HAW HAMBURG

BACHELOR THESIS
Abdullah Al Mohammad

Konzeption und Entwicklung eines generischen Services zur Datenfusion

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Abdullah Al Mohammad

Konzeption und Entwicklung eines generischen Services zur Datenfusion

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 28. Sept. 2023

Abdullah Al Mohammad

Thema der Arbeit

Konzeption und Entwicklung eines generischen Services zur Datenfusion

Stichworte

Datenfusion, Datenintegration, Service

Kurzzusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Konzeption und Entwicklung eines generischen Services zur Datenfusion. Der entwickelte Service soll in der Lage sein, Daten aus verschiedenen Quellen zu integrieren und fusionieren, um neue Erkenntnisse zu gewinnen und die Entscheidungsfindung zu unterstützen. Die Arbeit umfasst die Analyse der Anforderungen an den Service, die Konzeption und Implementierung der Servicearchitektur sowie die Evaluation der Servicefunktionalität. Basierend auf bestimmten Anforderungen und Vorschriften wird eine Lösung zur Erreichung des Ziels ausgewählt. Die Implementierung des Services erfolgt unter Verwendung moderner Technologien und Tools. Die Evaluation des Services erfolgt anhand von Testdaten und -szenarien. Die Arbeit schließt mit einer Zusammenfassung der Ergebnisse, einer Bewertung der Arbeit sowie einem Ausblick auf zukünftige Arbeiten.

Abdullah Al Mohammad

Title of Thesis

Conception and development of a generic service for data fusion

Keywords

Data fusion, Data integration, Service

Abstract

This bachelor thesis deals with the conception and development of a generic service for data fusion. The developed service should be able to integrate and merge data from different sources to gain new insights and support decision-making. The work includes the analysis of the service requirements, the conception and implementation of the service architecture and the evaluation of the service functionality. Based on specific requirements and regulations, a solution is selected to achieve this goal. The service is implemented using modern technologies and tools and is evaluated using test data and scenarios. The work closes with a summary of the results, an evaluation of the work and an outlook on future work.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Die Datenfusion ist ein wichtiger Prozess in vielen Anwendungsbereichen, um aus verschiedenen Datenquellen wertvolle Informationen zu gewinnen. Die Menge an verfügbaren Daten nimmt ständig zu, und es wird immer wichtiger, Daten aus verschiedenen Quellen zu kombinieren und analysieren. Datenfusion spielt in verschiedenen Anwendungsbereichen wie intelligenten Städten (eng. Smart Cities) [12], der Logistik [9] oder der Medizin [3] eine wichtige Rolle. Die Datenfusion der Daten aus verschiedenen Quellen ist jedoch eine komplexe Aufgabe, die spezielle Konzepte und Technologien erfordert.

1.2 Zielsetzung

Im Rahmen des Projektes 'Data Fusion Generator and Synthetic Data Generation for Cities' (DaFne) [2] zielt diese Bachelorarbeit darauf ab, einen generischen Service zur Datenfusion zu konzipieren und entwickeln. Dieser Service soll in der Plattform DaFne eingesetzt werden und ermöglichen, verschiedene Datensätze zu integrieren und fusionieren. Um dieses Ziel zu erreichen, sollte der Service auf Daten aus mehreren Quellen (CSV-Dateien, externe API) zugreifen können. Der Service sollte auch in der Lage sein, die Daten zu verarbeiten und zu einer einzigen Ansicht zusammenzuführen. Der Service soll es beherrschen, Daten von CSV-Datei und einem Wetter-API zu übernehmen und in einer einzigen Ansicht zu kombinieren. Schließlich sollte der Service Ergebnisse in einem leicht verständlichen Format bereitstellen können. Die Ergebnisse sollen in einem tabellarischen Format, wie z.B. CSV-Datei, gespeichert und angezeigt werden können.

1.3 Aufbau der Arbeit

Diese Arbeit besteht aus den folgenden Kapiteln, in denen jeder Aspekt der Arbeit besprochen wird.

- **Einleitung:** beinhaltet die Motivation sowie Zielsetzung und beschreibt, wie diese Arbeit strukturell aufgebaut ist.
- **Datenfusion:** besteht aus einem Überblick über die Grundlagen der Datenfusion. Dieses Kapitel erklärt Begriffe, zeigt Anwendungsgebiete auf und beschreibt, wie verschiedene Datenquellen miteinander kombiniert werden können. Außerdem werden Konfliktlösungsstrategien und verschiedene Fusionstechniken, wie Join- und Union-Ansätze, besprochen.
- **Konzeption:** Dieses Kapitel diskutiert und analysiert die funktionalen Anforderungen, die der Service leisten soll. Zudem wird die Architektur des Services vorgestellt und die verschiedenen Komponenten beschrieben. Die Interaktion zwischen den Komponenten und die Kommunikation mit externen APIs werden erläutert. Basierend auf den ermittelten Anforderungen erfolgt eine Auswahl der geeigneten Algorithmen und einer externen Wetter-API.
- **Umsetzung:** Dieses Kapitel beschreibt die konkrete technische Umsetzung des Services. Hierzu werden die Implementierungsdetails für jede Komponente, die Funktionalität des Services sowie die verwendeten Algorithmen erläutert.
- **Evaluation:** In diesem Kapitel wird die Evaluierung des implementierten Services dargestellt. Integrationstests, Unittests und manuelle Tests werden durchgeführt, um die Funktionalität, die Leistung, die Genauigkeit und die Robustheit des Services zu überprüfen.
- **Fazit und Ausblick:** Hier wird die Arbeit zusammenfassend illustriert und werden zukünftige Optmierungsmöglichkeiten diskutiert, um den Service weiter zu verbessern.

2 Datenfusion

Dieses Kapitel präsentiert die theoretischen Grundlagen der Arbeit. Hierfür wird der Begriff der Datenfusion erläutert und deren Anwendungsgebiete demonstriert. Zudem wird der Prozess der Datenfusion näher betrachtet, sodass der Umgang mit Konflikten und Strategien zur Konfliktbehandlung zum besseren Verständnis beleuchtet werden kann. Ansätze sowie Techniken der Datenfusion werden ebenfalls in diesem Kapitel illustriert.

2.1 Begrifflichkeit

In der Literatur wird der Begriff **Informationsfusion** hauptsächlich verwendet, um die Verknüpfung von Daten verschiedener Sensoren zu beschreiben. Zum Beispiel beschreiben Ruser und León die Informationsfusion als einen Prozess, bei dem Daten von verschiedenen Sensoren oder Quellen kombiniert werden, um ein vertieftes Verständnis von physikalischen Größen, Geschehnissen und Situationen zu erlangen. In einem breiteren Kontext bedeutet Informationsfusion, dass man ähnliche oder verschiedene Arten von Informationen zusammenfügt, um ein Gesamtbild zu erhalten [14].

Datenintegration und **Datenfusion** sind zwei Konzepte, die eng mit dem größeren Begriff **Informationsfusion** zusammenhängen, obwohl sie in der Literatur unterschiedlich verwendet werden. Conrad et al. betrachten Datenintegration als einen grundlegenden Bereich, der als Basis für die Informationsfusion dient [7]. Im Datenintegrationsmodell von Bleiholder und Naumann ist die **Datenfusion** ein Teil des Prozesses der Datenintegration. Sie benennen den Vorgang, bei dem Informationen aus unterschiedlichen Quellen zusammengeführt und eventuell auftretende Konflikte aufgelöst werden, als **Data Merging**, das auch einen Fusionsprozess darstellt [6]. Die Begriffe **Informationsintegration**, **Informationsfusion** und **Datenintegration** werden von Yao et al. synonym und austauschbar verwendet. Sie beschreiben damit die Aufgabe, Informationen im Web zu kombinieren [16]. Da in dieser Arbeit die Daten in Tabellenform

miteinander kombiniert werden und auch Web-Ressourcen eingesetzt werden, wird eine spezifische Art der Datenfusion betrachtet. Aus diesem Grund wird zwischen den oben genannten Begriffen nicht unterschieden, sodass stets von **Datenfusion** gesprochen wird, falls mehrere Datensätze zu einem einzigen Ergebnis zusammengeführt werden.

Die Datenfusion ist ein Teil des Prozesses, bei dem verschiedene Darstellungen derselben realen Entität zusammengefügt und kombiniert werden. Dies kann in einer einzigen Quelle oder über verschiedene Quellen hinweg geschehen. Es gibt viele Beispiele dafür im täglichen Internetgebrauch. Allerdings sind nicht alle Techniken der Datenfusion für jede Situation einsetzbar. Dementsprechend muss die geeignete Methode ausgewählt werden. Es gibt grundsätzlich zwei Arten der Datenfusion: die Fusion homogener und heterogener Daten. Die homogene Datenfusion ist die Zusammenführung von Daten in einer konsistenten Form. Dabei werden mehrere Datensätze zu einem Gesamtbild zusammengeführt. Im Fall der heterogenen Datenfusion werden die Daten aus unterschiedlichen Formen zu einem Gesamtbild zusammengefügt. Google Maps[1] ist ein Beipsiel hierfür. Durch die Abfrage der Web-Anwendung mit einer Postadresse werden dem Benutzer topologische Kartendaten, Geländedaten, Satellitenbilddaten, Verkehrsflussdaten und Bilder etc. im Webbrowser angezeigt. Mithilfe dieses Tools kann der Benutzer ein umfassendes Verständnis für die im Web verfügbaren Informationen über einen bestimmten Standort gewinnen [16]. Da in dieser Arbeit verschiedenen Quellen fusioniert werden, die unterschiedliche Strukturen besitzen, wird in dieser Arbeit lediglich die heterogene Datenfusion betrachtet.

2.2 Anwendungsgebiete

Es gibt viele verschiedene Bereiche, in denen Datenfusion verwendet werden kann. In der folgenden Liste sind nur einige Beispiele aufgeführt, um einen Überblick über die Vielfalt dieser Anwendungen zu geben.

- **Wettervorhersage:** In der Wettervorhersage werden Informationen von verschiedenen Quellen wie Satelliten, Wetterstationen und Radargeräten verwendet, um genaue Vorhersagen zu treffen [13].

[1] Google Maps https://www.google.com/maps

- **Biometrie:** In der Biometrie werden verschiedene Datenquellen zur Erkennung von Menschen eingesetzt. Merkmale wie Augenabdrücke, Fingerabdrücke oder Handschriften werden kombiniert, um eine Person eindeutig zu identifizieren [8].
- **Führungsinformationssysteme:** Führungsinformationssysteme kommen hauptsächlich im militärischen Kontext zur Unterstützung der schnellen strategischen Entscheidungsfindung zum Einsatz. Diese Systeme nutzen Datenfusionstechniken, um situationsabhängige Unterstützung bei solchen Entscheidungen zu ermöglichen, die die Genauigkeit der Ergebnisse beeinflussen kann [4, 15].
- **Robotersteuerung:** Eine intelligente Steuerung von Robotern erfordert das Zusammenführen von Informationen aus verschiedenen Quellen. Diese Informationen können entweder von Sensoren des Roboters erfasst werden oder aus einer Datenbank stammen. Es ist wichtig, dass diese Daten in Echtzeit verarbeitet werden, um eine schnelle Reaktion des Roboters zu ermöglichen. Eine ausführliche Übersicht über solche Systeme ist in [17] zu finden.

Während der Literaturrecherche wurde festgestellt, dass die Fusion von tabellarischen Daten nicht genügend in der Literatur untersucht wurde, sodass wenige Publikationen dafür existieren. Allerdings wurden verschiedene Konzepte zur Datenfusion im Bereich der Datenbanken entwickelt. Dementsprechend wird dieser Bereich den Fokus dieser Arbeit darstellen.

2.3 Kombination von Objektdarstellungen

In diesem Abschnitt wird die Verbindung zwischen den realen Objekten mit ihren digitalen Darstellungen betrachtet. Bei den realen Objekten wie Personen, Büchern oder anderen Objekten der materiellen Welt sind die digitalen Darstellungen oft nicht eindeutig zugeordnet. Informationen über diese realen Objekte können an verschiedenen Orten und in verschiedenen Systemen gespeichert sein. Deshalb wird den Begriff 'reale Identität' für das reale Objekt und 'Bezeichner' für seine digitale Version angewandt. Idealerweise sollte jedes reale Objekt einen eindeutigen Bezeichner haben. Allerdings ist dies oftmals in der Praxis nicht gegeben. Ein weiteres Problem stellt die Verwendung derselben Bezeichner für verschiedene Objekte dar. Um dies zu bewältigen, wird ein global eindeutiger digitaler Bezeichner benötigt. Er wird 'reale Welt ID' genannt. Die Abbildung 2.1 demonstriert ein Beispiel für die Beziehung zwischen einem Objekt der realen Welt,

das eine Identität besitzt, und seinen verschiedenen Darstellungen in der digitalen Welt (Bezeichner) [6].

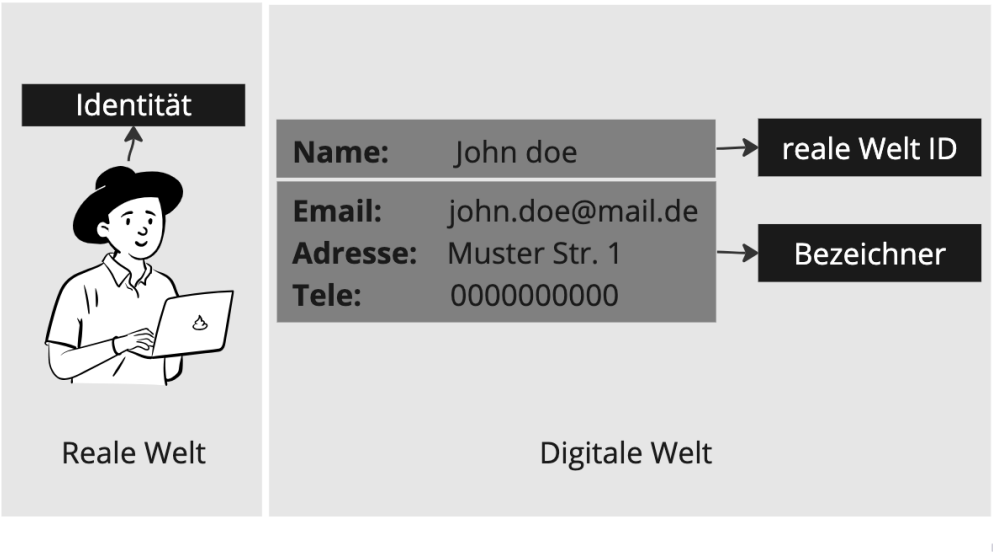


Abbildung 2.1: Beziehung zwischen einem realen Objekt und seinen verschiedenen digitalen Darstellungen. Quelle: Eigene Darstellung.

2.3.1 Der Datenintegrationsprozess

Nach Bleiholder und Naumann ist die Datenfusion nur ein Teil des umfassenden Prozesses der Datenintegration. Dieser Prozess besteht aus drei Phasen: *Daten-Transformation (Schema-Mapping)*, *Duplikaterkennung* und *Datenfusion* (siehe Abbildung 2.2). Die ersten beiden Phasen, Daten-Transformation (Schema-Mapping) und Duplikaterkennung, werden allgemein betrachtet, da der Fokus dieser Arbeit auf der Datenfusion liegt [6].

Integrierte Informationssysteme ermöglichen den Benutzern, eine gemeinsame Sicht auf unterschiedliche Datenquellen zu erhalten. Das Integrationssystem kümmert sich dabei um die Abfrage der Datenquellen, die Kombination und die Darstellung der Ergebnisse. In diesem Szenario werden die drei Phasen des Datenintegrationsprozesses aus Abbildung 2.2 verwendet [6].

Nachdem die Daten vorbereitet werden, existieren drei entscheidende Schritte für die Datenintegration. Im ersten Schritt (Phase 1, *'Schema-Mapping'*) werden die Attribute

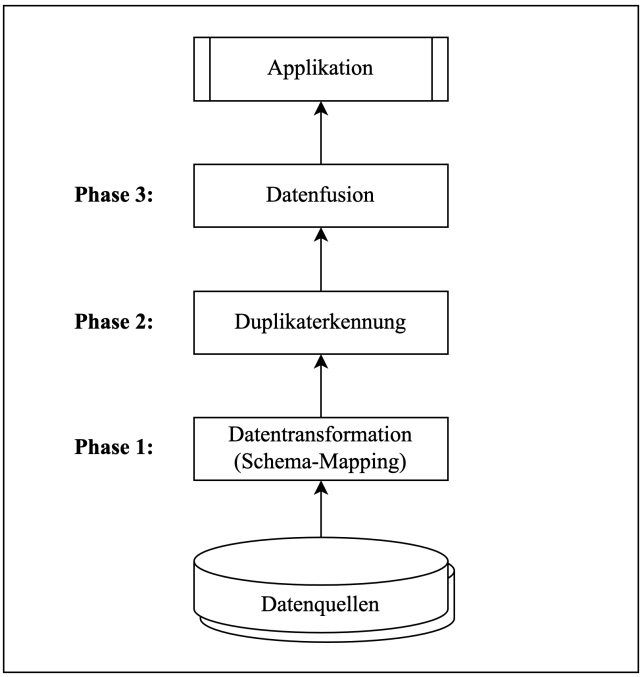


Abbildung 2.2: Die drei Phasen des Datenintegrationsprozesses. vgl. [6].

identifiziert, die für die Beschreibung der Objekte in den Datenquellen verwendet werden. Das Ergebnis ist ein Schemamapping, das die Daten aus den verschiedenen Quellen in eine einheitliche Darstellung transformiert und Namenskonflikte auflöst. Im zweiten Schritt (Phase 2, *'Duplikaterkennung'*) werden verschiedene in den Datenquellen beschriebene Objekte identifiziert und abgeglichen, um Identitätskonflikte zu lösen. Hierbei werden Duplikaterkennungstechniken eingesetzt, damit möglicherweise inkonsistente Darstellungen desselben realen Objekts gefunden und diesem ein eindeutiger Identifikator zugewiesen werden. Im dritten Schritt (Phase 3, *'Datenfusion'*) werden die Darstellungen desselben realen Objekts zu einer einzigen Darstellung zusammengeführt und Inkonsistenzen in den Daten behoben. Dieser Schritt steht im Mittelpunkt der vorliegenden Arbeit. Abschließend werden die Ergebnisse visualisiert beziehungsweise für die Verwendung in anderen Anwendungen geeignet exportiert. In den folgenden Abschnitten werden alle Schritte kurz erläutert. Die nachfolgende Beschreibung der Phasen wird aus [6] entnommen.

