓	✗
[Liu18]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Ansätze mit Edge-, bzw. Fog-Computing												
[VVE+17]	●	✗	●	✓	✗	●	●	●	●	✗	✗	✗
[ZWZ+18]	✓	✓	✗	✗	✗	✗	✓	●	●	●	●	●
[HBZ+06]	●	✗	✗	✗	✗	●	✓	✗	✗	✗	✗	✗
[DB18]	✓	✓	✓	✗	✗	✗	✓	●	✗	✗	✗	✗
[BBM21]	✓	✗	✗	✗	✗	●	✗	✗	✗	●	●	✗
[DHBD17]	●	✗	✓	✓	✗	●	✓	●	✓	✗	✓	✗
Anwendungsfallbezogene Architekturen												
[CAM+16]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
[PM20]	●	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[LSB20]	✗	✗	✗	✗	✗	✗	●	✗	✗	✗	✗	✗
[AGP17]	✓	✓	✓	✗	✗	✗	●	✗	●	✗	✗	✗
[GC17]	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
[NJ17]	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

4.2.1 Erfüllung der Big-Data-Eigenschaft (A1)

Unter allen Anforderungen wird die Big-Data-Eigenschaft am häufigsten von den Architekturen adressiert. Zur Speicherung der potenziell großen Datenmengen finden horizontal skalierbare Speichersysteme Verwendung. Diese basieren häufig auf dem HDFS. Vereinzelt werden aber auch andere horizontal gut skalierbare Datenbanktechnologien, wie beispielsweise Cassandra, eingesetzt. Im Hinblick auf skalierbare Frameworks zur Datenverarbeitung setzen viele Architekturen ebenso auf Open-Source-Technologien. Dazu gehören beispielsweise die verteilten Verarbeitungsangebote von Apache Spark und Apache Flink.

Bei den Architekturen, die Edge- und Fog-Computing in ihre Konzepte einbeziehen, wird die horizontale Skalierbarkeit vor allem durch die Verteilung der Rechen- und Speicherlasten auf möglichst viele Infrastrukturknoten des Connected-Cars-Systems ermöglicht. Die cloud-zentrierten Lösungen von Häberle et al. [HCF+15], Marosi et al. [MLKS18] und Mostefaoui et al. [MMH+22] zeigen zusätzlich, wie sich mittels Load Balancer die Datenlast horizontal in zentralen Cloudarchitekturen verteilen lässt.

Obwohl diese Maßnahmen für die Erfüllung der Volume-Anforderung (**A1.1**) in meinem Bewertungssystem ausreichen, gibt es Architekturen, die sie nicht erfüllen. Fünf Paper erfüllen die Anforderungen nur in Teilen. Dazu gehört die Streaming-Plattform von Van Raemdonck et al. [VVE+17], die keine Speicherung von großen Datenmengen ermöglicht, auch wenn die horizontale Skalierbarkeit der Verarbeitung durch Fog-Computing gewährleistet wird. Prehofer und Mehmood [PM20] berücksichtigen in ihrer Architektur ebenso keine Speicherung von Daten. Hull et al. [HBZ+06] verwenden zwar einzelne Datenbanken auf Edge-Knoten zur Zwischenspeicherung und ermöglichen lokale Datenverarbeitung, sehen aber im Backend eine PostgreSQL-Datenbank auf einem einzigen Serverknoten vor. Dieser kann zu einem Flaschenhals werden. Datta et al. [DHBD17] berücksichtigen neben Cloud- auch Edge-Computing, geben jedoch keine Details, ob und wie es um die Skalierung der Speichersysteme steht. Nkenyereye und Jang [NJ17] verwenden zwar HDFS zur verteilten Speicherung und Map-Reduce zur verteilten Verarbeitung, speichern jedoch zunächst alle ankommenden Echtzeitdaten und später alle Berechnungsergebnisse in einer MySQL-Datenbank.

Die Architektur von Lim et al. [LSB20] erfüllt die Teilanforderung nicht. Ihre Architektur ist nicht auf große Datenmengen ausgelegt. Auch sie speichern ihre Daten in einer einzelnen PostgreSQL-Datenbank. Zwar lassen sich viele relationale Datenbanken wie MySQL oder PostgreSQL horizontal mittels Partitionen und Sharding skalieren, jedoch stellt dies für sie grundsätzlich einen größeren manuellen Verwaltungsaufwand dar, als es beispielsweise bei dateibasierten Speichersystemen der Fall ist [CEP+21; SL22]. Da keine Details der Architektur hierzu mitgeteilt werden, sehe ich die Anforderung als nicht erfüllt an.

Die Velocity-Eigenschaft (**A1.2**) wird von acht Papern nicht erfüllt. Das liegt daran, dass die betroffenen Architekturen entweder ausschließlich Stream- oder Batchprocessing unterstützen, oder wie Häberle et al. [HCF+15], keine konkreten Details zur Art der Datenverarbeitung liefern.

Die Variety-Eigenschaft (**A1.3**) wurde von sieben Architekturen nicht erfüllt, weil keine Datenspeicherung vorgesehen ist oder die Speicherung von Daten notwendigerweise in relationalen Datenbanken erfolgt (wie bei [NJ17]). Die Streamprocessing-Plattform von Van Raemdonck et al.

[VVE+17] wird hier als Ausnahme aufgeführt und erfüllt die Anforderung zumindest teilweise. Multimodale Verarbeitungspipelines für heterogene Daten sind in ihrer Architektur explizit integriert. Lediglich ihre Persistierung wird nicht explizit unterstützt.

4.2.2 Berücksichtigung von Data Governance (A2)

Drei Architekturvorschläge thematisieren die Speicherung von Metadaten (Anforderung **A2.1**). Mostefaoui et al. [MMH+22] verwenden die Software Elasticsearch, um einen Data Catalog zu implementieren. Dieser indiziert alle Daten in ihrem verwendeten Data Lake unabhängig von den verwendeten Speichertechnologien. Entwicklern ist es dadurch möglich zu sehen, wo welche Daten im System gespeichert sind. Van Raemdonck et al. [VVE+17] sehen in ihrer Architektur eine Registry-Komponente vor, in der unter anderem Metadaten aller verfügbaren Datenstreams gespeichert werden sollen. Datta et al. [DHBD17] präsentieren ein umfangreiches Data-Discovery-Konzept für ihre IoV-Architektur. Metadaten werden in Form von JSON-LD auf jedem Fahrzeug gespeichert und von einer Resource-Discovery-Komponente genutzt.

Die Anforderung bezüglich der Ermöglichung von Data Accountability (**A2.2**) wird von keiner Architektur adressiert. Es werden von den Autoren keine Konzepte vorgestellt, die es ermöglichen festzuhalten, welche Entwicklerteams für die Bereitstellung und Wartung bestimmter Datensätze verantwortlich sind.

Data Quality (**A2.3**) steht im Fokus von zwei vorgeschlagenen Architekturen. Daniel et al. [DSP+17] führen explizite Data-Cleansing-Schritte im Rahmen von Extract Transform Load (ETL)-Vorverarbeitungsschritten durch. Außerdem klassifizieren sie die anfallenden Daten nach ihrer Relevanz für die nachfolgende Echtzeit- oder Batchverarbeitung. Dies adressiert explizit die Eigenschaft der Datenaktualität, weil verhindert wird, dass für das System wertlose Daten unnötig abgespeichert werden. Auch [GC17] sehen in der Architektur zur Umsetzung ihres konkreten Anwendungsfalls Datenpipelines zur Erhöhung der Datenqualität vor. Wertlose Daten werden ebenfalls schon nahe an der Edge des Netzwerks verworfen.

Sechs weitere Architekturen erfüllen die Teilanforderung A1.3 teilweise. Benaissa et al. [BBM21], Marosi et al. [MLKS18] und Mostefaoui et al. [MMH+22] ermöglichen zwar konzeptionell ETL, das zur Erhöhung der Datenqualität genutzt werden könnte, speichern jedoch alle Daten auf Verdacht ab, egal ob für sie ein Anwendungsfall existiert oder nicht. Datta et al. [DHBD17], Hull et al. [HBZ+06] und Van Raemdonck et al. [VVE+17] adressieren Datenqualität nicht explizit, ermöglichen aber gezielte Datenzugriffe bei Bedarf. Dies wird zumindest der Aktualitätseigenschaft teilweise gerecht, da somit keine wertlosen Daten abgespeichert werden müssen.

Zur Erfüllung des Security- und Privacy-Kriteriums (**A2.4**) ist es ausreichend, wenn solche Aspekte bei der Vorstellung der Architektur angesprochen werden. Marosi et al. [MLKS18] sehen sichere Kommunikationsprotokolle vor und erlauben es Fahrzeughaltern einzustellen, welche Daten sie mit der Plattform teilen wollen. Ähnliche Privacy-Policies sind auch für die Architektur von Mostefaoui et al. [MMH+22] eingeplant. Bezüglich Security beschäftigen sie sich mit Authentifizierung und Verschlüsselung für Speichersysteme und für die Zugriffs-APIs von dritten Parteien. Zhang et al. [ZWZ+18] erlauben innerhalb ihrer Data-Sharing-Komponente die Berücksichtigung von Datenschutzrichtlinien und diskutieren im Bereich von Security die Isolation mehrerer Tenants. Letztere soll mit der Anwendung von Trusted Execution Environments realisiert werden. Hull et al. [HBZ+06] ermöglichen Nutzerauthentifizierungen und haben bezüglich Datenschutz die allgemeine

Einschränkung, dass nur jeder Nutzer seine eigenen Daten sehen und analysieren kann. Darwish und Bakar [DB18] und Datta et al. [DHBD17] verweisen bei der Diskussion von Security- und Privacy-Aspekten auf bestehende Literatur. So beschäftigen sich beispielsweise Soryal und Saadawi [SS13] mit DoS-Attacken im Bereich von Connected Cars.

Zu den Architekturen, die Anforderung A2.4 nur teilweise erfüllen, gehören Amini et al. [AGP17] und Van Raemdonck et al. [VVE+17]. Sie gehen in nur kurzem Umfang auf Überlegungen im Kontext von Privacy ein. Lim et al. [LSB20] berücksichtigen hingegen ausschließlich die Nutzerauthentifizierung für ihre Systeme.

4.2.3 Verteiltes Daten- und Infrastrukturmanagement (A3)

Die Teilanforderung des Infrastrukturmanagements (**A3.1**) wird nur von Mostefaoui et al. [MMH+22] vollständig erfüllt. Sie sehen eine Gerätemanagement-Komponente vor, die Metadaten zu allen Geräten verwaltet und über die auch das Ausspielen von Updates auf Fahrzeuge möglich sein soll.

Fünf weitere Architekturen berücksichtigen die Anforderung teilweise. Marosi et al. [MLKS18] beschreiben eine Komponente für das Ausbringen von Softwareupdates. Darüber hinaus bietet sie jedoch keine Möglichkeiten zur Metadatenverwaltung dieser Geräte. Zhang et al. [ZWZ+18] ermöglichen zwar voneinander isolierte Services auf Fahrzeugen, zeigen aber nicht, wie die Ausbringung oder die Aktualisierung dieser Software vonstatten geht. Van Raemdonck et al. [VVE+17] verwalten den aktuellen Status von Rechenressourcen der verwendeten Fog-Infrastruktur, sehen aber abseits des automatischen Deployments von Datenflussgraphen keine Funktionalität vor, individuelle Softwarekomponenten auszubringen. Darwish und Bakar [DB18] erwähnen notwendiges Ressourcenmanagement ihrer verwendeten Fog-Knoten, sehen aber keine dedizierte Komponente für Updates oder dergleichen vor. Ähnlich ist es bei Datta et al. [DHBD17]. Sie verwalten zwar den aktuellen Status von verteilten Rechenressourcen, geben jedoch keine Details, ob und wie die Software auf diesen aktualisiert werden kann.

Das Discovery-System von Datta et al. [DHBD17] stellt zusätzlich auch Informationen darüber bereit, welche Daten auf Infrastrukturknoten vorhanden und nutzbar sind. Dies erfüllt Teilanforderung **A3.2**. Amini et al. [AGP17] und Mostefaoui et al. [MMH+22] erfüllen diese Anforderung teilweise, da ihre verwendeten Message Broker potenziell eine Zuordnung von Ressourcen über Topics ermöglichen. Sie zeigen jedoch nicht auf, wie Entwickler herausfinden können, welche Topics existieren oder zu welcher Infrastruktur sie gehören. Zhang et al. [ZWZ+18] sehen einen Data Collector vor, der automatisch alle relevanten Daten aus der Umgebung eines Fahrzeugs für dieses sammelt. Damit ist zumindest auf der Fahrzeugebene klar, welche Daten auf einem einzelnen Auto derzeit verfügbar sind. Van Raemdonck et al. [VVE+17] erfüllen die Anforderung ebenfalls teilweise über die schon mehrfach erwähnte Registry-Komponente. Mir fehlen jedoch Details darüber, was genau diese Komponente leistet, um die Anforderung komplett als erfüllt anzuerkennen.

4.2.4 Interoperabilität und Mandantenfähigkeit (A4)

Bezüglich der Berücksichtigung interner Interoperabilität (**A4.1**) sehen Marosi et al. [MLKS18] explizit das Teilen von Daten zwischen internen Services vor, die innerhalb der Application Tier der Architektur entwickelt werden. Anhand ihrer Data-Collector-Komponente zeigen sie beispielhaft, wie dies funktionieren kann. Benaissa et al. [BBM21] haben in ihrer Access Layer

auch die Funktionalität vorgesehen, dass Services auf verschiedenen Autos ihre Daten mittels Pull- und Push-Mechanismen teilen können. Der gleiche Mechanismus soll auch eine Kommunikation mit Services dritter Parteien ermöglichen (**A4.2**). Die Autoren nennen jedoch keine Details, wie die Access Layer konkret gestaltet sein soll und wie überhaupt auf Daten anderer Services oder Parteien aufmerksam gemacht werden kann. Mostefaoui et al. [MMH+22] ermöglichen interne Interoperabilität durch die Verwendung des Message Brokers Kafka, die alle Architekturschichten der Plattform verbinden soll. Auch können alle Entwickler auf die Daten im Data Lake zugreifen, wodurch ebenfalls eine Datenteilung umgesetzt wird. Der Datenzugriff für externe Parteien wird bei ihnen mittels dedizierten RESTful APIs oder MQTT-Topics gehandhabt. Diese Schnittstellen greifen auf Datenbanken zu, die ausschließlich für zu teilende Daten vorgesehen sind. Datta et al. [DHBD17] ermöglichen es externen Parteien, rohe Sensordaten von Fahrzeugen zu erhalten sowie auf ihnen berechnete Handlungsempfehlungen für Aktuatoren. Damit ist Anforderung **A4.2** erfüllt, auch wenn das Teilen von anders vorverarbeiteten Daten nicht möglich ist.

Die Architektur von Zhang et al. [ZWZ+18] erfüllt als einzige alle Teilanforderungen der Kernanforderung **A4** teilweise. Für die vollständige Erfüllung der ersten beiden Teilanforderungen sollten mehr Details gegeben werden, wie ihr Data-Sharing-Modul genau entworfen ist. Dieses Modul soll nicht nur das Teilen von Daten zwischen internen Services auf den Fahrzeugen ermöglichen, sondern auch für Fahrzeugservices dritter Parteien. Mit ihrer Architektur können externe Parteien auch die gleiche Edge-Infrastruktur wie die OEMs für ihre Services nutzen (**A4.3**). Eine Softwarebibliothek soll dabei die Entwicklung dieser Services erleichtern. Jedoch fehlen hier Details, über welche Komponente beispielsweise die externen Entwickler überhaupt Zugriffsrechte auf die Ressourcen erlangen können.

4.2.5 Zusammenfassung

Die in Tabelle 4.1 eingetragenen Erfüllungsgrade der Anforderungskriterien für die evaluierten Architekturen zeigen, dass Anforderung A1, die Berücksichtigung der Big-Data-Eigenschaften, als einzige vollständig von den Architekturvorschlägen berücksichtigt wird. Die restlichen Anforderungen A1 (Data Governance), A2 (Verteiltes Daten- und Infrastrukturmanagement) und A3 (Interoperabilität und Mandantenfähigkeit) werden von keiner Architektur vollständig berücksichtigt. Somit gibt es unter den untersuchten wissenschaftlichen Papers keinen Vorschlag, der als ganzheitlicher Architekturansatz im Sinne des in Kapitel 3.3 definierten Anforderungskatalogs verwendet werden kann. Die Teilanforderungen A2.2 (Data Accountability) und A4.3 (Unterstützung und Verwaltung einer mandantenfähigen Fog-Infrastruktur) sind besonders hervorzuheben. Als einzige Teilanforderungen werden sie von keiner Architektur vollständig erfüllt.

4.3 Domänenunabhängige Architekturen

Neben konkreten Vorschlägen für Datenarchitekturen in der Connected-Cars-Domäne gibt es auch allgemeine Konzepte im Kontext des Data Engineerings. Zu solchen gehören Data Warehouses, Data Lakes, Lakehouses und die Architektur des Data Mesh. Sie beschäftigen sich vordergründig mit der geeigneten Verwaltung und Aufbereitung von Daten für analytische Anwendungsfälle [RH22]. In diesem Abschnitt diskutiere ich die Eignung dieser Konzepte für die Connected-Cars-Domäne. Anders als im vorherigen Abschnitt 4.2, wird hier nicht der Anforderungskatalog von Kapitel 3.3

als Evaluationsgrundlage verwendet. Auf diese Weise lassen sich besser die Unterschiede der Architekturen herausarbeiten, da sie aufgrund ihrer Domänenunabhängigkeit ohnehin keine der Anforderungskriterien erfüllen, die spezifisch für die Connected-Cars-Domäne sind (wie zum Beispiel die Ermöglichung von OTA-Updates).

4.3.1 Data Warehouse

Das Ziel von Data Warehouses ist es, Daten aus dem operationalen Betrieb eines Unternehmens zu sammeln und in geeigneter Weise bereitzustellen, sodass Analysten aus ihnen geschäftsrelevante Erkenntnisse ziehen können. Die operationalen Daten werden dazu beispielsweise aus Datenbanken extrahiert, in geeignete Datenformate für das Warehouse überführt und anschließend dort abgespeichert. Dieser Prozess wird als ETL bezeichnet. Die gesammelten Daten können dann ausgehend vom Warehouse OLAP-Systemen zur Verarbeitung zugänglich gemacht werden. Der Begriff Data Warehouse bezieht sich auf die Speicherung aller für die Analyse relevanten Daten einer Organisation. Werden mehrere kleinere Data Warehouses für einzelne Subdomänen verwendet, spricht man von Data Marts [VZ14].

Typischerweise arbeiten Data Warehouses nach dem Schema-On-Write-Prinzip. Zunächst wird ein Datenschema für die zu speichernden Daten modelliert und implementiert. Erst dann können die operationalen Daten dort integriert werden [VZ14]. In Kombination mit den vorangehenden ETL-Prozessen kann dieses Vorgehen ein Nachtteil für die Entwicklung statistischer Modelle wie im Bereich des maschinellen Lernens sein. Dort arbeitet man gerne auf der Basis von Rohdaten, um experimentell Transformationen durchführen zu können, beispielsweise im Rahmen des Feature-Engineerings [GGH+20]. Gleichzeitig bietet der ETL-Prozess in der Connected-Cars-Domäne den Vorteil, dass die Transformationen in geografischer Nähe der Datenquellen, wie auf Edge- und Fogknoten, stattfinden können. Das kann Rechenressourcen sparen, da so geringere Datenmengen über Netzwerke übertragen und zu Analysezwecken zentral gespeichert werden müssen.

Die Schemas von Data Warehouses beziehen sich meistens auf relationale Modelle. So ist beispielsweise der Data Cube ein von vielen OLAP-Systemen verwendetes Konzept, das mehrdimensionale Datenoperationen auf strukturierten Daten ermöglicht [VZ14]. Solche Technologien inklusive dafür vorgesehener Abfragesprachen ermöglichen vergleichsweise kurze Antwortzeiten der Datenanalysen sowie interaktive Analysen für viele gleichzeitige Nutzer [Fan15]. Speziell für die Connected-Cars-Domäne ist diese relationale Ausrichtung von Data Warehouses jedoch auch ein entscheidender Nachteil. Die Verarbeitung von unstrukturierten Daten wird von Data Warehouses grundsätzlich eher nicht unterstützt. Schon aus diesem Grund sind Data Warehouses für die Connected-Cars-Domäne als alleinige Datenplattform nicht gut geeignet.

Einschränkend zu bemerken ist, dass es durchaus Konzepte für Data Warehouses gibt, die versuchen, die angesprochenen Nachtteile zu beheben. Vaisman und Zimányi [VZ14] stellen beispielsweise Data Warehouses für räumliche und graphenbasierte Daten vor. Gröger et al. [GSM14] entwerfen einen Ansatz, um unstrukturierte Daten in relationalen Data Warehouses in Form von Links zu ihren Dateipfaden einzubetten. Reis und Housley [RH22] stellen darüber hinaus Extract Load Transform (ELT)-Data-Warehouses vor, die Daten auch in ihrer Rohform in einer Staging Area vorabspeichern. Dies sind weiterentwickelte Varianten der ursprünglichen Data-Warehouse-Konzepte, die einen

erhöhten Implementierungsaufwand erfordern. Insbesondere für die letzten beiden angesprochenen Varianten von Warehouses hat sich stattdessen ein eigenes Konzept begonnen zu etablieren, das als Data Lake bezeichnet wird.

4.3.2 Data Lake

Das Konzept des Data Lakes sieht eine Speicherung aller zu analysierenden Daten in ihrem Rohformat vor. Dabei werden explizit sowohl strukturierte als auch unstrukturierte Daten zur Speicherung unterstützt. Nutzer des Data Lakes führen auf Grundlage dieser Rohdaten anwendungsfallabhängige Transformationsschritte durch und verarbeiten die Daten erst dann für das eigentliche Ziel [Fan15]. Dieses Vorgehen wird auch als ELT-Prozess bezeichnet [VZ14].

Um zu verhindern, dass auf diese Weise immer wiederkehrend gleiche Transformationsprozesse auf den gleichen Daten ausgeführt werden müssen, kommen Datenmanagementkonzepte wie Landing Zones oder Data Ponds zum Einsatz. Bei Landing Zones können die Daten nicht nur in ihrem Rohformat, sondern auch in weiterverarbeiteten Formaten innerhalb dafür vorgesehener Verwaltungszonen gespeichert werden. Data Ponds verfolgen ein ähnliches Ziel, tragen aber dafür Sorge, dass die gespeicherten Informationen in den Ponds jeweils disjunkt sind [GGH+20]. Da Data Lakes anders als Data Warehouses auf Schema-On-Read setzen, ist es bei ihnen besonders notwendig Metadatenmanagement zu betreiben, um Daten für die Nutzer entdeckbar und verwendbar zu machen [GGH+21].

Für die Anwendung in der Connected-Cars-Domäne ist die Möglichkeit der Speicherung heterogener Daten der größte Vorteil des Data Lakes gegenüber Warehouses. Ein weiterer Vorteil ist die Möglichkeit für Data Scientisten mit Rohdaten experimentieren zu können. Dies kann bei der Erstellung von Machine-Learning-Modellen hilfreich sein. Grundsätzlich ermöglicht Schema-On-Read auch eine höhere Flexibilität bezüglich der Erweiterbarkeit des Systems. Es muss nicht von Beginn an klar sein, welche Daten für welchen Anwendungsfall in Zukunft benötigt werden könnten.

Allerdings hat der Data Lake für die Connected-Cars-Domäne auch Nachteile. Den größten sehe ich im Zusammenhang mit dem ELT-Prozess. Alle für die Analyse relevanten Daten im Connected-Cars-System müssten vollständig in ihrem Rohformat einem zentralen Data Lake zugesendet und dort gespeichert werden. Bei der Verwendung von Landing Zones würde anschließend auch noch eine zusätzliche Replikation von Teilen der Daten erfolgen. Insbesondere für sehr große Daten wie Videos von Fahrzeugkameras würde dies insgesamt einen enormen Ressourcenverbrauch bedeuten. In Anbetracht der sehr großen Datenmengen von zahlreichen Datenquellen, die im Kontext von Connected Cars zu erwarten sind, ist der Data Lake deshalb nicht realistisch für alle Analytics-Anwendungsfälle einsetzbar.

4.3.3 Lakehouse

Lakehouses sind ein Konzept mit dem Ziel, die Eigenschaften von Data Warehouses und Data Lakes vorteilhaft in einer Datenplattform zu vereinen. Die Idee dieser Art von Datenplattform entstand dabei aus der Begebenheit, dass Data Lake und Warehouse in der Praxis oft gemeinsam verwendet wurden, um die Vorteile beider Architekturen zu erhalten [AGXZ21]. So beispielsweise auch die Connected-Cars-Datenarchitektur eines Autoherstellers, beschrieben von Mostefaoui

et al. [MMH+22], die in Abschnitt 4.1.1 vorgestellt wurde. Ihr Ziel ist es, zusätzlich zu den Vorteilen eines Data Lakes, mit einem Warehouse schnelle SQL-Abfragen für viele gleichzeitige Nutzer zu ermöglichen. Durch eine Verbindung von Warehouses und Data Lakes in einer einzigen Plattform, soll die Notwendigkeit komplexer Two-Tier-Architekturen hingegen vermieden werden können [AGXZ21].

Eine Lakehouse-Architektur wird meistens so gestaltet, dass auf der Basis der Technologie eines Data Lakes ein zusätzliches Lakehouse-Framework als Softwareschicht darüber positioniert wird. Die Aufgabe dieses Frameworks ist es dann, unter anderem die Metadaten der gespeicherten Daten so zu berücksichtigen, dass neben den unstrukturierten Daten auch relationale Daten verwaltet werden können. Die Definition des Lakehouses nach Schneider et al. [SSM23] sieht außerdem eine Abfragesprache, Garantien der Konsistenz, Atomizität und Isolation (Teile der sogenannten ACID-Eigenschaften) vor sowie Möglichkeiten, Stream- und Batchverarbeitungsaufgaben zu unterstützen. Um Abfragesprachen zum Beispiel für die relational gespeicherten Daten im Lakehouse zu ermöglichen, ist eine konkrete Aufgabe des Frameworks beispielsweise die Verwaltung von Caches und Indizes [AGXZ21].

Gemäß der Definition von Armbrust et al. [AGXZ21] unterstützen Lakehouses ETL und ELT als Datenintegrations- und Transformationspattern. Für die Connected-Cars-Domäne ist diese hybride Herangehensweise gut geeignet. Daten, die relevant für Machine-Learning-Modelle sind, können in ihrer Rohform gespeichert werden, während Daten abseits dieses Anwendungsfallunsterstützen schon im Voraus mittels ETL vorverarbeitet werden können. Dies ermöglicht eine effizientere Ressourcennutzung, auch im Hinblick auf die Möglichkeit des Einbezugs von Edge- oder Fog-rechenressourcen. Die Speicherung von heterogenen Daten und die gleichzeitige Möglichkeit Analysen auf Basis relationaler Abfragesprachen vielen Nutzern gleichzeitig zu ermöglichen, spricht ebenso für eine Eignung für die Connected-Cars-Domäne. Beispiele für Anwendungsfälle aus Kapitel 3.1, in denen Analysen von einer Vielzahl verschiedener Nutzer durchgeführt werden, sind beispielsweise das Live-Reporting für Fahrzeughalter und Flottenmanager. Hier kommt auch die Isolationseigenschaft von Lakehouse besonders wichtig zu tragen.

Die Unterstützung von Batch- und Streamverarbeitung, die Teil der Definition von Schneider et al. [SSM23] ist, spricht ebenso für eine Verwendung in der Connected-Cars-Domäne. Die in Kapitel 3.3 definierte Teilanfoderung Velocity (**A1.2**) macht unter anderem die Ermöglichung beider Verarbeitungsarten zum Kriterium.

Zusammenfassend ist im Vergleich der Architekturen des Data Warehouses, Data Lakes und Lakehouses das Lakehouse konzeptionell am besten für die Connected-Cars-Domäne geeignet.

Um die Lakehouse-Architektur noch besser an die Connected-Cars-Domäne anzupassen, schlage ich zusätzliche Funktionalitäten für die Berücksichtigung von räumlichen Daten vor. Wie in Kapitel 3.2 ausgeführt, spielen räumliche Daten (zum Beispiel geografische Koordinaten) eine wichtige Rolle für die Implementierung vieler Anwendungsfälle der Connected-Cars-Vision. Eine Möglichkeit der Anpassung des Lakehouses an diese Daten ist die Wahl eines geeigneten Speicherformats. Data Lakes und Lakehouses nutzen häufig das spaltenorientierte Speicherformat Parquet [SSM23]. Für dieses Speicherformat existiert eine Erweiterung namens Geoparquet[1], die zusätzliche Datentypen für raumbezogene Daten anbietet. Verteilte Technologien für Batch- oder Streamprocessing wie

[1] https://geoparquet.org/

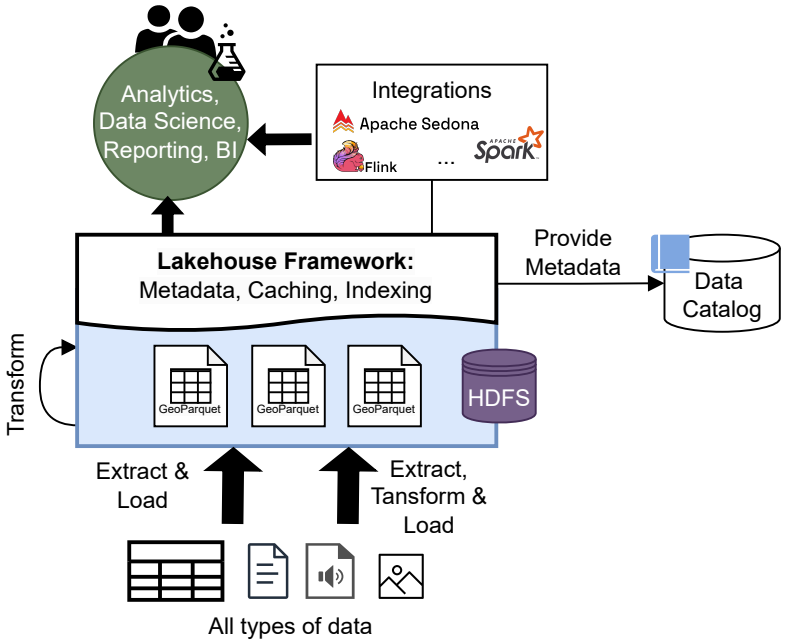


Abbildung 4.1: Lakehouse-Architektur basierend auf HDFS mit Anpassungen für die Connected-Cars-Domäne, um räumliche Daten einfacher berücksichtigen zu können.

Apache Spark und Apache Flink können mit Erweiterungen, wie Apache Sedona[2], direkt auf Basis dieses Datenformats agieren. Dadurch werden räumliche Datenanalysen und -abfragen möglich, ohne eigene Transformationen durchführen zu müssen. Aus meiner Sicht kann dies eine sinnvolle Technologieauswahl für eine Lakehouse-Architektur in der Connected-Cars-Domäne darstellen.

Abbildung 4.1 veranschaulicht die beschriebene Lakehouse-Architektur bildlich. Als verteiltes Speichersystem wurde in diesem Fall HDFS gewählt. Die Data-Catalog-Komponente wird zusätzlich von mir für die Architektur vorgesehen und registriert alle im Lakehouse gespeicherten Daten, um sie für die Connected-Cars-Organisation auffindbar zu machen (Berücksichtigung der Anforderung Discoverability, **A1.1**). Zur Integration solcher Datenkataloge bieten einige Lakehouse-Frameworks Integrationsschnittstellen an. Deltalake und Apache Hudi ermöglichen beispielsweise die Anbindung des Cloud-Datenkatalogs AWS Glue[3].

4.3.4 Data Mesh

Anders als die bisher vorgestellten Datenarchitekturen legt das Konzept des Data Meshs einen verstärkten Fokus auf die Organisation von Entwicklerteams, die die analytischen Daten bereitstellen. Zhamak Dehghani [Deh22], die den Begriff im Jahr 2019 initial definierte, beschreibt in ihrem Buch den Data Mesh folgendermaßen:

[2] https://sedona.apache.org/1.5.1/

[3] https://www.databricks.com/product/aws/glue, https://hudi.apache.org/docs/syncing_aws_glue_data_catalog/ (Letzter Abruf: 06.03.2024)

> *„Data mesh is a decentralized sociotechnical approach to share, access, and manage analytical data in complex and large-scale environments—within or across organizations.“* (Zhamak Dehghani [Deh22, Seite 3])

Eine der Grundannahmen des Konzepts ist, dass monolithische Datenarchitekturen, wie die ausschließliche Verwendung eines Data Lakes, für große Organisationen mit vielen beteiligten Entwicklern und verschiedenen Subdomänen mit Nachteilen verbunden seien. Diese Nachteile ließen sich durch eine dezentralisierte Datenarchitektur beheben. Dazu werden vier Grundprinzipien des Data-Mesh-Konzepts definiert.

Domain Ownership Statt die Erzeugung und Verwaltung analytischer Datensätze einem zentralisiertem Team von Dateningenieuren zu überlassen, sollen im Data Mesh Daten in den Verantwortlichkeiten der Domänen bleiben, aus denen sie ursprünglich stammen. Dadurch sollen Abhängigkeiten zwischen Entwicklerteams vermieden werden, die ansonsten domänenübergreifend agieren hätten müssen. Auf diese Weise werden die Erweiterbarkeit und Wartbarkeit des Systems erhöht. Änderungen können entweder selbst von Domänenteams unternommen werden oder neue Domänen können dem Gesamtsystem zu diesem Zweck hinzugefügt werden. Das Anpassen einer zentralen Datenpipeline beispielsweise, die für die Daten der gesamten Organisation verantwortlich ist, ist im Data Mesh nicht notwendig. Die durch dieses Prinzip entstehende dezentrale Architektur kann als Anwendung des Architekturprinzips *Domain-Driven Design* verstanden werden [GKD+23].

Data as a Product Die analytischen Daten, die nach dem ersten Prinzip der Verantwortung ihrer Ursprungsdomäne unterliegen, sollen mit anderen Domänen, Kunden oder externen Organisationen geteilt werden können. Dadurch sollen Datensilos, also Datenansammlungen, die von dem Rest der Organisation isoliert sind, in den einzelnen Subdomänen vermieden werden. Die Daten werden gemäß des zweiten Prinzips *Data as a Product* als Produkt gesehen, das hohen Qualitätsstandards genügen soll. Um Datenprodukte erstellen und teilen zu können, stellt die Data-Mesh-Architektur automatisierte Mechanismen bereit, die alle Artefakte eines Datenprodukts zu diesem Zweck verwalten. Dazu gehören neben den Daten selbst beispielsweise Metadaten, der Code und Anforderungsbeschreibungen an die benötigte Infrastruktur, um die Daten bereitzustellen. *Data Contracts* legen darüber hinaus fest, zu welchen Bedingungen und mit welchen Garantien die Daten zwischen Produzenten und Konsumenten ausgetauscht werden.

Self-Serve Platform Die *Self-Serve Platform* ist im Data Mesh hauptverantwortlich für die technische Umsetzung aller Prinzipien. Sie stellt den Entwicklern domänenagnostische Mechanismen und Infrastruktur zur Verfügung, um Datenprodukte erstellen, teilen und warten zu können. Zu dieser Infrastruktur können beispielsweise Speicherangebote von Datenplattformen oder Datenverarbeitungsframeworks gehören. Diese Angebote können dann von den Domänenteams genutzt werden, um ihre eigenen Datenarchitekturen zu bauen. Durch die Self-Serve Platform wird das notwendige Spezialwissen für die Installation und Wartung einer eigenen Dateninfrastruktur in den Domänen reduziert. Auch werden durch geteilte Plattformangebote Technologien den Teams vorgegeben, wodurch die Interoperabilität von Datenprodukten erhöht werden kann.

Federated Computational Governance Bei der Verwendung des Data-Mesh-Ansatzes muss sichergestellt werden, dass die Datenprodukte aller Domänen interoperabel austauschbar sind und Organisationsrichtlinien nicht verletzt werden. Das Ziel des vierten Prinzips *Federated Computational Governance* ist es deshalb, die Einhaltung solcher Richtlinien automatisiert zu erzwingen. Dadurch wird verhindert, dass die Domänenteams Entscheidungen bei dem Bau der dezentralen Architekturen für ihre Datenprodukte treffen, die nicht im Einklang mit den Interessen oder Standards der Organisation sind. Umgesetzt werden die Governance-Richtlinien insbesondere mithilfe der Self-Serve Platform.

Die Anwendung der Prinzipien des Data-Mesh-Konzepts auf die Connected-Cars-Domäne ist aus mehreren Gründen interessant. Zunächst adressiert der Data Mesh große Systemumgebungen mit vielen Subdomänen. Die Vorstellung der Anwendungsfälle aus Kapitel 3.1 hat gezeigt, dass die Connected-Cars-Vision eine Vielzahl unterschiedlicher Anwendungsfälle vorsieht mit einer Vielzahl von beteiligten Geräten und Parteien. Eine solche große Datenlandschaft mit vielen Datenproduzenten und -konsumenten, in der auch Erweiterbarkeit und Änderbarkeit für weitere Anwendungsfälle eine große Rolle spielen, ist nach [GKD+23] eine Umgebung, die gemäß grauer Literatur als geeignet für den Data-Mesh-Ansatz gilt. Ein weiterer Umstand, der dazu führt, dass die dezentrale Datenarchitektur des Data Mesh nach Goedegebuure et al. [GKD+23] vorteilhaft gegenüber einem zentralisiertem Ansatz sein kann, ist, wenn ein hohes Maß an Data Governance erfordert wird. Dies trifft auf die Connected-Cars-Domäne zu, wie in der Anforderungsanalyse von Kapitel 3.2 ausgeführt wurde. Data Governance ist eine der drei identifizierten Kernanforderungen des Anforderungskatalogs von Kapitel 3.3 und wird von keiner der in Abschnitt 4.2 evaluierten Architekturvorschlägen vollständig erfüllt. Der Data-Mesh kann hier mit dem Prinzip der Federated Computational Governance eine Umsetzungsstrategie anbieten. Die Teilanforderung Data Accountability (A2.2) wird dabei schon explizit durch das Data-Mesh-Prinzip der Domänenunterteilung berücksichtigt, wodurch Datenprodukte klaren Zuständigkeitsgruppen zuordenbar sind.

Das Teilen von Daten innerhalb der OEMs-Organisation und mit externen Parteien ist darüber hinaus Gegenstand von Anforderung A4, die ebenfalls von keiner bestehenden Architektur vollständig berücksichtigt wird. Auch für diesen Aspekt liefert der Data Mesh ein Konzept mit unter Domänen austauschbaren Datenprodukten. Diese Datenprodukte können, aufgrund ihres geforderten Qualitätsstandards, ohne zusätzliche Qualitätsaufbereitung an dritte Parteien übergeben werden, wodurch auch die externe Interoperabilität (A4.2) vom Data-Mesh-Konzept profitieren kann. Insgesamt hat der Data-Mesh folglich das Potenzial, eine sinnvolle Architektur für die Connected-Cars-Domäne darzustellen.

4.4 Zusammenfassung

Die Evaluation existenter Datenarchitekturen in Abschnitt 4.1 und 4.2 hat gezeigt, dass unter ihnen kein Vorschlag existiert, der vollständig alle von mir erhobenen Anforderungen berücksichtigt. Zwar wurden über alle wissenschaftlichen Paper hinweg viele der Anforderungen angesprochen oder teilweise adressiert, jedoch lässt sich keine ganzheitliche Datenarchitektur ausmachen, die alle Anforderungen abdeckt.

In Abschnitt 4.3 diskutierte ich darüber hinaus die Anwendung verschiedener allgemeiner Datenarchitekturen auf die Connected-Cars-Domäne. Dabei kam ich zum Schluss, dass das Konzept des Lakehouse und der etwas abstraktere Architekturansatz des Data Mesh vielversprechende Potenziale

für den Einsatz in der Domäne aufweisen. Die Anwendung des Data Mesh zeigt sich dabei vor allem geeignet, um die Anforderungenzu erfüllen, auf denen in der Literatur bisher ein weniger starker Fokus lag.

Die gewonnenen Erkenntnisse aus der Evaluation nehme ich für das folgende Kapitel 5 zum Anlass, um einen eigenen Vorschlag für eine Connected-Cars-Datenarchitektur zu beschreiben. Für diese wende ich das Data-Mesh-Konzept an und zeige auf, wie es sich für die Connected-Cars-Domäne umsetzen lässt. Dazu lasse ich Konzepte aus den vorgestellten domänenbezogenen Architekturen einfließen, berücksichtige die gewonnenen Erkenntnisse bezüglich des Lakehouses und bringe neue eigene Überlegungen ein.

5 Anwendung des Data-Mesh-Konzepts auf die Connected-Cars-Domäne

In diesem Kapitel zeige ich, wie die Prinzipien des Data Mesh auf die Connected-Cars-Domäne angewendet werden können. In Kombination mit den in Kapitel 4 vorgestellten Konzepten des Lakehouses, bestehenden Architekturvorschlägen aus der Literatur und eigenen Überlegungen soll daraus ein Gesamtkonzept entstehen, mit welchem die Kriterien des Anforderungskatalogs (siehe Kapitel 3.3) für eine Datenarchitektur erfüllt werden können. Ob das gelingt, ist Thema von Kapitel 8. Der Hauptunterschied meines Vorschlags gegenüber allgemein diskutierten Data-Mesh-Konzepten in der Literatur ist, dass ich die Fog als verfügbare Infrastruktur miteinbeziehe und die Besonderheiten herausarbeite, die sich für die Anwendung der Mesh-Prinzipien dabei ergeben. Dadurch wird das Data-Mesh-Konzept speziell für die Connected-Cars-Domäne konkretisiert.

Der Aufbau des Kapitels orientiert sich an den vier Grundprinzipien des Data Mesh (Domain Ownership, Data as a Product, Self-Serve Platform, Federated Computational Governance). In Abschnitt 5.1 zeige ich anhand von Beispielen, wie die Connected-Cars-Domäne in Subdomänen unterteilt werden kann und was die Vorteile des Einbezugs von Fog Computing für diese Domänen sind. Im Anschluss diskutiere ich in Abschnitt 5.2, welche Konsequenzen diese geografische Verteilung auf die grundlegende Architektur und Verwaltung von Datenprodukten hat. In Abschnitt 5.3 schlage ich dann eine eigene Architektur für Self-Serve Platforms vor, die in der Lage ist, verteilte Datenprodukte zu ermöglichen und gleichzeitig die Infrastrukturknoten zu verwalten. Zuletzt gehe ich in Abschnitt 5.4 in kurzem Umfang darauf ein, von welchen Komponenten der Self-Serve Platform Governance-Richtlinien automatisiert umgesetzt werden können, wo die Richtlinien gespeichert werden, und zeige, wie ein bestehendes Datenschutzkonzept für Connected Cars in das Data-Mesh-Konzept eingebettet werden kann.

Jedes Unterkapitel startet mit einem Grundlagenteil, der allgemeine Data-Mesh-Konzepte erklärt, die für den Abschnitt relevant sind. Danach folgen eigene Überlegungen.

5.1 Domain Ownership

5.1.1 Grundlagen

Zur Anwendung des Prinzips Domain Ownership muss eine Organisation in Domänen unterteilt werden. Jede einzelne Domäne ist dann selbst für den Entwurf und die Umsetzung einer eigenen Datenarchitektur zur Realisierung ihrer Anwendungsfälle zuständig. Werden bestimmte analytische Daten aus einer Domäne auch von anderen Domänen benötigt, so müssen von ihr außerdem entsprechende Datenprodukte bereitgestellt werden [Deh22].

5.1.2 Bedeutung und Beispiele für die Connected-Cars-Domäne

In Kapitel 3.1 stelle ich Anwendungsfälle für Connected-Cars-Systeme vor. Um Domain Ownership zu etablieren, müssen diese Anwendungsfälle in den Zuständigkeitsbereich von einzelnen Subdomänen einsortiert werden. Zur Identifizierung dieser Domänen ist es sinnvoll, Anwendungsfälle zu gruppieren, die ähnliche Anforderungen an Technologien und eingesetzte Geräte im IoV haben. Damit kann das Ziel erreicht werden, die Abhängigkeiten zwischen den einzelnen Domänen zur Realisierung ihrer Anwendungsfälle zu minimieren. Anwendungsfälle der direkten V2V- und V2I-Kommunikation können beispielsweise gut als eigene Domäne modelliert werden, da sie ähnlich hohe Anforderungen an die Verarbeitungsperformanz haben. Die Verwendung von V2V- und V2I-Netzwerkprotokollen wird dann zu einem Expertengebiet der Entwickler in diesen Domänen. Weiter eignet sich die Formierung einer Infotainment-Domäne. Entwickler in dieser Domäne können sich auf alle Funktionalitäten fokussieren, die Gebrauch von den Infotainment-Geräten der Fahrzeuge machen. Zu solchen Geräten gehören zum Beispiel Musikanlagen und Head-Up-Displays. Ihre Programmierung wird damit zu ihrem unabhängigen Zuständigkeitsgebiet.

Auf der Basis dieser Überlegungen zur Vermeidung von Abhängigkeiten können auch die restlichen Anwendungsfälle in die Zuständigkeiten von einzelnen Domänen gruppiert werden. Abbildung 5.1 zeigt anhand eines Beispiels, wie verschiedene Subdomänen der Connected-Cars-Domäne interagieren. Die Domänen, dargestellt in Ellipsen, bieten Datenschnittstellen anderen Domänen an, damit sie ihre Anwendungsfälle implementieren können. Die verwendete Ball-Notation unterscheidet in der Abbildung zwischen Schnittstellen für operationale Daten (gekennzeichnet mit einem großen *O*) und für analytische Daten (gekennzeichnet mit einem *A*). Die Domänen `Infotainment` und `High-Resolution Traffic Map` deuten zusätzlich noch in abstrakter Weise Bestandteile ihrer Datenarchitektur an. Anwendungen (`Apps`) sind hierbei die operationalen Services zur Realisierung konkreter Anwendungsfälle. Datenprodukte sind als Sechsecke eingezeichnet. Sie enthalten Anwendungen, die analytische Daten sammeln, transformieren, speichern und in geeigneter Weise anderen Domänen bereitstellen.

Die Besonderheit bei der Anwendung des Prinzips Domain Ownership auf die Connected-Cars-Domäne in meinem Data-Mesh-Konzept ist, dass alle verfügbaren Rechenkomponenten des IoVs (siehe Kapitel 2.1) von Domänen genutzt werden können. Durch die lokale Ausbringung von Anwendungen auf Fahrzeuge oder lokale Rechenknoten der Edge, können die hohen Anforderungen an die Datengeschwindigkeiten bei den Anwendungsfällen mit V2V- und V2I-Kommunikation realisiert werden. Gleichzeitig werden Skalierungseffekte erzielt und die Ressourcen des Gesamtsystems geschont. Es ist dabei Domänen auch möglich, mehrere Arten von Infrastrukturknoten gleichzeitig zu verwenden. Die blau dargestellte `High-Resolution-Traffic-Map`-Domäne ist in Abbildung 5.1 ein Beispiel hierfür. Sie stellt Datenprodukte für hochauflösende regionale Karten auf Fog-Knoten zur Verfügung, die für die Verkehrsüberwachung einer Stadt verwendet werden können. In der Cloud aggregiert die Domäne alle regionalen Karten, um eine große, dafür aber weniger hochauflösende Karte von ganz Deutschland bereitzustellen. Diese Datenarchitektur erzielt mehrere Vorteile. Zunächst ist das System durch die Verwendung von regionalen Fog-Knoten automatisch horizontal skaliert und daher eher in der Lage, mit großen Datenmengen zurechtzukommen. Gleichzeitig dient die Bereitstellung lokaler Karten als CDN und verringert die Latenzen zu lokalen Datenkonsumenten, wie beispielsweise Fahrzeugen in den Regionen. Zuletzt müssen für die Zusammenstellung der überregionalen Deutschlandkarte nur Daten bis zur Cloud übertragen werden, die wirklich für diese relevant sind. Details zu einzelnen Fahrzeugen könnten beispielsweise verworfen werden. Dieses Vorgehen spart Netzwerkressourcen im Vergleich zu einem cloudzentrierten Ansatz.

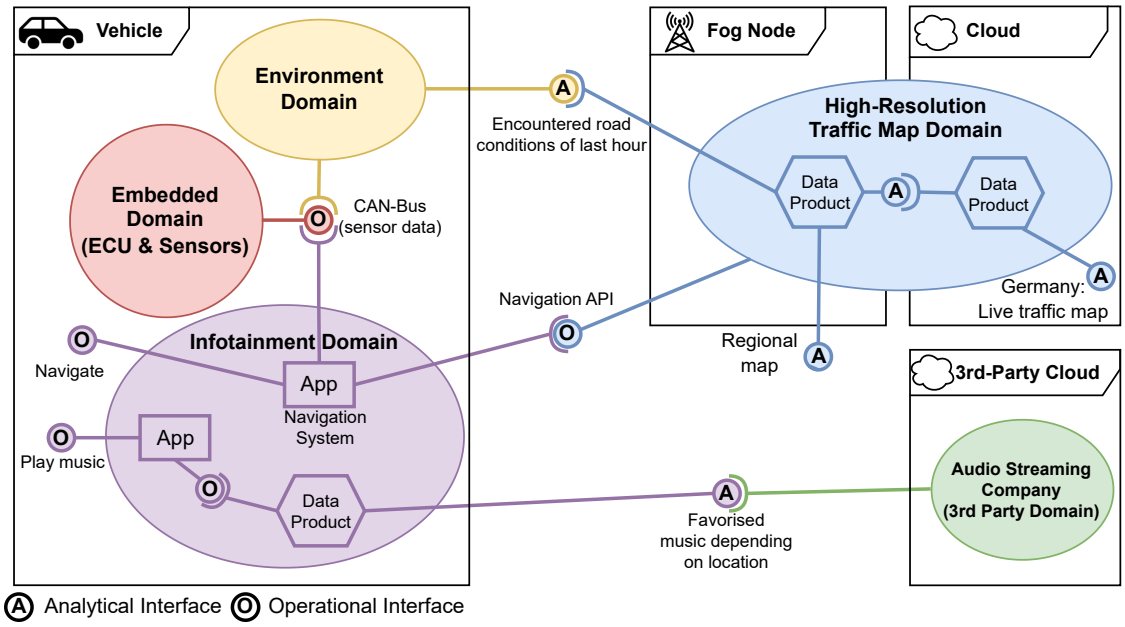


Abbildung 5.1: Zusammenspiel dezentralisierter Datenarchitekturen einzelner Subdomänen. Die Darstellungsweise von interagierenden Domänen ist von Dehghani [Deh22] übernommen.

Parteien, die nicht zum Fahrzeughersteller gehören, für den der Data Mesh implementiert ist, werden als externe Domäne modelliert. In Abbildung 5.1 erhält ein Musikstreamingdienst analytische Daten darüber, welche Musiktitel an welchen geografischen Standorten von Fahrzeughaltern favorisiert abgespielt wurden. Die Mechanismen zum Teilen von Datenprodukten sind hierbei die gleichen wie beim Produktaustausch zwischen internen Domänen (siehe Abschnitt 5.3.4).

5.2 Data as a Product

5.2.1 Grundlagen

Datenprodukte werden in einer eigenen isolierten Laufzeitumgebung, einem sogenannten *Product Container* auf die verwaltete Infrastruktur ausgebracht [GKD+23]. Dehghani [Deh22] unterscheidet in ihrem Buch zwischen vier grundlegenden Schnittstellen, die diese Container standardisiert anbieten sollen. Diese sind auch in Abbildung 5.2 für zwei Container zu sehen.