Phase 1: Daten-Transformation (Schema-Mapping)

Wenn Daten aus verschiedenen Quellen fusioniert werden, haben sie oft unterschiedliche Formate oder Schemata. Die Umwandlung der zu fusionierenden Daten in ein einheitliches Format beziehungsweise Schema wird als *Daten-Transformation* bezeichnet. Es gibt zwei gängige Ansätze, um die Unterschiede zwischen den Datenquellen zu überbrücken und die Daten-Transformationen zu realisieren: *Schema-Integration* und *Schema-Mapping*. Beim ersten Ansatz der 'Schema-Integration' wird eine bekannte Menge von Datenquellen kombiniert. Dabei werden die einzelnen Schemata betrachtet und es wird versucht, ein neues, einfaches und verständliches Schema zu erstellen, das vollständig und korrekt im Vergleich zu den Quellschemata interpretierbar ist. Der Ansatz des 'Schema-Mapping' bei der Daten-Transformation geht davon aus, dass ein bestimmtes Ziel-Schema bereits vorhanden ist. Das Ziel ist es, eine Menge von Quellen in ein Informationssystem zu integrieren. Dazu wird eine Liste von Gemeinsamkeiten zwischen den Elementen des Quellschemas und denen des globalen Schemas erstellt, um die notwendigen Transformationen zu definieren. Diese Gemeinsamkeiten geben an, wie die Elemente des Quellschemas in das Ziel-Schema übertragen werden sollen. Manchmal kann diese Zuordnung manuell erstellt werden. Jedoch existieren auch Schema-Matching-Techniken, die automatisch Ähnlichkeiten zwischen zwei Schemata finden können. Beide Ansätze, Schema-Integration und Schema-Mapping, haben das gleiche Ziel, Daten von verschiedenen Quellen so zu transformieren, dass sie einem gemeinsamen globalen Schema entsprechen. Das Finden einer vollständigen und korrekten Transformation, sei es zu einem bereits vorhandenen oder einem neuen Schema, ist jedoch eine schwierige Aufgabe. Nach diesem Schritt im Datenintegrationsprozess sind alle Objekte des gleichen Typs einheitlich repräsentiert [6].

Phase 2: Duplikaterkennung

Im nächsten Schritt des Datenintegrationsprozesses werden Duplikate erkannt. Das Ziel dieses Schritts besteht darin, mehrere Darstellungen desselben realen Objekts zu identifizieren und diesen Objekten eindeutige IDs zuzuweisen. Auf den ersten Blick scheint die Erkennung von Duplikaten ein einfacher Schritt zu sein. Hierzu werden einfach alle möglichen Paare von Datenobjekten mithilfe eines Ähnlichkeitsmaßes verglichen und es wird anhand einer festgelegten Grenze entschieden, ob sie als Duplikate betrachtet werden sollten. In diesem Fall erhalten beide Objekte die gleiche ID. Aber in der Praxis

gibt es zwei Herausforderungen, die bewältigt werden müssen: 'Effektivität' und 'Effizienz'. Die Effektivität hängt vor allem von der Qualität des Ähnlichkeitsmaßes und der richtigen Wahl einer passenden Ähnlichkeitsschwelle ab. Ein Ähnlichkeitsmaß ist eine Funktion, die die Ähnlichkeit zwischen zwei Darstellungen bestimmt. Normalerweise ist das Ähnlichkeitsmaß speziell für eine bestimmte Anwendung entworfen, beispielsweise um doppelte Kundendaten zu finden. Die Effizienz hängt davon ab, wie viel Zeit und wie viel Speicherplatz der Algorithmus zur Duplikaterkennung benötigt. Bei großen Datensätzen kann der Vergleich aller möglichen Paare von Datenobjekten sehr zeitaufwendig und speicherintensiv sein, vor allem wenn die Ähnlichkeitsfunktion viel Rechenleistung erfordert.
Nach der Feststellung der digitalen Darstellungen desselben Objektes wird jeder Darstellung eine Objekt-ID zugewiesen. Wenn zwei Darstellungen dieselbe ID haben, bedeutet das, dass sie Duplikate sind. Es ist ebenfalls möglich, dass mehr als zwei Repräsentationen dieselbe ID teilen und eine Gruppe von Duplikaten bilden. Das Ziel ist es, all diese verschiedenen Repräsentationen zu einer einzigen zusammenzuführen. Jedoch ist keine Duplikaterkennungstechnik optimal. Daher kann die zugewiesene Objekt-ID nur eine Annäherung an die tatsächliche ID sein [6].

Phase 3: Datenfusion

Nachdem alle Daten in eine einheitliche Form gebracht und Duplikate erkannt werden, gibt es immer noch einige Unterschiede in den Eigenschaften der Objekte. Zum Beispiel könnten verschiedene Adressen oder Schreibweisen von Namen vorkommen. Der nächste Schritt ist die Datenfusion, bei der diese Unterschiede behoben werden, um eine endgültige Darstellung aller Objekte zu erstellen. Demzufolge entspricht diese Darstellung den Anforderungen der Benutzer bei der Datenintegration. In den Daten können zwei Arten von Konflikten auftreten: 'Unsicherheiten' und 'Widersprüche'. Eine Unsicherheit tritt auf, wenn ein Datensatz einen Wert hat, der in einem anderen Datensatz fehlt. Ein Widerspruch hingegen entsteht, wenn es zwei verschiedene Werte, die nicht gleich `Null` sind, für dieselbe Information gibt. Bleiholder und Naumann haben verschiedene Konfliktlösungsstrategien zusammengestellt, um diese Konflikte zu lösen, und wir werden diese später im Abschnitt zur Konfliktbehandlung 2.4.2 genauer besprechen [5].

Während der Vorbereitung der Daten müssen einige Probleme berücksichtigt werden, wie zum Beispiel unterschiedliche Formate von Datumsangaben. Diese Herausforderungen können später auch als Konflikte betrachtet und während der Datenfusion behandelt

werden. Einige dieser späteren Konflikte entstehen durch nicht standardisierte Werte und unstrukturierte Daten.

2.3.2 Vollständigkeit und Prägnanz

Die Hauptziele der Datenintegration stellen die Verfügbarkeit aller Informationen zu den Objekten (*Vollständigkeit*) (eng. completeness) sowie die Darstellung der Informationen ohne Widersprüche und überflüssige Daten (*Prägnanz*) (eng. conciseness) dar. Um die *Vollständigkeit* zu erreichen, werden weitere Datenquellen hinzugefügt, damit alle Informationen über die Objekte verfügbar sind. Für die Erhöhung der *Prägnanz* werden redundante Daten entfernt, duplizierte Einträge zusammengefasst und gemeinsame Attribute von Objekten vereint [6].

Bleiholder und Naumann haben Maße für Prägnanz und Vollständigkeit definiert, um zu messen, wie gut die Daten zusammengeführt wurden, ähnlich wie bei der Messung von Precision und Recall im Bereich des Information Retrievals[2]. Sie betrachten dabei die Anzahl der einzigartigen und zusätzlichen Objektrepräsentationen im gesamten Datensatz. Dies unterstützt die Bewertung der Vollständigkeit und der Prägnanz bei der Durchführung der Datenfusion (siehe Abbildung 2.3) [6].

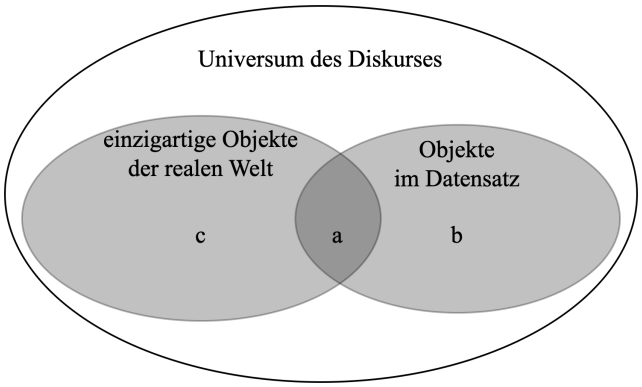


Abbildung 2.3: Die Messung der Vollständigkeit und Prägnanz. vgl. [6].

[2]Der Begriff Information Retrieval beschreibt die (erneute) Beschaffung von Informationen , die eine Person in einer spezifischen Situation benötigt, um Probleme zu lösen [10].

Vollständigkeit

Die Vollständigkeit eines Datensatzes, wie eine Suchanfrage oder eine Tabelle, kann ähnlich wie die *Recall*-Metrik gemessen werden. Dabei geht es darum, wie viele der verfügbaren Daten in diesem Satz im Vergleich zu allen Daten im gesamten Diskurs vorhanden sind. Dies kann auf der Ebene der Datensätze (Tupel) oder auf der Ebene der Attribute (Spalten) gemessen werden. Eine besondere Form der Vollständigkeit ist die sogenannte **extensionale Vollständigkeit**. Hierbei wird gezählt, wie viele einzigartige Darstellungen von realen Objekten in einem Datensatz (a) vorhanden sind, verglichen mit der Gesamtzahl der einzigartigen realen Objekte ($a + c$) in der realen Welt, z. B. in allen Quellen eines integrierten Systems (Abbildung 2.3). Dies gibt an, wie gut dieser Datensatz die reale Welt abdeckt, wobei es angenommen wird, dass dieselben realen Objekte identifiziert werden können [6].

$$extensionale\ Vollständigkeit = \frac{\|eindeutige\ Objekte\ im\ Datensatz\|}{\|alle\ eindeutige\ Objekte\ in\ der\ realen\ Welt\|} = \frac{a}{a+c}$$

Die **intensionale Vollständigkeit** betrifft die Anzahl der eindeutigen Attribute in einem Datensatz im Verhältnis zur Gesamtzahl der eindeutigen Attribute. Ein höherer Wert wird erreicht, wenn zusätzliche Datenquellen hinzugefügt werden, die zusätzliche Attribute liefern. In Abbildung 2.4 wird ein Beispiel für die Kombination von Datenquellen illustriert. Die kombinierte Darstellung hat eine hohe extensionale Vollständigkeit von 1, was bedeutet, dass alle relevanten realen Objekte abgedeckt werden. Die Quellen S und T haben eine geringere extensionale Vollständigkeit von 3/4. Die intensionale Vollständigkeit ist ebenfalls 1, da alle eindeutigen Attribute abgedeckt sind [6].

Prägnanz

Die Prägnanz misst die Einzigartigkeit von Objekten in einem Datensatz. Ähnlich zur *Präzision* in anderen Bereichen betrachtet die Prägnanz, wie eindeutig die Objekte in einem Ergebnis sind [6].

$$extensionale\ Prägnanz = \frac{\|eindeutige\ Objekte\ im\ Datensatz\|}{\|alle\ Objekte\ im\ Datensatz\|} = \frac{a}{a+b}$$

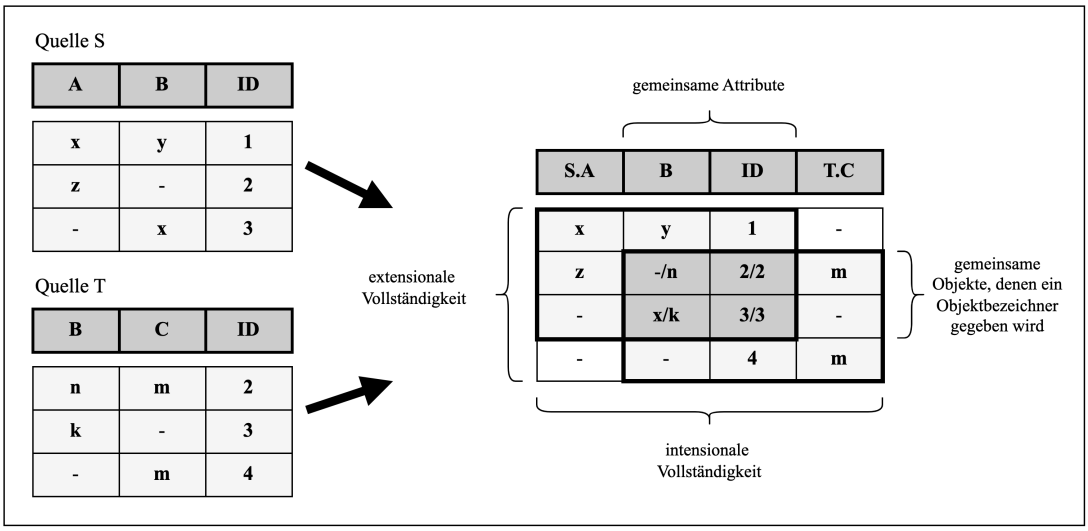


Abbildung 2.4: Extensionale und intensionale Vollständigkeit. vgl. [6].

In Abbildung 2.3 wird die **extensionale Prägnanz** als das Verhältnis der Anzahl der eindeutigen Objekte (a) zur Gesamtzahl der Objektrepräsentationen im Datensatz ($a+b$) definiert. Wenn zum Beispiel vier unterschiedliche Objekte (anhand ihrer Objekt-ID) aus vier verschiedenen Tupelrepräsentationen im Ergebnis stammen, ergibt sich eine extensionale Prägnanz von 1 (= 4/4). Die **intensionale Prägnanz** misst die Einzigartigkeit der Attribute in einem Datensatz im Verhältnis zur Gesamtzahl der Attribute. Auch hier ergibt sich eine intensionale Prägnanz von 1 (= 4/4), wenn alle vier Attribute im Ergebnis unterschiedlich sind. Um diese Maße sinnvoll anzuwenden, müssen die Definitionen der Objekte im Universum und im betrachteten Datensatz übereinstimmen. Bei Operationen wie *Join*, die das Schema ändern, können Probleme auftreten [6].

Vier Grade der Integration

In Abbildung 2.5 werden vier verschiedene Wege illustriert, wie die Beispiel-Tabellen (Quellen S und T) aus Abbildung 2.3 kombiniert werden können. Dies hilft dabei, die Ideen von Vollständigkeit und Prägnanz bei der Zusammenführung von Datenquellen besser nachzuvollziehen. Obwohl sich die Übertragung dieser Vorstellung auf mehr als zwei Quellen als schwierig ergibt, ist diese konzeptionell einfach.
Ohne Kenntnis über das Schema-Mapping und ohne Informationen über Identifikatoren des Objekts liefert das Integrationssystem das beste Ergebnis, wie in Abbildung

2.5(a) dargestellt wird. Dieses Ergebnis ist zwar in Bezug auf Vollständigkeit hoch, jedoch nicht prägnant. Mit dem Wissen über gemeinsame Attribute durch das Schema-Mapping können Ergebnisse wie in Abbildung 2.5(b) erzielt werden. Hierbei werden die beiden Quell-Tabellen durch eine 'outer union'-Operation vereint, wobei gleichnamige Attribute zugeordnet werden. Solche Ergebnisse werden als intensionell prägnant bezeichnet, da jede Eigenschaft der realen Welt nur durch ein Attribut repräsentiert wird und das vollständige Schema-Mapping genutzt wird. Durch die Kenntnis über gemeinsame Objekte, beispielsweise durch die Verwendung einer global konsistenten ID, können Ergebnisse wie in Abbildung 2.5(c) erzielt werden. Hierbei wird das einzige gemeinsame Attribut, die ID, als Objekt-Identifikator in den Quellen S und T verwendet. Solche Ergebnisse werden als extensional prägnant bezeichnet, da kein reales Objekt durch mehr als ein Tupel repräsentiert wird. Schließlich zeigt Abbildung 2.5(d) das Ergebnis nach dem Abgleich von gleichnamigen Attributen unter Verwendung eines Schema-Mappings und gemeinsamen Objekten unter Verwendung eines Objekt-Identifikators. Dieses Ergebnis enthält pro dargestelltem realen Objekt nur ein Tupel und pro Attribut nur einen Wert. Ein solches Ergebnis, das sowohl intensional als auch extensional prägnant ist, stellt das ultimative Ziel der Datenfusion dar. Es ist zu beobachten, dass das ID-Attribut ein besonderer Fall ist und keine Datenkonflikte aufweist, da es zur Identifikation von realen Objekten dient [6].

Die meisten Integrationsmethoden und integrierten Informationssysteme sind in der Lage, sowohl die extensionale als auch die intensionale Vollständigkeit zu erhöhen. Dies ist eine Hauptfunktion dieser Systeme. Jedoch wird das Problem der Prägnanz nur wenig berücksichtigt [6].

S.A	S.B	S.ID	T.ID	T.B	T.C
x	y	1	-	-	-
z	-	2	-	-	-
-	x	3	-	-	-
-	-	-	2	n	m
-	-	-	3	k	-
-	-	-	4	-	m

(a) Es wird kein Mapping und kein Objektidentifikator verwendet.

S.A	B	ID	T.C
x	y	1	-
z	-	2	-
-	x	3	-
-	n	2	m
-	k	3	-
-	-	4	m

(b) Mapping (S.B <-> T.B, S.ID <-> T.ID) verwendet, aber kein Objektidentifikator verwendet.

S.A	S.B	ID	T.B	T.C
x	y	1	-	-
z	-	2	n	m
-	x	3	k	-
-	-	4	-	m

(c) Partielles Mapping (S.ID <-> T.ID) verwendet, Objektidentifikator (S.ID <-> T.ID) verwendet.

S.A	B	ID	T.C
x	y	1	-
z	n	2	m
-	x	3	-
-	-	4	m

(d) Vollständige Mapping (S.B <-> T.B, S.ID <-> T.ID) und Objektidentifikator (S.ID <-> T.ID) verwendet.

Abbildung 2.5: Vier Grade der Integration. vlg. [6].

2.4 Umgang mit Konflikten

Dieser Abschnitt befasst sich mit widersprüchlichen Informationen, die während der Datenintegration auftreten können. Hierbei liegt der Fokus auf Datenkonflikten und Mechanismen zur Bewältigung dieser Konflikte.