Der *Input Port* ist dafür zuständig, alle Daten aus operationalen Systemen oder von anderen Datenprodukten zu sammeln, die für die Erstellung des Datenprodukts notwendig sind. Der *Output Port* ist dann für die Bereitstellung der fertigen Produktdaten an andere Parteien verantwortlich. Dazu kann er multimodale Möglichkeiten des Datenzugriffs anbieten. Beispielsweise via Streaming, SQL-APIs oder einer API für direkte Dateizugriffe. Über den *Control Port* lassen sich Konfigurationen des Datenprodukts verändern. Zuletzt stellt der *Discover- und Observe-Port* eine Schnittstelle bereit, die für die Produktüberwachung und zur Abfrage von Metadaten genutzt werden kann [Deh22].

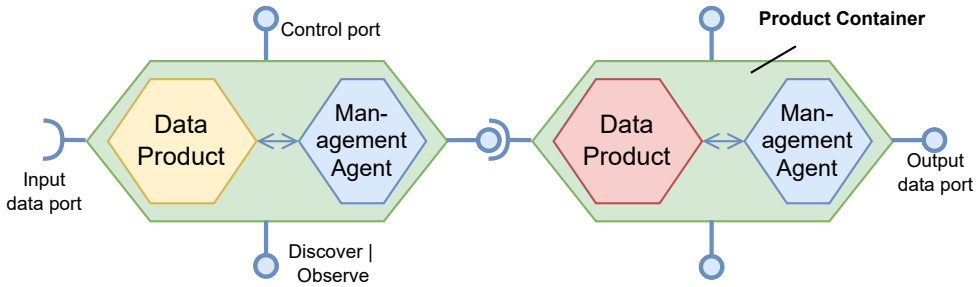


Abbildung 5.2: Zwei Product Container zur Bereitstellung einer standardisierten Laufzeitumgebung für Datenprodukte. Das rechte Datenprodukt verwendet das linke Datenprodukt als Dateninput.

Management Agents sind nach Goedegebuure et al. [GKD+23] Softwarekomponenten, die parallel neben den Softwarekomponenten des Datenprodukts laufen. Sie werden für alle Datenprodukte im Data Mesh entwickelt und können Funktionen implementieren, die für die Verwaltung der Product Container zuständig sind, oder das äußere Verhalten von Datenprodukten standardisieren. Beispielsweise können Governance-Richtlinien von solchen Agents erzwungen werden, die über den Control Port einstellbar sind. Management Agents werden von Dehghani [Deh22] auch als *Sidecar* bezeichnet, da sie dem gleichnamigen Entwurfsmuster von Burns [Bur18] folgen.

5.2.2 Bedeutung und Besonderheiten für die Connected-Cars-Domäne

Damit die in Abschnitt 5.1 vorgestellten geografisch verteilten Datenarchitekturen in der Fog möglich sind, müssen Datenprodukte auf einzelne Fog-Knoten ausgebracht werden können. Daraus ergeben sich drei verschiedene Verteilungsmodelle für Datenprodukte. Diese sind in Abbildung 5.2 dargestellt.

1. **Data Product:** Ein konventionelles Datenprodukt, das in einem einzelnen Container auf eine Rechenkomponente ausgebracht werden kann. Es gibt nur je einen Control- und Observation-Port, die zur Verwaltung des Datenprodukts genutzt werden können.
2. **Compound Data Product:** Ein zusammengesetztes Datenprodukt läuft gleichzeitig auf mehreren Fog-Knoten. Alle Produktartefakte, wie der Programmcode, sind für jede Instanz des Datenprodukts gleich. Lediglich die bereitgestellten Daten des Produkts können sich voneinander unterscheiden, da sie jeweils aus der lokalen Umgebung des Knotens erhoben werden. Jede Instanz hat seinen eigenen Product Container mit allen vier Schnittstellen. Ein Beispiel für ein zusammengesetztes Datenprodukt ist die lokale Bereitstellung von Geschwindigkeitsdaten auf Fahrzeugen einer Fahrzeugflotte zu Analysezwecken.
3. **Distributed Data Product:** Ein verteiltes Datenprodukt implementiert eine Datenarchitektur, die sich über mehrere Typen Fog-Knoten erstreckt. In Abbildung 5.3 ist das gelbe Datenprodukt über einen Fog-Knoten und eine Cloud verteilt. Ein verteiltes Datenprodukt ist dann sinnvoll, wenn beispielsweise eine ETL-Pipeline über mehrere Typen von Infrastrukturknoten hinweg ausgebracht werden soll, um Ressourcen zu schonen. Verteilte Datenprodukte haben jeweils nur einen Input- und Outputport, aber möglicherweise Control- und Observe-Ports

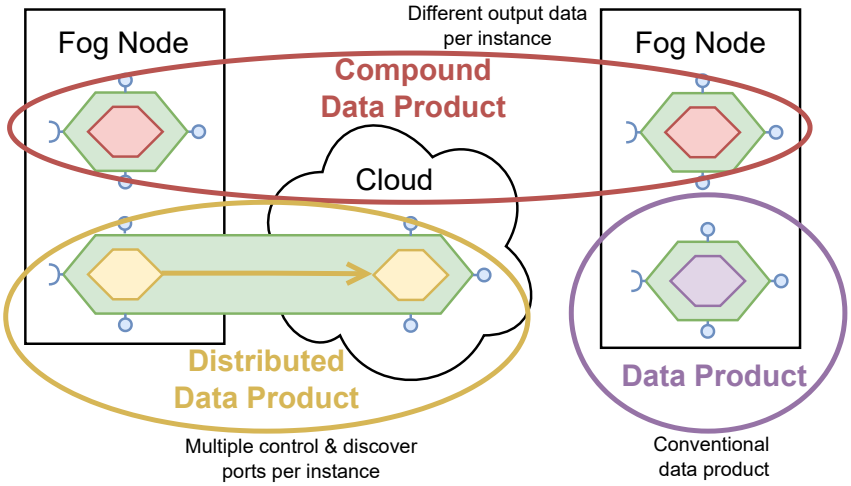


Abbildung 5.3: Geografische Verteilung von Datenprodukten

auf jedem beteiligten Infrastrukturknoten. Verteilte Datenprodukte können gleichzeitig auch zusammengesetzt sein. Ein Beispiel hierfür ist der Anwendungsfall der Kennzeichenerkennung. Dashcam-Videos werden auf Fahrzeugen zur Kennzeichenerkennung verarbeitet. Die Kennzeichen werden anschließend an regionale Fog-Komponenten gesendet, die dann alle Kennzeichen einer Region aggregieren und gesammelt bereitstellen können.

5.2.3 Auswirkungen auf den Lebenszyklus der Datenprodukte

Datenprodukte haben einen dynamischen Lebenszyklus. Das allgemeine Data-Mesh-Konzept sieht vor, dass sie dynamisch bei Bedarf auf die Infrastruktur in Product Container ausgebracht, gestartet, gestoppt und wieder entfernt werden können [GKD+23]. Dieser Produktlebenszyklus ist aufgrund seines Potenzials der Ressourcenschonung besonders für die Connected-Cars-Domäne vorteilhaft. Datenprodukte werden nur auf die Infrastruktur ausgebracht und gestartet, wenn auch wirklich ein Anwendungsfall für die Daten vorhanden ist. Sollte keine Nachfrage nach dem Datenprodukt mehr existieren, entfernt oder stoppt man den Container einfach wieder. Das spart Rechen-, Speicher- und Energieressourcen, die besonders auf Fog-Knoten beschränkt sein können.

Das dynamische Management von verteilten und zusammengesetzten Datenprodukten in der Fog bringt jedoch auch zusätzliche Komplexität mit sich, die bei der Anwendung des Data-Mesh-Konzepts auf die Connected-Cars-Domäne berücksichtigt werden muss.

Bei dem Einsatz von verteilten Datenprodukten muss sichergestellt werden, dass die Container, die gemeinsam ein verteiltes Datenprodukt bilden, in einem atomaren, transaktionalen Rahmen auf die jeweiligen Infrastrukturknoten ausgebracht, modifiziert beziehungsweise wieder entfernt werden. Andernfalls läuft man in Gefahr, dass unvollständige Datenprodukte ausgebracht werden.

Bei zusammengesetzten Datenprodukten muss hingegen abgewägt werden, ob ein atomares Ausbringen von Datenprodukten auf alle Infrastrukturziele überhaupt realistisch umsetzbar ist. Gerade wenn die Instanzen des Datenprodukts auf sehr viele Knoten eines Typs ausgebracht werden sollen,

muss man mit Netzwerkpartitionen rechnen. Das ist zum Beispiel der Fall, wenn ein Datenprodukt auf alle Fahrzeuge eines Herstellers ausgebracht werden soll. In diesem Szenario wird es mit hoher Wahrscheinlichkeit immer Fahrzeuge geben, die aufgrund von mangelnder Energieversorgung oder ihrer aktuellen Position keine Netzwerkanbindung haben. Auch muss man mit Fahrzeugen rechnen, die trotz aktiver Internetverbindung Hardwarefehler haben und deshalb keine weiteren Datenprodukte unterstützen können. Für zusammengesetzte Datenprodukte mit vielen Instanzen muss mit solchen Inkonsistenzen der Datenprodukte umgegangen werden können. Der Data Mesh muss dazu in der Lage sein, für jedes Infrastrukturziel den aktuellen Deployment-Status von Datenprodukten festzuhalten. Darüber hinaus muss Datenkonsumenten transparent kommuniziert werden, welche Infrastrukturknoten welche Datenprodukte aktuell zur Verfügung stellen.

Eine analoge Diskussion zu der Ausbringung oder dem Entfernen von Datenprodukten ergibt sich bei der dynamischen Konfiguration der Datenprodukte über die Control Ports. Auch hier besteht bei verteilten und zusammengesetzten Datenprodukten eine zu berücksichtigende Gefahr für Inkonsistenzen. Bei verteilten Datenprodukten ist es darüber hinaus möglich, dass das Datenprodukt mehrere Control Ports auf unterschiedlicher Infrastruktur besitzt. Sollte die Instanz nur einen Control Port besitzen, muss der Product Container selbst dafür Sorge tragen, dass Nachrichten von diesem auch auf Softwarekomponenten des Datenprodukts auf andere Infrastrukturknoten weitergeleitet werden. Existieren mehrere Kontrollports, muss auch hier pro Datenprodukt eine atomare Transaktion verwendet werden, falls die Einstellung alle Komponenten betrifft.

Im Allgemeinen hängt die Form von Aktualisierungen von Datenprodukten auch immer von der Infrastruktur und ihre dafür bereitgestellten Mechanismen ab. Datenzentren könnten beispielsweise Orchestrierungstechnologien wie Kubernetes unterstützen, während für Fahrzeuge oder RSDs andere Mechanismen wie zum Beispiel OTAs geeignet sind. Die Architektur der Self-Serve Platform (siehe Abschnitt 5.3.4) muss folglich potenziell unterschiedliche Deployment-Systeme unterstützen.

5.3 Self-Serve Platform

5.3.1 Grundlagen

Die Self-Serve Platform ist im Data Mesh dafür verantwortlich, alle Technologien und Werkzeuge bereitzustellen, die von den Domänenteams benötigt werden, um ihre lokalen Datenarchitekturen zu bauen. Außerdem bietet sie Funktionalitäten zur Verwaltung (Erstellen, Teilen, Suchen, Entfernen) von Datenprodukten an [Deh22; GKD+23]. Die grundsätzliche Ausrichtung des Data Mesh, mit seinem Fokus auf Datenprodukte, liegt auf der Berücksichtigung von analytischen Daten. Dennoch betont Dehghani [Deh22], dass ein Gesamtziel der Self-Serve Platform auch darin besteht, standardisierte Mechanismen für Entwickler bereitzustellen, mit denen sie sowohl ihre operationalen als auch analytischen Anwendungsfälle implementieren können.

Die verschiedenen zu erfüllenden Aufgaben der Self-Serve Platform werden von Dehghani [Deh22] in drei *Planes* eingeteilt. Die *Data Infrastructure Plane* ist dafür verantwortlich, allen Domänenteams zur Entwicklung ihrer dezentralen Architekturen Ressourcen, Technologien und Werkzeuge bereitzustellen. Sowohl operationale Systeme als auch die Datenprodukte sollen mithilfe ihr entwickelt werden können. Die Angebote der Plane werden von domänenunabhängigen *Platform Engineers* entwickelt und bereitgestellt [Deh22]. Dadurch wird die Interoperabilität der Systeme in den Domänen erhöht, da alle Teams auf die gleichen standardisierten Technologien zurückgreifen. Auch

wird Spezialwissen über die Inbetriebnahme dieser Systeme nicht in jeder Domäne benötigt und somit doppelter Entwicklungsaufwand reduziert [GKD+23]. Die *Data Product Experience Plane* bietet den Entwicklern Funktionen an, um den Lebenszyklus von Datenprodukten zu verwalten. Datenprodukte können von angebotenen Funktionen in dieser Schicht erstellt, auf die Infrastruktur ausgebracht, ausgeführt, gestoppt und wieder entfernt werden [Deh22]. Die letzte Plane ist die *Mesh Experience Plane*. Sie bietet Funktionalitäten an, die mehrere Datenprodukte gleichzeitig berücksichtigen. Dazu gehört beispielsweise die Ermöglichung von Suchanfragen und Vergleichen für alle verfügbaren Datenprodukte durch Entwickler. Auch globale Governance-Richtlinien können in dieser Plane berücksichtigt werden [Deh22].

Wie genau die Self-Serve Platform entworfen wird, ist abhängig von der Einsatzdomäne des Data Mesh [Deh22]. Goedegebuure et al. [GKD+23] nennen allgemeine Komponenten, die gemäß grauer Literatur für die Plattform häufig vorgesehen werden. Dazu gehören beispielsweise Datenkataloge, Speicher- und Rechenangebote sowie Komponenten für das Lebenszyklusmanagement für Datenprodukte. Die Beschreibungen der Funktionsweisen oder der zu leistenden Aufgaben der Angebote bleiben jedoch auf einer eher unkonkreten und abstrakten Ebene, da sie letztlich auch domänenabhängig sind.

5.3.2 Bedeutung für die Connected-Cars-Domäne

Um das Data-Mesh-Konzept auf die Connected-Cars-Domäne anzuwenden, muss eine Self-Serve Platform entworfen werden, die speziell auf die Anforderungen dieser Domäne zugeschnitten ist. Im Unterschied zu dem allgemeinen Data-Mesh-Konzept muss sie dabei in der Lage sein, neben Ressourcen in der Cloud, auch Fog-Ressourcen bereitzustellen, um geografisch lokale Datenarchitekturen und -produkte zu ermöglichen. Darüber hinaus muss sie allgemeine Verwaltungskomponenten für die stark verteilte Infrastruktur anbieten sowie Möglichkeiten für interne und externe Parteien Datenprodukte und Fog-Infrastruktur zu teilen. Diese Verwaltungsstrukturen sind Teil der Anforderungskriterien an eine Datenarchitektur aus dem Anforderungskatalog von Kapitel 3.3.

In den folgenden zwei Unterkapiteln (Abschnitt 5.3.4 und 5.3.3) stelle ich meine Architektur für eine Self-Serve Platform im Kontext Connected-Cars-Domäne vor. Konkret handelt es sich um zwei Typen von Plattformen, die gemeinsam miteinander interagieren. Die konzeptionell größere Plattform ist in der Cloud des jeweiligen OEM verortet. Sie bietet Rechen- und Speicherressourcen für skalierbare Applikationen an und darüber hinaus zentrale Verwaltungsservices, um die Gesamtfunktionalität des Data Mesh zu ermöglichen. Für diese Plattform in der Cloud konkretisiere ich die von Dehghani [Deh22] vorgeschlagenen Planes einer Self-Serve Platform, indem ich ihnen Softwarekomponenten zuordne. Die zweite Self-Serve Platform umfasst weniger Komponenten, muss dafür jedoch auf jedem Fog-Knoten der Connected-Cars-Infrastruktur ausgebracht werden, der für Datenverarbeitung und -speicherung genutzt werden soll. Sie stellt die Grundlage dafür dar, lokale Datenprodukte auf geografisch verteilten Fog-Knoten zu ermöglichen.

Für beide Plattformen benenne ich konkrete Komponenten, erkläre ihre Aufgaben und gehe, sofern vorhanden, auf die Besonderheiten im Kontext der Connected-Cars-Domäne ein. Ein Detailentwurf jeder einzelnen Komponenten ist im Rahmen dieser Arbeit jedoch nicht möglich. Stattdessen stelle ich in Kapitel 6 Implementierungskonzepte für zwei konkrete Aspekte der Architektur vor.

5.3.3 Architektur von lokalen Self-Serve Platforms in der Fog

Auf jedem Fog-Knoten, dessen Rechen- und Speicherressourcen für das Connected-Cars-System verwendet werden können, wird eine lokale Self-Serve Platform installiert. Die angebotenen Funktionalitäten und Services dieser Plattform können von den Entwicklern der Datenprodukte immer als präsent angenommen werden, wenn sie diese für die Fog entwickeln.

Abbildung 5.4 zeigt einen abstrakten Entwurf der Self-Serve Plattform der Fog. Unterschieden werden Plattformangebote, die standardmäßig auf jedem Knoten liegen (*managed*), und unverwaltete (*unmanaged*) Ressourcen.

Die verwalteten Plattformangebote sind Services, die allen Mandanten der Plattform, wie zum Beispiel einzelnen Subdomänen, zur Verfügung stehen. Dazu gehören Kommunikationsschnittstellen, um mit Fahrzeugen, RSDs, Applikationsservern oder Cloud-Komponenten der eigenen Infrastruktur zu kommunizieren. *Data Collection Interfaces* sind Datenschnittstellen, die besonders für Fahrzeuge oder RSDs relevant sind. Sie stellen Zugriffe auf Sensor- und Umgebungsdaten bereit. Für diese Komponente kann bei Fahrzeugen der *Driving Data Integrator* verwendet werden, der von Zhang et al. [ZWZ+18] entworfen wurde. *Streaming Offerings* bezeichnen darüber hinaus Angebote von Stream-Processing-Frameworks, auf die die Entwickler lokal zugreifen können. Dazu gehören:

- Frameworks, um den Fog-Knoten in ein allgemeines Fog-Streaming-Cluster einzubinden. Verwendet man die World-Wide-Streams-Plattform von Van Raemdonck et al. [VVE+17], so können hier beispielsweise die `Processing-Site`-Komponenten untergebracht werden.
- Stream-Processing-Frameworks als Single-Node-Deployment, wie zum Beispiel Apache Spark SQL als integriertes Tool für Datenzugriffe auf das lokale Lakehouse.
- Stream-Processing-Frameworks als Cluster-Deployment, wenn es sich bei dem Fogknoten um ein Servercluster handeln sollte (wie ein regionales Datencenter). Dies können zum Beispiel Apache Flink und Spark mit der Sedona-Erweiterung sein.

In der Speicherschicht der Plattform werden Datenmanagementsysteme für operationale und analytische Daten angeboten. Für operationale Daten eignen sich Datenbanken mit schnellen Lese- und Schreibzugriffen auf kleine Datensätze. RDBMS, NoSQL-Datenbanken, In-Memory-Datenbanken zur Realisierung von Caches oder Objektspeicher für unstrukturierte Daten werden je nach Anwendungsfall hier als Angebote den Entwicklern vorgegeben. Für den lokalen analytischen Speicher wird ein Lakehouse verwendet. Dadurch erhält man die Vorteile bei der Verwendung dieses Systems, die Kapitel 4.3.3 aufgezeigt wurden.

Sowohl für operationale, als auch analytische Daten ist ein Metadatenspeicher vorgesehen. Durch diese Komponente haben lokale Applikationen die Möglichkeit, alle Daten und Datenprodukte auf dem Infrastrukturknoten zu entdecken. Um das Prinzip der *Computational Federated Governance* für Datenprodukte umsetzen zu können, existiert außerdem ein Speicher für Richtlinien, die von allen Datenprodukten auf dem Infrastrukturknoten berücksichtigt werden müssen. Dieser Speicher interagiert mit dem Control Port der Product Container, um die Richtlinien jeweils umzusetzen.

Die Datenprodukte selbst laufen in der Ausführungsumgebung einer *Analytical Computing*-Komponente. Sie ist für die Verwaltung der Product Container zuständig. Der Bereich, in dem diese Komponente liegt, wird in Abbildung 5.4 als *unmanaged* bezeichnet. Dies soll unterstreichen, dass die Services dort nicht von der Plattform selbst vorgegeben werden, sondern hier die Entwickler für

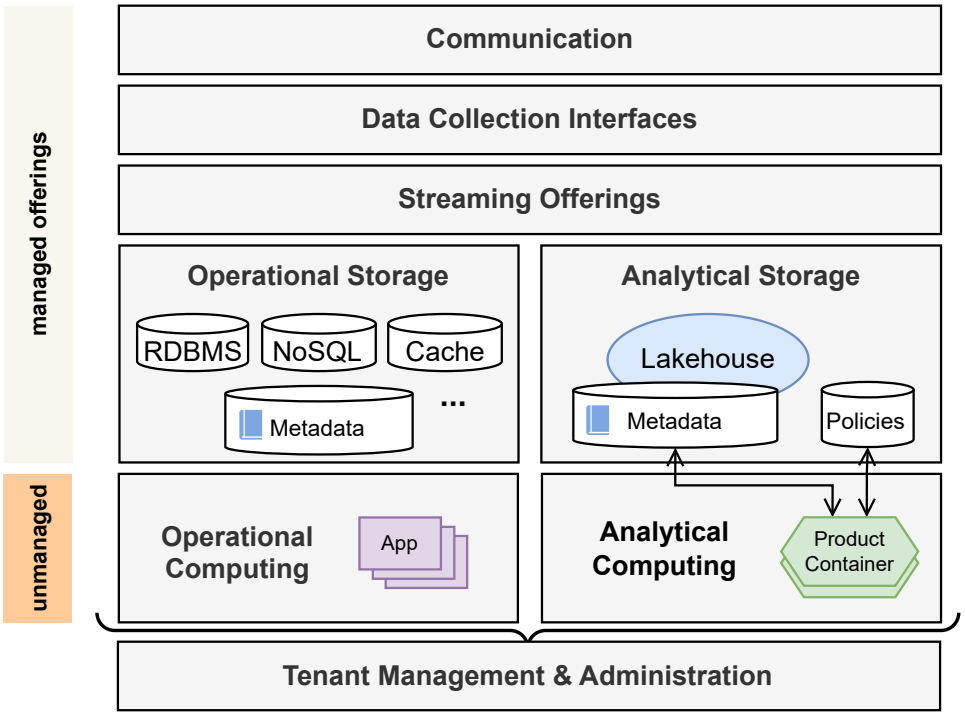


Abbildung 5.4: Self-Serve Platform für Fog-Komponenten

die konkrete Implementierung ihrer Anwendungsfälle zuständig sind. Von dieser Schicht aus können die Entwickler auf alle anderen Plattformangebote zugreifen. Für operationale Anwendungen steht ebenso eine Ausführungsumgebung zur Verfügung. Den Mandanten der Infrastrukturkomponente werden hier gemäß des Prinzips Infrastructure as a Service (IAAS) voneinander isolierte Ressourcen zur Verfügung gestellt, um ihre konkreten Anwendungsfälle mithilfe der Plattformangebote zu implementieren.

Zuletzt ist die in Abbildung 5.4 unterste abgebildete Komponente, *Tenant Management & Administration*, für alle Belange zuständig, die für die Plattformadministration und die Zuweisung von Ressourcen an verschiedene Mandanten benötigt werden.

Wie IAAS- oder Platform as a Service (PAAS)-Angebote auf Edge- beziehungsweise Fogkomponenten konkret realisiert werden können, ist ein eigenes Forschungsgebiet, das hierfür die Begrifflichkeit Mobile Edge Computing oder MEC verwendet [GSS+18; WWG+19]. Bezogen wird sich im Kontext des IoV meist auf Infrastruktur von Mobilfunkanbietern, die ihre geografisch verteilten Rechenressourcen verschiedenen Tenants der Automobilindustrie zur Verfügung stellen. Mein Architekturvorschlag sieht zusätzlich ein Angebot von Rechenressourcen auf Fahrzeugen und RSDs selbst vor. Wie solche On-Board-Frameworks technisch, beispielsweise mit geeigneten Isolationsmechanismen, realisiert werden können, wird in der Publikation von Zhang et al. [ZWZ+18] diskutiert.

5.3.4 Architektur der Self-Serve Platform in der Cloud

Auf der obersten Ebene der IoV-Infrastrukturhierarchie (siehe Abbildung 2.1), sieht mein Konzept eine umfangreichere Self-Serve Platform vor. Anders als bei der lokalen Self-Serve Platform in der Fog, existiert diese Plattform nur als einzelne Instanz für den gesamten Data Mesh des OEMs. Ihre Komponenten und deren Einteilung in die drei Planes einer Self-Serve Platform (siehe 5.3.1) sind in Abbildung 5.5 zu sehen.

Data Infrastructure Plane

In der Infrastructure Plane meiner Architektur werden den Entwicklern Infrastrukturressourcen sowohl in der Cloud als auch in der Fog zum Zugriff angeboten.

Eine ***Cloud Computing Platform*** stellt den Entwicklern Dienste bereit, die Entwicklern dabei helfen, ihre Anwendungen und Datenprodukte in der Cloud entwickeln zu können. Durch die Verwendung von Mechanismen des Cloud Computings kann eine Skalierbarkeit und Elastizität der Anwendungen erreicht werden, sodass auch mit großen und unvorhersehbaren Workloads umgegangen werden kann. Cloud-Konzepte für die Connected-Cars-Domäne habe ich bereits in Kapitel 4 bei der Evaluation von cloudzentrierten Architekturen vorgestellt. Von diesen Konzepten können einzelne Architekturdetails übernommen werden. Zum Beispiel bietet sich die Availability-Tier von Marosi et al. [MLKS18] dafür an, die mittels Load Balancern implementiert wird.

Ausgewählte Cloud-Dienste, die allen Nutzern zur Verfügung stehen, wie zum Beispiel bestimmte PAAS-, Software as a Service (SAAS) oder Function as a Service (FAAS)-Angebote, werden in Abbildung 5.5 unter *Chosen Offerings* verstanden. Zu solchen Angeboten zählen zum Beispiel Datenbanken für operationale Anwendungsfälle. Continuous Integration Pipelines und Entwicklungsplattformen werden unter der Komponente *DevOps Platform* zusammengefasst. Neben den allgemeinen Cloudangeboten, deren konkrete Architektur und Zusammenspiel im Rahmen dieser Arbeit nicht näher ausgeführt wird, wird auch eine analytische Datenplattform bereitgestellt. Konkret handelt es sich um die Lakehouse-Architektur, die in Kapitel 4.3.3 vorgestellt wurde. Frameworks für die Stream- und Batchverarbeitung wie Apache Sedona gehören zu den integrierten Tools des Lakehouses.

Um die Entwicklung lokaler Datenprodukte und Anwendungen auf Fog-Knoten zu ermöglichen, ist die Komponente ***Fog Computing Platform*** vorgesehen. Für die Orchestrierung der Anwendungen und Datenprodukte ist dabei konkret die Unterkomponente *Fog Placement & Orchestration* zuständig. Sie implementiert die Mechanismen, die OTA-Updates von Fahrzeugen und RSDs ermöglichen, aber auch für die Softwareorchestrierung auf regionale Server und Datenzentren notwendig sind. Dazu kommuniziert die Komponente mit den in Abschnitt 5.3.3 vorgestellten Administrierungsschichten der Self-Serve Platform der Fog.

Die *Stream Processing Platform* beinhaltet die Managementkomponenten, die für Streamverarbeitungs-Frameworks in der Fog benötigt werden. Bei der Streaming-Plattform von Van Raemdonck et al. [VVE+17] wird diese Komponente als *Orchestration Layer* bezeichnet. Die verteilte Verarbeitung von Datenstreams in der Edge oder der Fog sind eigene Forschungsthemen. Neben Van Raemdonck et al. stellen beispielsweise auch Wang et al. [WZH+20] ein Konzept hierfür für die Connected-Cars-Domäne vor. Bei ihnen können beliebige Fog-Knoten als Datenquelle und -senke dienen. Die Stream Processing Platform innerhalb der Fog Computing Platform soll in der

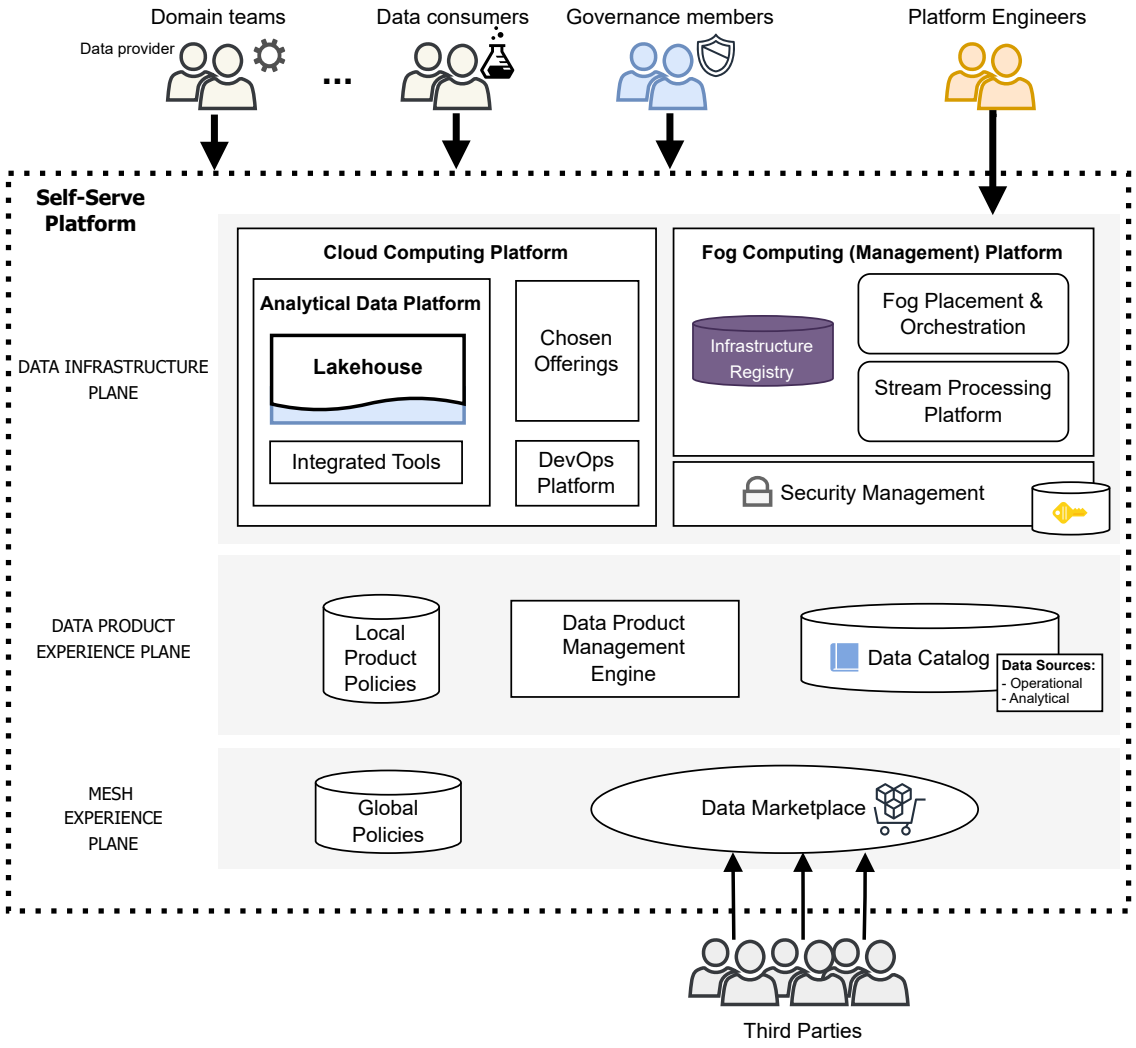


Abbildung 5.5: Self-Serve Platform in der Cloud

Lage sein, auf einfache Weise Datentransformationen bereits in der Fog zu ermöglichen. Dadurch kann beispielsweise eine erhöhte Datenqualität erreicht werden, bevor etwaige nicht verwertbare Daten weiter in höhere Fog-Hierarchien übermittelt werden müssen.

Damit die Komponenten der Fog Computing Platform überhaupt wissen, wo welche Software ausgebracht und verwaltet werden muss, gibt es die *Infrastructure Registry*. In ihr werden Informationen und der aktuelle Status aller Infrastrukturknoten der Fog gespeichert, auf denen die lokale Self-Serve Platform installiert ist. Zu wichtigen Informationen gehören:

- Art des Infrastrukturknotens: Fahrzeug, RSD, regionaler Datencenter, ...
- Geografischer Ort, Netzwerkzellenzugehörigkeit
- Verfügbare Rechen-, Speicher- und Kommunikationsressourcen
- Status laufender Anwendungen und Datenprodukte (Versionen, Deployment-Status)

- Kommunikationsdetails zur Erreichbarkeit und Adressierung der Knoten
- Liste aller Mandanten, die Anwendungen auf dem Knoten betreiben

Zuletzt gibt es in der Data Infrastructure Plane noch eine Komponente, die sich mit der Sicherheit aller Applikationen beschäftigt. In Abbildung 5.5 wird sie ***Security Management*** genannt. Hier sollen beispielsweise plattformübergreifende Authentifizierungs- und Authorisierungsmethoden, Management Agents, die für die Verschlüsselung von Datenprodukten sorgen, und digitale Signaturmechanismen für die Softwareorchestrierung bereitgestellt werden. Die Details hierzu können im Rahmen dieser Arbeit jedoch nicht behandelt werden.

Data Product Experience Plane

Innerhalb Data Product Experience Plane implementiert die Komponente ***Data Product Management Engine*** alle Funktionalitäten zum Verwalten von Datenprodukten. Es werden von ihr Schnittstellen bereitgestellt, um Datenprodukte zu bauen, zu testen, in die Infrastruktur auszubringen, bei Bedarf zu starten, zu stoppen und wieder zu entfernen. In ihr Aufgabengebiet fällt deshalb auch die Umsetzung der besonderen Fallstricke bei der Ausbringung verteilter oder zusammengesetzter Datenprodukte, die in Abschnitt 5.2.3 geäußert wurden. Um diese Verwaltungsaufgaben umzusetzen, arbeitet sie mit den Infrastrukturservices der Infrastructure Plane zusammen.

Die Management Engine ist auch dafür zuständig, Governance-Richtlinien automatisiert umzusetzen, wenn diese bereits im Erstellungsprozess von Datenprodukten berücksichtigt werden können. Solche Richtlinien können global für alle Datenprodukte gelten oder nur für ein bestimmtes Datenprodukt. Die lokalen Richtlinien für Datenprodukte werden bei ihrer Erstellung festgelegt und anschließend in einer Datenbank für ***Local Product Policies*** abgelegt. In Abschnitt 5.4 gehe ich näher auf solche Richtlinien ein.

Alle Informationen, die für das Verwalten eines Datenprodukts relevant sind, werden nach Erstellung des Produkts in einem ***Data Catalog*** abgelegt. Er organisiert neben den Datenprodukten auch alle anderen Metadaten im Data Mesh und stellt sie den Domänenteams bereit, um Daten suchen und finden zu können. Insbesondere beim Erstellen neuer Datenprodukte oder operationalen Anwendungen ist es wichtig, in Erfahrung bringen zu können, wo im System schon welche Daten vorliegen und wie sie für den eigenen Anwendungsfall geeignet verwendet werden können. Unter anderem aus diesen Gründen ist ein Datenkatalog nach Olesen-Bagneux [Ole23] geeignet für die Vermeidung von Datensilos in Unternehmen.

Ein Beispiel für andere Metadaten, abseits der von Datenprodukten, die vom Datenkatalog verwaltet werden müssen, sind Informationen über operationale Daten auf Infrastrukturknoten. Mittels Referenzen auf Daten der Infrastructure Registry macht der Katalog nachvollziehbar, welche Typen von Infrastrukturknoten welche Daten (wie beispielsweise Sensordaten von bestimmten Fahrzeugmodellen) über ihre Schnittstellen anbieten. Auch Auskunft darüber, welche Daten bereits in operationalen Datenbanken persistiert werden, ist insbesondere für die Erstellung neuer Datenprodukte eine wichtige Information. Die Kategorisierung von Daten in Datenkatalogen nach Subdomänen, oder hier auch nach dem Typ der Infrastruktur, nennt Olesen-Bagneux [Ole23] vertikale Organisation. Abbildung 5.6 zeigt beispielhaft einen Suchablauf, wie die Zusammenarbeit zwischen Data Catalog und Infrastructure Registry implementiert werden könnte.

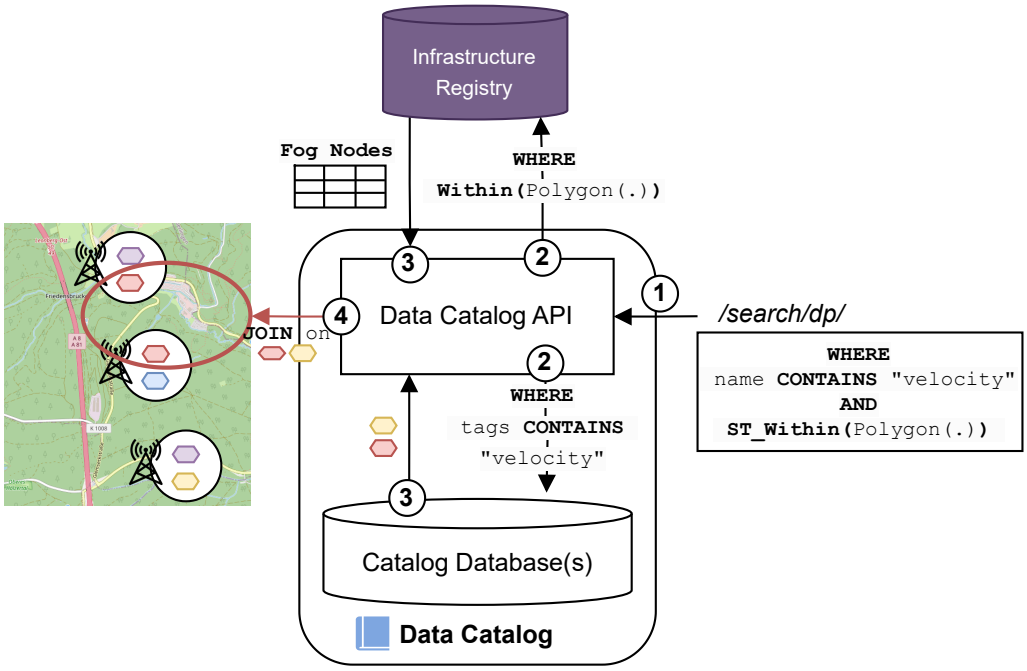


Abbildung 5.6: Beispielhafter Ablauf bei der Zusammenarbeit von Data Catalog und Infrastructure Registry, um eine Suchanfrage zu beantworten. Es werden Fog-Knoten in einer bestimmten Region mit Datenprodukten gesucht, die Geschwindigkeitsdaten anbieten. Die dargestellte Abfragesprache ist pseudohaft.

Horizontale Organisation im Data Catalog ermöglicht darüber hinaus, darzustellen, wie einzelne Daten durch die Systeme fließen. Umgesetzt wird diese Art der Organisation durch *Data Lineage*, also dem Nachverfolgen der Datenflüsse mit allen Transformationsschritten, die von Subsystem zu Subsystem existieren [Ole23]. Diese Art der Datennachverfolgbarkeit hat verschiedene Anwendungsgebiete und Vorteile. Dazu zählen beispielsweise die langfristige Erhöhung der Datenqualität durch die Identifikation von fehlerbehafteten Datenquellen und die Erhöhung der Transparenz und das Vertrauen von und in Machine-Learning-Modelle [BBFM23; IW09]. Die Erstellung des Data Lineages ist für die Connected-Cars-Domäne besonders herausfordernd, da jede Fog-Komponente mit einer lokalen Self-Serve Plattform eigene Technologien und Architekturen verwendet, die für den Gesamtlineage berücksichtigt werden müssen. In Kapitel 6.1.2 stelle ich eine Methodologie vor, die sich mit dieser Aufgabe befasst.

Die letzte der Art Organisation von Metadaten in Datenkatalogen nach Olesen-Bagneux [Ole23] ist die relationale Organisation. Hier werden semantische Beziehungen zwischen einzelnen Datenpunkten hergestellt und beschrieben. Ein Beispiel für eine solche Beziehung in der Connected-Cars-Domäne sind unterschiedliche Arten der Abstandsmessungen (wie Ultraschall und optische Verfahren). Sie werden im Data Catalog miteinander in Relation gesetzt, da sie die gleiche Umgebungsinformation auf verschiedene Weisen messen.

Mesh Experience Plane

In meinem Architekturvorschlag der Mesh Experience Plane sehe ich, analog zur Speicherung lokaler Governance-Richtlinien in der Data Product Experience Plane, die Speicherung globaler Richtlinien vor. Sie gelten für alle Datenprodukte und werden in der Komponente ***Global Policies*** gespeichert.

Darüber hinaus ist in dieser Schicht ein ***Data Marketplace*** verortet. Er ist dafür da, allen Datenproduzenten eine Plattform zu bieten, ihre Datenprodukte unter eigens definierten Bedingungen Datenkonsumenten anbieten zu können. Zu den Datenproduzenten und -konsumenten können dabei sowohl andere Domänen im Unternehmen des OEMs gehören, als auch dritte Parteien wie kooperierende Unternehmen.

Bezüglich der Architektur des Data Marketplaces stütze ich mich auf einen Vorschlag von Eichler et al. [EGH+23]. Sie stellen eine Architektur für einen sogenannten *Enterprise Data Marketplace* vor. In dieser verwenden sie einen Data Catalog als Grundlage, der gemeinsam mit weiteren Komponenten die zusätzlichen Funktionen bereitstellt, die für das Teilen von Datenprodukten notwendig sind. Dazu gehören die Festlegung von Preisen und Lizenz- und Nutzungsbestimmungen, wie beispielsweise die vereinbarte Nutzungsdauer von Datenprodukten. Auch die Übergabe von Informationen, die einen Datenzugriff ermöglichen, also beispielsweise die Übergabe von API-Keys oder Informationen für anderweitige Authentifizierungsmethoden, ist Aufgabe des Data Marketplaces [EGH+23].

Die Funktionalität des Data Marketplaces geht über das Teilen von Datenprodukten hinaus. Auch Infrastruktur oder Softwareservices können über den Marktplatz vertrieben werden (IAAS- und PAAS-Angebote) [EGH+23]. Für die Connected-Cars-Domäne bedeutet dies, dass hier geeignete Mechanismen implementiert werden können, um mehreren Parteien Zugriffe auf die Fog-Infrastruktur zu gewähren. Auf diese Weise können neue Mandanten in das angewandte MEC-Konzept (siehe Abschnitt 5.3.3) eingebunden werden, um ihre eigenen Services lokal und geografisch verteilt auszubringen. In meiner Data-Mesh-Architektur sind diese Mandanten entweder Teil von Domänenteams des OEMs, der den Data Mesh implementiert, oder externe Unternehmen. Dadurch können mehrere OEMs ihre Infrastruktur koordiniert unter den jeweiligen gewünschten Konditionen teilen. Dies ist besonders für RSDs wichtig, die so nicht flächendeckend von einem Unternehmen installiert werden müssen. Aber auch für die Entwicklung von Anwendungen, die über verschiedene Fahrzeugmodelle hinweg interagieren sollen, kann der Data Marketplace Verwendung finden. Der Einbezug von externen Parteien ist der Hauptunterschied zwischen dem Data Marketplace für meine vorgeschlagene Data-Mesh-Architektur und dem Enterprise Data Marketplace von Eichler et al. [EGH+23].

5.4 Federated Computational Governance

5.4.1 Grundlagen

In Abschnitt 5.3.4 stelle ich zwei Speicherkomponenten für Governance-Richtlinien in der Self-Serve Platform der Cloud vor. Eine für lokale und eine für globale Richtlinien. Lokale Richtlinien sind der Grund, weshalb das vierte Data-Mesh-Prinzip nach Dehghani [Deh22] als *federated* bezeichnet wird. Die Idee ist, dass die Ersteller der Datenprodukte selbst am besten wissen, welche

Governance-Eigenschaften oder Richtlinien für ihre Datenprodukte oder Domänen gelten. Beispiele für lokale Policies sind daher Angaben zur Datenqualität der Produkte und Angaben zu Vorschriften, unter welchen Bedingungen die Daten genutzt werden dürfen. Versprochene Angaben zu den Eigenschaften und Leistungen von Datenprodukten können auch als Service Level Agreements (SLAs) bezeichnet werden, die aus mehreren Service Level Objectivess (SLOs) zusammengesetzt sind [DHM23]. Globale Policies gelten für alle Datenprodukte [Deh22]. Dies können beispielsweise allgemeine Compliance-Richtlinien eines Unternehmens sein [GKD+23], wie etwa allgemeine Datenschutzbestimmungen, die immer von allen Domänenteams zu berücksichtigen sind.

Dehghani [Deh22] schlägt drei verschiedene Methoden vor, um Governance-Richtlinien automatisiert im Data-Mesh umzusetzen: Policies as Code, standardisierte Protokolle und Schnittstellen sowie automatisierte Tests und Monitoring. Policy as Code kann mittels der vom Data Mesh bereitgestellten Implementierung des Product Containers oder von Management Agents realisiert werden. Standardisierte Protokolle beziehen sich auf alle Mechanismen zur Erstellung und Verwaltung von Datenprodukten, die im Data Mesh für alle Domänen normiert werden. Zuletzt sollen automatisierte Tests und Monitoring als Kontrollmechanismen zur Überprüfung dienen, ob bestimmte definierte Richtlinien von den Datenprodukten auch eingehalten werden [Deh22].

5.4.2 Bedeutung und Umsetzung für die Connected-Cars-Domäne

Policy as Code

Um Policy as Code umzusetzen, sind konkrete Implementierungen des Product Containers und Management Agents notwendig, die in der Lage sind, gespeicherte Richtlinien zu berücksichtigen. Im Rahmen dieser Arbeit werden hierfür keine eigenen Konzepte vorgestellt. Stattdessen schlage ich vor, ein System aus der Literatur zu verwenden, das zumindest Privacy-Richtlinien für Connected Cars dynamisch berücksichtigen kann. Dieses System von Li et al. [LHSM22], genannt SAPP4C, eignet sich als Management Agent. In Abbildung 5.7 ist es, verkürzt dargestellt, in der Funktion eines solchen Agents, eingebettet im Product Container, zu sehen. Der Agent wird auf alle Datenprodukte auf Fahrzeugen ausgebracht und ermöglicht es Fahrzeughaltern, eigene Datenschutzrichtlinien festzulegen. Es werden somit nur Daten von dem Fahrzeug durch das Datenprodukt geteilt, zu denen der Fahrer seine Zustimmung gegeben hat. Zusätzlich können auch situationale Richtlinien, basierend auf dem Umgebungskontext des Fahrzeugs, definiert werden. Hat der Fahrzeughalter keine oder eingeschränkte Erlaubnisse erteilt, kommen dynamische Anonymisierungsfilter zum Einsatz, die vor der Datenversendung angewandt werden [LHSM22]. Der Policy Store als Teil der Self-Serve Platform auf Fahrzeugen (siehe Abbildung 5.4), ermöglicht die Speicherung aller Richtlinien für ein Fahrzeug. In Abbildung 5.7 ist außerdem noch ergänzend der Metadatenspeicher der Self-Serve Platform der Fog zu sehen. Er erhält über den Observe Port die Metadaten aller auf dem Auto verfügbaren Datenprodukte und speichert sie lokal zu Discovery-Zwecken ab.

Standardisierte Protokolle und Monitoring

Standardisierte Protokolle, die Richtlinien durch Vorgaben der Verwaltungskomponenten in der Self-Serve Platform erzwingen, sind in meiner Architektur vor allem durch den Data Marketplace und die Data Product Management Engine gegeben. Der Data Marketplace legt fest, wie Datenprodukte untereinander ausgetauscht werden sollen und kann damit dafür Sorge tragen, dass Richtlinien

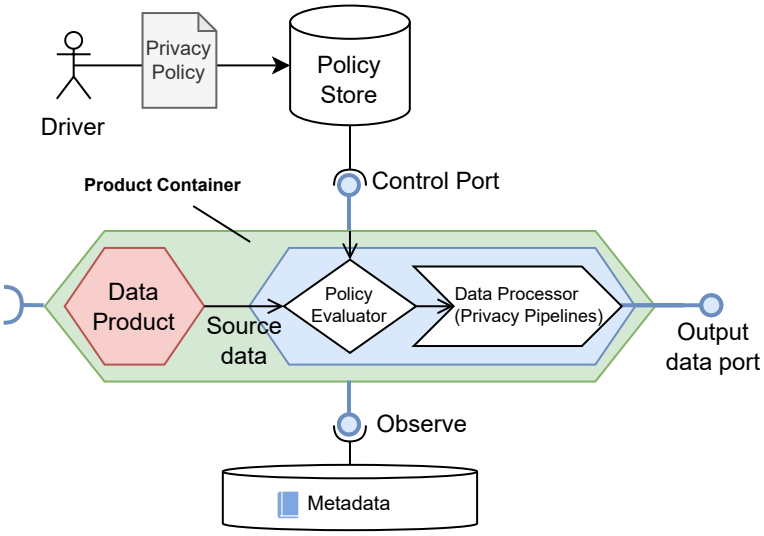


Abbildung 5.7: Ein auf ein Fahrzeug ausgebrachtes Datenprodukt verwendet das SAPP4C-Framework von Li et al. [LHSM22] als Management Agent.

in Bezug auf die Produktverwendung durch interne Parteien, aber auch externe Unternehmen umgesetzt werden. Die Data Product Management Engine ist unter anderem für das Erstellen von Datenprodukten zuständig. Hier können schon vorab nur Datenprodukte zugelassen werden, die auch alle definierten Richtlinien erfüllen.

In Kapitel 6.2 schlage ich ein konkretes Implementierungskonzept zur Umsetzung von Richtlinien vor, die bei der Erstellung neuer Datenprodukte überprüft werden können. Auch zeige ich zum Thema Monitoring in Kapitel 7.1, wie Endpunkte zur Überwachung bestimmter SLOs für Datenprodukte definiert werden können.

Speicherinhalte der Policy Stores

Insgesamt wurden in Abschnitt 5.3 bei der Vorstellung der Architektur der Self-Serve Platform drei Speicher für Governance-Richtlinien vorgestellt. Der erste befindet sich in der Fog-Plattform, um Richtlinien-Einstellungen speichern zu können, die für die ganze Fog-Komponente (wie ein Fahrzeug) gelten. Die zwei weiteren sind in der Plattform der Cloud verortet und speichern lokale sowie globale Richtlinien. Zusammenfassend speichern sie dabei folgende Daten:

- Versprochene Qualitätseigenschaften von Datenprodukten (SLA)
- API-Endpunkte zum Monitoring von versprochenen Qualitätseigenschaften (SLOs)
- Richtlinien von Datenprodukten, die bei der Erstellung auf ihnen basierender überprüft werden können (z.B. Zugriffsbeschränkungen für bestimmte andere Domänen)
- Agent-Implementierungen und -konfigurationen, die mit allen oder bestimmten Datenprodukten ausgebracht werden müssen

6 Implementierung ausgewählter Aspekte

Im vorherigen Kapitel 5 stelle ich Überlegungen und Konzepte vor, die es erlauben, dezentral organisierte Datenarchitekturen mithilfe der Anwendung der Data-Mesh-Prinzipien für die Connected-Cars-Domäne zu realisieren. Aufgrund der Größe des Konzeptumfangs ist es jedoch im Rahmen dieser Arbeit nicht möglich, alle Details auszuarbeiten. Dazu gehört zum Beispiel die genaue Architektur der einzelnen Komponenten der Self-Serve Platforms. Stattdessen beschränke ich mich in diesem Kapitel auf zwei ausgewählte Aspekte des Gesamtkonzepts, für die ich detailliertere Implementierungskonzepte vorstelle. Die prototypische Umsetzung dieser Konzepte ist im Anschluss das Thema von Kapitel 7.