2.4.1 Datenkonflikte bei der Datenintegration

Im Kontext der Datenintegration treten aufgrund unterschiedlicher Darstellungen desselben realen Objektes in den Quellen drei Arten von Konflikten auf [6]:

- **Schematische Konflikte** treten auf, wenn verschiedene Namen für Attribute verwendet werden, um dasselbe Objekt zu beschreiben, beziehungsweise wenn die Struktur der Datensätze unterschiedlich ist.
- **Identitätskonflikte** entstehen, wenn dieselben realen Objekte in einer oder mehreren Datenquellen mehrfach beschrieben werden oder wenn verschiedene Repräsentationen derselben realen Identität in einer oder mehreren Quellen vorliegen. Diese Unterschiede werden in den ersten beiden Phasen des Datenintegrationsprozesses, dem *Schema-Mapping* und der *Duplikaterkennung*, gelöst, wie im Abschnitt 2.3.1 beschrieben wird.
- **Datenkonflikte** werden erst im Schritt der Datenfusion angegangen. Solche Konflikte treten auf, wenn für dieselbe reale Identität (z. B. ein Student) semantisch ähnliche Attribute in einer oder mehreren Quellen nicht übereinstimmen. Ein Beispiel wäre, wenn erste Quelle das Alter eines Studenten als '23' angibt, während die zweite Quelle '25' angibt.

Wie im Abschnitt 2.3.1 bereits erwähnt wird, werden zwei Arten von Konflikten in den Daten erkannt: (a) *Unsicherheiten* und (b) *Widersprüche*.

Unsicherheiten entstehen, wenn Informationen fehlen. Das bedeutet, dass ein bestimmter Wert für ein Attribut gesetzt ist, während in anderen Teilen der Daten keine Informationen vorliegen. Ein Beispiel hierfür sind Lücken in den Daten oder fehlende Attribute. In Abbildung 2.4 sind solche Unsicherheiten in den Attributen $S.A$ und $T.C$ (wo das Attribut fehlt) sowie in B (wo in Quelle S Informationen fehlen) für das Objekt mit der $ID = 2$ zu sehen. Solche Unsicherheiten sind in gewisser Weise auch eine Art von Konflikten zwischen den Daten. Allerdings sind sie in der Regel einfacher zu behandeln

als Widersprüche. **Widersprüche** treten auf, wenn es zwei oder mehr unterschiedliche nicht-leere Werte für dasselbe Attribut eines Objekts gibt. Ein Beispiel in Abbildung 2.4 zeigt einen Widerspruch im Attribut B für das Objekt mit der $ID = 3$, wo es unterschiedliche Werte wie x und k gibt [6].

2.4.2 Strategien zur Konfliktbehandlung

Strategien zur Konfliktbehandlung sind grundlegende Herangehensweisen, wie widersprüchliche Daten behandelt werden können. Sie bieten Richtlinien dafür, wie mit inkonsistenten Daten umzugehen ist. Einige dieser Strategien helfen sogar bei der Entscheidung, welcher Wert bevorzugt werden sollte, wie Werte miteinander kombiniert werden können oder wie ein neuer Wert erzeugt werden kann, um eine einzige und stimmige Darstellung bei der Datenfusion zu erreichen. In diesem Abschnitt wird die Kategorisierung dieser Strategien vonBleiholder und Naumann betrachtet [5].
Die Strategien zur Konfliktbehandlung lassen sich grundsätzlich in drei Kategorien einteilen, wie in Abbildung 2.6 dargestellt wird: *Konfliktignoranz*, *Konfliktvermeidung* und *Konfliktauflösung*.

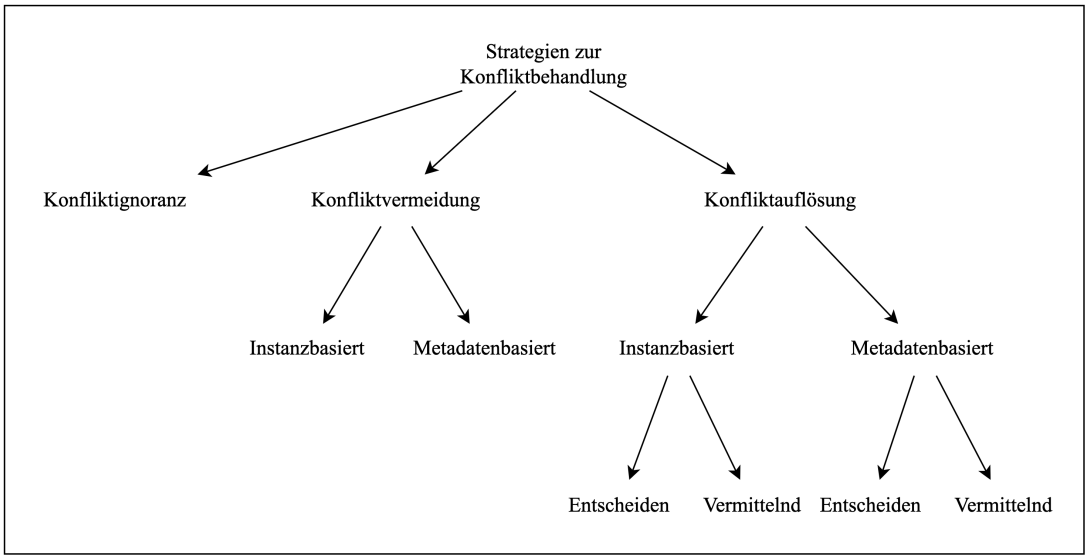


Abbildung 2.6: Klassifikation der Strategien zur Konfliktbehandlung. vgl. [6].

Konfliktignoranz:

Strategien der Kategorie Konfliktignoranz behandeln Konflikte, indem sie sie einfach ignorieren, ohne spezielle Algorithmen zur Konfliktauflösung zu verwenden. Hierbei wird eine externe Konfliktlösung notwendig. Diese Strategien erfordern weder die Erkennung noch die Lösung von Datenkonflikten, da diese Informationen nicht relevant sind. Die Anwendung dieser Strategien ist unkompliziert. Zwei Beispiele solcher Strategien sind [5]:

Weitergabe (Pass it on): Bei dieser Strategie werden alle widersprüchlichen Werte gesammelt und an den Benutzer oder eine andere Anwendung weitergegeben. Es liegt in der Verantwortung des Benutzers oder der Anwendung zu entscheiden, wie mit den möglichen Konflikten zwischen den Werten umgegangen werden soll. Die Strategie selbst enthält keine speziellen Algorithmen zur Konfliktlösung. Dadurch bleiben Unstimmigkeiten in den Daten bestehen, nachdem die Strategie angewendet wurde [5].

Alle Möglichkeiten betrachten (Consider all possibilities): Ziel dieser Strategie ist es, so umfassend wie möglich zu sein. Dazu werden alle denkbaren Kombinationen von Attributwerten aufgezählt und dem Benutzer zur Auswahl gestellt. Es kann notwendig sein, Kombinationen von Attributwerten zu erstellen, die in den ursprünglichen Quelldaten nicht vorhanden waren. Allerdings bleiben Datenkonflikte trotz Anwendung dieser Strategie weiterhin in den Daten bestehen [5].

Konfliktvermeidung:

Diese Strategien lösen Konflikte nicht direkt auf, sondern entfernen zumindest inkonsistente Daten. Das ermöglicht in der Regel schnellere Entscheidungen im Vergleich zu den Konfliktauflösungsstrategien. Jedoch basieren die Entscheidungen der Software oft auf vordefinierten Regeln und berücksichtigen eventuell auftretende Konflikte nicht. Dieses Fehlen der Betrachtung der Daten kann dazu führen, dass die Software nicht alle verfügbaren Informationen optimal nutzt, was ein Problem sein kann. Konfliktvermeidungsstrategien lassen sich im Allgemeinen in zwei Kategorien einteilen: solche, die ihre Entscheidungen auf Metadaten stützen (metadatenbasierte), und solche, die das nicht tun (instanzbasierte) [5].

Zwei Beispiele für instanzbasierte Strategien sind [5]:

Nimm die Informationen (Take the information): Diese Strategie basiert auf der Idee, vorhandene Informationen zu nutzen und fehlende Informationen zu ignorieren. Sie ignoriert Null-Werte und stellt eine natürliche Methode dar, um mit Unsicherheiten umzugehen. Wenn es nur Unsicherheiten gibt, aber keine Konflikte in den Daten, ist es sinnvoll, diese Strategie anzuwenden. Sie wird oft eingesetzt, um überflüssige Null-Werte in einer Abfrage zu entfernen.

Kein Weitererzählen (No Gossiping): Bei dieser Strategie werden nur sichere Fakten in das Ergebnis einer Abfrage aufgenommen, während inkonsistente Antworten ausgeschlossen werden. Diese Entscheidung basiert auf den vorhandenen Daten und zielt darauf ab, inkonsistente Daten bewusst zu ignorieren. Im Gegensatz zu Strategien, die Konflikte ignorieren, identifiziert und berücksichtigt diese Methode Konflikte optimal.

Ein Beispiel für eine metadatenbasierte Konfliktvermeidung ist:

Vertraue deinen Freunden (Trust your friends): Diese Strategie beruht darauf, sich auf andere zu verlassen, um entweder den richtigen Wert oder die richtige Vorgehensweise bereitzustellen. Nachdem es entschieden wurde, welcher Datenquelle vertraut werden soll, wird diese Entscheidung für alle Datensätze getroffen, unabhängig davon, ob ein Konflikt vorliegt oder nicht. Diese Strategie kann dazu führen, dass Daten aus einer bestimmten Quelle bevorzugt werden. Die Quellenpräferenz kann vom Benutzer festgelegt werden oder automatisch anhand von Qualitätskriterien bestimmt werden, um die zuverlässigste oder umfangreichste Quelle auszuwählen.

Konfliktvermeidungsstrategien sind Entscheidungswege, die zuerst festlegen, ob widersprüchliche Informationen berücksichtigt werden sollen oder nicht, ohne dabei die konkreten widersprüchlichen Werte anzusehen. Falls es entschieden wird, dass Widersprüche zugelassen werden, wird zuerst festgelegt, welcher Wert bevorzugt wird, bevor das Endergebnis ausgegeben wird. Das bedeutet, ein System, das solche Strategien anwendet, erkennt eventuelle Konflikte in den Daten nicht zwangsläufig, sondern trifft Entscheidungen auf einer allgemeinen Ebene, ob Inkonsistenzen akzeptiert werden sollen oder nicht [5].

Konfliktauflösung:

Konfliktauflösungsstrategien sind anders als die vorherigen Strategien, da sie Konflikte tatsächlich automatisch lösen. Diese Strategien können in zwei Arten unterteilt werden: auf Instanzen basierend (welcher der gegebenen Werte genommen wird) und auf Metadaten basierend (wenn ein neuer Wert erstellt wird). Zusätzlich können sie in entscheidende und vermittelnde Strategien unterteilt werden. Die entscheidenden Strategien wählen einfach einen der bereits vorhandenen Werte aus. Diese ermöglichen meist eine klare Verfolgung, von woher der Wert stammt. Die vermittelnden Strategien hingegen können neue Werte erstellen, die zuvor in den Daten nicht vorhanden waren. Dies kann bedeuten, dass die genaue Quelle oder Herkunft des neuen Werts nicht immer klar ist [5].

Beispiele für instanzbasierte entscheidende Strategien sind [5]:

Mit den Wölfen heulen (Cry with the wolves): Diese Strategie bevorzugt korrekte Werte gegenüber falschen, wenn genügend Beweise dafür vorhanden sind. Sie wählt den am häufigsten vorkommenden Wert unter den konkurrierenden Werten aus.

Würfeln (Roll the dice): Diese Strategie wählt zufällig einen der konkurrierenden Werte aus. Obwohl diese Strategie als einfach erscheint, ist sie akzeptabel, besonders wenn keine weiteren Informationen zur Verfügung stehen.

Besser verbiegen als brechen (Better bend than break): Hier wird versucht, einen neuen Wert zu generieren, der eine Kompromisslösung zwischen den vorhandenen Werten darstellt. Dieser neue Wert soll irgendwie ‘repräsentativ‘ für die vorhandenen Werte sein.

Beispiele für entscheidungsbasierte Strategien, die auf Metadaten basieren, sind [5]:

In der Mitte treffen (Meet in the middle): Diese Strategie versucht, einen Wert zu generieren, der möglichst nah an allen vorhandenen Werten liegt, ohne einen Wert gegenüber einem anderen zu bevorzugen.

Auf dem Laufenden halten (Keep up to date): Diese Strategie verwendet den neuesten Wert und benötigt dazu Zeitstempelinformationen, um die Aktualität der Daten zu berücksichtigen. Dies kann in Tabellen als separates Attribut

oder durch Datenfluss-Funktionen bereitgestellt werden, besonders in Datenstrom-Umgebungen, wo die Reihenfolge der Daten natürlich ist.

2.5 Operatoren und Techniken für die Datenfusion

Bleiholder und Naumann haben verschiedene grundlegende und erweiterte Methoden vorgestellt, die in der Lage sind, Daten aus verschiedenen Quellen zu verbinden und zu vereinen. Diese Methoden werden als Operatoren bezeichnet. Diese Operatoren ermöglichen es, Informationen aus verschiedenen Tabellen oder Datensätzen in eine einzige Tabelle zu kombinieren. Da in diesem Abschnitt speziell relationale Techniken betrachtet werden, werden Daten aus Quelltabellen (möglicherweise aus verschiedenen Datenquellen) in eine integrierte Tabelle zusammengeführt. Zuerst schauen wir uns Standardmethoden an, wie zum Beispiel *Union* und *Join*. Join-basierte Methoden kombinieren normalerweise Zeilen aus verschiedenen Tabellen, indem sie Kriterien in bestimmten Spalten vergleichen. Union-basierte Methoden erstellen normalerweise zunächst ein gemeinsames Schema und fügen dann die verschiedenen Tupel aus verschiedenen Quelltabellen an [6].

2.5.1 Join-Ansätze

Generell erhöhen Join-Ansätze die intensionale Vollständigkeit, da sie Attribute aus verschiedenen Quellen getrennt in das Ergebnis einbeziehen. Allerdings reduzieren sie oft die extensionale Vollständigkeit, außer im Fall eines *Full Outer Join*. Join-basierte Ansätze sind in Bezug auf die Eindeutigkeit der Attribute sehr wirksam, solange eindeutige Identifikatoren vorhanden sind, die beispielsweise durch die Erkennung von Duplikaten erstellt werden und keine Duplikate innerhalb einer Quelle vorhanden sind [6].

Standard-Joins:

Equi-Join ($\bowtie$): ist eine Methode, um Tupel aus zwei Tabellen zu kombinieren. Dabei werden die Tupel miteinander verknüpft, wenn bestimmte Spalten in beiden Tabellen übereinstimmen. Wenn diese Bedingung erfüllt ist, werden die Tupel kombiniert und das Ergebnis enthält Informationen aus beiden Quellen. In einem Szenario mit dem Ziel, Tupel zu verknüpfen, die dieselben Objekte repräsentieren, könnte ein

'Equi-Join' auf bestimmten Attributen, die eine eindeutige Identifikation erlauben, eine sinnvolle Wahl sein. Dies kann beispielsweise der Fall sein, falls zwei Tabellen vorhanden sind, die auf einer gemeinsamen ID basieren. Dieser Fall wird als Schlüssel-Join ($\bowtie_{id=id}$) bezeichnet. Allerdings sind im Allgemeinen beim Kombinieren von zwei Tabellen durch einen Join andere Bedingungen möglich, sowie auch die Verwendung von Attributen, die keine real-world-ID bilden [6].

Natural Join ($\bowtie$): ist eine Methode, um Tupel aus zwei Tabellen zu verknüpfen. Hierbei werden die Tupel aufgrund von gemeinsamen Attributen mit denselben Namen kombiniert. Wenn die Werte dieser gemeinsamen Attribute in beiden Tabellen übereinstimmen, werden die Tupel miteinander verbunden. Attribute, die in einer der Tabellen fehlen, werden bei der Verknüpfung nicht berücksichtigt, sind aber trotzdem im Ergebnis enthalten [6].

Full Outer Join ($\bowtie\!\!\!\!\subset$): erweitert die Standard Natural Join-Operation. Hierbei werden nicht nur die übereinstimmenden Tupel kombiniert, sondern auch die Tupel, die nur in einer der beiden Quelltabellen vorhanden sind. Die Attribute, die nur in einer Tabelle vorhanden sind, werden in den nicht übereinstimmenden Tupeln der anderen Tabelle mit leeren Werten (*Null*) aufgefüllt. Es gibt auch Varianten des Outer Joins, die auf dem Schlüssel-Join ($\bowtie\!\!\!\!\subset_{id=id}$) anstelle des Natural Join basieren, sowie links ($\supset\!\!\!\!\bowtie_{id=id}$) und rechts Outer Join ($\bowtie\!\!\!\!\subset_{id=id}$) Varianten [6].

Obwohl Joins bei der Datenfusion nützlich sind, besitzen sie Nachteile, die im Folgenden darstellt werden:
Erstens werden Datenkonflikte normalerweise von Joins nicht direkt angegangen. Das bedeutet, dass Objekte entweder gar nicht im Ergebnis auftauchen oder in verschiedenen Tupeln des Ergebnisses erscheinen können. Das tritt bei *Standard-Joins* und *Outer-Joins* auf, vor allem wenn es Konflikte in den verknüpften Attributen gibt.Zweitens ist es allein mit Joins nicht möglich, Informationen aus dem Schema-Matching zu integrieren. Das bedeutet, selbst wenn zwei Spalten in verschiedenen Tabellen semantisch gleich sind, können sie im Ergebnis als separate Spalten erscheinen [6].

Vollständige Disjunktion:

Die Reihenfolge, in der Tabellen durch *outer Join* kombiniert werden, kann unterschiedliche Ergebnisse erzeugen, weil der Operator nicht assoziativ ist. Bei der Verbindung von zwei oder mehr Tabellen wird der *full outer Join* benutzt. Dieser kombiniert passende

Zeilen nach den Verbindungsattributen zu einer einzelnen Zeile. Im Unterschied zum *full outer Join* mit zwei Tabellen ist es jedoch im Allgemeinen nicht möglich, eine vollständige Disjunktion allein durch eine Kette von *outer Joins* zu erreichen. Im Allgemeinen ist die Anwendung einer vollständigen Disjunktion in Situationen mit mehr als zwei Tabellen vorteilhaft, in denen sonst ein *full outer Join* mit nur zwei Tabellen verwendet werden würde. Ein weiterer Vorteil des vollständigen Disjunktionsoperators ist, dass er semantisch äquivalente Spalten aus unterschiedlichen Quelltabellen im Ergebnis kombiniert, was bei normalen Joins normalerweise nicht der Fall ist [6].