6.1 Anlegen von Datenprodukten

Der erste ausgewählte Aspekt ist das Anlegen von Datenprodukten durch Entwickler der Domänenteams. An ihm lässt sich besonders gut veranschaulichen, welche Artefakte für die Erstellung neuer Datenprodukte notwendig sind und wie die Konfiguration der in Kapitel 5 vorgestellten geografisch verteilten Datenprodukte für die Connected-Cars-Domäne umgesetzt werden kann.

Die grundsätzliche Anlegung von Datenprodukten ist in meinem Konzept der erste Schritt für Domänenteams, um ihr entwickeltes Datenprodukt auf den Weg zur Veröffentlichung zu bringen. Wie in gängigen Prozessmodellen des Software Engineerings üblich [LL13], sehe auch ich für Datenprodukte Maßnahmen zur Qualitätssicherung via geforderten Tests und Abnahmereviews vor. Nachdem alle Konfigurationen für das Datenprodukt angelegt wurden (*Plan*), wird es auf eine Testumgebung ausgebracht, um seine Funktionsweise final zu testen. Anschließend findet ein Abnahmereview statt, in dem alle Artefakte sowie die Testberichte überprüft werden. Dieser Abnahmeprozess dient als Kontrollmechanismus, um zu verhindern, dass vor allem in die ressourcenbegrenzte Fog disfunktionale Datenprodukte ausgebracht werden, die dort Schaden anrichten könnten. Zu sehen ist der Gesamtprozess in Abbildung 6.1. Als Maßnahme zur Anwendung des Prinzips Computational Governance habe ich zwischen Plan und Test den Schritt einer automatischen Richtlinienüberprüfung eingefügt. Hier werden alle Richtlinien vorab automatisiert überprüft, die kein manuelles Review erfordern. Der zweite Implementierungsaspekt in diesem Kapitel (siehe Abschnitt 6.2) befasst sich genau mit einem solchen Konzept der automatisierten Überprüfung.

6.1.1 Metamodell

Für alle Metadaten und Artefakte, die beim Anlegen von Datenprodukten berücksichtigt werden, habe ich ein Datenmodell entworfen. Die Entitäten des Modells und ihre Beziehungen zueinander sind in Abbildung 6.2 als UML-Diagramm zu sehen. In der wissenschaftlichen Literatur gibt es bereits von Driessen et al. [DHM23] einen Vorschlag für ein Metamodell für Datenprodukte

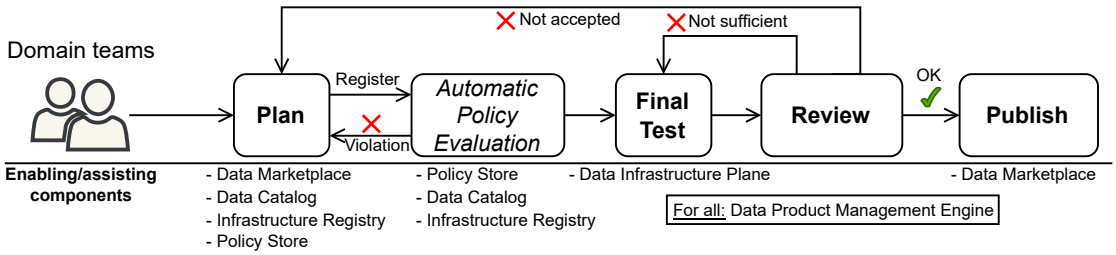


Abbildung 6.1: Anlageprozess neuer Datenprodukte von dem Erstellen von Konfigurationen und der Angabe der Artefakte (Plan) bis zur Veröffentlichung (Publish).

im Data Mesh. Es beschreibt Daten, die zu einem Datenprodukt in Datenkatalogen abgelegt werden. Dieses Metamodell, genannt Data Product Model Template (ProMoTe), hat, da es auch im Kontext des Data Meshs entworfen wurde, Gemeinsamkeiten mit meinem Datenmodell. Es ist jedoch nicht in Gänze für die Connected-Cars-Domäne gemäß meines Konzepts geeignet. Im Modell von Driessen et al. [DHM23] ist es nicht vorgesehen, verschiedene Verteilungsmodelle für Datenprodukte (siehe Kapitel 5.2.2), ausgebracht auf Infrastrukturknoten mit unterschiedlichen Eigenschaften, zu unterstützen. Zusätzlich verfeinert mein Konzept die Konfiguration von einzelnen Artefakten des Datenprodukts, da der Fokus auf der konstruktiven Anlegung der Produkte liegt.

Im Folgenden stelle ich die Entitäten meines Metamodells und ihre Zusammenhänge vor. Konkrete Attribute, die zu diesen Entitäten konfiguriert werden, zeige ich in Kapitel 7.1 bei der Vorstellung des zugehörigen Prototypen.

Zentrale Entitäten um das Data Product

Zentral für das Metamodell ist die dargestellte Entität des `Data Products`. Jedes Datenprodukt gehört genau einer Domäne (`Domain`) und kann selbst Daten aus anderen Datenprodukten nutzen, zu denen es über den Data Marketplace Zugriff angefordert hat. Darüber hinaus können zu jedem Datenprodukt `Local Policies` definiert werden. Sie legen Richtlinien fest, die eingehalten werden müssen, wenn das Datenprodukt von Datenkonsumenten genutzt wird. Dazu gehören auch Nutzungseinschränkungen des Datenprodukts für bestimmte Domänen, weshalb den `Local Policies` im UML-Diagramm mehreren Domänen zugeordnet werden können.

Die Entität `Distribution` ist dafür da, um zu konfigurieren, von welchem Verteilungsmodell (siehe Kapitel 5.2.2) das Datenprodukt Gebrauch macht. Zusätzlich wird mit ihr festgelegt, auf welche Infrastrukturknoten das Datenprodukt ausgebracht werden soll (`Infrastructure Type`) und in welcher geografischen Region diese Infrastrukturknoten liegen müssen, sofern es sich um Fog-Knoten handelt. Auf diese Weise wird dem System mitgeteilt, auf welche Infrastruktur die Datenprodukte ausgebracht werden sollen. Zuletzt kann ein Datenprodukt beliebig viele Qualitätsversprechen für spätere Datenkonsumenten in Form von SLOs angeben (Entität `Quality SLO`).

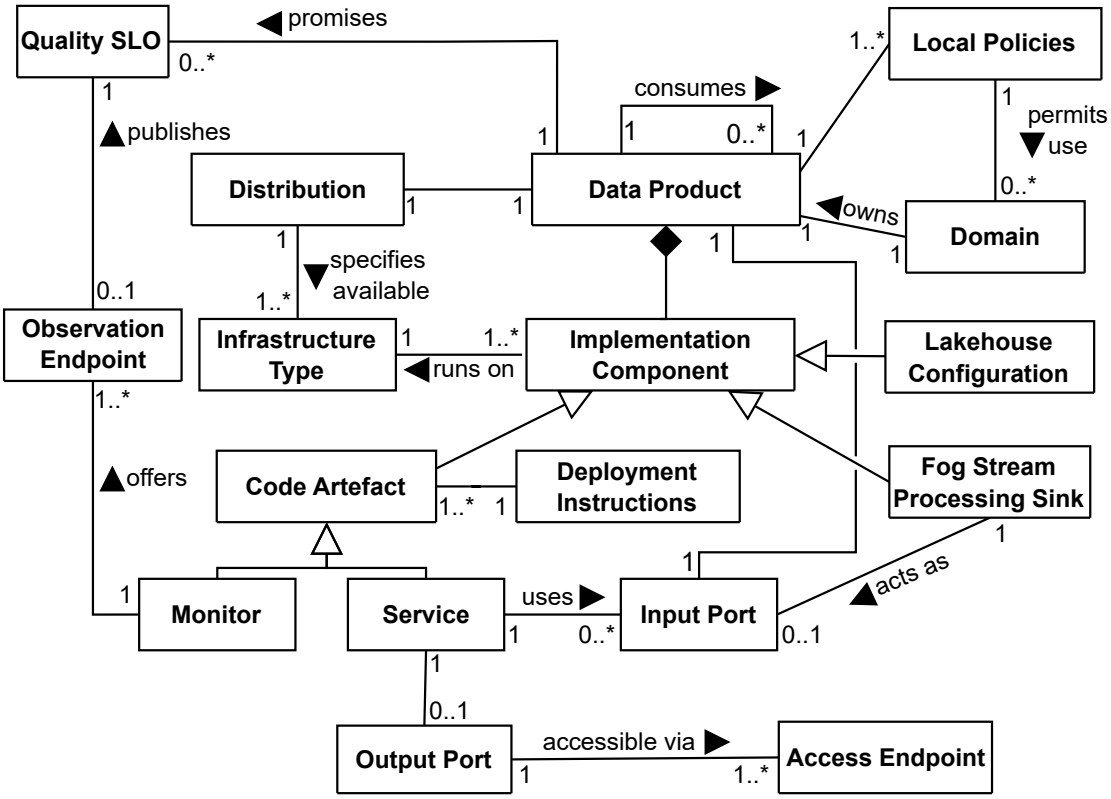


Abbildung 6.2: UML-Modell für zu erhebende Metadaten bei der Datenproduktregistrierung

Entitäten zur Modellierung der Datenproduktimplementierungen

Für das Anlegen von Datenprodukten müssen Entwickler genau angeben, welche Softwareservices Teil der Datenproduktimplementierung sind. Das Herunterbrechen von Datenprodukten auf einzelne Services, die es implementieren, hat mehrere Vorteile:

- Datenprodukte sollen gemäß der Idee des Produktlebenszyklus (siehe Kapitel 5.2.3) automatisiert in Test- und Produktionsumgebungen ausgebracht werden können. Durch die Definition einzelner Services ist es möglich, für unterschiedliche Teile des Datenprodukts unterschiedliche Ausbringungsmethoden zu konfigurieren. Das ist insbesondere bei zusammengesetzten Datenprodukten notwendig, wenn die verwendeten Infrastrukturknoten jeweils andere Update-Mechanismen zur Verfügung stellen.
- Data Lineage lässt sich auf diese Weise nicht nur in Form von Abhängigkeiten zwischen Datenprodukten modellieren, sondern auch auf Grundlage einzelner Services, die das Datenprodukt implementieren. Damit wird eine höhere Auflösung des Data Lineages erreicht.
- Die Vorgabe klarer und einheitlicher Definitionen für beteiligte Services erleichtert den Review- und Dokumentationsprozess eines Datenprodukts, insbesondere für einen domänenfremden Reviewer. Sie können als Maßnahme der Richtlinienerzwingung mittels standardisierter Protokolle gesehen werden (siehe Kapitel 5.4).

Insgesamt gibt es vier unterschiedliche Service-, beziehungsweise Konfigurationskomponenten, um die Implementierung eines Datenprodukts zu modellieren. In dem Metamodell von Abbildung 6.2 werden sie deshalb auch als `Implementation Component` bezeichnet. Zu jeder Komponente muss festgelegt werden, auf welchem Typ von Infrastruktur, die Teil der vorher definierten `Distribution` ist, sie ausgebracht werden soll (`Infrastructure Type`). Dabei kann der gleiche `Infrastructure Type` mehreren Infrastrukturknoten zugeordnet sein, um zusammengesetzte oder verteilte Datenprodukte (siehe Kapitel 5.2.2) zu ermöglichen.

- **Service:** Ein `Service` implementiert beliebige Softwarefunktionen. Von allen Arten der `Implementation Components` lässt er die meisten Freiheiten bezüglich seiner Funktionalität. Zu ihm werden Softwareartefakte wie Programmcode oder ausführbare Dateien konfiguriert. Auch werden Anweisungen zur Ausbringung der Software gegeben, wie beispielsweise Konfigurationsdateien für Orchestrierungssoftware. Im Metamodell werden diese Daten von der Entität `Deployment Instructions` repräsentiert. Darüber hinaus können für Services die Input- und Output-Ports des Datenprodukts definiert werden. Für die Konfiguration von `Input Ports` werden dabei Datenprodukte verlinkt, die das Datenprodukt als Konsumentenabhängigkeit definiert hat. Die genauen vereinbarten Zugriffsmechanismen und Datenkontrakte werden über den Data Marketplace vorher vereinbart. Für die Definition von `Output Ports` werden konkrete `Access Endpoints` definiert. Diese geben an, welche Daten (Schemas) auf welche Weise (SQL-API, Message-Broker, ...) Datenkonsumenten verfügbar gemacht werden.
- **Monitor:** Ein `Monitor` hat die gleichen Eigenschaften wie ein Service (im Metamodell ebenso Kindklasse von `Code Artefact`), ist aber nicht in der Lage, als Input- oder Output-Port zu agieren. Stattdessen können über ihn `Observation Endpoints` konfiguriert werden, die den Observe Port des Datenprodukts (siehe Kapitel 5.2.1) mit konkreten Zugriffspfaden konfigurieren. Diese Endpunkte können dafür verwendet werden, um allgemeine Metriken des Datenprodukts zu überwachen oder um konkrete `Quality SLOs` die das Datenprodukt verspricht, für Konsumenten überprüfbar zu machen. Die Logik diese Daten bereitzustellen, wird durch die Software des Monitors implementiert.
- **Fog Stream Processing Sink:** Diese Komponente erlaubt es, Transformationspipelines mithilfe der Fog Stream Processing Engine zu erstellen, die von der Self-Serve Platform angeboten wird (siehe Kapitel 5.3.4). Sie agiert dabei als Datensenke und kann auch als Input Port für ein Datenprodukt verwendet werden. Dazu muss ein anderes Datenprodukt einen Output Port mit entsprechendem `Access Endpoint` als Datenquelle bereitstellen.
- **Lakehouse Configuration:** Die letzte Konfigurationskomponente eines Datenprodukts ist die `Lakehouse Configuration`. Sie ermöglicht es, etwaige Schemas oder Pfadstrukturen für das Lakehouse der zentralen oder lokalen Self-Serve Platform zu definieren. Damit stellt sie keine Software bereit, sondern dient lediglich zur Konfiguration der Metadaten des Lakehouses.

6.1.2 Erfassen des Data Lineage

Im besten Fall sollte ein Data Catalog selbst nach geeigneter Anfangskonfiguration in der Lage sein, Lineage-Graphen zu bilden, indem er vorgesehene Schnittstellen bei den verwendeten Datenverarbeitungstechnologien abfragt. Dies ist im Allgemeinen keine einfache Aufgabe, da viele Frameworks und Tools unterschiedliche oder auch eingeschränkte Funktionalitäten zum Tracken von

Lineage anbieten [Ole23]. Ein Ansatz zur Standardisierung solcher Lineage-APIs ist OpenLineage[1]. Tools wie Apache Spark und Airflow unterstützen sie bereits. Für die Connected-Cars-Domäne existieren jedoch noch keine Tools, die in der Lage sind, verschiedene Technologien verteilt auf mehreren Fog-Knoten zu berücksichtigen und zu einem gemeinsamen Graphen zu verbinden. Deshalb schlage ich ein System zur manuellen Konfiguration des Data Lineage beim Anlegen neuer Datenprodukte vor.

Die Idee ist, dass basierend auf den genauen Konfigurationen der Datenproduktimplementierung durch die `Implementation Components` des Metamodells den Entwicklern ein Lineage-Template angeboten wird. Dieses Template zeigt visuell alle relevanten Komponenten der Datenarchitektur der Datenprodukte, aber auch Komponenten, die im Kontext der Datenarchitektur relevant sein können. Zu solchen Kontextkomponenten gehören die Eigenschaften der lokalen Self-Serve Platform wie verfügbare Message Broker (Informationen hierfür werden aus der Infrastructure Registry, siehe Kapitel 5.3.4, entnommen). Darüber hinaus werden auch Datenschnittstellen dargestellt, die direkt auf dem Infrastrukturknoten verfügbar sind. Sie sind Teil der Data Collection Interfaces, der Self-Serve Platform in der Fog. Solche Datenschnittstellen sind beispielsweise Sensorwerte, die der Infrastrukturknoten erhebt.

Die Aufgabe der Entwickler des Datenprodukts ist es, basierend auf diesem Template den Datenfluss zwischen den `Implementation Components` einzuzeichnen sowie etwaige Verbindungen zu Schnittstellen der Data Collection Interfaces. Für jede Verbindung wird dabei, im Falle von Transformationen, das Datenmodell angegeben, das die neuen Daten repräsentiert. Auf diese Weise erhält man den geplanten Data Lineage eines Datenprodukts. Sobald dieses das Review (siehe Abbildung 6.1) überstanden hat und veröffentlicht wird, kann der Lineage dann dem gesamten Lineage aller Connected-Car-Datenprodukte hinzugefügt werden. Im Metadatenmodell von Abbildung 6.2 sind die Data-Lineage-Beziehungen zwischen den `Implementation Components` aus Gründen der Übersichtlichkeit nicht eingezeichnet.

Das beschriebene Konzept beschränkt sich an dieser Stelle auf die initiale Erhebung des Lineages bei Produktanlegung. Für die Aktualisierung von Datenprodukten oder für operationale Systeme, für die der Lineage ebenfalls nachvollziehbar sein soll, benötigt es ergänzende Konzepte, die hier nicht behandelt werden. Die Umsetzung kann jedoch in ähnlicher Weise erfolgen.

6.2 Richtlinienüberprüfung via Data Lineage

Der zweite Aspekt, zu dem ich ein konkretes Implementierungskonzept vorstelle, befasst sich mit der automatisierten Richtlinienüberprüfung für Datenprodukte unter Verwendung des Data-Lineage-Graphen. Ausgewählt habe ich ihn erstens, da er gut veranschaulicht, wie die in Abschnitt 5.4.2 vorgestellten Konzepte der Richtlinienerzwingung via Policy as Code und standardisierte Protokolle konkret in der Praxis implementiert werden können. Zweitens bietet das Konzept Anwendungen, die besonders für die Connected-Cars-Domäne vorteilhaft sind.

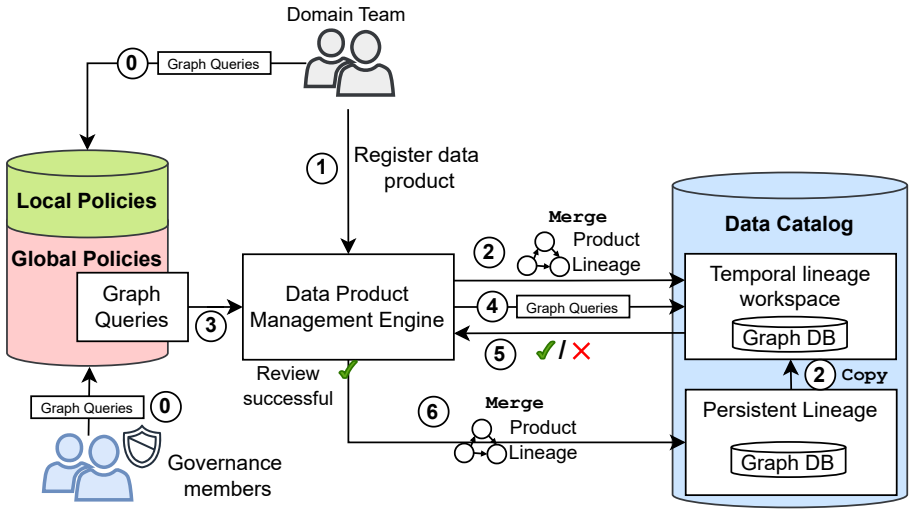


Abbildung 6.3: Prozess, in welchem Graphabfragen für den Data-Lineage-Graphen genutzt werden, um Governance-Richtlinien des Data Mesh automatisiert zu überprüfen.

6.2.1 Allgemeines Konzept

Das Konzept ist im Anlegungsprozess neuer Datenprodukte (siehe Abbildung 6.1) im Schritt `Automatic Policy Evaluation` verortet. Es basiert darauf, dass der Data-Lineage-Graph, der im Schritt `Plan` von den Entwickern für das Datenprodukt erstellt wurde (siehe Abschnitt 6.1.2), genutzt wird, um ihn auf Eigenschaften zu überprüfen, die eine Richtlinienverletzung darstellen. Diese Richtlinien können dabei sowohl globale Richtlinien sein, die von dem Data-Governance-Team des Data Mesh festgelegt wurden und für alle Datenprodukte gelten, als auch lokale Richtlinien von Datenprodukten, die das neu anzulegende Datenprodukt als Abhängigkeit verwendet. Wird eine Richtlinienverletzung auf Basis des Lineage-Graphen erkannt, wird das Datenprodukt erst gar nicht für Tests zur Ausbringung des Datenprodukts oder manuelle Review-Schritte (siehe Abbildung 6.1) freigegeben. Auf diesem Weg erhalten die Entwickler des Datenprodukts schnelles Feedback und gleichzeitig werden Reviewer zur manuellen Überprüfung von Richtlinien entlastet. Grundannahme hierbei ist, dass der Lineage von den Entwicklern korrekt eingezeichnet wurde. Dies muss deshalb im Rahmen des Reviews überprüft werden.

Zur automatisierten Überprüfung der Richtlinien werden sie in Form von Abfragen für graphenbasierte Datenbanken in den Policy Stores der zentralen Self-Serve Platform der Cloud (siehe Kapitel 5.3.4) abgelegt. Die Graphdatenbanken werden genutzt, um den Data Lineage des Systems zu speichern. Immer wenn die Richtlinien bei der Anlegung neuer Datenprodukte überprüft werden müssen, werden sie von der Data Product Management Engine aus den Policy Stores gelesen und auf dem Data-Lineage-Graphen des neuen Datenprodukts ausgeführt. Der gesamte Prozess dieses Konzepts ist in Abbildung 6.3 zu sehen. Von Domänenteams werden lokale Richtlinien in Form von Graphabfragen für bestimmte Datenprodukte abgelegt und von Teams, die globale Richtlinien festlegen (`Governance Members`), globale. Sobald ein Domänenteam ein neues Datenprodukt zur Registrierung einreicht, wird der neu definierte Lineage-Graph mit dem Gesamtlineage des Systems

[1] https://openlineage.io/

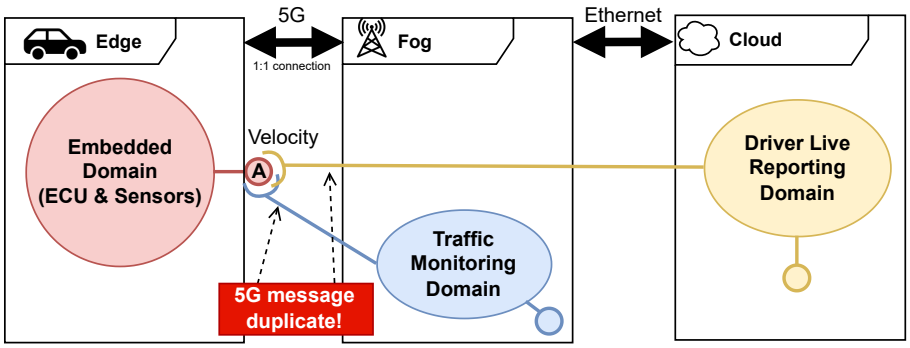


Abbildung 6.4: Zwei Domänen beziehen die gleichen Datensätze über die gleichen Fog-Knoten und erzeugen damit ein unnötiges Nachrichtenduplikat.

verbunden (Schritt 2). Dieser Graph wird nun temporär in einer eigenen Datenbank gespeichert und dafür verwendet werden, um die Graphabfragen auszuführen. Die Data Product Management Engine wertet aus, welche Richtlinien für die automatisierte Überprüfung relevant sind und führt die Abfragen anschließend auf diesem Graphen aus (Schritt 3 und 4). Werden Richtlinienverletzungen auf Grundlage dieser Abfragen erkannt, wird das Datenprodukt abgewiesen. Wenn nicht, wird, sobald auch die Tests und das manuelle Softwarereview erfolgreich waren, der Lineage des neuen Datenprodukts in den persistenten Lineage des Gesamtsystems eingetragen.

Das vorgestellte Konzept ist eine Umsetzung des Prinzips Policy as Code (siehe Kapitel 5.4), da die Richtlinienüberprüfung mit dynamisch programmierbaren Abfragen umgesetzt wird. Es macht gleichzeitig von standardisierten Protokollen Gebrauch (siehe Kapitel 5.4), da der Prozess der Lineage-Erhebung (siehe Abschnitt 6.1.2) und seine Einpflegung in den gesamten Lineage-Graph für alle Datenprodukte durch die Data Product Management Engine normiert wird. Somit können keine nicht kompatiblen Lineage-Definitionen zu Datenprodukten angegeben werden.

6.2.2 Anwendungsgebiete

Je mehr Informationen zum Datenfluss im Lineage-Graphen modelliert werden, desto mehr Anwendungen ergeben sich. Solche können zum Beispiel sein:

- Erzwingung von Architekturmustern
- Kontrolle der Einhaltung ortsabhängiger Datenschutzrichtlinien
- Vermeidung von Datenduplikaten in den Domänen

Das Problem der Datenprodukte wird von Goedegebuure et al. [GKD+23] als ein häufig aufgeführtes Problem bei der Anwendung des Data-Mesh-Konzepts identifiziert. Aufgrund der dezentralen Datenarchitekturen, die von weitestgehend unabhängigen Domänenteams entwickelt werden, besteht durch mangelnde Absprachen die Gefahr, dass die gleichen Daten verwaltet oder verarbeitet werden. Solche Duplikate erzeugen zusätzliche Managementkosten für die Daten [GKD+23]. Für die Connected-Cars-Domäne, mit dem Einbezug von Fog-Ressourcen, ist dieses Problem noch relevanter, da vor allem die Ressourcen nahe der Edge begrenzt sind und durch Duplikate

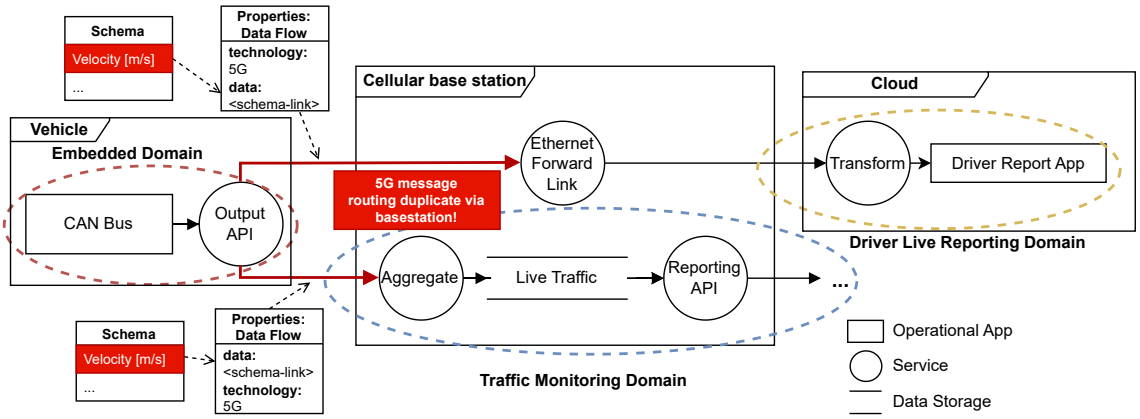


Abbildung 6.5: Beispiel, wie ein Data-Lineage-Graph für das Szenario konzeptionell aussehen könnte

verschwendet werden. Im Folgenden stelle ich anhand eines konkreten Szenarios vor, wie das Problem der Datenduplikate mit dem vorgestellten Lineage-Überprüfungskonzept gelöst werden kann.