Match Join und konflikttolerante Abfragen

Der Match Join ist eine Methode, um konfliktfreie Ergebnisse aus mehreren Datenquellen zu erhalten. Hierbei wird zuerst eine große Tabelle erstellt, indem die Quelltabellen mit ihren realen IDs zusammengeführt werden. Dann werden aus dieser Tabelle Tupel ausgewählt, um ein konfliktfreies Ergebnis gemäß eines Modells für konflikttolerante Abfragen zu erzielen. Die Ausführungskomplexität des Operators ist abhängig von der Anzahl der eindeutigen Werte pro Attribut und kann im schlimmsten Fall exponentiell sein. Die Auswahl der Tupel erfolgt durch detaillierte Operationen, die das gewöhnliche relationale Modell erweitern. Es ist zu beachten, dass dieser Operator in manchen Fällen Tupel mit Attributwertkombinationen enthält, die in keiner der Quellen vorkamen, um neue Informationen zu generieren [6].

Erhöhung der Prägnanz in Join-Ergebnissen

Bei regulären Join-Ergebnissen sind normalerweise nicht alle Attribute, die semantisch gleich sind, in einer einzigen Gruppe vorhanden. Das bedeutet, dass die Ergebnisse vollständig sind, aber nicht prägnant. Dies bedeutet wiederum, dass alle Datenwerte aus den Quellen, die möglicherweise widersprüchlich sind, im Ergebnis weiterhin vorhanden sind. Um diese Konflikte zu lösen und die intensionale Prägnanz zu erhöhen, kann eine zusätzliche Operation verwendet werden, um diese Attribute zu kombinieren [6].

2.5.2 Union Ansätze

Bei Ansätzen mit *Union*-Operationen handelt es sich um eine Möglichkeit, um sicherzustellen, dass alle vorhandenen Tupel aus beiden Tabellen im Ergebnis enthalten sind,

sodass die Gesamtheit der Daten sichergestellt wird. Im Vergleich zu *Join*-Ansätzen sind diese jedoch bezüglich der Prägnanz nicht so effektiv [6].

Union, Outer Union, Minimum Union und Complement Union

Die *Union*-Operation ($\cup$) vereint zwei Tabellen mit Mengensemantik und entfernt dabei exakte Duplikate. Die *Outer-Union* ($\uplus$) ermöglicht es, nicht kompatible Tabellen zu kombinieren, indem sie fehlende Attribute mit *Null*-Werten ergänzt. Der *Minimum-Union*-Operator ($\oplus$) entsteht aus der *Outer-Union*, indem untergeordnete Tupel entfernt werden. Diese Operatoren sind nützlich, um sicherzustellen, dass alle Daten erfasst werden (extensionale Vollständigkeit) [6].

Merge und Prioritized Merge

Der *Merge*-Operator ($\boxtimes$) ist eine weitere Methode, um Daten zu vereinen und Unsicherheiten zu beseitigen, indem er *Coalesce*-Funktionen[3] verwendet. Es gibt auch eine priorisierte Variante, um Werte aus einer Quelle vorrangig zu behandeln. *Merge* eignet sich besonders für Szenarien mit vielen Nullwerten und leicht unterschiedlichen Schemata [6].

Erhöhung der Prägnanz in Union-Ergebnissen:

Da *Union*-Operationen unterschiedliche Darstellungen desselben realen Objekts in verschiedenen Zeilen erlauben, treten oft Datenkonflikte auf. Um solche Konflikte zu bewältigen, werden bei *Union*-Ergebnissen die unterschiedlichen Darstellungen gruppiert und die Werte der Attribute aggregiert. Das Ergebnis ist ähnlich wie das *Union*-Ergebnis, jedoch ohne zusätzliche Schemaänderung. Um die Prägnanz in *Union*-Ergebnissen zu erhöhen, werden Attribute als Identifikatoren für dieselben realen Objekte benötigt. Die Aggregation führt dann alle Tupel einer Gruppe zu einem zusammen, indem Aggregatfunktionen auf die Attributwerte angewendet werden. Standard-Aggregatfunktionen wie (`max, min, sum, count, average`) bieten begrenzte Funktionalität bei der Konfliktlösung [6].

[3] Die Funktion COALESCE() gibt den ersten Nicht-Null-Wert in einer Liste zurück.

2.5.3 Andere Techniken

Es gibt andere Methoden neben Joins und Unions, um Daten zusammenzuführen. Diese Ansätze erweitern das gewöhnliche Datenbankmodell oder bestehende Operationen, um Daten zu fusionieren. Oftmals beinhalten diese Methoden zusätzliche Informationen [6].

Alle Möglichkeiten betrachten (Consider all possibilities)

Eine Methode, mit unsicheren Daten umzugehen, ist 'Alle Möglichkeiten betrachten' (Consider all possibilities). Hierbei werden unsichere Daten in einer Tabelle dargestellt und zur Beantwortung von Fragen genutzt. Dies geschieht, indem eine zusätzliche Spalte hinzugefügt wird, die jedem Datensatz eine Entscheidung zuweist, ob er im Ergebnis erscheinen sollte oder nicht. Dies kann mithilfe von Wahrscheinlichkeiten oder diskreten Werten erreicht werden. Die Datenbankoperationen werden angepasst, um diese zusätzlichen Informationen zu verwenden. In einigen Fällen wird auch eine zusätzliche Spalte eingeführt, die Informationen über die Herkunft des Datensatzes enthält. Viele Ansätze verwenden diese Consider all possibilities Strategie und erlauben dem Benutzer, bewusst zwischen den Möglichkeiten zu wählen oder den wahrscheinlichsten Wert zu erhalten. Dabei entsteht die Problematik, dass die Unsicherheit nur auf der Ebene einzelner Datensätze modelliert wird. Zudem ist es herausfordernd, die Wahrscheinlichkeiten optimal zu bestimmen [6].

Betrachtung nur konsistenter Möglichkeiten

Die Idee hier ist, lediglich Informationen zu liefern, die keine Konflikte aufweisen, wenn eine Abfrage gestellt wird. Wenn jemand eine Frage stellt, erhält er nur Antworten, die unter bestimmten Bedingungen konsistent sind und keine Widersprüche haben. Eine konsistente Antwort ist die Gruppe von Datensätzen, die in allen möglichen Lösungen enthalten sind [6].

3 Konzeption

3.1 Anwendungsfall

In diesem Abschnitt wird ein Anwendungsbeispiel demonstriert, damit die Anforderungen für den Service formuliert werden können. Dieses Beispiel beschreibt eine angenommene Situation, in der der Service eingesetzt werden soll.

3.1.1 Ergänzung Hamburg-Brücken-Daten

Die Stadt Hamburg möchte Informationen über den Verkehr und das Wetter an den Brücken haben. Dazu werden die Brücken-Daten ergänzt, indem Daten wie Verkehrsengpässe, -aufkommen, und Wetterdetails wie Temperatur und Windgeschwindigkeit hinzugefügt werden. In diesem Fall soll der Service die Fähigkeit haben, Daten aus verschiedenen Quellen zusammenzuführen. Dies bedeutet, die Daten von Hamburgs Brücken und die Verkehrsdaten werden basierend auf den vom Nutzer ausgewählten Kriterien zusammengefügt. Danach werden die Wetterdaten, wie Temperatur, Luftfeuchtigkeit und Windgeschwindigkeit, von einer externen Datenquelle mit Hilfe von Ortsangaben (Breitengrad, Längengrad) zu den kombinierten Daten hinzugefügt.

Beschreibung:

Ein Benutzer möchte Daten aus zwei CSV-Dateien, 'Hamburg-Brücken-Daten' und 'Verkehrsdaten' (siehe Tabellen 3.1a und 3.1b) zusammenführen und Wetterdaten von einer externen API hinzufügen. Das Ziel ist es, eine umfassende Tabelle mit allen relevanten Informationen zu erstellen.

Vorbedingungen:

- Der Benutzer hat einen Zugriff auf die genannten CSV-Dateien.

- Beide Dateien haben mindestens eine gemeinsame Spalte, die als Schlüssel verwendet werden kann.
- Der Benutzer hat die benötigten Informationen wie Startdatum, Enddatum, maximale Entfernung und Spaltennamen für Ortsinformationen (Breitengrad, Längengrad).
- Mindestens eine der Dateien enthält die Ortsinformationen (*Breitengrad, Längengrad*).

Nachbedingungen:

- Der Benutzer erhält eine neue CSV-Datei, die Daten aus den beiden ursprünglichen Dateien und zusätzlich Wetterdaten enthält.
- Die Daten in der neuen Datei sind mithilfe des ausgewählten Verknüpfungstyps (Join-Art) zusammengeführt.

Ablauf:

1. Der Benutzer sendet eine Anfrage mit den erforderlichen Parametern an den Service, um die CSV-Dateien zusammenzuführen und Wetterdaten hinzuzufügen.
2. Der Service erhält die Anfrage und liest die Parameter.
3. Der Service prüft, ob die Anfrage gültig ist.
4. Der Benutzer erhält eine Antwort mit einem Link zum späteren Abrufen des Ergebnisses.
5. Der Service liest die beiden CSV-Dateien ein und wandelt die Daten in ein geeignetes Format um.
6. Der Service vergleicht die Schlüsselspalten in den Datensätzen und fügt passende Zeilen zusammen.
7. Die Ortsinformationen (Breitengrad, Längengrad) werden aus den Daten extrahiert.
8. Mithilfe der Ortsinformationen und der Parameter (Startdatum, Enddatum, maximale Entfernung) werden Wetterdaten von einer externen API abgerufen und in das Datenmodell integriert.

9. Die Wetterdaten werden mit den Ergebnis-Daten aus Schritt 6 zusammengeführt.
10. Die kombinierten Daten werden in eine neue CSV-Datei geschrieben.
11. Der Service speichert das Ergebnis der Anfrage und den Zustand.
12. Der Benutzer kann den Link aus Schritt 4 verwenden, um das Ergebnis herunterzuladen oder den Zustand der Anfrage zu überprüfen.

Alternativer Ablauf:

3a Falls die Anfrage ungültige Parameter enthält, gibt der Service eine entsprechende Fehlermeldung zurück.

6a Falls keine der Dateien die benötigten Schlüsselspalten enthält, wird eine Fehlermeldung erstellt.

Brückenname	Brückentyp	Breitengrad	Längengrad
Koehlbrandbruecke	Suspension Bridge	53.4871	9.9573
Lombardsbruecke	Beam Bridge	53.5567	9.9899
Kennedybruecke	Beam Bridge	53.5452	9.9782
Freihafenelbbruecken	Beam Bridge	53.5283	9.9906
Norderelbbruecken	Beam Bridge	53.5572	9.9688
Alter Elbtunnel	Tunnel	53.5444	9.9666

(a) Hamburg Brücken Beispieldaten

Breitengrad	Längengrad	Verkehrszeitstempel	Verkehrsvolumen
53.4871	9.9573	2023-05-01 12:00:00	100
53.5567	9.9899	2023-05-01 12:00:00	75
53.5452	9.9782	2023-05-01 12:00:00	50
53.5283	9.9906	2023-05-01 12:00:00	200
53.5572	9.9688	2023-05-01 12:00:00	150
53.5444	9.9666	2023-05-01 12:00:00	Null

(b) Hamburg Verkehrsvolumen Beispieldaten

Tabelle 3.1: Hamburg Brücken- und Verkehrsvolumen- Beispieldaten

3.2 Anforderungen

In diesem Abschnitt werden die Anforderungen aufgelistet, die der Service erfüllen muss. Diese Anforderungen sind die Grundlage für die Entwicklung der Softwarearchitektur, der Funktionslogik, des Designs der Kommunikationsschnittstelle (REST-API) und des Datenmodells des Services. Die funktionalen Anforderungen beschreiben die verschiedenen Aktionen oder Aufgaben, die der zu entwickelnde Service ausführen soll. Diese Aufgaben oder Funktionen werden in der Abbildung 3.1 dargestellt. Es ist entscheidend, dass der Service in das bestehende System (**'Dafne'**) integriert werden kann. Dies bedeutet, dass der Service nahtlos mit anderen bereits vorhandenen Systemen arbeiten kann.

F1 Der Service muss eine REST-Schnittstelle bereitstellen, um Anfragen zu empfangen, die die erforderlichen Parameter und Dateien enthalten.

F2 Der Service soll in der Lage sein, Daten aus verschiedenen Quellen zu verarbeiten und in einer einzigen Ansicht zusammenzuführen, basierend auf der vom Benutzer ausgewählten Joinart wie 'Inner_Join', 'left_join', 'right_join', 'Full_outer_join'.

F3 Der Service soll mehrere CSV-Dateien als Eingabe akzeptieren.

F4 Der Benutzer soll die Möglichkeit haben, die Spalten der CSV-Dateien auszuwählen, die als Schlüssel für die Verbindung dienen sollen.

F5 Der Benutzer muss in der Lage sein, die Art der Verbindung (z.B. inner join, left join, right join usw.) auszuwählen.

F6 Der Service wird Wetterdaten von einer externen API abrufen und in die Datenfusion integrieren.

F7 Der Service soll die Daten in geeigneten Datenmodellen speichern, um sie später miteinander zu verbinden.

F8 Der Service soll die Datensätze nach dem vom Benutzer ausgewählten Join-Art fusionieren.

F9 Der Service soll den Zustand des Prozesses speichern.

F10 Der Service soll eine REST-Schnittstelle bereitstellen, um den Zustand der Anfrage oder die Ergebnisse abzurufen.

F11 Der Service soll Ergebnisse als CSV-Datei speichern.

F12 Der Service soll den Zustand und die Ergebnisse für eine festgelegte Zeitspanne speichern. Nach Ablauf dieser Zeitspanne sollen der Zustand und die Ergebnisse automatisch gelöscht werden.

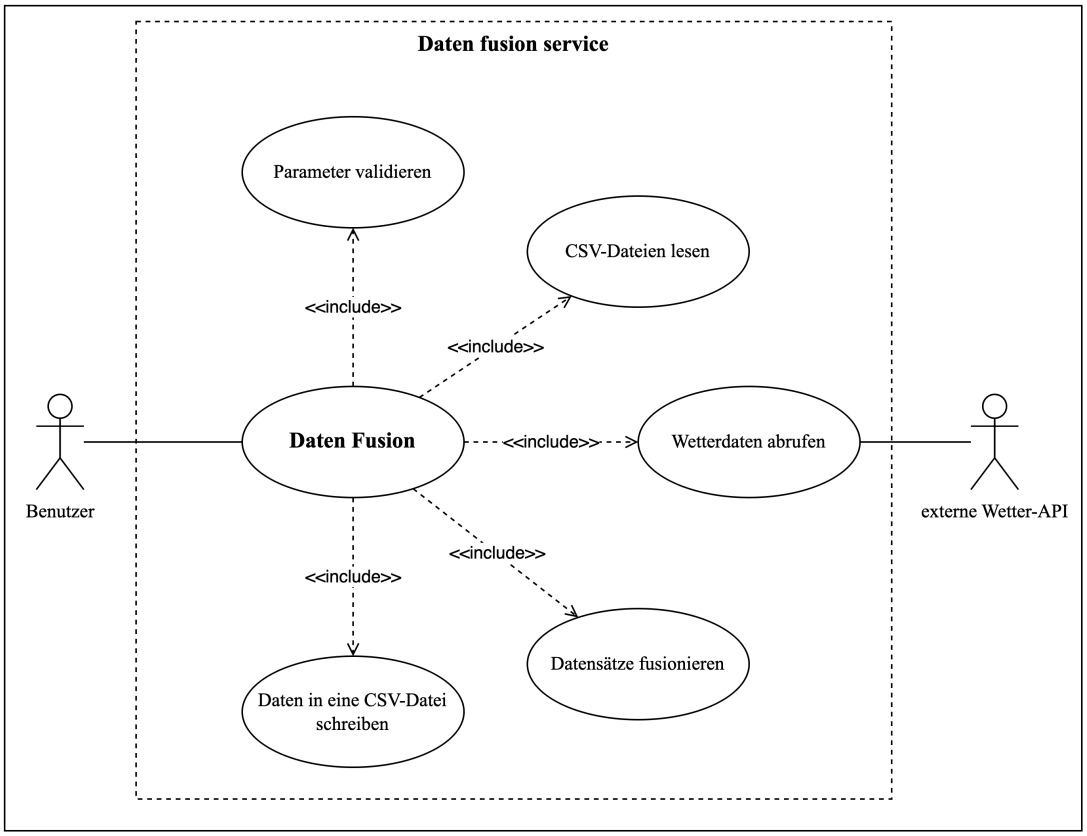


Abbildung 3.1: Anwendungsfalldiagramm zur Funktionalität des Services

3.3 Architektur

Nachdem die Anforderungen definiert werden, wird die Architektur sowie der Entwurf des Services vorgestellt. Das Ziel ist dabei, einen umfassenden Überblick über den Service zu vermitteln.

3.3.1 Kontextsicht

Abbildung 3.2 veranschaulicht die Kontextsicht der Architektur des Datafusion-Services mit einer REST-API und der Integration einer externen REST-API für den Abruf von Wetterdaten. Die Kontextsicht definiert die Systemgrenzen, die externen Systeme sowie Benutzer, die Schnittstellen, die Kommunikationsprotokolle und die Abhängigkeiten des Systems. Zusätzlich wird die grundlegende Systemarchitektur beschrieben.

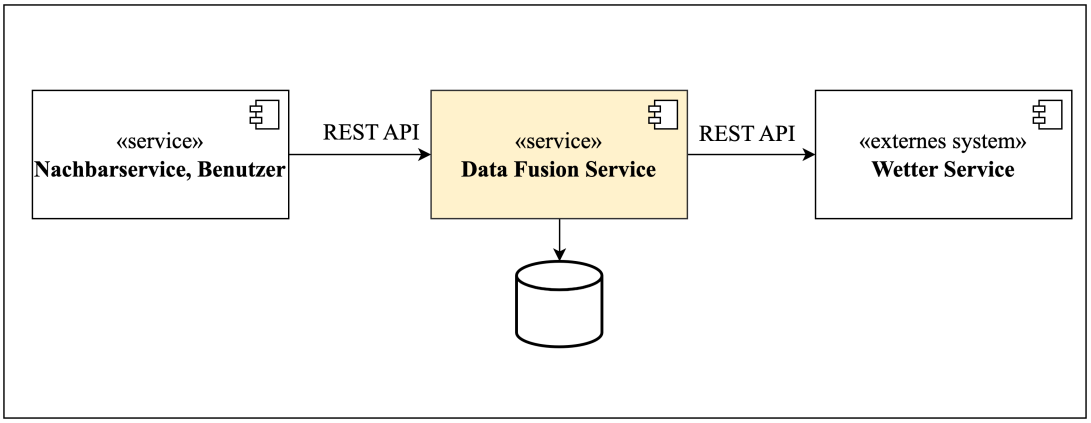


Abbildung 3.2: Kontextsicht des Datenfusion-Services.

Systemgrenzen:

Der Datenfusion-Service wird in einem Docker-Container ausgeführt. Nachbarservices oder Benutzer interagieren mit dem Datenfusion-Service über eine RESTful API. Er greift auf eine externe REST-API zu, um Wetterdaten abzurufen und verwendet eine Datenbank, um die Ergebnisse und den Zustand der Anfragen zu speichern

Externe Systeme und Benutzer:

Benutzer: Personen oder andere Systeme, die den Datenfusion-Service verwenden möchten, um Daten zu fusionieren.

Externe REST-API: Eine Drittanbieter-API, die Wetterdaten bereitstellt und über eine RESTful-Schnittstelle zugänglich ist.

Schnittstellen:

Datenfusion-Service REST-API: Eine RESTful-Schnittstelle, über die Benutzer mit dem Datenfusion-Service interagieren können.

Externe REST-API-Schnittstelle: Eine RESTful-Schnittstelle der Wetterdienst-API, über die der Datafusion-Service Wetterdaten abrufen kann.

Kommunikationsprotokolle:

Datenfusion-Service REST-API: Kommunikation über das HTTP-Protokoll mit JSON, Multipart und Octet-Stream als Datenformat für Anfragen und Antworten.

Externe REST-API-Schnittstelle: Kommunikation über das HTTP-Protokoll mit JSON als Datenformat für Anfragen und Antworten.

Abhängigkeiten:

Der Datenfusion-Service hängt von der Verfügbarkeit und Stabilität der Datenbank und der externen REST-API des Wetterdienstes ab.

Datenbank:

Speichert und verwaltet die relevanten Daten für den Anfragezustand und Fusionergebnisse.

3.3.2 DDD-Schichtstruktur

Aufgrund der Komplexität des Services wird das Schichtentwurfmuster des Domain Driven Designs (DDD)[1] eingesetzt. Die Abbildung 3.3 zeigt diese Schichten wie folgt:

- **API-Schicht:** diese Schicht verarbeitet eingehende Anfragen von Benutzern und leitet sie an die entsprechenden internen Komponenten weiter.
- **Anwendungsschicht:** verwaltet den Ablauf des Anwendungsfalls und koordiniert die Anwendungsaktivitäten.
- **Domänenschicht:** enthält die wichtigen Komponenten und Datenstrukturen, die die Funktionalität des Services Implementieren, wie z.B. Lesen oder Schreiben der CSV-Dateien, Fusionsfunktionalität und Hinzufügen der Wetterdaten.
- **Infrastrukturschicht:** fungiert als unterstützende Bibliothek für alle anderen Schichten und stellt die Integration mit der externen REST-API für den Abruf von Wetterdaten sowie mit dem Datenbank zur Speicherung und Verwaltung von Anfragezustand und Fusionsergebnisse sicher.

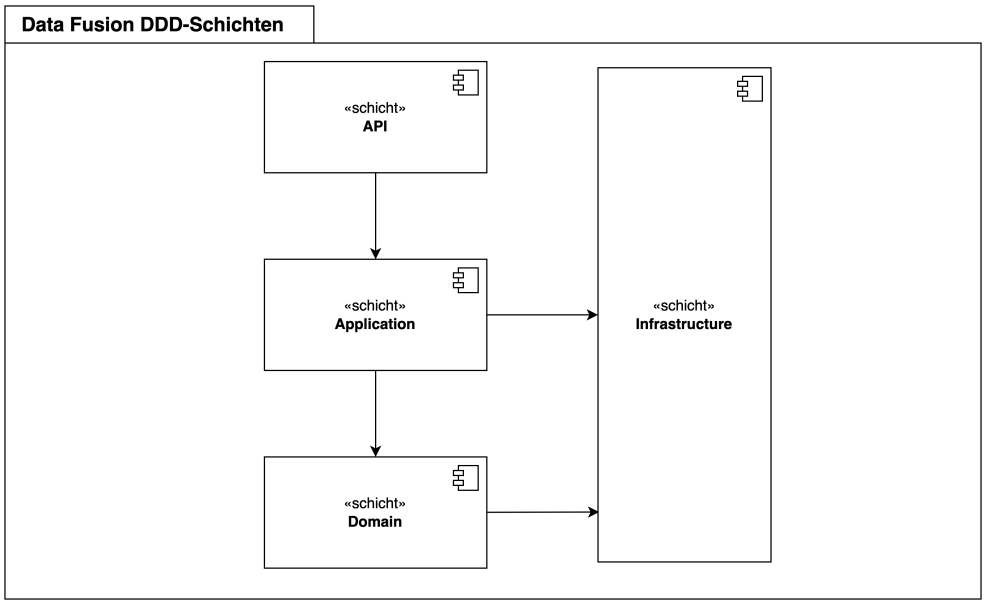


Abbildung 3.3: Das Schichtenentwurfmuster des Domain-Driven Designs (DDD).

[1] http://uniknow.github.io/AgileDev/site/0.1.8-SNAPSHOT/parent/ddd/core/layered_architecture.html

3.3.3 Bausteinsicht

Die Abbildung 3.4 zeigt die folgenden Komponenten des Services:

1. **API Komponente:** Die API-Komponente stellt die Implementierung von Endpunkten bereit, um Anfragen zu verarbeiten und Ergebnisse oder Anfragezustände abzurufen. Diese Komponente ist eng mit der Applikationskomponente verbunden und hängt von ihr ab, um die erforderlichen Funktionalitäten bereitzustellen.

2. **Applikationskomponente:** Die Applikationskomponente übernimmt die Verantwortung für die Implementierung der Anwendungsfälle und koordiniert die Interaktionen zwischen den verschiedenen Komponenten des Services.

3. **Zustand-Komponente:** Die Zustand-Komponente ist zuständig für die Verwaltung der Anfragebearbeitungszustände und die Speicherung der Ergebnisse. Sie ist ebenfalls zuständig für die Aktualisierung des Zustandes während der Verarbeitung von Anfragen.

4. **CSV-Verarbeitung Komponente:** Die CSV-Verarbeitung-Komponente ermöglicht das Lesen und Schreiben von CSV-Dateien. Sie stellt Funktionen zur Verfügung, um Daten aus CSV-Dateien zu extrahieren und in das interne Datenformat zu konvertieren sowie umgekehrt. Ein Datenformat wird eingesetzt, das aus einer Liste von Maps besteht, wobei jede Zeile der CSV-Datei durch eine Map repräsentiert wird. Dabei dienen die Schlüssel der Map den Spaltennamen der CSV-Datei.

5. **Fusion-Komponente:** Die Fusion-Komponente hat die zentrale Aufgabe, die Datenfusion durchzuführen. Sie nutzt die ausgewählte Fusion Strategy, um Datensätze zu kombinieren und ein optimales Ergebnis zu erzeugen. Diese Komponente bietet verschiedene Implementierungen des JOIN-Algorithmus, um unterschiedliche Datenfusionsansätze zu ermöglichen. Die Fusion-Strategien stellen Methoden bereit, um die Fusion von Datensätzen unter Verwendung verschiedener Ansätze durchzuführen.

6. **Wetter-Komponente:** Die Wetterkomponente ist für die Integration von einem externen Wetter-API verantwörtlich, um Wetterdaten abzurufen. Sie bietet Funktionen zur Abfrage und Konvertierung der Wetterdaten in das interne Datenformat.

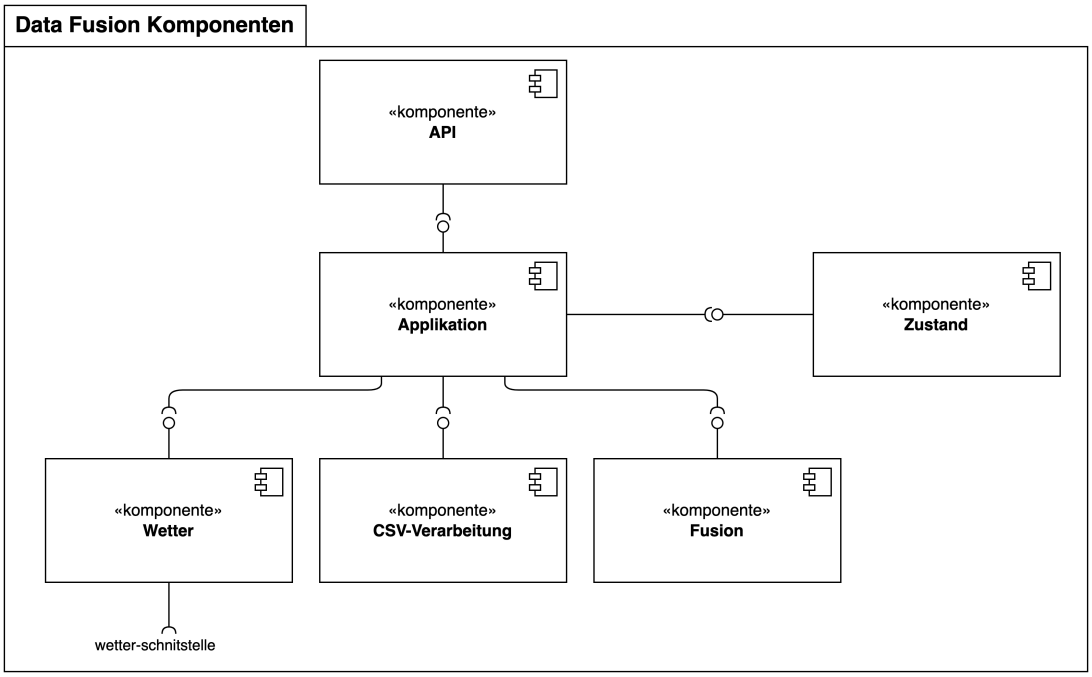


Abbildung 3.4: Bausteinsicht des Datenfusion-Services

3.4 Auswahl der geeigneten Algorithmen und Methoden

In diesem Abschnitt liegt der Fokus auf der Auswahl geeigneter Algorithmen und Methoden für die Umsetzung des generischen Services zur Datenfusion, insbesondere im Hinblick auf Join-Ansätze (Abschnitt 2.5.1). Join-Ansätze sind vielversprechende Methoden, die die gestellten Anforderungen erfüllen und die gewünschten Fusionsergebnisse erzielen. Basierend auf den Anforderungen aus Abschnitt 3.2 wird festgestellt, dass Join-Ansätze die gewünschten Fusionsergebnisse liefern. Sie ermöglichen, Daten aus verschiedenen Quellen zu kombinieren und zu verknüpfen, basierend auf gemeinsamen Merkmalen oder Schlüsselattributen. Durch die Verwendung von Joins können verschiedene Datenobjekte miteinander verbunden werden, um eine umfassendere und konsolidierte Darstellung zu erreichen.

3.5 Auswahl der Wetter-API

Bei der Auswahl der externen Wetter-API wurde Bright Sky[2] ausgewählt. Bright Sky bietet eine JSON-API für die Wetterdaten des Deutschen Wetterdienstes[3] (DWD) an. Es ermöglicht den Zugriff auf verschiedene meteorologische Beobachtungen und Berechnungen, darunter Wettervorhersagen und Wetterdaten von DWD-Stationen. Bright Sky ist ein Open-Source-Projekt, das darauf abzielt, beliebte Wetterdaten frei verfügbar und einfach über eine JSON-API abrufbar zu machen. Es ist kostenlos, erfordert keine Registrierung und bietet eine umfassende Dokumentation, die den Zugriff auf Wetterdatensätze beschreibt. Die Auswahl von Bright Sky als externe Wetter-API ermöglicht es, auf verlässliche und umfangreiche Wetterdaten zuzugreifen. Die API wird somit verwendet, um Wetterdaten abzurufen und sie in das interne Datenformat des Services zu konvertieren. Die API-Dokumentation [1] enthält alle erforderlichen Informationen, um die Bright Sky API zu verstehen und einzusetzen. Es gibt einen Überblick über die Datenquelle, die Abdeckung, die verfügbaren Endpunkte und die erforderlichen Parameter für jede Anfrage. Die Dokumentation enthält auch Beispiele für Anfragen und Antworten sowie Links zu weiteren relevanten Informationen.

Die Bright Sky API bietet vier Hauptendpunkte:

1. ***/weather*:** ermöglicht den Zugriff auf beobachtete und/oder prognostizierte Wetterdaten für einen bestimmten Zeitraum.
2. ***/current_weather*:** liefert aktuelle Wetterdaten für einen bestimmten Standort.
3. ***/synop*:** ermöglicht den Zugriff auf zehnminütige SYNOP-Beobachtungen[4].
4. ***/sources*:** ruft eine Liste aller Bright Sky-Quellen ab, die den angegebenen Kriterien für den Standort entsprechen. Die Quellen werden nach Entfernung sortiert.

Für jede API-Anfrage werden die erforderlichen Parameter und deren mögliche Werte beschrieben. Die Dokumentation enthält ebenfalls Informationen über die API-Server-URL und das Antwortformat (JSON). Insgesamt bietet die vorgeschlagene API-Dokumentation alle Informationen, die Entwickler benötigen, um die Bright Sky API erfolgreich anzuwenden.

[2] Bright Sky: https://brightsky.dev/
[3] Deutscher Wetterdienst: https://www.dwd.de/DE/Home/home_node.html
[4] Synoptische Beobachtungen (SYNOP) sind regelmäßige Wetterbeobachtungen zu bestimmten Zeitstandards gemäß international gültiger Regelungen [11].

4 Umsetzung

In diesem Kapitel folgt die technische Umsetzung des generischen Services zur Datenfusion. Hierfür wird im Abschnitt Ïmplementierung"die geeigneten Werkzeuge und Tools, wie Famework, Umgebeung oder Programmiersprache, im Bezug auf den vorgesehenen Einsatz diskutiert.

4.1 Technische Umsetzung

Für die Implementierung des generischen Services zur Datenfusion wird eine sorgfältige Auswahl der Technologien getroffen, um eine effiziente und leistungsfähige Umsetzung sicherzustellen. In diesem Fall werden Quarkus[1] als Framework, Kotlin[2] als Programmiersprache, IntelliJ[3] als integrierte Entwicklungsumgebung (IDE), Docker[4] für die Containerisierung, Gradle[5] als Build-Tool und PostgreSQL[6] als Datenbank für die Speicherung von Ergebnissen und Anfragenzuständen ausgewählt.

- **Framework:** Quarkus ist ein effizientes Hochleistungs-Framework für die Entwicklung von Javaanwendungen. Es bietet eine effiziente Verarbeitung und optimierte Ressourcennutzung. Quarkus ermöglicht eine schnelle Entwicklung und Ausführung von Microservices und unterstützt eine Vielzahl an Funktionen, die für die Datenfusion benötigt werden, wie zum Beispiel das Verarbeiten von Datenströmen, das Verbinden mit Datenquellen sowie das Bereitstellen von APIs.
- **Programmiersprache:** Als Programmiersprache wird Kotlin angewandt und ist eine hochentwickelte Sprache, die nahtlos mit Java interagiert. Sie bietet eine hohe

[1] www.quarkus.io
[2] www.kotlinlang.org
[3] www.jetbrains.com/idea/features/
[4] www.docker.com
[5] www.gradle.org
[6] www.postgresql.org

Produktivität bei der Entwicklung und Implementierung komplexer Logik. Durch die Verwendung von Kotlin können die Vorteile der Java Virtuelle Maschine (JVM) genutzt werden, sodass eine optimale Leistung und Kompatibilität gewährleistet werden.

- **Entwicklungsumgebung (IDE):** IntelliJ wird als integrierte Entwicklungsumgebung (IDE) eingesetzt. Sie ist eine leistungsstarke IDE, die speziell für die Entwicklung von Java- und Kotlin-Anwendungen vorgesehen wird. Sie bietet umfangreiche Funktionen wie intelligentes Code-Editing, Debugging-Tools, automatische Code-Vervollständigung und eine nahtlose Integration mit Build-Tools und Versionskontrollsystemen.

- **Containerisierung:** Docker wird zur Containerisierung des generischen Services eingesetzt und ermöglicht die Erstellung, Verteilung und Ausführung von Anwendungen in isolierten Containern. Durch die Verwendung von Docker können der Service und seine Abhängigkeiten in einem konsistenten und skalierbaren Umfeld bereitgestellt werden. Dies erleichtert die Portabilität, Skalierbarkeit und das Management des Services.

- **Build-Tool:** Gradle stellt ein leistungsfähiges und flexibles Build-Tool dar, das die Automatisierung des Build-Prozesses und die Verwaltung von Abhängigkeiten ermöglicht. Mit Gradle können Entwickler den Build-Prozess des generischen Services zur Datenfusion effizient konfigurieren und steuern.

- **Datenbank:** PostgreSQL ist eine leistungsstarke und relationale Datenbank mit umfangreichen Funktionen und bietet Unterstützung für komplexe Abfragen an. Sie wird im Kontext des Anwendungsfalls für die Speicherung von Ergebnissen und Anfragenzuständen eingesetzt und bietet eine zuverlässige und skalierbare Datenbanklösung, um die Fusionsergebnisse und den Zustand der Anfragen effizient zu speichern und verwalten.

4.2 Implementierung

In diesem Abschnitt wird die Umsetzung des generischen Datenfusionsservices beschrieben. Hierzu werden verschiedene Komponenten und Funktionen vorgestellt, die für den reibungslosen Betrieb des Services erforderlich sind. Das Klassendiagramm (siehe Abbildung 4.1) veranschaulicht die verschiedenen Klassen des Services. Um die Übersichtlichkeit zu verbessern, werden die Klassen entsprechend ihrer jeweiligen Komponenten farblich hervorgehoben.