Szenario

Die Schonung von Rechen-, Netzwerk- und Speicherressourcen ist in der Connected-Cars-Domäne wichtig, um mit der großen Menge an zu verarbeitenden Daten zurechtzukommen. Besonders bei Edge- und Fog-Computing sind die einzelnen Knoten häufig in ihren Ressourcen beschränkt. Domänenteams, die Connected-Cars-Anwendungsfälle implementieren, sind deshalb dazu angehalten, keine nicht zwingend benötigten Daten in ihren Datenarchitekturen zu verwalten. Ein Problem, das in der dezentralen Datenarchitektur des Data Mesh auftreten kann, ist, dass unterschiedliche Domänenteams auf der gleichen Infrastruktur die gleichen Daten verarbeiten wollen, um sie in ihren Anwendungen oder Datenprodukten zu verwenden. Dies sind potenzielle Datenduplikate, die unnötige Ressourcen verbrauchen.

Abbildung 6.4 zeigt ein konkretes Szenario. Zwei Domänen (`Traffic Monitoring` und `Driver Live Reporting`) benötigen für ihre Anwendungsfälle aktuelle Geschwindigkeiten der Fahrzeuge. Dazu greifen sie auf ein Datenprodukt der `Embedded Domain` zu, die dafür eine API direkt auf den Netzwerkschnittstellen der Fahrzeuge anbietet. Da im Szenario die Fahrzeuge nur über Mobilfunkinternet erreichbar sind, müssen die Daten für beide Datenkonsumenten über Mobilfunkmäste geroutet werden. Dies erzeugt folglich Nachrichtenduplikate bei eben diesen.

Erkennung der Duplikate

Im Zuge des Anlegeprozesses neuer Datenprodukte, der im vorherigen Abschnitt 6.1 beschrieben wurde, legt eine der beiden Domänen ein neues Datenprodukt ab. Dieses Datenprodukt würde bei Ausbringung das zu vermeidende Datenduplikat erzeugen. Wie in Abschnitt 6.1.2 beschrieben, wird im Zuge des Anlegeprozesses ein neuer Lineage-Graph für das Datenprodukt erstellt. Ein Ausschnitt davon, wie dieser Graph für das betrachtete Szenario aussehen könnte, ist in Abbildung 6.5 zu

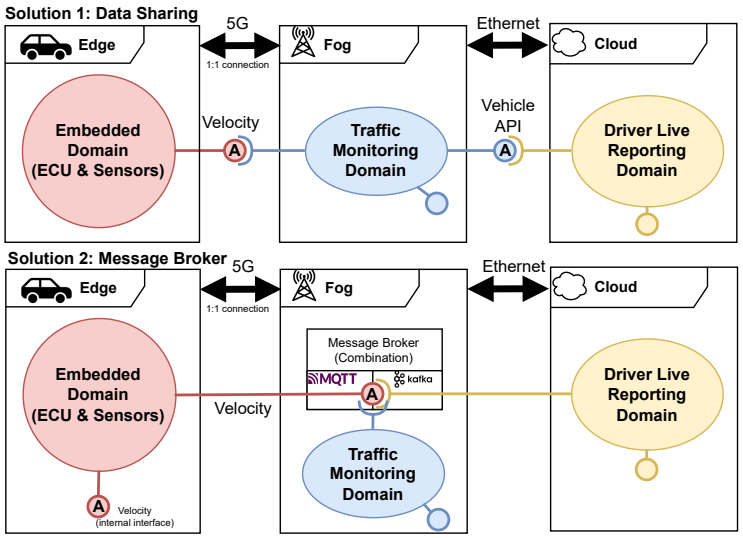


Abbildung 6.6: Mögliche Lösungen für die beteiligten Domäne zur Einhaltung der Richtlinie

sehen. Der Lineage-Graph enthält Referenzen auf gespeicherte Datenschemas von Output Ports und einzelnen `Implementation Components` (siehe Abschnitt 6.1.2). Mithilfe dieser Angaben wird bei der anschließenden automatisierten Richtlinienüberprüfung berechnet, dass zwei Datenflüsse mit den gleichen Daten zu dem gleichen Typ von Infrastrukturknoten führen. Grundlage dieser Berechnung ist dabei, wie in Abschnitt 6.2.1 beschrieben, eine Graphabfrage, die im globalen Policy Store von Entscheidungsträgern des OEM gespeichert wurde.

Auflösung der Duplikate

Nachdem die Duplikate im Szenario erkannt wurden, sind die Domänenteams gezwungen, alternative Architekturansätze zu etablieren. In Abbildung 6.6 sind zwei Möglichkeiten abgebildet, wie beide Domänen die Geschwindigkeitsdaten erhalten können, ohne Datenduplikate auf den Basisstationen zu verursachen. Die erste Lösung sieht vor, dass die Domäne `Traffic Monitoring` ihrerseits ein Datenprodukt über Ethernet-Schnittstellen anbietet. Dieses Datenprodukt kann von der `Driver Live Reporting`-Domäne verwendet werden. Die zweite Lösung besteht darin, dass die rot eingezeichnete `Embedded`-Domäne eine zusätzliche Zugriffsmöglichkeit ihrer Daten über einen Message-Broker auf Fog-Komponenten ermöglicht. Dazu können beispielsweise ein MQTT-Broker für die 5G-Kommunikation und Apache Kafka für die Kommunikation im Ethernet-Netzwerk gewählt werden. Die Gründe, weshalb diese Zweiteilung sinnvoll sein kann, wurden in Kapitel 4.1 bei der Vorstellung der Connected-Cars-Architektur von Mostefaoui et al. [MMH+22] angesprochen.

7 Prototypen

In diesem Kapitel stelle ich die prototypische Umsetzung meiner in Kapitel 6 eingeführten Konzepte vor. Für die Anlegung von Datenprodukten ist der Prototyp eine Benutzeroberfläche, die es Entwicklern nutzerfreundlich ermöglicht, ihre Datenprodukte für die Connected-Cars-Domäne anzulegen. Da abseits der Oberfläche keine tatsächliche Funktionalität (wie ein persistenter Eintrag im Data Catalog) implementiert wird, handelt es sich um einen horizontalen Prototypen [CV97]. Der zweite Prototyp zeigt, wie das Konzept zur automatischen Überprüfung von Governance-Richtlinien technisch umgesetzt werden kann.

7.1 Benutzeroberfläche zur Anlegung neuer Datenprodukte

Die Benutzeroberfläche habe ich mithilfe von Vue.js, einem Framework für JavaScript, entwickelt. Ihre erste Ansicht ist in Abbildung **??** zu sehen. Der Nutzer befindet sich im Schritt *Plan*, welcher der erste Schritt bis zur tatsächlichen Veröffentlichung eines Datenprodukts ist sowie der einzige, der im Prototypen konkret abgebildet wird (siehe Kapitel 6.1). Rechts daneben befinden sich eingeklappt alle Formulare, die zur Anlegung des Datenprodukts benötigt werden. Die Vorstellung dieser Formulare ist Thema der nächsten Unterabschnitte.

7.1.1 Allgemeine Informationen und geografische Produktverteilung

Abbildung 7.2 zeigt die ersten beiden Formulare zum Planen des Datenprodukts. In Abbildung 7.2a werden allgemeine Informationen über das Datenprodukt wie Name, Beschreibung, Domänenzugehörigkeit und Tags für die Kategorisierung des Produkts eingegeben. In Abbildung 7.2b bestimmt der Entwickler, um was für ein Datenprodukt es sich handelt (normal, zusammengesetzt, verteilt) und auf welche Infrastruktur es ausgebracht werden soll (zum Beispiel bestimmte Fahrzeugmodelle, RSDs, bestimmte Cloud-Regionen). Er kann außerdem geografische Regionen auf einer Karte eingrenzen, wenn er möchte, dass nur ganz bestimmte örtliche Infrastrukturknoten das Datenprodukt erhalten.

7.1.2 Auswahl von Datenproduktabhängigkeiten

Der nächste Formularschritt ist in Abbildung 7.3 zu sehen. In dieser Ansicht kann der Entwickler auswählen, welche anderen Datenprodukte das neue Datenprodukt als Abhängigkeit verwendet. Die konkrete Auswahl wird durch eine Verlinkung der Benutzeroberfläche des Data Marketplaces realisiert (nicht Teil des Prototyps). Der Nutzer hat im gezeigten Beispiel von Abbildung 7.3 drei Datenprodukte zur Verwendung ausgewählt und ihren Zugriff über den Datenmarktplatz konfiguriert. Für diese Produkte sind jeweils ihre Tags, der Status ihres Lebenszyklus (`deployed`,

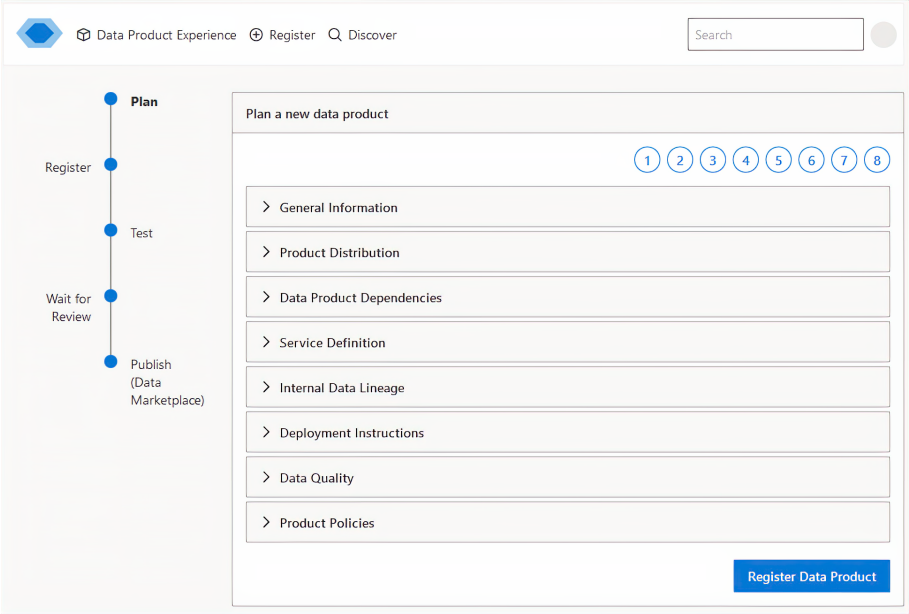
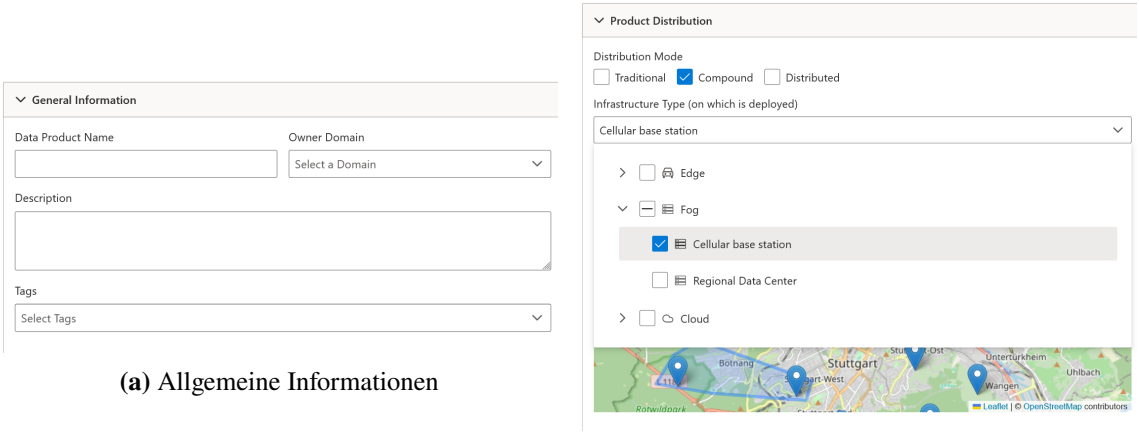


Abbildung 7.1: Startseite zur Anlegung neuer Datenprodukte



(a) Allgemeine Informationen

(b) Geografische Verteilung

Abbildung 7.2: Allgemeine Produktinformationen und Auswahl der Infrastruktur

`stopped`, ...) sowie eine Anzahl an Bewertungssternen angegeben. Letztere sind als Implementierung der von Dehghani [Deh22] angedachten *Feedback Loop* gedacht. Beliebte Datenprodukte werden dabei durch Bewertungen von Entwicklern anderer Domänen mit einem höheren Suchrang belohnt, um sie zum Beispiel von weniger qualitativ hochwertigen Produkten hervorzuheben.

7.1.3 Definition der Datenproduktimplementierung

Der vierte Schritt ist das Definieren aller beteiligten Services sowie Konfigurationen, die Teil der Datenproduktimplementierung sind. Hierzu können auf der linken Seite des Formulars von Abbildung 7.4 `Implementation Components` (siehe Kapitel 6.1.1) hinzugefügt werden.

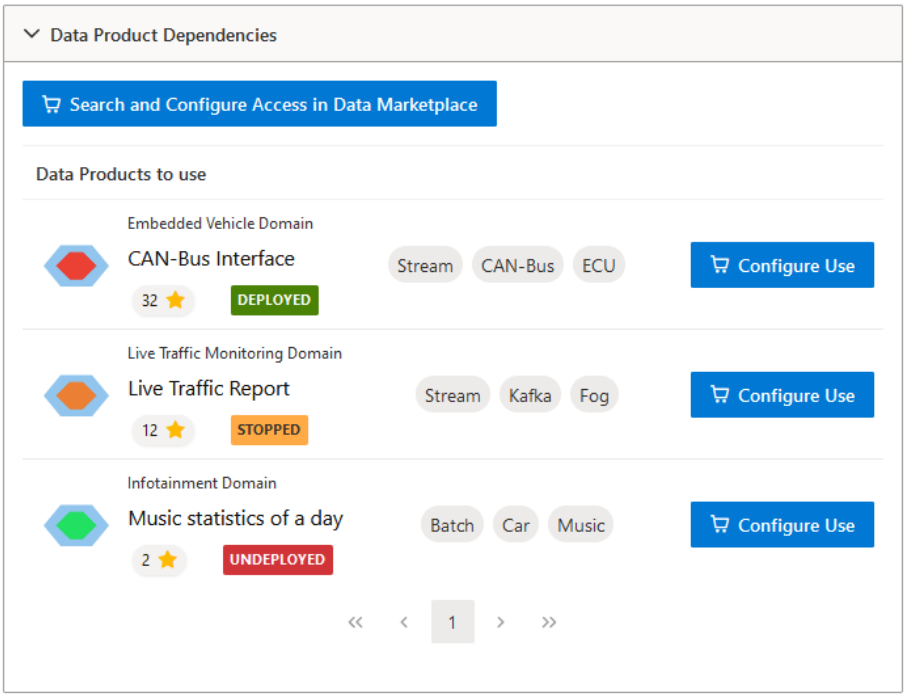


Abbildung 7.3: Auswahl von anderen Datenprodukten als Abhängigkeiten

Auf der rechten Seite des Formulars werden alle Daten zu den einzelnen Services und Konfigurationskomponenten eingegeben. Für jede Komponente muss dabei spezifiziert werden, auf welchen Typen von Infrastruktur sie später laufen soll (`Runs on Infrastructure`). Dabei kann nur eine Untermenge von den im Formular 7.2b bereits vorausgewählten Infrastrukturtypen ausgewählt werden. Die restlichen Formularinhalte unterscheiden sich je nach Implementierungskomponente:

- **Service:** In Abbildung 7.4 sieht man, dass zu einem Service der zugehörige Softwarecode, ausführbare Softwareartefakte und ein Link zur Dokumentation konfiguriert werden. Zur Konfiguration von Input Ports kann ein Link auf den jeweiligen Namen des verwendeten Datenprodukts gegeben werden. Die genauen vereinbarten Zugriffsmechanismen und Nutzungsbedingungen wurden bereits vorher über den Data Marketplace (siehe Formular 7.3) vereinbart. Die Definition von Output Ports wird in Abbildung 7.5 gezeigt. Es wird ein logisches Datenmodell angegeben. Dieses gibt auf semantischer Ebene an, welche Daten von dem Port bereitgestellt werden. Danach wird konfiguriert, welche Zugriffsmöglichkeiten auf diese Daten existieren sollen (SQL, Dateizugriff, Pub/Sub, HTTP-API, ...). Für jede dieser Zugriffsmöglichkeiten, im Metamodell von Abbildung 6.2 `Access Endpoints` genannt, werden Detaileinstellungen konfiguriert. Für einen Pub/Sub-Mechanismus ist das in Abbildung 7.5b zu sehen. Im Beispiel wird ein Message-Broker ausgewählt, der von der Self-Serve Platform bereits auf der Infrastruktur angeboten wird, und ein Topic festgelegt. Anschließend kann ein physisches Datenmodell angehängt werden, das konkret definiert, wie das logische Datenmodell auf die physische Nachrichtenrepräsentation abgebildet wird. Zuletzt können noch Einstellungen konfiguriert werden, die technische Eigenschaften des `Access Endpoints` beschreiben, wie zum Beispiel die maximale Payload-Größe. Die Checkboxen auf der rechten

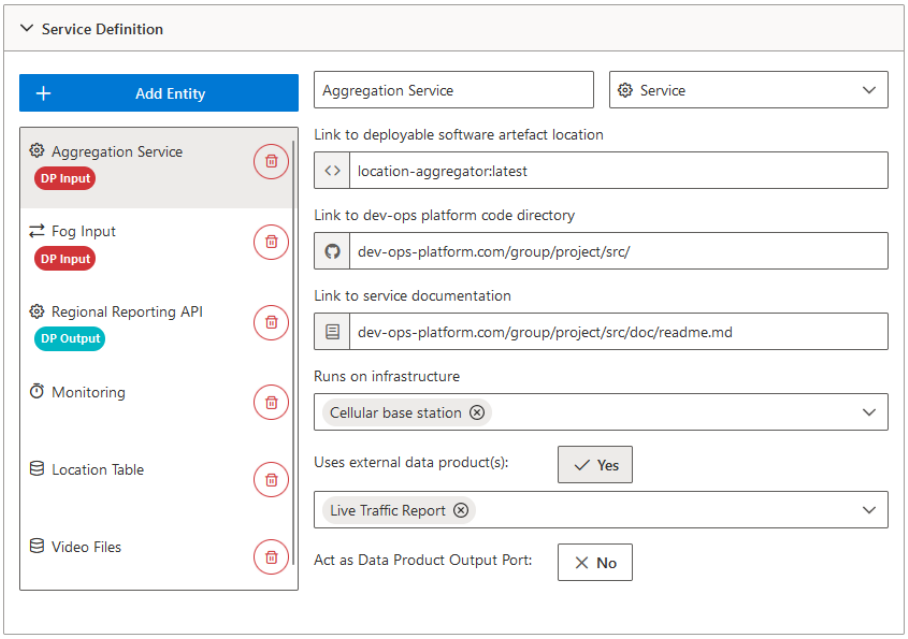
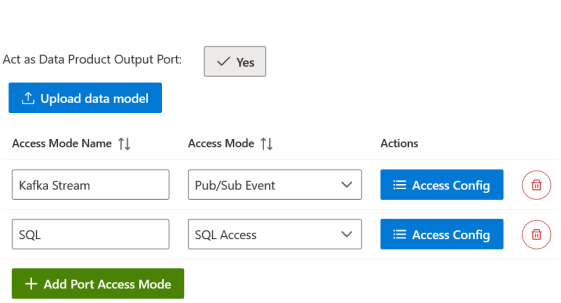


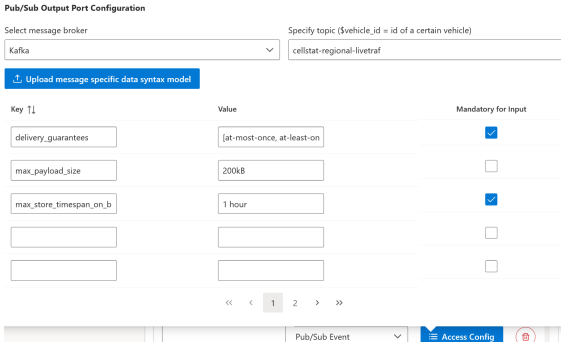
Abbildung 7.4: Konfiguration eines Softwareservices einer Datenproduktimplementierung

Seite definieren, ob die Einstellungen für Input Ports anderer Datenprodukte immer gültig sind oder ob sie nach Bedarf für diese Datenprodukte von Datenkonsumenten konfiguriert werden können sollen.

- **Monitor:** Das Formular zur Definition von Monitoren gleicht dem von Services, bis auf dass statt Input- und Output-Ports API-Pfade für den Observation Port des Data Products definiert werden können. Die Definition dieser Pfade ist in Abbildung 7.6b zu sehen. Zu jedem Pfad können auch Key-Value-Paare definiert werden, die jeweils eine für das Monitoring des Datenprodukts relevante Metrik repräsentieren können.
- **Fog Stream Processing Sink:** Abbildung 7.6a zeigt die Konfiguration der Komponente zur Interaktion mit der Fog Stream Processing Engine der Self-Serve Platform (siehe Kapitel 5.3.4), die den dynamischen Bau von Datenpipelines in der Fog ermöglicht. Es kann ausgewählt werden, ob ein interner Service des Datenprodukts als Datenquelle oder ein anderes Datenprodukt genutzt werden soll. Ganz unten wird außerdem ein Topic vorgegeben, das von anderen Services abonniert werden kann, um die Ergebnisse der Verarbeitung zu erhalten. In der Mitte besteht die Möglichkeit, die Verarbeitungsabfrage für die Processing Engine einzufügen. Die Gestalt dieser Abfrage hängt von der konkreten Implementierung des genutzten Streamverarbeitungsframeworks ab. In meinem Prototyp habe ich mit der Verwendung von `XStream` beispielhaft die Streamingplattform von Van Raemdonck et al. [VVE+17] referenziert.
- **Lakehouse Configuration:** Abbildung 7.7 zeigt das Formular zur Konfiguration von Metadaten der Lakehouses. Der Entwickler, der das Datenprodukt plant, muss eine Datei mit Metadaten einfügen, die etwaige Schemas und Pfadstrukturen definieren, die er für das Datenprodukt nutzen möchte.

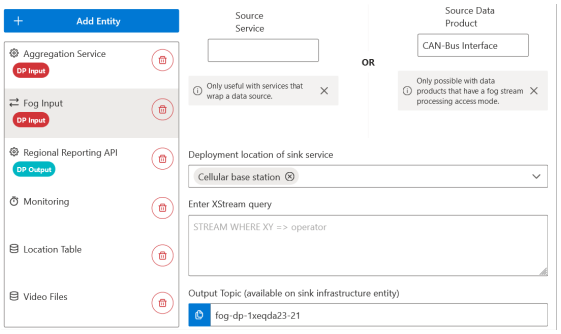


(a) Auswahl multimodaler Zugriffsmöglichkeiten

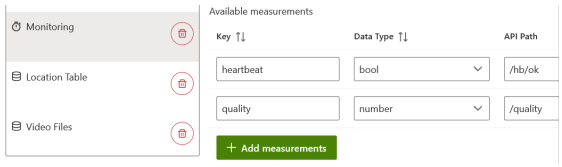


(b) Konfiguration einer Pub/Sub-Zugriffsmöglichkeit

Abbildung 7.5: Konfiguration eines Output Ports



(a) Konfiguration der Streamverarbeitung in der Fog



(b) Konfiguration des Observe Ports eines Monitors

Abbildung 7.6: Formular zur Konfiguration von Fog Stream Processing und eines Monitors

7.1.4 Konfiguration des Data Lineages

Der fünfte Schritt im Planen und Registrieren eines Datenprodukts ist die Beschreibung von Abhängigkeiten zwischen den Services. Basierend auf den Definitionen von Schritt vier wird, gemäß des vorgestellten Konzepts in Kapitel 6.1.2, ein Lineage-Template berechnet[1]. Abbildung 7.8 zeigt ein solches Template. Darauf zu sehen sind Output Ports der verwendeten Datenprodukte (in gelb) und die konfigurierten Input- und Output-Ports der Datenproduktservices (in grün). Die einzelnen `Implementation Components` sind als Kästen dargestellt, wobei sie wiederum von einem äußeren Kasten umgeben sind, der die Infrastrukturtypen angibt, auf denen die Services ausgebracht sind. Im Beispiel von Abbildung 7.8 ist das eine Basistation einer Mobilfunkzelle.

Da auf der Basisstation sowohl ein Lakehouse der lokalen Self-Serve Platform wie auch ein Message Broker (Kafka) zur Verfügung steht und von den `Implementation Components` zuvor zur Verwendung konfiguriert wurde, werden diese beiden Komponenten zusätzlich angezeigt. Kafka wurde dabei zuvor als `Access Endpoint` des Output Ports von Service `Regional Reporting API` definiert. Außerdem wurden für das Lakehouse zwei `Lakehouse Configuration`-Komponenten angelegt, die Metadaten zur Speicherung von Geodaten und Videodateien angeben. Darüber hinaus

[1] Für den horizontalen Prototyp ist diese Berechnung nicht implementiert und wird nur angedeutet.