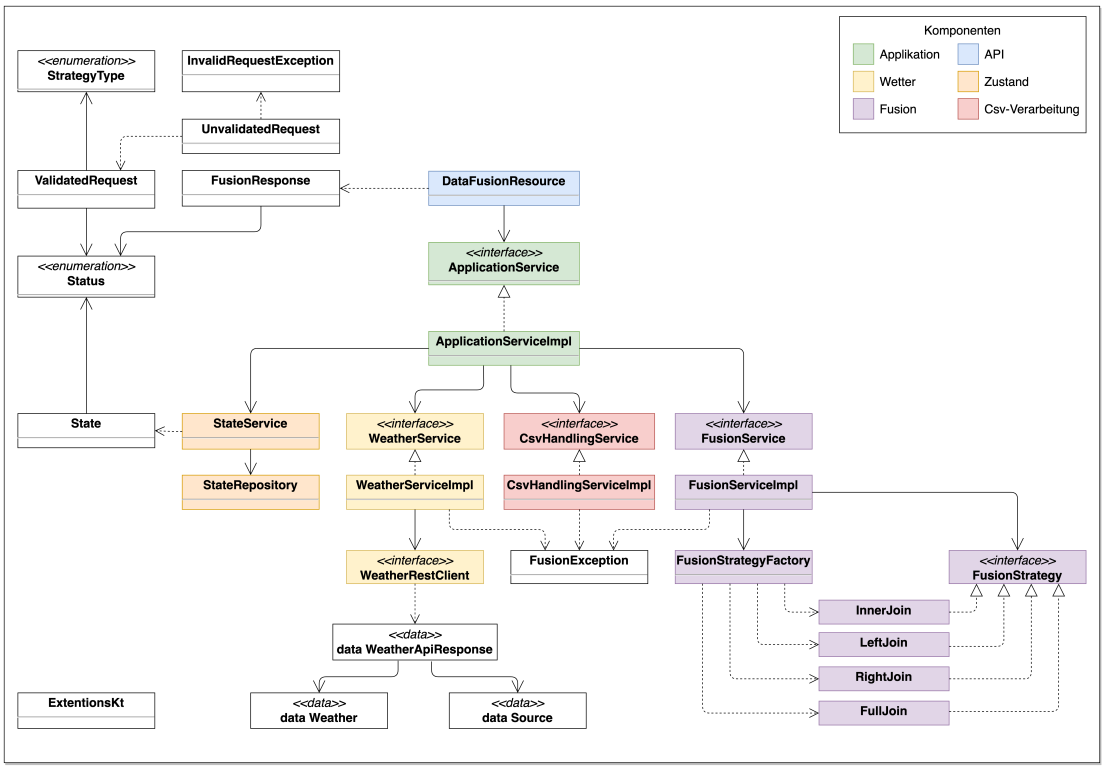


Abbildung 4.1: Klassendiagramm des Datenfusion-Services (Eigene Darstellung)

4.2.1 API Komponente

Die API-Komponente übernimmt die Implementierung der Endpunkte für die Verarbeitung von Anfragen und den Abruf von Ergebnissen oder Anfragezuständen. In der Abbildung 4.1 repräsentiert die Klasse *DataFusionResource* den Kern dieser API-Komponente.

Diese Klasse ist abhängig von der Applikationskomponente und innerhalb der API-Komponente werden zwei Endpunkte zur Verfügung gestellt:

Anfrage einreichen

Die Methode *submitRequest* ist ein POST-Endpunkt, der auf dem Pfad */api/v1.0/fusion* reagiert und Daten im *multipart/form-data* Format akzeptiert. Sie nimmt eine unvalidierte Anfrage entgegen und generiert ein eindeutiger *Request-ID*, sodass die unvalidierte Anfrage in eine validierte Anfrage umgewandelt wird. Falls bei der Umwandlung eine Ausnahme auftritt, wird eine entsprechende Fehlerantwort erstellt und zurückgegeben. Anschließend wird die validierte Anfrage an den *ApplicationService* übergeben, der die Datenfusion durchführt. Hierzu wird ein Link erstellt, um das Ergebnis der Anfrage abzurufen. Schließlich wird eine JSON-Antwort mit dem Statuscode *202 Accepted* zurückgegeben, die den Link und den Status der Anfrage (VALIDATED) enthält. In der nachfolgenden Tabelle 4.1 sind die erforderlichen Parameter für eine Datenfusion-Anfrage aufgeführt.

Param	Typ	Beschreibung
files	array	CSV-Dateien, die fusioniert werden sollen.
identifiers	array	Spaltennamen, die für die Datenübereinstimmung verwendet werden sollen. Die Spaltennamen sollten in allen CSV-Dateien enthalten sein. z.B. 'ID'.
fuse_weather	boolean	Falls 'true' werden die Wetterdaten hinzugefügt.
start_date	string	Das Startdatum für den Data-Fusion-Request der Wetterdaten. Das Format ist ISO 8601, z.B. 2022-01-01T00:00:00+01:00.
end_date	string	Das Enddatum für den Data-Fusion-Request der Wetterdaten. end_date sollte nach oder gleich start_date sein. Das Format ist ISO 8601, z.B. 2022-01-31T23:59:59+01:00.
max_distance	integer	Die maximale Entfernung von Wetterstation in Metern für die Datenübereinstimmung.
lat_column_name	string	Der Name der Spalte, die die Ortsinformationen (Breitengrad) enthält. Standardmäßig ist es 'LATITUDE'.
lon_column_name	string	Der Name der Spalte, die die Ortsinformationen (Längengrad) enthält. Standardmäßig ist es 'LONGITUDE'.
strategy	string	Die Data-Fusion-Strategie, die verwendet werden soll. Gültige Optionen sind INNER_JOIN, LEFT_JOIN, RIGHT_JOIN, OUTER_JOIN. Standardmäßig ist es LEFT_JOIN.

Tabelle 4.1: Die erforderlichen Parameter für Datenfusion-Anfrage

Ergebnis / Zustand abrufen

Die Methode *getResult* ist ein GET-Endpunkt, der auf dem Pfad */api/v1.0/fusion/result/{requestId}* reagiert und den Zustand und das Ergebnis einer Anfrage anhand der *Request-ID* abruft. Je nach Zustand der Anfrage wird eine entsprechende Antwort generiert. Falls die Anfrage als VALIDATED oder IN_PROGRESS gekennzeichnet ist, wird eine FusionResponse mit einem Link zur Überprüfung des Status zurückgegeben. Andernfalls wird die Anfrage als 'COMPLETED' gekennzeichnet. Dementsprechend wird das fusionierte Ergebnis als Datei im CSV-Format zurückgegeben. Wenn ein Fehler auftritt, während der Verarbeitung oder der Anfragezustand nicht gefunden wird, wird eine entsprechende Fehlerantwort ausgegeben.

4.2.2 Applikation Komponente

Die Applikation-Komponente ist für die Umsetzung der Anwendungsfälle zuständig und verwaltet den Zustand der Anfrageverarbeitung sowie die Speicherung der Ergebnisse. Sie kümmert sich um die Aktualisierung des Zustandes während der Anfrageverarbeitung und koordiniert die verschiedenen Komponenten des Services. In der Abbildung 4.1 repräsentiert die Klasse *ApplicationServiceImpl* den Kern dieser Komponente.

Bearbeitung einer Anfrage

Die Methode *fuseDataToCsv* führt die Datenfusion durch. Zunächst wird ein Zustand für die Anfrage erstellt und der Status auf IN_PROGRESS gesetzt. Anschließend werden die Daten aus den eingereichten CSV-Dateien mithilfe der CSV-Verarbeitungskomponente gelesen und durch die Fusionskomponente unter Verwendung der ausgewählten Fusionsstrategie komibiniert. Falls die Option *fuseWeather* in der Anfrage aktiviert ist, werden zusätzlich Wetterdaten durch die Wetterkomponente hinzugefügt. Nach der Datenfusion wird das Ergebnis in eine CSV-Datei geschrieben und der Zustand der Anfrage auf COMPLETED gesetzt. Falls bei der Bearbeitung der Anfrage ein Fehler auftritt, wird der Zustand der Anfrage auf FAILED gesetzt und der Anfragezustand entsprechend aktualisiert. Der Anfragezustand wird durch die Zustandskomponente erstellt, aktualisiert und gespeichert. Diese Komponente ist für die Verwaltung des Zustandes während der Verarbeitung der Anfrage verantwortlich.

Zustand/Ergebnis einer Anfrage

Die Methode *getResult* ermöglicht es, den aktuellen Zustand einer Anfrage anhand ihrer ID mithilfe der Zustandskomponente abzurufen. Dies dient dazu, den Fortschritt der Anfrage zu überprüfen oder das fertige Ergebnis abzurufen.

4.2.3 Zustand Komponente

Die Zustandskomponente ist für die Verwaltung des Anfragezustandes verantwortlich und durch die Klasse *StateService* in Abbildung 4.1 illustriert. Sie bietet Methoden zum Erstellen, Aktualisieren und Abrufen des Zustandes einer Anfrage. Die Klasse *StateService* arbeitet mit dem *StateRepository* zusammen, der den Zugriff auf die persistenten Zustandsdaten in der Datenbank ermöglicht. Das *StateRepository* bietet Funktionen zum Speichern und Abrufen von Zuständen und Ergebnissen in einer Datenbank.

Erstellung eines Zustandes

Die Methode *createState* erstellt einen neuen Zustandseintrag für eine Anfrage. Dabei werden relevante Informationen aus der ‘ValidatedRequest‘ für die Erstellung eines neuen *State*-Objektes genutzt. Im nächsten Schritt wird dieses in der Datenbank gespeichert.

Aktualisierung eines Zustandes

Die Methode *updateState* aktualisiert den Zustand einer vorhandenen Anfrage. Sie ruft den entsprechenden Eintrag aus der Datenbank ab und aktualisiert die Status- und Datenfelder basierend auf den Informationen in der ‘ValidatedRequest‘. Wenn ein Fehlerobjekt übergeben wird, wird auch die Fehlermeldung im Zustand gespeichert.

Abrufen eines Zustandes

Die Methode *getState* ermöglicht den Abruf des Zustandes einer Anfrage anhand ihrer ID. Sie ruft den entsprechenden Eintrag aus der Datenbank ab und gibt ihn zurück.

Löschen eines Zustandes

Zusätzlich enthält die Komponente die geplante Aufgabe *deleteStates*, die regelmäßig ausgeführt wird, um alte Zustandseinträge zu löschen, die z.B. vor einer Woche erstellt sind.

4.2.4 Komponente der CSV-Verarbeitung

Die CSV-Verarbeitungs-Komponente ermöglicht das Lesen und Schreiben von CSV-Dateien. Sie stellt Funktionen zur Verfügung, um Daten aus CSV-Dateien zu extrahieren und in das interne Datenformat zu konvertieren sowie umgekehrt. Die Klasse *CsvHandlingServiceImpl* in der Abbildung 4.1 repräsentiert die Implementierung dieser Komponente. Die Klasse *CsvHandlingServiceImpl* instanziiert ein *CsvMapper*-Objekt, das für das Lesen und Schreiben von CSV-Daten notwendig ist. Dabei werden verschiedene Konfigurationsoptionen festgelegt, wie das Trimmen von Leerzeichen, das Überspringen leerer Zeilen und die Behandlung leerer Zeichenfolgen als null. Zusätzlich wird das Jackson-Kotlin-Modul registriert, damit die Unterstützung für Kotlin-spezifische Funktionen bereitgestellt wird.

Lesen

Die Methode *read* liest eine CSV-Datei anhand des angegebenen Pfades und gibt eine Liste von *Map<String, Any?>* zurück. Dabei liest der *CsvMapper* die CSV-Daten und wandelt diese in eine Liste von *Maps* um. Jedes Map demonstriert eine Zeile in der CSV-Datei, wobei die Schlüssel die Spaltennamen darstellen.

Schreiben

Die Methode *write* schreibt die angegebenen Daten in eine CSV-Datei und gibt die erstellte Datei zurück. Zunächst wird überprüft, ob die Datensätze nicht leer sind. Anschließend wird ein CSV-Schema erstellt, das auf den angegebenen Spaltennamen basiert. Anschließend wird eine temporäre Datei erstellt, in die die CSV-Daten geschrieben werden. Der *CsvMapper* wird die Daten im nächsten Schritt in die Datei schreiben, wobei das Schema und die Header-Informationen berücksichtigt werden.

4.2.5 Fusionskomponente

Die Fusionskomponente hat die Hauptaufgabe, die Datenfusion durchzuführen. Sie nutzt die ausgewählte Fusionsstrategie, um Datensätze zu kombinieren und ein optimales Ergebnis zu erzeugen. Diese Komponente bietet verschiedene Implementierungen des JOIN-Algorithmus, sodass unterschiedliche Ansätze für die Datenfusion ermöglicht werden. Die Fusionsstrategien stellen Methoden für die Durchführung der Fusion von Datensätzen unter Verwendung verschiedener Ansätze bereit.

Fusionsservice

Der Fusionsservice ist für die tatsächliche Durchführung der Datenfusion verantwortlich. Er verwendet die ausgewählte Fusionsstrategie, um Datensätze zusammenzuführen und ein konsolidiertes Ergebnis zu generieren.

Fusionsstrategien

Die Fusionsstrategien verwenden das Strategy Entwurfmuster, um verschiedene Implementierungen des JOIN-Algorithmus für die Datenfusion zur Verfügung zu stellen. Diese Strategien bieten Methoden an, die die Realisierung der Fusion von Datensätzen mit unterschiedlichen Ansätzen ermöglichen.

Die Methoden (*inner* 1, *left* 2, *right* 3, *getMapByHash* 4, *getHashBasedOnKeys* 5 und *mergeMaps* 6) sind Hilfsmethoden, die von den spezifischen JOIN-Implementierungen verwendet werden:

- **InnerJoin**: Diese Implementierung ruft die Methode *inner* 1 auf, die die übereinstimmenden Maps in beiden Datensätzen zurückgibt. Das Ergebnis ist eine Liste von fusionierten Maps basierend auf dem INNER JOIN.
- **LeftJoin:** Diese Implementierung ruft die Methoden *inner* 1 und *left* 2 auf, das dem LEFT JOIN-Algorithmus entspricht. Das Ergebnis ist eine Liste von fusionierten Maps basierend auf dem LEFT JOIN.
- **RightJoin:** Diese Implementierung ruft die Methoden *inner* 1 und *right* 3 auf, das RIGHT JOIN-Algorithmus entspricht. Das Ergebnis ist eine Liste von fusionierten Maps basierend auf dem RIGHT JOIN.

- **OuterJoin:** Diese Implementierung ruft die Methoden *inner* 1, *left* 2 und *right* 3 auf, das den OUTER JOIN-Algorithmus darstellt. Das Ergebnis ist eine Liste von fusionierten Maps basierend auf dem OUTER JOIN, bei dem sowohl die Datensätze aus den beiden Datensätze beibehalten werden, selbst wenn keine Übereinstimmung vorhanden ist.

Algorithm 1 inner

```
1: function INNER(receiverDataSet, donorDataSet, identifiers)
2:     matchingMaps ← leere Liste von Maps
3:     secondMapByHash ← GETMAPBYHASH(donorDataSet, identifiers)
4:     for map in receiverDataSet do
5:         hash ← GETHASHBASEDONKEYS(map, identifiers)
6:         matchingMap ← secondMapByHash[hash]
7:         if ¬ ISNULLOREMPTY(matchingMap) then
8:             for mat in matchingMap do
9:                 MERGEMAPS(map, mat) hinzufügen zu matchingMaps
10:            end for
11:        end if
12:    end for
13:    return matchingMaps
14: end function
```

Algorithm 2 left

```
1: function LEFT(receiverDataSet, donorDataSet, identifiers)
2:     matchingMaps ← leere Liste von Maps
3:     secondMapByHash ← GETMAPBYHASH(donorDataSet, identifiers)
4:     for map in receiverDataSet do
5:         hash ← GETHASHBASEDONKEYS(map, identifiers)
6:         matchingMap ← secondMapByHash[hash]
7:         if matchingMap ist null then
8:             map hinzufügen zu matchingMaps
9:         end if
10:    end for
11:    return matchingMaps
12: end function
```

Algorithm 3 right

```
1: function RIGHT(receiverDataSet, donorDataSet, identifiers)
2:     matchingMaps ← leere Liste von Maps
3:     secondMapByHash ← GETMAPBYHASH(receiverDataSet, identifiers)
4:     for map in donorDataSet do
5:         hash ← GETHASHBASEDONKEYS(map, identifiers)
6:         matchingMap ← secondMapByHash[hash]
7:         if matchingMap ist null then
8:             map hinzufügen zu matchingMaps
9:         end if
10:    end for
11:    return matchingMaps
12: end function
```

Algorithm 4 getMapByHash

```
1: function GETMAPBYHASH(dataSet, keys)
2:     mapByHash ← leeres Map mit Key-Typ Long und Value-Typ List von Mutable-
   Maps
3:     for map in dataSet do
4:         hash ← GETHASHBASEDONKEYS(map, keys)
5:         mapByHash.computeIfAbsent(hash, λk → leere MutableList).add(map)
6:     end for
7:     return mapByHash
8: end function
```

Algorithm 5 getHashBasedOnKeys

```
1: function GETHASHBASEDONKEYS(map, keys)
2:     hash ← 17
3:     for key in keys do
4:         value ← map[key]
5:         hashCode ← HASHCODEORDEFAULT(value, 0)
6:         hash ← 31 · hash + hashCode
7:     end for
8:     return hash
9: end function
```

Algorithm 6 mergeMaps

```
1: function MERGEMAPS(map1, map2)
2:     mergedMap ← leeres MutableMap von Strings zu beliebigen Werten
3:     allKeys ← Vereinigung der Schlüssel von map1 und map2
4:     for key in allKeys do
5:         value1 ← map1[key]
6:         value2 ← map2[key]
7:         if value1 ist null und value2 ist null then
8:             mergedMap[key] ← null
9:         else if value2 ist null then
10:            mergedMap[key] ← value1
11:        else if value1 ist null then
12:            mergedMap[key] ← value2
13:        else
14:            mergedMap[key] ← value1          ▷ Standardverhalten: Wert von map1 übernehmen
15:        end if
16:    end for
17:    return mergedMap
18: end function
```

4.2.6 Wetterkomponente

Die vorliegende Wetterkomponente ist darauf ausgelegt, Wetterdaten von einer externen API anhand spezifischer Parameter abzurufen und sie in das interne Datenformat der Anwendung zu konvertieren. Dabei wird ein Datensatz, Start- und Endzeitpunkte, eine maximale Entfernung sowie die Spaltennamen für Breiten- und Längengrade berücksichtigt. Die Komponente bietet zudem Funktionalitäten zur Verarbeitung und Zusammenführung der abgerufenen Wetterdaten mit den vorliegenden Datensätzen. Für den Abruf der Wetterdaten wird die API 'Bright Sky' (Abschnitt 3.5) genutzt, die verschiedene Endpunkte zur Verfügung stellt. In dieser Implementierung werden nur zwei Endpunkte verwendet: */sources* und */weather*, um eine effiziente Abfrage der Wetterdaten zu gewährleisten. Um die hohe Anzahl an Wetteranfragen zu bewältigen, die sich aus einem umfangreichen Datensatz ergeben könnte, werden zunächst die Wetterstationen (Quellen) abgerufen, die sich in einem bestimmten Bereich befinden. Dieser Bereich wird als Kreis definiert, dessen Parameter durch die Konfigurationswerte als Mitte des Kreises *weather.coordinates.latitude*, *weather.coordinates.longitude* und den Radius *weather.max_distance* festgelegt werden. Standardmäßig sind diese Parameter auf die geografischen Koordinaten von Hamburg gesetzt. Anschließend werden diejenigen Wetterstationen aus-

gewählt, die sich innerhalb der angegebenen Entfernung von den Koordinaten jedes Datensatzes befinden und während des vorgegebenen Zeitraums Wetterdaten liefern. Für jede ausgewählte Wetterstation und den angegebenen Zeitraum werden daraufhin die entsprechenden Wetterdaten abgerufen. Im letzten Schritt erfolgt die Fusion der abgerufenen Wetterdaten mit den übergebenen Datensätzen. Dabei werden den Datensätzen die Wetterdatensätze hinzugefügt, die sich minimal von dem jeweiligen Datensatz entfernt befinden und einen Zeitstempel innerhalb des vorgegebenen Zeitraums aufweisen. Das Ergebnis ist eine Liste von fusionierten Datensätzen, die sowohl die ursprünglichen Daten als auch die Wetterdaten umfassen. Die Kommunikation mit der externen Wetter-API erfolgt über einen REST-Client, der von der Wetterkomponente genutzt wird.

4.3 Anwendungsablauf

Der folgende Abschnitt skizziert den Ablauf der Anwendung, der sich von der Einreichung einer Anfrage bis zur Abrufung von Ergebnissen oder Anfragezuständen erstreckt. Die detaillierte Veranschaulichung dieses Prozesses erfolgt mithilfe der Sequenzdiagramme in der Abbildung 4.2. Die dargestellten Sequenzdiagramme veranschaulichen den Ablauf der Anwendung. Sie sind in zwei wesentliche Abschnitte gegliedert: 'Anfrage einreichen' (Abbildung 4.2a) und 'Ergebnis oder Anfragezustand abrufen' (Abbildung 4.2b).

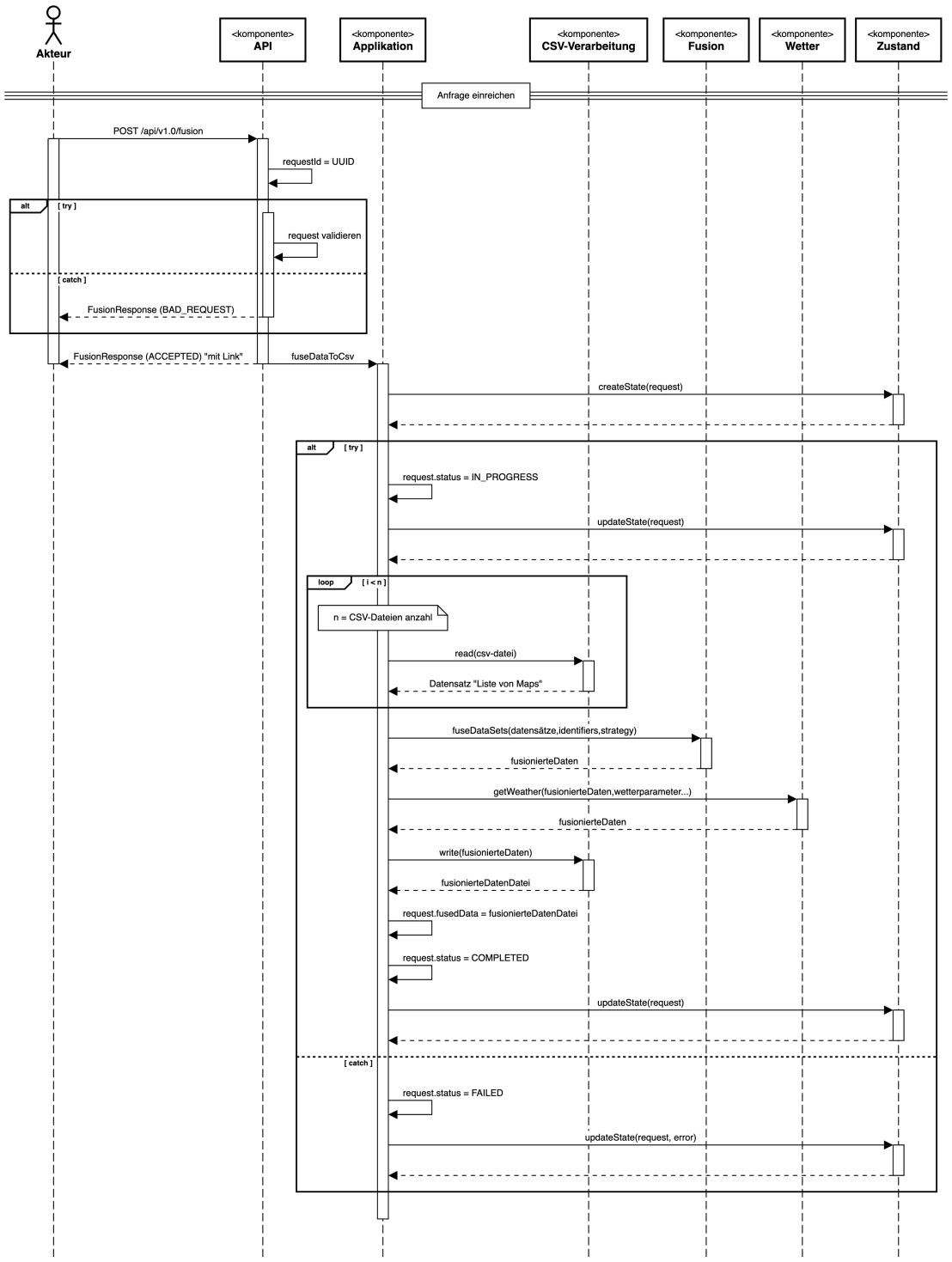
Anfrage einreichen:

- Der Akteur, ein Benutzer oder ein externer Service, sendet eine POST-Anfrage an den Endpunkt *(/api/v1.0/fusion)*.
- Die API empfängt die Anfrage und generiert eine eindeutige Anfrage-ID (UUID).
- Die API validiert die Anfrage, um sicherzustellen, dass ihre Parameter korrekt und gültig sind.
- Falls die Anfrage ungültig ist (z. B. ein fehlender oder fehlerhafter Parameter), wird eine BAD_REQUEST-Fehlerantwort mit einr passenden Nachricht an den Akteur gesendet.
- Wenn die Anfrage erfolgreich validiert wurde, sendet die API eine ACCEPTED-Antwort an den Akteur. Diese Antwort enthält in der Regel einen Link, der später verwendet wird, um das Ergebnis oder den Zustand abzurufen.
- Gleichzeitig beginnt die API, die Fusion als parallele Aufgabe auszuführen und an die Applikation-Komponente weiterzugeben.
- Die Applikations-Komponente erstellt einen Zustand für die Anfrage und setzt ihren Status auf (IN_PROGRESS).
- Die CSV-Verarbeitungs-Komponente liest die CSV-Dateien ein und gibt eine Liste von Maps zurück.
- Die Applikations-Komponente verwendet die Fusions-Komponente, um diese Datensätze basierend auf den angegebenen Identifikatoren und Strategien zu fusionieren.

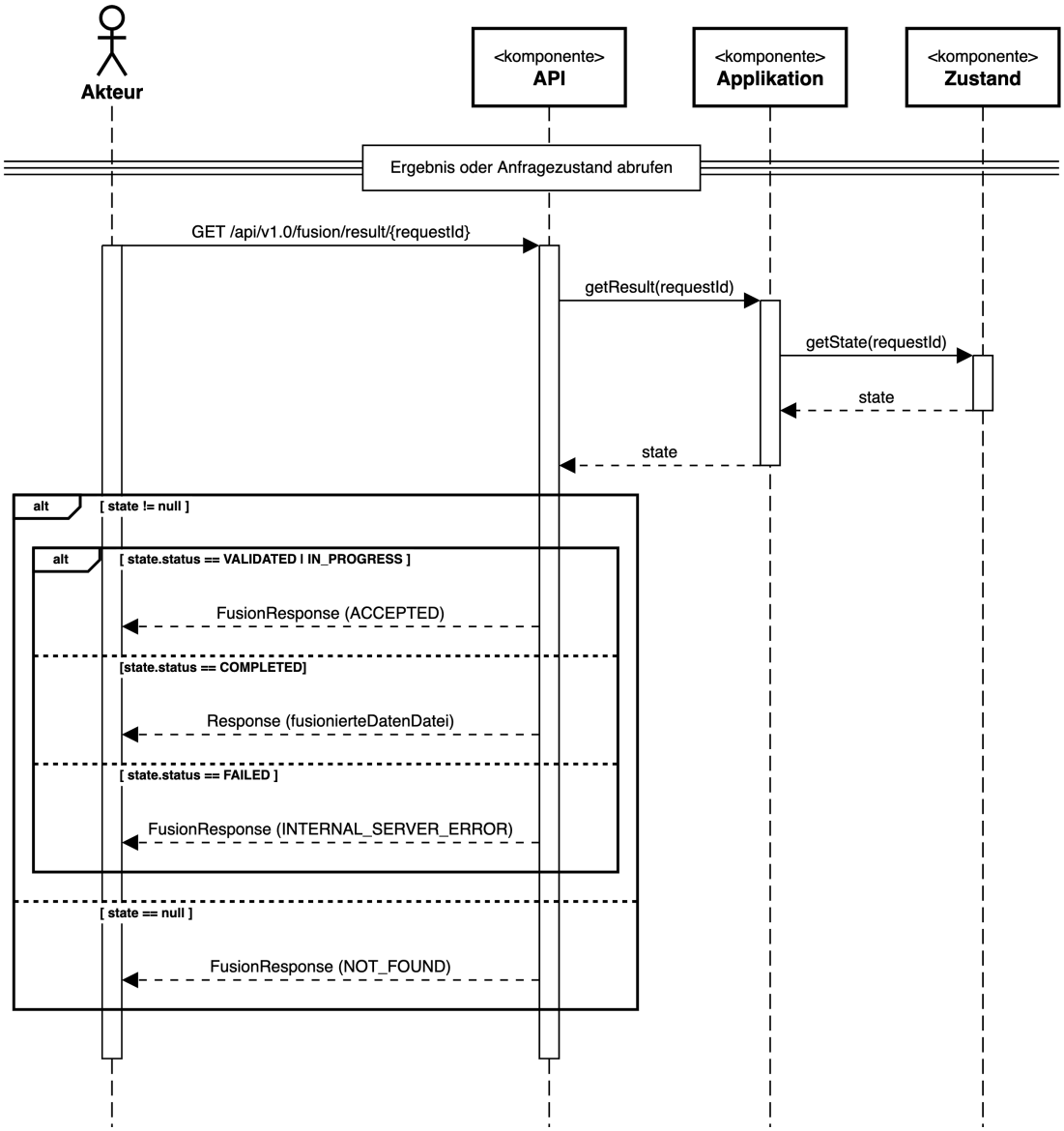
- Falls in der Anfrage der Parameter *fuseWeather* auf 'true' gesetzt ist, ruft die Applikation die "WetterKomponente auf, um Wetterdaten zu den fusionierten Daten hinzuzufügen.
- Die fusionierten Daten werden anschließend von der CSV-Verarbeitungs-Komponente in eine Datei geschrieben.
- Die Applikation aktualisiert den Status der Anfrage auf COMPLETED und den Zustand entsprechend.
- Falls im Prozess ein Fehler auftritt, wird der Status auf FAILED gesetzt und der Fehler wird ebenfalls im Zustand gespeichert.

Ergebnis oder Anfragezustand abrufen:

- Der Akteur sendet eine GET-Anfrage an die API, um den Status oder das Ergebnis einer zuvor eingereichten Anfrage abzurufen. Dies erfolgt über den Endpunkt '/api/v1.0/fusion/result/requestId', wobei 'requestId' die eindeutige Anfrage-ID ist.
- Die API leitet die Anfrage an die Applikations-Komponente weiter, um den Status oder das Ergebnis der Anfrage abzurufen.
- Die Applikations-Komponente fragt den Zustand der Anfrage ab.
- Wenn der Zustand nicht null ist und sich im Status VALIDATED oder IN_PROGRESS befindet, sendet die API eine ACCEPTED-Antwort zurück. Dies bedeutet, dass die Anfrage noch in Bearbeitung ist.
- Wenn der Zustand den Status COMPLETED besitzt, wird das fusionierte Ergebnis (CSV-Datei) als Antwort zurückgegeben.
- Wenn der Status der Anfrage als 'FAILED' markiert ist, wird eine 'INTERNAL_SERVER_ERROR'-Antwort zurückgegeben. Dieser Status weist darauf hin, dass während der Verarbeitung der Anfrage ein interner Serverfehler aufgetreten ist, der dazu geführt hat, dass die Anfrage nicht erfolgreich bearbeitet werden konnte.
- Wenn der Zustand der Anfrage null ist, wird eine NOT_FOUND-Antwort zurückgegeben, da die Anfrage nicht gefunden wurde.



(a) Anfrage einreichen.



(b) Ergebnis oder Anfragezustand abrufen

Abbildung 4.2: Anwendungsablauf des Services zur Datenfusion.

5 Evaluation

Im vorliegenden Kapitel liegt der Fokus auf der Evaluierung des implementierten Services. Hierzu werden Unittests, Integrationstests sowie Manuelltests durchgeführt und verschiedene Szenarien durchgelaufen.

5.1 Integrationstest

Im Rahmen der Integrationstests wird der Datenfusionsservice gründlich auf Leistung und Zuverlässigkeit geprüft, indem Szenarien mit erfolgreichen als auch fehlerhaften Testfällen simuliert werden, sodass verschiedene Fusionierungsstrategien evaluiert werden können, sowohl mit als auch ohne Einbeziehung von Wetterdaten. Zunächst werden acht erfolgreiche Testfälle mit verschiedenen Strategien (Inner Join, Left Join, Right Join, Full Outer Join) durchgeführt. Dabei wurde überprüft, ob der Service korrekte Ergebnisse liefert. In den erfolgreichen Testfällen steht die Überprüfung im Vordergrund, ob der Datenfusionsservice ordnungsgemäß funktioniert, wenn zwei CSV-Dateien miteinander kombiniert werden. Der Testprozess beginnt damit, dass Testdaten in Form von zwei CSV-Dateien bereitgestellt werden. Diese Dateien enthalten Beispieldaten mit Informationen zu geografischen Standorten. Im Anschluss wird eine Anfrage an den Service gesendet, bei der diese beiden CSV-Dateien anhand spezifischer Parameter fusioniert werden. Nachdem die Daten fusioniert sind, wird die Antwort des Services überprüft. Dabei wird erwartet, dass die Antwort den HTTP-Statuscode 'ACCEPTED' zurückgibt. Als Nächstes wird der Link aus der Antwort abgelesen, der AnfrageId enthält, um das Ergebnis abzurufen. Im nächsten Schritt wird eine kurze Verzögerung eingeführt, um sicherzustellen, dass der Service ausreichend Zeit hat, die Anfrage zu verarbeiten, bevor die Ergebnisse überprüft werden. Abschließend wird eine neue Anfrage an den Ergebnislink gesendet, sodass die Antwort auf das Vorhandensein der erwarteten Ergebnisse geprüft wird. Darüber hinaus werden ebenfalls fehlerhafte Testfälle durchgeführt, um zu prüfen, wie der Service auf Anfragen mit ungültigen Parametern reagiert.

Insgesamt umfasste der Integrationstest dreizehn Tests, die verschiedene Aspekte des implementierten Services abdeckten. Diese Tests werden mit unterschiedlichen Parametern und Szenarien durchgeführt, um sicherzustellen, dass der Service robust und zuverlässig ist. Ein detaillierter Testbericht im Anhang A.1 dokumentiert die erfolgreiche Durchführung der Integrationstests und bestätigt die Qualität des Services in Bezug auf die Integration seiner Komponenten.

5.2 Unittests

Im Verlauf der Softwareentwicklung werden spezifische Unittests für jede einzelne Komponente erstellt und durchgeführt. Unittests sind eine bewährte Methode zur Evaluierung, mit der die Korrektheit und Robustheit des Programmcodes überprüft wird. Sie sind ein wesentlicher Bestandteil der Teststrategie und dienen dazu, einzelne Funktionen oder Methoden isoliert von anderen Teilen des Codes zu testen. Das Hauptziel dabei ist sicherzustellen, dass diese Funktionen und Methoden den definierten Anforderungen und Spezifikationen des Services entsprechen.
Insgesamt werden achtundzwanzig Unittests durchgeführt, um die Funktionalität der einzelnen Komponenten zu validieren. Ein ausführlicher Testbericht im Anhang A.2 dokumentiert die erfolgreiche Durchführung der Unittests.

5.3 Manuelle Tests

In manuellen Tests werden die Anfragen und Testschritte manuell durchgeführt, ohne auf automatisierte Testskripte oder Testwerkzeuge zurückzugreifen. Der Service wurde zuerst in einem Docker-Container gestartet. Anschließend wird die Software Postman[1] genutzt, um eine Anfrage an den Service zu senden. Dabei werden die erforderlichen Parameter als Objekt im Format *multipart/form-data* übermittelt. Für jede Anfrage wird jeweils zwei CSV-Dateien ausgewählt, um sie anschließend miteinander zu fusionieren und/oder Wetterdaten hinzuzufügen. Diese Dateien weisen identische Spalten auf, unterscheiden sich jedoch in ihren Werten. Zudem enthalten sie jeweils die gleiche Anzahl von Zeilen. Bei dem Fusionieren werden unterschiedliche Strategien angewandt. Diese Vorgehensweise wird für vier Paare von CSV-Dateien wiederholt, wobei jede Gruppe eine unterschiedliche Anzahl von Zeilen aufwies.

[1] https://www.postman.com/

Die Abbildungen 5.1 und 5.2 zeigen die Leistung verschiedener Fusionsstrategien (Inner Join, Left Join, Right Join und Outer Join) in Bezug auf die Laufzeit in Millisekunden für verschiedene Datensatzgrößen (1.000, 10.000, 100.000, 500.000 und 1.000.000 Zeilen). Die Leistung ist in zwei Szenarien dargestellt: sowohl mit als auch ohne Wetterdaten.

Aus den Abbildungen 5.1 und 5.2 können folgende Erkenntnisse gezogen werden:

- **Datensatzgröße:** Je größer der Datensatz ist, desto länger dauert die Verarbeitung. Dies ist zu erwarten, da größere Datensätze mehr Rechenleistung und Speicher erfordern.
- **Strategien:** Die verschiedenen Strategien haben unterschiedliche Laufzeiten. In den meisten Fällen sind Inner Join und Right Join schneller als Left Join und Full Join. Dies könnte darauf hinweisen, dass die Datensätze häufiger Übereinstimmungen auf der rechten Seite (der zweiten Datei) haben.
- **Einbeziehung von Wetterdaten:** Wenn Wetterdaten in die Verarbeitung einbezogen werden, steigt die Laufzeit erheblich an. Dies ist in den Zeiten in der Abbildung 5.2 gut sichtbar. Die Verarbeitung von Daten mit Wetterdaten erfordert mehr Rechenleistung und Speicher.

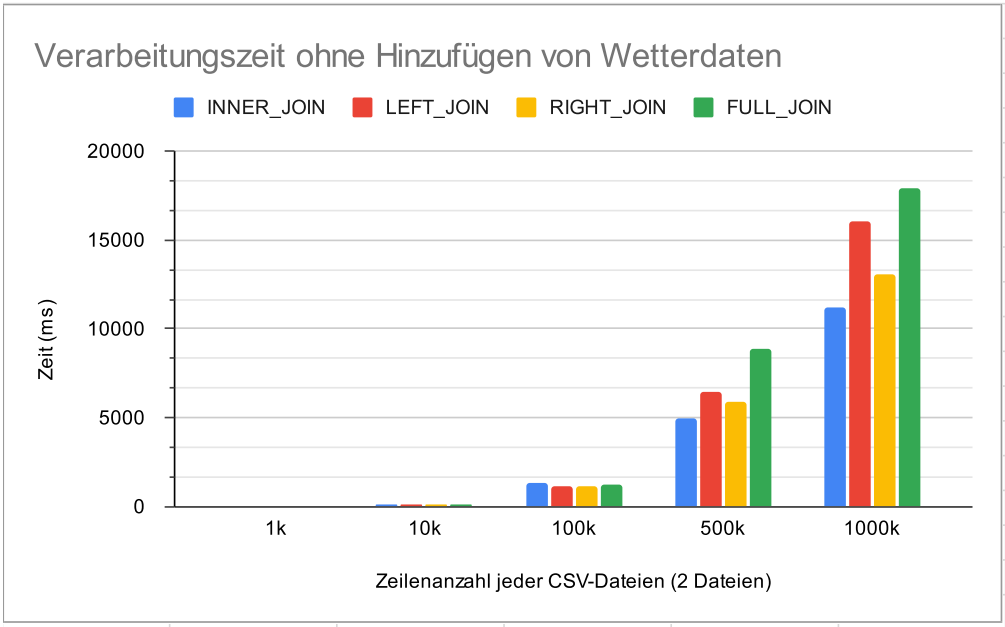


Abbildung 5.1: Verarbeitungszeit für das Zusammenführen von zwei CSV-Dateien mit unterschiedlich großen Datensätzen.

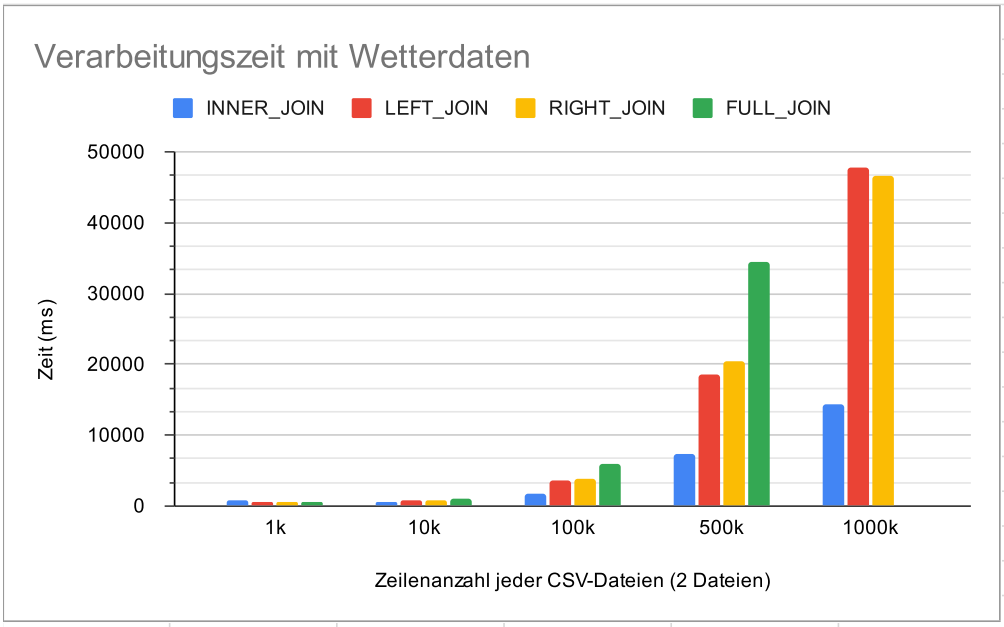


Abbildung 5.2: Verarbeitungszeit für das Zusammenführen von zwei CSV-Dateien mit unterschiedlich großen Datensätzen und das Hinzufügen von Wetterdaten.

5.4 Ergebnisse

Die Evaluation zeigt, dass der implementierte Service seine Ziele erreicht und erfolgreich funktioniert. Das Fazit aus den durchgeführten Tests ist positiv und hebt die folgenden Punkte hervor:

1. **Funktionalität:**
    - Die Zustandskomponente bietet die Möglichkeit, den Zustand einer Anfrage anhand ihrer eindeutigen ID abzurufen. Mithilfe dieser Komponente ist es möglich, den Fortschritt einer Anfrage zu überwachen und das endgültige Ergebnis abzurufen, sobald es verfügbar ist.
    - Die CSV-Verarbeitungs-Komponente funktioniert zuverlässig und liest CSV-Dateien korrekt ein. Sie extrahiert die Daten und konvertiert sie in das interne Datenformat des Services. Beim Schreiben in die CSV-Dateien werden die Daten fehlerfrei geschrieben.

- Die Fusionskomponente erfüllt ihre Hauptaufgabe der Datenfusion erfolgreich. Die verschiedenen Implementierungen des JOIN-Algorithmus (Inner Join, Left Join, Right Join und Outer Join) liefern die erwarteten fusionierten Datensätze, abhängig von der ausgewählten Fusionsstrategie.
- Die Wetterkomponente ruft erfolgreich Wetterdaten von der externen API 'Bright Sky' ab. Die Wetterdaten werden basierend auf den angegebenen Parametern wie Datensatz, Zeitraum und Entfernung richtig verarbeitet und den Datensätzen hinzugefügt.

2. **Leistung:**
   - Der Service bietet eine relativ optimale Leistung. Die Bearbeitungszeit für Anfragen hängt von der Datenmenge und der Komplexität der Datenfusion ab, bewegt sich jedoch in akzeptablen Zeitrahmen, wie aus den Abbildungen 5.1 und 5.2 hervorgeht. Es ist jedoch wichtig zu beachten, dass der Service optimal für die Verarbeitung kleinerer Datenmengen geeignet ist. Bei großen Datenmengen erfordert er erhebliche Speicher- und Prozessorressourcen.
3. **Genauigkeit:**
   - Die Datenfusion ist genau und liefert präzise Ergebnisse. Die Fusionierung der Datensätze erfolgt basierend auf den ausgewählten Fusionsstrategien. Die resultierenden fusionierten Datensätze sind korrekt.
4. **Robustheit:**
   - Der Service zeigt sich robust gegenüber fehlerhaften Anfragen und CSV-Dateien. Sie reagiert angemessen auf ungültige Eingaben und verhindert potenzielle Abstürze oder Datenverluste.
   - Die Zustandskomponente arbeitet zuverlässig und speichert Anfragezustände und Ergebnisse in der Datenbank.
5. **Benutzerfreundlichkeit:**
   - Die Benutzerschnittstelle des Services ist gut gestaltet und benutzerfreundlich. Die verschiedenen Komponenten sind klar voneinander getrennt und einfach zu bedienen.

- Die CSV-Verarbeitungs-Komponente bietet Funktionen zum einfachen Lesen und Schreiben von CSV-Dateien. Dies führt zur Vereinfachung der Arbeit mit externen Datenquellen.
- Die Wetterkomponente ermöglicht den Benutzern, Wetterdaten für bestimmte Datensätze und geografische Koordinaten abzurufen und fügt diese Daten nahtlos den vorhandenen Datensätzen hinzu.

6. **Herausforderungen, Risiken**

   - **Speicher- und Ressourcenbedarf:** Es ist zu beachten, dass der Service bei sehr großen Datensätzen einen erheblichen Speicherplatz- und Prozessorbedarf benötigt. Dies kann zu Herausforderungen führen und erfordert möglicherweise die Optimierung des Codes oder den Einsatz von leistungsstärkerer Hardware.
   - **Abhängigkeit von Externen:** Es besteht eine wesentliche Abhängigkeit von externen Datenquellen, insbesondere von der externen Wetter-API. Änderungen, Ausfälle oder Unterbrechungen dieser Quellen können sich nachteilig auf die Verfügbarkeit und Leistung des Services auswirken.
   - **Wartungsaufwand:** Änderungen in der externen API erfordern möglicherweise Anpassungen in dem Service, um die Kompatibilität sicherzustellen. Dies kann zusätzlichen Wartungsaufwand bedeuten.
   - **Verfügbarkeit der externen API:** Die Verfügbarkeit der externen API kann zu einem kritischen Punkt werden. Ausfälle oder Wartungsarbeiten an der API können den Service beeinträchtigen.

Zusammenfassend zeigt die Evaluation, dass der implementierte Service seine Ziele erfüllt und erfolgreich funktioniert. Er bietet eine adäquate Leistung und Genauigkeit bei der Datenfusion, ist robust gegenüber Fehlern und benutzerfreundlich. Es wurden jedoch auch Herausforderungen und Risiken identifiziert, insbesondere im Zusammenhang mit dem Speicher- und Ressourcenbedarf bei großen Datensätzen sowie der Abhängigkeit von externen Datenquellen. Diese Aspekte müssen bei der weiteren Entwicklung und Wartung des Services berücksichtigt werden.

6 Fazit und Ausblick

6.1 Fazit

Das Hauptziel dieser Bachelorarbeit bestand darin, einen generischen Service zur Datenfusion zu entwickeln. Das Kapitel 2 führt wichtige Grundlagen ein, die das Verständnis der Arbeit unterstützten. Es erklärt den Prozess der Datenfusion, bei dem die Daten aus verschiedenen Quellen zu einer einheitlichen Darstellung zusammengeführt werden. Konflikte und Strategien zur Konfliktbehandlung wurden betrachtet und verschiedene Techniken und Operatoren wie Join- und Union-Ansätze wurden präsentiert, um die Datenfusion effizient durchzuführen.

Aufbauend auf diesen Grundlagen stellt Kapitel 3 die Konzepte des entwickelten Services vor. Es beschreibt die Architektur des Datafusion-Services, einschließlich Schnittstellen, Kommunikationsprotokollen und Abhängigkeiten. Dabei wurde das Domain-Driven Design (DDD) Schichtenentwurfsmuster verwendet, um die verschiedenen Komponenten des Services zu strukturieren. Die Bausteinsicht zeigt die einzelnen Komponenten, wie die API-Komponente, die Applikationskomponente und die Fusions-Komponente. Zudem wurden geeignete Algorithmen und die Auswahl der Bright Sky Wetter-API für das Projekt erläutert.

Das Kapitel 4 veranschaulicht die praktische Realisierung der Konzepte. Dabei wurden die relevanten Technologien vorgestellt, die eine bedeutende Rolle während der Umsetzung spielten. Des Weiteren wurde das Klassendiagramm dargelegt und diskutiert. Zudem wurden die Funktionen der unterschiedlichen Komponenten detailliert beschrieben.

Die Evaluation im Kapitel 5 zeigt, dass der Service seine Ziele erfüllt. Die Funktionalität der Datenfusion, die CSV-Verarbeitung und die Abfrage von Wetterdaten wurden erfolgreich überprüft. Der Service zeigt eine relativ optimale Leistung sowie eine angemessene Genauigkeit und Robustheit. Der Service besitzt das Potenzial, wertvolle Erkenntnisse aus den fusionierten Daten zu liefern.

Die vorliegende Arbeit präsentiert die Implementierung und Evaluierung eines generischen Services zur Datenfusion. Dieser Service bietet den Benutzern die Möglichkeit, CSV-Dateien zu fusionieren und Wetterdaten hinzuzufügen. Obwohl der Service eine relativ optimale Leistung bietet, erfordert er jedoch bei großen Datenmengen erhebliche Speicher- und Ressourcenanforderungen.

6.2 Ausblick

Wie bereits erwähnt, besteht eine der wichtigsten Herausforderungen des Services in seiner Leistungsfähigkeit bei sehr großen Datensätzen. Zukünftige Entwicklungen könnten sich auf die Optimierung des Codes und die Implementierung von Parallelverarbeitung konzentrieren, um die Verarbeitungszeiten weiter zu reduzieren und den Speicherbedarf zu minimieren. Derzeit bietet der Service vier grundlegende Fusionsstrategien (Inner Join, Left Join, Right Join, Full Outer Join) an. Weitere Forschungen könnten unterschiedliche Strategien untersuchen, um die Flexibilität und Anpassungsfähigkeit des Services zu erweitern. Die Konfliktbehandlung könnte durch die Implementierung dieser Strategien optimiert werden, die die speziellen Anforderungen unterschiedlicher Szenarien berücksichtigen, beispielsweise durch die Einbeziehung von Prioritäten oder zusätzlicher Metadaten. Die Erweiterung der unterstützten Datenformate auf JSON und XML sowie die Integration weiterer externer Datenquellen wie Verkehrsdaten würden die Anwendbarkeit des Services in verschiedenen Szenarien erweitern und die Datenbasis diversifizieren.

Literaturverzeichnis

[1] : *Bright Sky API Documentation*. 2023. – URL https://brightsky.dev/docs/#/. – Zugriffsdatum: 2023-03-27

[2] Ayse Glass ; Kübra Tokuç ; Jörg Rainer Noennig ; Ulrike Steffens ; Burak Bek: *DaFne: Data Fusion Generator and Synthetic Data Generation for Cities*. 2023

[3] Baloch, Zartasha ; Shaikh, Faisal K. ; Unar, Mukhtiar A.: A context-aware data fusion approach for health-IoT. 10 (2018), Nr. 3, S. 241–245. – URL https://doi.org/10.1007/s41870-018-0116-1. – ISSN 2511-2112

[4] Benaskeur, Abder R. ; Rhéaume, François: Adaptive data fusion and sensor management for military applications. 11 (2007), Nr. 4, S. 327–338. – URL https://www.sciencedirect.com/science/article/pii/S1270963807000120. – ISSN 1270-9638

[5] Bleiholder, Jens ; Naumann, Felix: Conflict Handling Strategies in an Integrated Information System.

[6] Bleiholder, Jens ; Naumann, Felix: Data Fusion. In: *ACM Comput. Surv.* 41 (2008), 12. – URL https://www.researchgate.net/publication/220566228_Data_Fusion. – Zugriffsdatum: 2023-03-18

[7] Conrad, Stefan ; Saake, Gunter ; Sattler, Kai-Uwe: Informationsfusion — Herausforderungen an die Datenbanktechnologie — Kurzbeitrag —. In: Buchmann, Alejandro P. (Hrsg.): *Datenbanksysteme in Büro, Technik und Wissenschaft*. Springer Berlin Heidelberg, 1999, S. 307–316. – URL http://link.springer.com/10.1007/978-3-642-60119-4_18. – Zugriffsdatum: 2023-03-18. – Series Title: Informatik aktuell. – ISBN 978-3-540-65606-7 978-3-642-60119-4

[8] Esposito, Anna ; Hussain, Amir ; M, Marinaro ; R, Martone: *Multimodal Signals: Cognitive and Algorithmic Issues*. 2009. – ISBN 978-3-642-00524-4

[9] Gehring, Maximilian ; Rüppel, Uwe: Data fusion approach for a digital construction logistics twin. In: *Frontiers in Built Environment* 9 (2023). – URL https://www.frontiersin.org/articles/10.3389/fbuil.2023.1145250. – ISSN 2297-3362

[10] Henrich, Andreas: Information Retrieval 1: Grundlagen, Modelle und Anwendungen. (2008)

[11] Kupfer, Heike ; Herber, Andreas ; König-Langlo, Gert: *Strahlungsmessungen und synoptische Beobachtungen in Ny-Ålesund*. 2003

[12] Lau, Billy Pik L. ; Marakkalage, Sumudu H. ; Zhou, Yuren ; Hassan, Naveed U. ; Yuen, Chau ; Zhang, Meng ; Tan, U-Xuan: A survey of data fusion in smart city applications. 52 (2019), S. 357–374. – URL https://www.sciencedirect.com/science/article/pii/S1566253519300326. – Zugriffsdatum: 2023-08-27. – ISSN 1566-2535

[13] Mitchell, Patrick J. ; Waldner, François ; Horan, Heidi ; Brown, Jaclyn N. ; Hochman, Zvi: Data fusion using climatology and seasonal climate forecasts improves estimates of Australian national wheat yields. 320 (2022), S. 108932. – URL https://www.sciencedirect.com/science/article/pii/S0168192322001253. – Zugriffsdatum: 2023-04-19. – ISSN 0168-1923

[14] Ruser, Heinrich ; Puente León, Fernando: Informationsfusion – Eine Übersicht (Information Fusion – An Overview). 74 (2007), Nr. 3, S. 93–102. – URL https://www.degruyter.com/document/doi/10.1524/teme.2007.74.3.93/html. – Zugriffsdatum: 2023-03-29. – ISSN 0171-8096

[15] Wunder, Michael ; Grosche, Jürgen: *Verteilte Führungsinformationssysteme*. Springer Science & Business Media, 2009. – Google-Books-ID: SIIKrcDPqhoC. – ISBN 978-3-642-00508-4

[16] Yao, JingTao ; Raghavan, Vijay V. ; Wu, Zonghuan