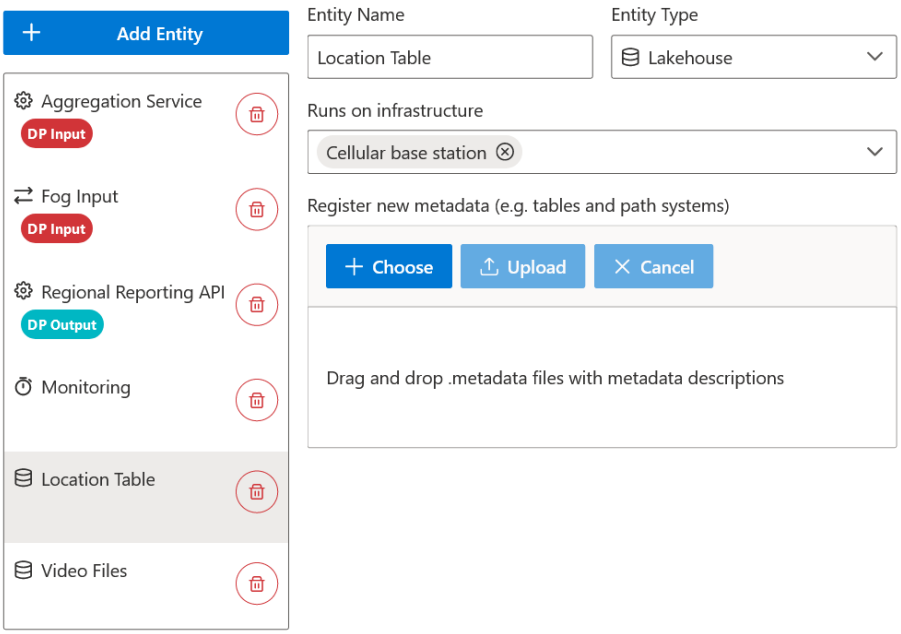


Abbildung 7.7: Konfiguration von Metadaten für ein Lakehouse

werden auch Datenschnittstellen dargestellt, die direkt auf dem Infrastrukturknoten verfügbar sind. Im Beispiel von Abbildung 7.8 sind Daten über die Anzahl an Fahrzeugen, die sich in einer Mobilfunkzelle befinden, sowie aktuelle Netzwerkparameter verfügbar.

Die Aufgabe der Entwickler des Datenprodukts ist es nun, den Datenfluss zwischen den `Implementation Components` einzuzeichnen sowie etwaige Verbindungen zu den Schnittstellen der Data Collection Interfaces. Für jede Verbindung wird dabei im Falle von Transformationen das Datenmodell angegeben, das die neuen Daten repräsentiert. Auf diese Weise wird der geplante Data Lineage des Datenprodukts modelliert.

7.1.5 Konfiguration des Deployments

Zur Ermöglichung der Ausbringung des Datenprodukts kann in diesem Schritt ausgewählt werden, welche `Implementation Component` auf welche Weise auf die Infrastruktur ausgebracht wird. In dem in Abbildung 7.9 dargestellten Beispiel für ein zu diesem Zweck auszufüllendes Formular kann man zwischen Kubernetes und TOSCA-Deployments auswählen. Zusätzlich gibt es OTA-Updates für Infrastrukturknoten wie Fahrzeuge, die keinem Deployment-Cluster angehörig sind. Läuft die `Implementation Component` auf solchen Knoten, gibt es keine alternative Update-Auswahlmöglichkeit.

7.1.6 Angaben zur Datenqualität

Zu einem Datenprodukt kann angegeben werden, welche Qualitätseigenschaften die Daten versprechen zu erfüllen (SLA). Abbildung 7.10 zeigt die Oberfläche hierfür. Neben statischen Angaben ist es auch möglich, einzelne SLOs mit einer konkreten Monitoring-Metrik zu assoziieren. Sie

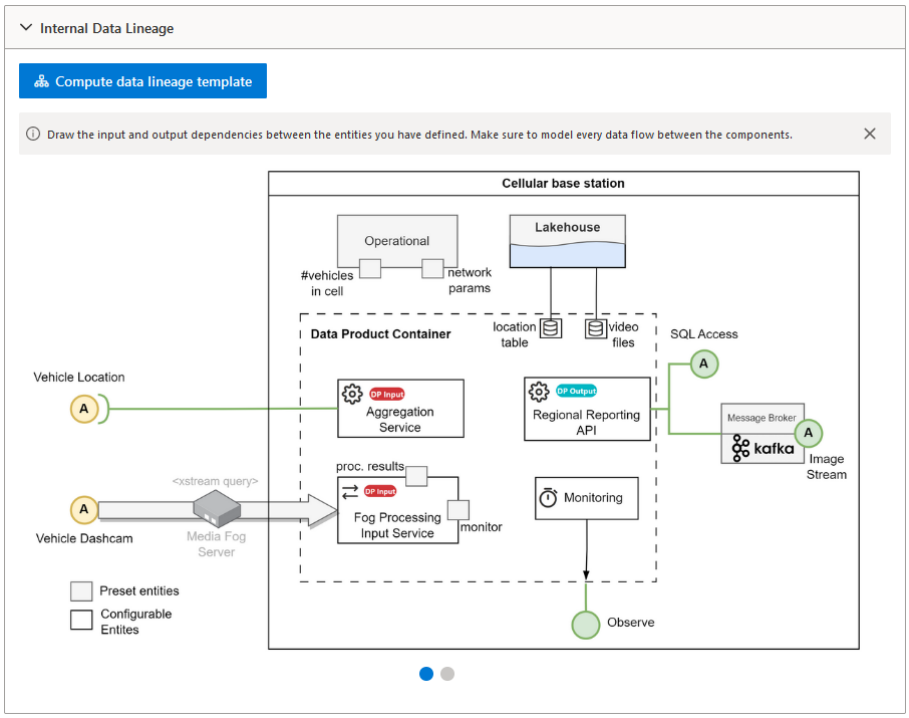


Abbildung 7.8: Beispiel eines Templates zum Einzeichnen von Datenabhängigkeiten zwischen Komponenten eines Datenprodukts

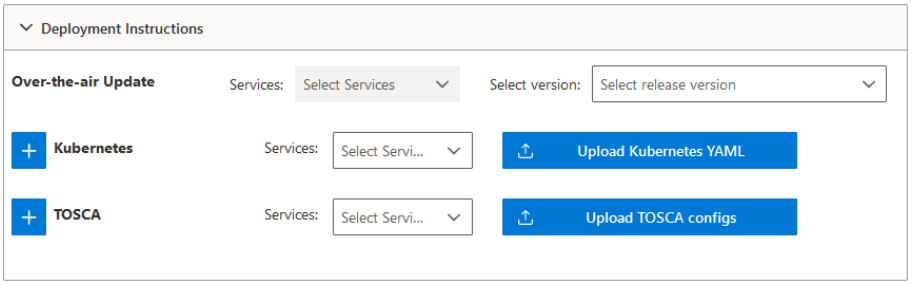


Abbildung 7.9: Auswahl von Technologien zur Ausbringung des Datenprodukts

können dann über die `Observation Endpoints` der Monitore überwacht werden. Auf diese Weise ist es den Nutzern von Datenprodukten möglich nachzuvollziehen, ob im laufenden Betrieb die Qualitätsversprechungen auch eingehalten werden.

7.1.7 Einstellung lokaler Governance-Richtlinien

Der letzte Schritt zum Anlegen von Datenprodukten ist die Möglichkeit, lokale Richtlinien zu definieren, die für das Produkt gelten sollen. In Abbildung 7.11 sind eine Reihe solcher Einstellungsmöglichkeiten zu sehen, die für den Prototyp berücksichtigt wurden:

- **Access:** Auswahl, beziehungsweise Ausschluss, von Domänen, die dieses Datenprodukt nicht verwenden dürfen.

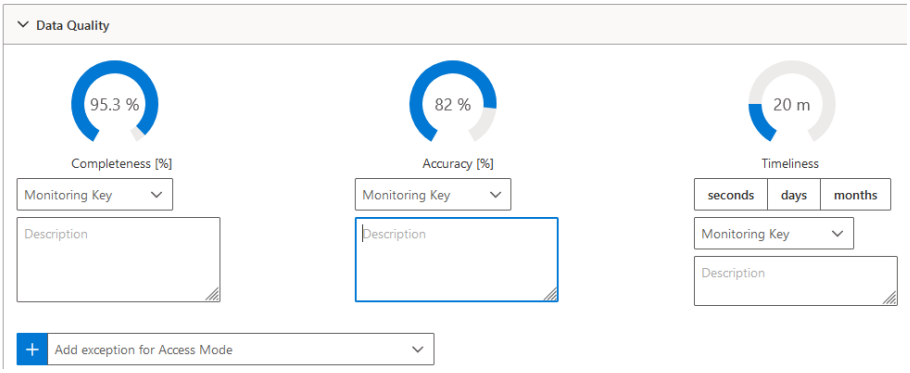


Abbildung 7.10: Angaben zur Datenqualität des Datenprodukts

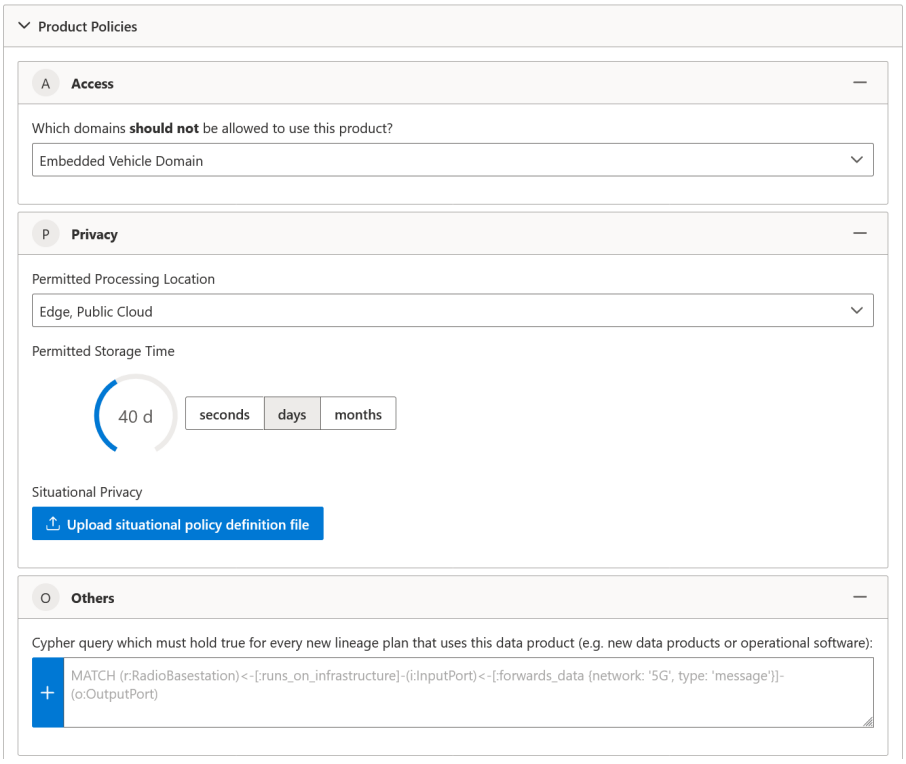


Abbildung 7.11: Konfiguration lokaler Governance-Richtlinien für das Datenprodukt

- **Privacy:** Auswahl, auf welcher Infrastruktur die Datenprodukte verwendet werden dürfen und wie lange die erhaltenen Daten des Datenprodukts im Anschluss im eigenen System gespeichert werden dürfen. Außerdem können hier Definitionen situationaler Datenschutzrichtlinien hochgeladen werden, wie sie von Li et al. [LHSM22] vorgeschlagen werden (siehe Kapitel 5.4).
- **Lineage-bezogene Richtlinien:** Cypher-Abfragen, die auf dem Data-Lineage-Graphen eines neuen Datenprodukts ausgeführt werden sollen, wenn es das aktuell erstellte als Abhängigkeit verwendet. Mehr zu diesem System wird in Abschnitt 7.2 beschrieben.

7.2 Umsetzung der lineage-basierten Richtlinienüberprüfung

Zur Umsetzung des zweiten Implementierungskonzepts der automatisierten Richtlinienüberprüfung via Data Lineage (siehe Kapitel 6.2) habe ich Neo4J[2] als Graphdatenbank verwendet. Sie speichert sowohl den gesamten Data Lineage des Systems ab als auch in einer zweiten Datenbankinstanz temporäre Graphen, auf denen die Richtlinien ausgewertet werden. Als Beispiel, wie der Data-Lineage-Graph in Neo4J repräsentiert werden kann, habe ich den konzeptionell dargestellten Lineage-Graph von Abbildung 7.8 vom beschriebenen Szenario in Kapitel 6.2.2 für das Datenmodell in Neo4J formalisiert. Eine Visualisierung dieses Graphen ist in Abbildung 7.12 zu sehen.

In der Abbildung bilden für das Szenario die rot markierten Knoten die `Embedded`-Domäne ab, die blauen Knoten die einzelnen Services der `Traffic Monitoring`-Domäne. Aus Darstellungsgründen wurde die dritte Domäne in der Cloud nicht eingezeichnet. Mittels Output- und Input-Knoten werden die Datenschemas den einzelnen Datenflussverbindungen (wie Netzwerkverbindungen, gekennzeichnet mit dem Label `forwards_data`) zugeordnet. Diese Knoten verweisen wiederum auf eindeutig identifizierbare Datenpunkte im Lineage-Graphen (`DataPoint`). Die grau eingezeichnete `Infrastructure Proxy` ist die Komponente auf allen Mobilfunk-Basisstationen, die die Daten an das weitere Netzwerk über Ethernet weiterleitet (zum Beispiel zur Cloud). Fortgesetzt an diesem Knoten würde, wenn der Graph vollständig abgebildet wäre, die `Driver Live Reporting`-Domäne eingezeichnet werden. Ganz oben befinden sich zwei Knoten, die jeweils eine Infrastrukturkomponente repräsentieren. Über sie lässt sich herausfinden, welche Knoten des Lineage-Graphen auf welchem Typ von Infrastruktur verortet sind.

Neo4J unterstützt die deklarative Abfragesprache Cypher. Die Operation `Merge` dieser Sprache eignet sich, um den neuen Lineage-Graph, der durch die Anlage neuer Datenprodukte entsteht, mit dem persistenten Lineage des Gesamtsystems zu verbinden. Anschließend können Richtlinien, die als Cypher-Abfragen in den Policy Stores gespeichert werden, auf diesem temporären Graphen ausgeführt werden. Listing 7.1 zeigt eine Cypher-Abfrage, die als Richtlinie für die Vermeidung von 5G-Nachrichtenduplikaten auf Mobilfunkbasisstationen verwendet werden kann (siehe Szenariobeschreibung von Kapitel 6.2.2).

In der Abfrage werden zunächst alle Pfade im Graph identifiziert (`MATCH`), die von einem beliebigen `OutputPort`-Knoten über eine 5G-Netzwerkverbindung zu einer Mobilfunkbasistation führen (`path1`). Für die Knoten in diesem Pfad, die angeben, direkt auf einer solchen Infrastruktur zu laufen (Variable `i`), wird nachverfolgt, welche `DataPoints` sie über diese Netzwerkverbindung konsumieren (`path2`). Anschließend werden im `WITH`-Statement Aggregationen durchgeführt, die das Vorkommen gleicher Datenpunkte in den gefundenen Pfaden zählen und die Namen der betroffenen Knoten sammeln. Basierend auf diesen Aggregationsergebnissen werden als Nächstes durch das `WHERE`-Statement die Pfade herausgefiltert, bei denen mindestens zwei `DataPoints` zu dem gleichen Knoten auf dem Infrastrukturknoten führen. Existieren solche Pfade, enthalten diese die zu vermeidenden Datenduplikate. In diesem Fall wird mittels `RETURN` eine Fehlermeldung ausgegeben.

Führt man diese Abfrage auf dem in Abbildung 7.12 gezeigten Graph aus, so erhält man eine Ausgabe, die in Tabelle 7.1 zu sehen ist. In diesem Fall wird eine Richtlinienverletzung erkannt und eine Fehlermeldung bereitgestellt.

[2] https://neo4j.com/

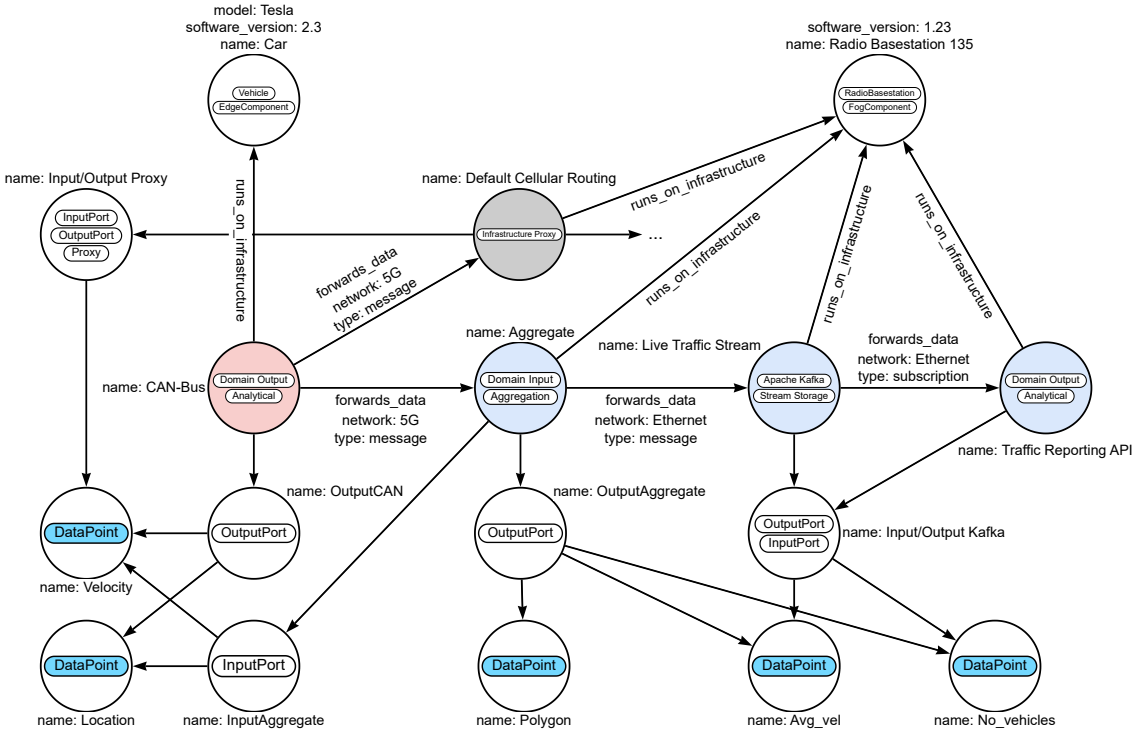


Abbildung 7.12: Beispiel, wie ein Lineage-Graph für das Szenario in Neo4J aussehen kann.

Listing 7.1 Cypher-Abfrage zur Erkennung von doppelt empfangenen Daten auf Mobilfunkbasisstationen im Data-Lineage-Graphen von Abbildung 7.12

```
MATCH
    path1=(r:RadioBasestation)<-[:runs_on_infrastructure]-(i)<-[relation:forwards_data {
    network: "5G", type: "message"}]-(b)--(out:OutputPort),
    path2=(i)--(inp:InputPort)--(d:DataPoint)
WITH  r as Basestation, count(d) as no_duplicates, d as datapoint,
      Collect(i.name) as DomainPort, Collect(b.name) as DomainOutput
WHERE no_duplicates >= 2
RETURN "Global policy violation: The DataPoint " + datapoint.name + " is routed duplicated via
       5G to radio basestation of type " + Basestation.name + "." as Policy_Violation, [
       datapoint.name, Basestation.name, DomainPort, DomainOutput] as Conflicted_Entities
```

Tabelle 7.1: Ausgabe der Cypher-Query von Listing 7.1, ausgeführt auf dem Graphen von Abbildung 7.12

Policy_Violation	Conflicted_Entities
"Global policy violation: The DataPoint Velocity is routed duplicated via 5G to radio basestation of type Radio Basestation 135."	["Velocity", "Radio Basestation 135", ["Aggregate", "Default Cellular Routing"], ["CAN-Bus", "CAN-Bus"]]

8 Evaluation des Konzepts

In gleicher Weise wie ich in Kapitel 4 die Architekturen aus der wissenschaftlichen Literatur auf die Erfüllung der Kriterien des Anforderungskatalogs untersucht habe, diskutiere ich nun die Erfüllung dieser durch meine in Kapitel 5 bis 7 vorgestellten Konzepte und Prototypen. Eine Übersicht über meine Einschätzung der Anforderungsüberdeckung ist in Tabelle 8.1 zu sehen.

A1: Big-Data-Fähigkeit

- **A1.1, Volume:** Durch die Möglichkeit für Domänenteams dezentrale Datenarchitekturen in der Fog und Cloud zu bauen, können Daten im ganzen Connected-Cars-System geografisch verteilt gespeichert und verarbeitet werden. Lokale und bei Bedarf ausbringbare Datenprodukte auf die Fog-Infrastruktur, inklusive der Fahrzeuge selbst, sparen Rechen-, Speicher- und Netzwerkressourcen. Zusätzlich kommen in der Cloud Skalierungsmechanismen wie Load Balancer zum Einsatz sowie horizontal skalierbare Datenplattformen und Verarbeitungstechnologien. Letztere werden insbesondere durch das Lakehouse unterstützt.
- **A1.2, Velocity:** Stream- und Batchverarbeitung von Daten unterstützen die integrierten Tools des Lakehouses. Zusätzlich zu Cluster-Verarbeitungsframeworks in der Cloud bietet der Data Mesh auch Technologien für die Streamverarbeitung in der Fog an.
- **A1.3, Variety:** Analytische Daten jeglicher Form können in den Lakehouses gespeichert werden. Für operationale Daten gibt es heterogene Datenbankangebote in der Fog und Cloud (zum Beispiel Objektspeicher für Binärdateien, RDBMS für relationale Daten).

A2: Berücksichtigung von Data Governance

- **A2.1, Disoverability:** Metadaten zu allen im System verfügbaren und operationalen Daten werden in einem zentralen Datenkatalog der Self-Serve Platform abgelegt. Ein Data Marketplace kann von Entwicklern (aber auch dritten Parteien) genutzt werden, um Datenprodukte im Data Mesh zu entdecken und zu nutzen. Zusätzlich befinden sich auf jedem Fog-Knoten Metadatenspeicher für vor Ort verfügbare Daten. Diese bieten lokalen Diensten in der Fog eine Möglichkeit des Datendiscoverys.
- **A2.2, Accountability:** Das domänengetriebene Design des Data Mesh sorgt für klare Zuständigkeiten der verwalteten Daten. Das Erstellen und Pflegen von Datenprodukten liegt in dem Verantwortungsbereich der Domäne, aus der die Daten stammen. Im Data Catalog und im Data Marketplace ist klar vermerkt, von wem das Datenprodukt entwickelt wird.
- **A2.3, Quality:** Datenpipelines zur Qualitätserhöhung können mittels zusammengesetzten Datenprodukten über mehrere Infrastrukturknoten hinweg entwickelt werden. Unterstützt werden die Entwickler dabei von den Streamverarbeitungstechnologien. Zusätzlich vereinbaren Datenprodukte klare SLAs bezüglich versprochener Datenqualität. Anhand des Prototypen wurde demonstriert, wie der Observation Port eines Datenprodukts konfiguriert werden kann, um einzelne SLOs zu überwachen. Neben diesen Maßnahmen zur Qualitätssicherung sieht

der Data Marketplace eine Feedback Loop vor. Qualitativ minderwertige Datenprodukte erhalten niedrigere Suchränge bei der Discovery. Des Weiteren wird durch den dynamischen Lebenszyklus von Datenprodukten verhindert, dass wertlose Daten im System gespeichert sind. Die geografische Lokalität der Datenprodukte ermöglicht darüber hinaus auch, zeitliche und geografische Aktualität der Daten zu gewährleisten. Lokale Datenkonsumenten können direkt auf Daten ihrer Umgebung zugreifen. Ein Beispiel hierfür könnte beispielsweise die Verkehrsbehörde einer Stadt sein, die über regionale Netzwerke auf die für sie relevanten Verkehrsdaten zugreifen kann.

- **A2.4, Security und Privacy:** Privacy wird im vorgestellten Data Mesh durch die Integration des Privacy-Frameworks von Li et al. [LHSM22] als Management Agent in Produkt-Containern berücksichtigt (siehe Kapitel 5.4). Security kann ebenso via Agents, aber auch mittels etablierten Standards durch die Self-Serve Platform berücksichtigt werden. Die Security-Komponente, die Teil dieser Plattform ist, soll beispielsweise auch für die Authentifizierungsmechanismen aller Plattformangebote zuständig sein. Genaue Details zur Umsetzung dieser Maßnahmen übersteigen jedoch den Rahmen dieser Arbeit.

A3: Verteiltes Infrastruktur- und Datenmanagement

- **A3.1, Infrastrukturmanagement** In der vorgestellten Architektur für die Self-Serve Platform ist die Komponente der Infrastructure Registry für die Verwaltung aller Metadaten von Infrastrukturgeräten zuständig. Darüber hinaus ist die Komponente Fog Placement & Orchestration damit beauftragt, Updates auf die verwaltete Infrastruktur auszuspielen. Wie eine Konfiguration dieser Updates bei der Registrierung von Datenprodukten erfolgen kann, wird in Kapitel 7.1 behandelt.
- **A3.2, Datenmanagement:** Verteiltes Datenmanagement wird durch das Zusammenspiel der Infrastructure Registry mit dem Datenkatalog ermöglicht. Zu jedem Typ von Infrastruktur werden entsprechende Metadaten zur Verfügbarkeit von Daten auf den Knoten gespeichert. Ein Beispiel, wie Fog-Knoten mit bestimmten Datenprodukten auffindbar sind, wird in Abbildung 5.6 gezeigt.

A4: Interoperabilität und Mandantenfähigkeit

- **A4.1/4.2, Interne/Externe Interoperabilität:** Der Datenmarktplatz erlaubt sowohl das domänenübergreifende Teilen von Datenprodukten im Data Mesh als auch für externe Parteien.
- **A4.3, Mandantenfähigkeit:** Der Datenmarktplatz kann Ressourcen der Infrastruktur sowohl an Domänenteams als auch an dritte Parteien vermieten. Die Fog-Knoten implementieren dazu das Konzept des MEC, welches durch die Architektur der lokalen Self-Serve Platform der Fog unterstützt wird (siehe Kapitel 5.3.3).

Tabelle 8.1: Eigene Einschätzung des Konzepts hinsichtlich der Erfüllung der Kriterien des in Kapitel 3.3 definierten Anforderungskatalogs.

A1			A2				A3		A4		
A1.1	**A1.2**	**A1.3**	**A2.1**	**A2.2**	**A2.3**	**A2.4**	**A3.1**	**A3.2**	**A4.1**	**A4.2**	**A4.3**
✓	✓	✓	✓	✓	✓	●	✓	✓	✓	✓	✓

9 Zusammenfassung, Fazit und Ausblick

Zusammenfassung und Fazit

Diese Arbeit verfolgte das Ziel, eine Datenarchitektur zu finden, die in der Lage ist, die Anforderungen der Connected-Cars-Domäne zu erfüllen. Mit einer solchen Datenarchitektur ist es möglich, die Anwendungsfälle zu realisieren, die für Connected Cars in der Literatur formuliert werden. Dieses definierte Ziel konnte bis auf eine Einschränkung erreicht werden. Dazu wurde ein Vorschlag einer dezentralen Datenarchitektur erarbeitet, die die Prinzipien des Data Mesh auf die Connected-Cars-Domäne anwendet. Anders als bestehende Vorschläge für Datenarchitekturen, die sich mehrheitlich auf die Erfüllung der Big-Data-Eigenschaften fokussieren, werden mit diesem Konzept verstärkt Schwerpunkte auf die Berücksichtigung von Data Governance und der Interoperabilität der Systeme, beziehungsweise dem Datenaustausch gelegt. Neben Big-Data-Fähigkeit und der Verwaltung der geografisch verteilten Daten und Infrastruktur sind dies die vier Kernanforderungen, die in dieser Arbeit für eine Datenarchitektur der Connected-Cars-Domäne identifiziert wurden. Arbeitet man in die Architektur der Self-Serve Platform des Data Mesh zusätzlich Komponenten ein, die die Verwaltung der geografisch verteilten IoV-Infrastruktur übernehmen, so erhält man ein Gesamtkonzept, das in der Lage ist, alle vier Anforderungen zu erfüllen. Lediglich eine Teilanforderung von Data Governance, Security, wurde in dieser Arbeit nicht ausreichend behandelt, um sie als erfüllt anerkennen zu können.

Die größte Besonderheit des vorgestellten Konzepts zur Umsetzung der Data-Mesh-Prinzipien für die Connected-Cars-Domäne ist die Möglichkeit des Einbezugs von Fog Computing. Dadurch erhalten die vom Data Mesh verwalteten dezentralen Datenarchitekturen Vorteile der Ressourcenschonung und Übertragungslatenz, die mit rein zentralisierten Ansätzen wie ausschließlichem Cloud Computing nicht möglich wären. Realisiert wird die geografische Verteilung der Datenarchitektur durch eine Zweiteilung der Self-Serve Platform des Data Mesh. Neben einer zentralen Self-Serve Platform in der Cloud, werden den Entwicklern lokale Self-Serve Platforms auf Fog-Knoten angeboten. In dieser Arbeit wurden für diese Plattformen Architekturen ausgearbeitet. Ein wichtiger Bestandteil dieser ist ein Data Lakehouse zur Verwaltung von analytischen Daten. Im Vergleich zu Data Warehouses und Data Lakes bietet es die umfangreichsten Funktionalitäten und damit auch die größte Flexibilität für die diversen Anwendungsfälle von Connected-Cars-Systemen.

Ob die vorgeschlagene dezentrale Datenarchitektur für die Connected-Cars-Domäne tatsächlich in die Praxis umgesetzt werden kann, hängt davon ab, ob den OEMs der Mehraufwand wert ist, die Architektur der Self-Serve Platforms umzusetzen. Neben diesem initialen Mehraufwand ist ein weiterer potenzieller Nachtteil des Data Mesh die Gefahr der Dopplung von Entwicklungsaufwand in den voneinander unabhängigen Domänen. Zur Behebung dieses Problems wird in dieser Arbeit ein Implementierungskonzept vorgeschlagen und prototypisch implementiert. Es ist in der Lage, architekturbedingte Datenduplikate zu erkennen und für Datenprodukte zu verhindern. Ein weiteres Implementierungskonzept mit Prototyp zeigt, wie das konzeptionelle Anlegen von geografisch

verteilten Datenprodukten umgesetzt werden kann und welche Konfiguration dafür notwendig sind. Diese beiden Implementierungskonzepte haben einen Nutzen, der auch abseits der Connected-Cars-Domäne Anwendung finden kann.

Ausblick

Aufgrund des großen Konzeptumfangs, der bei der Anwendung der Data-Mesh-Prinzipien entsteht, konnten viele Teilaspekte der vorgestellten Datenarchitektur in dieser Arbeit nicht bis ins Detail entworfen werden. Dadurch ergeben sich viele Ansatzpunkte für fortführende Arbeiten. Zunächst sind hier die einzelnen Komponenten der Self-Serve Platform zu nennen. Sowohl für die Teilkomponenten der lokalen Plattformen in der Fog als auch für die der Cloud können Detailentwürfe geliefert werden. Für den Data Catalog und die Infrastructure Registry werden konkrete Datenmodelle und Ontologien benötigt, die in der Lage sind, alle relevanten Metadaten und Daten zur Infrastrukturverwaltung zu beschreiben. Die Data Product Management Engine kann in Teilkomponenten unterteilt werden, die detaillierter die Zuständigkeitsbereiche für die verschiedenen zu erfüllenden Funktionalitäten festlegen. Unter anderem kann genauer ausgearbeitet werden, wie genau die Data Product Management Engine in Kooperation mit den Infrastrukturkomponenten die verteilten und zusammengesetzten Datenprodukte ausbringt und ihren Lebenszyklus verwaltet. So muss es möglich sein, mehrere Deployment-Methoden (OTA-Updates und Clusterorchestrierungstechnologien) zu kombinieren, um verteilte Datenprodukte zu realisieren. In diesem Zusammenhang muss ein Implementierungskonzept für die Product Container festgelegt werden, sodass diese geeignete standardisierte Schnittstellen anbieten, um die geografisch verteilten Datenprodukte zu ermöglichen sowie unterstützende Funktionalitäten speziell für die Connected-Cars-Domäne bereitzustellen. Zur Gestaltung der Product Container gehören auch Konzepte für Management Agents, von denen in dieser Arbeit nur ein einzelnes für die Erzwingung von Datenschutzrichtlinien konkret beschrieben wurde. Schließlich muss auch die Komponente des Data Marketplace vor allem hinsichtlich seiner Funktionalitäten der Produktnutzungskonfiguration und der Infrastrukturvergabe ausgestaltet werden.

Auch die zwei in dieser Arbeit vorgestellten Implementierungskonzepte lassen Anknüpfungspunkte für fortführende Arbeiten offen. Das in Kapitel 6.1.2 beschriebene System zur Erfassung des Data Lineage kann genauer ausgearbeitet werden, indem ein Data-Lineage-Modell für die Connected-Cars-Domäne entworfen wird. In diesem Zusammenhang kann auch die Berechnung der Lineage-Templates konkret ausgearbeitet werden. Bei dem Metamodell zur Anlegung von neuen Datenprodukten, beziehungsweise dem zugehörigen Prototypen (siehe Kapitel 7.1), blieben ebenfalls Details offen. Dazu gehören Metadatenmodelle wie Schemas für Lakehouse-Konfigurationen und logische und physische Schemas für die Definition der Output-Port-Endpunkte.

Zuletzt hat die Evaluation des eigenen Konzepts in Kapitel 8 gezeigt, dass der Themenkomplex Security im vorgestellten Konzept zu großen Teilen offen blieb. Zukünftige Arbeiten können sich mit dem Zusammentragen bestehender Sicherheitskonzepte für die Connected-Cars-Domäne beschäftigen und sie in das vorgeschlagene Data-Mesh-Konzept integrieren. Management Agents, Product-Container-Implementierungen und die zentrale Sicherheitskomponente der Self-Serve Platform lassen sich zu diesem Zweck ausgestalten.

Literaturverzeichnis

[5GP15] 5GPPP White Paper. *5G Automotive Vision*. 20. Okt. 2015. URL: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Automotive-Vertical-Sectors.pdf (zitiert auf S. 21–24).

[AGP17] S. Amini, I. Gerostathopoulos, C. Prehofer. „Big data analytics architecture for real-time traffic control“. In: *2017 5th IEEE international conference on models and technologies for intelligent transportation systems (MT-ITS)*. IEEE. 2017, S. 710–715 (zitiert auf S. 37, 38, 41).

[AGXZ21] M. Armbrust, A. Ghodsi, R. Xin, M. Zaharia. „Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics“. In: *Proceedings of CIDR*. Bd. 8. 2021 (zitiert auf S. 44, 45).

[BBFM23] E. Bertino, S. Bhattacharya, E. Ferrari, D. Milojicic. „Trustworthy AI and Data Lineage“. In: *IEEE Internet Computing* 27.6 (2023), S. 5–6 (zitiert auf S. 63).

[BBM21] K. Benaissa, S. Bitam, A. Mellouk. „On-Board Data Management Layer: Connected Vehicle as Data Platform“. In: *Electronics* 10.15 (2021), S. 1810 (zitiert auf S. 35, 38, 40, 41).

[BJH17] M. Bosler, C. Jud, G. Herzwurm. „Platforms and Ecosystems for Connected Car Services.“ In: *IWSECO*. 2017, S. 16–27 (zitiert auf S. 22, 23).

[Bur18] B. Burns. *Designing distributed systems: patterns and paradigms for scalable, reliable services*. O'Reilly Media, Inc., 2018 (zitiert auf S. 54).

[CAM+16] N. Cárdenas-Benítez, R. Aquino-Santos, P. Magaña-Espinoza, J. Aguilar-Velazco, A. Edwards-Block, A. Medina Cass. „Traffic congestion detection system through connected vehicles and big data“. In: *Sensors* 16.5 (2016), S. 599 (zitiert auf S. 36, 38).

[CEP+21] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, M. Slot. „Citus: Distributed postgresql for data-intensive applications“. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, S. 2490–2502 (zitiert auf S. 39).

[CM16] R. Coppola, M. Morisio. „Connected car: technologies, issues, future trends“. In: *ACM Computing Surveys (CSUR)* 49.3 (2016), S. 1–36 (zitiert auf S. 15, 17, 22, 23).

[CV97] M. Carr, J. Verner. „Prototyping and software development approaches“. In: *Department of Information Systems, City University of Hong Kong, Hong Kong* (1997), S. 319–338 (zitiert auf S. 77).

[DB18] T. S. Darwish, K. A. Bakar. „Fog based intelligent transportation big data analytics in the internet of vehicles environment: motivations, architecture, challenges, and critical issues“. In: *IEEE Access* 6 (2018), S. 15679–15701 (zitiert auf S. 27, 28, 34, 35, 38, 41).

[DDM14] Y. Demchenko, C. De Laat, P. Membrey. „Defining architecture components of the Big Data Ecosystem“. In: *2014 International conference on collaboration technologies and systems (CTS)*. IEEE. 2014, S. 104–112 (zitiert auf S. 25).

[Deh22] Z. Dehghani. *Data Mesh: Delivering Data-Driven Value at Scale*. Hrsg. von O'Reilly Media, Inc. 8. März 2022 (zitiert auf S. 21, 22, 46, 47, 51, 53, 54, 56, 57, 64, 65, 78).

[DGDM13] Y. Demchenko, P. Grosso, C. De Laat, P. Membrey. „Addressing big data issues in scientific data infrastructure“. In: *2013 International conference on collaboration technologies and systems (CTS)*. IEEE. 2013, S. 48–55 (zitiert auf S. 25).

[DHBD17] S. K. Datta, J. Haerri, C. Bonnet, R. F. Da Costa. „Vehicles as connected resources: Opportunities and challenges for the future“. In: *IEEE Vehicular Technology Magazine* 12.2 (2017), S. 26–35 (zitiert auf S. 35, 38–42).

[DHM23] S. Driessen, W.-J. v. den Heuvel, G. Monsieur. „ProMoTe: A Data Product Model Template for Data Meshes“. In: *International Conference on Conceptual Modeling*. Springer. 2023, S. 125–142 (zitiert auf S. 65, 67, 68).

[DSP+17] A. Daniel, K. Subburathinam, A. Paul, N. Rajkumar, S. Rho. „Big autonomous vehicular data classifications: Towards procuring intelligence in ITS“. In: *Vehicular Communications* 9 (2017), S. 306–312 (zitiert auf S. 32, 38, 40).

[DTD19] M. D. Donno, K. Tange, N. Dragoni. „Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog“. In: *IEEE Access* 7 (2019), S. 150936–150948. URL: https://api.semanticscholar.org/CorpusID:204970757 (zitiert auf S. 18, 19).

[EGH+23] R. Eichler, C. Gröger, E. Hoos, C. Stach, H. Schwarz, B. Mitschang. „Introducing the enterprise data marketplace: a platform for democratizing company data“. In: *Journal of Big Data* 10.1 (2023), S. 173 (zitiert auf S. 64).

[EGL+21] E. Eryurek, U. Gilad, V. Lakshmanan, A. Kibunguchy-Grant, J. Ashdown. *Data Governance: The Definitive Guide*. O'Reilly Media, Inc., 2021 (zitiert auf S. 27).

[Fan15] H. Fang. „Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem“. In: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE. 2015, S. 820–824 (zitiert auf S. 43, 44).

[FDA+18] M. Fallgren, M. Dillinger, J. Alonso-Zarate, M. Boban, T. Abbas, K. Manolakis, T. Mahmoodi, T. Svensson, A. Laya, R. Vilalta. „Fifth-generation technologies for the connected car: Capable systems for vehicle-to-anything communications“. In: *IEEE vehicular technology magazine* 13.3 (2018), S. 28–38 (zitiert auf S. 17).

[FHS+23] A. Fieschi, P. Hirmer, R. Sturm, M. Eisele, B. Mitschang. „Anonymization Use Cases for Data Transfer in the Automotive Domain“. In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2023, S. 98–103 (zitiert auf S. 23, 24, 27).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, 2014 (zitiert auf S. 32).

[Gar] Gartner Inc. *Big Data defintion*. URL: http://www.gartner.com/it-glossary/big-data/ (zitiert auf S. 25).

[GBC22] A. Guerna, S. Bitam, C. T. Calafate. „Roadside unit deployment in internet of vehicles systems: A survey“. In: *Sensors* 22.9 (2022), S. 3190 (zitiert auf S. 18).

[GC17] C. Gerloff, C. Cleophas. „Excavating the treasure of IoT data: An architecture to empower rapid data analytics for predictive maintenance of connected vehicles“. In: (2017) (zitiert auf S. 37, 38, 40).

[GDO+17] L. Guo, M. Dong, K. Ota, Q. Li, T. Ye, J. Wu, J. Li. „A secure mechanism for big data collection in large scale internet of vehicle“. In: *IEEE Internet of Things Journal* 4.2 (2017), S. 601–610 (zitiert auf S. 27).

[GGH+20] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, B. Mitschang. „A zone reference model for enterprise-grade data lake management“. In: *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE. 2020, S. 57–66 (zitiert auf S. 43, 44).

[GGH+21] C. Giebler, C. Gröger, E. Hoos, R. Eichler, H. Schwarz, B. Mitschang. „The data lake architecture framework: a foundation for building a comprehensive data lake architecture“. In: *Conference for Database Systems for Business, Technology and Web (BTW)*. Bd. 70469. 2021 (zitiert auf S. 44).

[GKD+23] A. Goedegebuure, I. Kumara, S. Driessen, D. Di Nucci, G. Monsieur, W.-j. v. d. Heuvel, D. A. Tamburri. „Data mesh: a systematic gray literature review“. In: *arXiv preprint arXiv:2304.01062* (2023) (zitiert auf S. 47, 48, 53–57, 65, 73).

[Gre15] S. Greengard. „Automotive systems get smarter“. In: *Communications of the ACM* 58.10 (2015), S. 18–20 (zitiert auf S. 15).

[GSM14] C. Gröger, H. Schwarz, B. Mitschang. „The deep data warehouse: link-based integration and enrichment of warehouse data and unstructured content“. In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference*. IEEE. 2014, S. 210–217 (zitiert auf S. 43).

[GSS+18] F. Giust, V. Sciancalepore, D. Sabella, M. C. Filippou, S. Mangiante, W. Featherstone, D. Munaretto. „Multi-access edge computing: The driver behind the wheel of 5G-connected cars“. In: *IEEE Communications Standards Magazine* 2.3 (2018), S. 66–73 (zitiert auf S. 22, 59).

[HBZ+06] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, S. Madden. „Cartel: a distributed mobile sensor computing system“. In: *Proceedings of the 4th international conference on Embedded networked sensor systems*. 2006, S. 125–138 (zitiert auf S. 34, 38–40).

[HCF+15] T. Häberle, L. Charissis, C. Fehling, J. Nahm, F. Leymann. „The connected car in the cloud: a platform for prototyping telematics services“. In: *IEEE Software* 32.6 (2015), S. 11–17 (zitiert auf S. 32, 38, 39).

[HMD17] A. Haroun, A. Mostefaoui, F. Dessables. „A big data architecture for automotive applications: PSA group deployment experience“. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, S. 921–928 (zitiert auf S. 32).

[HMKA19] D. Hetzer, M. Muehleisen, A. Kousaridas, J. Alonso-Zarate. „5g connected and automated driving: Use cases and technologies in cross-border environments“. In: *2019 European conference on networks and communications (EuCNC)*. IEEE. 2019, S. 78–82 (zitiert auf S. 22).

[IBM24] IBM. *Was ist eine Datenarchitektur?* Online. 2024. URL: https://www.ibm.com/de-de/topics/data-architecture (zitiert auf S. 19).

[IW09] R. Ikeda, J. Widom. „Data lineage: A survey“. In: *Stanford University Publications. http://ilpubs. stanford. edu* 8090.918 (2009), S. 1 (zitiert auf S. 63).

[JZM+20] B. Ji, X. Zhang, S. Mumtaz, C. Han, C. Li, H. Wen, D. Wang. „Survey on the internet of vehicles: Network architectures and applications“. In: *IEEE Communications Standards Magazine* 4.1 (2020), S. 34–41 (zitiert auf S. 17).

[LHSM22] Y. Li, P. Hirmer, C. Stach, B. Mitschang. „Ensuring situation-aware privacy for connected vehicles“. In: *Proceedings of the 12th International Conference on the Internet of Things*. 2022, S. 135–138 (zitiert auf S. 26, 65, 66, 84, 88).

[Liu18] D. Liu. „Big data analytics architecture for internet-of-vehicles based on the spark“. In: *2018 International conference on intelligent transportation, big data & smart city (ICITBS)*. IEEE. 2018, S. 13–16 (zitiert auf S. 33, 38).

[LL13] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. 3. Aufl. dpunkt.verlag, 2013. ISBN: 978-3-86490-092-1 (zitiert auf S. 67).

[LLZ+20] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, W. Shi. „Computing systems for autonomous driving: State of the art and challenges“. In: *IEEE Internet of Things Journal* 8.8 (2020), S. 6469–6486 (zitiert auf S. 15).

[LSB20] K. L. Lim, S. Speidel, T. Bräunl. „A unified telemetry platform for electric vehicles and charging infrastructure“. In: *Connected Vehicles in the Internet of Things: Concepts, Technologies and Frameworks for the IoV* (2020), S. 167–219 (zitiert auf S. 36, 38, 39, 41).

[LWJZ21] K. L. Lim, J. Whitehead, D. Jia, Z. Zheng. „State of data platforms for connected vehicles and infrastructures“. In: *Communications in transportation research* 1 (2021), S. 100013 (zitiert auf S. 15, 18, 22, 23).

[MLKS18] A. C. Marosi, R. Lovas, Á. Kisari, E. Simonyi. „A novel IoT platform for the era of connected cars“. In: *2018 IEEE international conference on future IoT technologies (Future IoT)*. IEEE. 2018, S. 1–11 (zitiert auf S. 23, 31, 38–41, 60).

[MMH+22] A. Mostefaoui, M. A. Merzoug, A. Haroun, A. Nassar, F. Dessables. „Big data architecture for connected vehicles: Feedback and application examples from an automotive group“. In: *Future Generation Computer Systems* 134 (2022), S. 374–387 (zitiert auf S. 32, 38–42, 44, 75).

[NJ17] L. Nkenyereye, J.-W. Jang. „Integration of big data for querying CAN bus data from connected car“. In: *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. 2017, S. 946–950 (zitiert auf S. 37–39).

[Ole23] O. Olesen-Bagneux. *The Enterprise Data Catalog*. O'Reilly Media, Inc., 2023 (zitiert auf S. 62, 63, 71).

[PM20] C. Prehofer, S. Mehmood. „Big data architectures for vehicle data analysis“. In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. 2020, S. 3404–3412 (zitiert auf S. 36, 38, 39).

[RH22] J. Reis, M. Housley. *Fundamentals of Data Engineering*. 1. Aufl. O'Reilly Media, Inc., Juli 2022. ISBN: 978-1-098-10830-4 (zitiert auf S. 16, 19, 22, 23, 26, 42, 43).

[Saa22] M. Saarinen. *A literature review on connected vehicle use cases*. Bachelor's Thesis. University of Oulu: Faculty of Information Technology and Electrical Engineering. 7. Mai 2022 (zitiert auf S. 23).

[SCB16] M. Shojafar, N. Cordeschi, E. Baccarelli. „Energy-efficient adaptive resource management for real-time vehicular cloud services“. In: *IEEE Transactions on Cloud computing* 7.1 (2016), S. 196–209 (zitiert auf S. 35).

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge computing: Vision and challenges“. In: *IEEE internet of things journal* 3.5 (2016), S. 637–646 (zitiert auf S. 18).

[SEA06] R. Santos, A. Edwards, O. Alvarez. „Towards an inter-vehicle communication algorithm“. In: *2006 3rd International Conference on Electrical and Electronics Engineering*. IEEE. 2006, S. 1–4 (zitiert auf S. 36).

[SES17] J. E. Siegel, D. C. Erb, S. E. Sarma. „A survey of the connected vehicle landscape—Architectures, enabling technologies, applications, and development areas“. In: *IEEE Transactions on Intelligent Transportation Systems* 19.8 (2017), S. 2391–2406 (zitiert auf S. 15, 22, 23).

[SL22] R. Y. S. Samidi, A. B. Lesmana. „Implementation of Database Distributed Sharding Horizontal Partition in MySQL. Case Study of Application of Food Serving On Kemkes“. In: *JURNAL SISFOTEK GLOBAL* 12.1 (2022), S. 50–57 (zitiert auf S. 39).

[SPV22] A. Simonetta, M. C. Paoletti, A. Venticinque. „The use of Maximum Completeness to Estimate Bias in AI-based Recommendation Systems“. In: *CEUR Workshop Proceedings*. Bd. 3360. 2022, S. 76–84 (zitiert auf S. 27).

[SS13] J. Soryal, T. Saadawi. „DoS attack detection in Internet-connected vehicles“. In: *2013 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE. 2013, S. 7–13 (zitiert auf S. 27, 41).

[SSM23] J. Schneider, H. Schwarz, B. Mitschang. „Assessing the Lakehouse: Analysis, Requirements and Definition“. In: (2023) (zitiert auf S. 45).

[TK22] M. N. Tahir, M. Katz. „Performance evaluation of IEEE 802.11 p, LTE and 5G in connected vehicles for cooperative awareness“. In: *Engineering Reports* 4.4 (2022), e12467 (zitiert auf S. 17).

[VVE+17] W. Van Raemdonck, T. Van Cutsem, K. S. Esmaili, M. Cortes, P. Dobbelaere, L. Hoste, E. Philips, M. Roelands, L. Trappeniers. „Building connected car applications on top of the world-wide streams platform“. In: *Proceedings of the 11th ACM international conference on distributed and event-based systems*. 2017, S. 315–318 (zitiert auf S. 23, 33, 38–41, 58, 60, 80).

[VZ14] A. Vaisman, E. Zimányi. „Data warehouse systems“. In: *Data-Centric Systems and Applications* (2014) (zitiert auf S. 43, 44).

[WM15] J. Warren, N. Marz. *Big Data: Principles and best practices of scalable realtime data systems*. Simon und Schuster, 2015 (zitiert auf S. 33).

[Wol16] D. Wollschläger. „Preconditions, requirements & prospects of the connected car“. In: *Auto Tech Review* 5.1 (2016), S. 30–35 (zitiert auf S. 25).

[WWG+19] Y. Wang, J. Wang, Y. Ge, B. Yu, C. Li, L. Li. „MEC support for C-V2X system architecture“. In: *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. IEEE. 2019, S. 1375–1379 (zitiert auf S. 22, 28, 29, 59).

[WZH+20] X. Wang, Z. Zhou, P. Han, T. Meng, G. Sun, J. Zhai. „Edge-stream: A stream processing approach for distributed applications on a hierarchical edge-computing system“. In: *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2020, S. 14–27 (zitiert auf S. 60).

[ZWZ+18] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, H. Zhong. „OpenVDAP: An open vehicular data analytics platform for CAVs“. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, S. 1310–1320 (zitiert auf S. 34, 38, 40–42, 58, 59).

[ZXCW20] H. Zhou, W. Xu, J. Chen, W. Wang. „Evolutionary V2X technologies toward the Internet of vehicles: Challenges and opportunities“. In: *Proceedings of the IEEE* 108.2 (2020), S. 308–323 (zitiert auf S. 17).

Alle URLs wurden zuletzt am 13. 05. 2024 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift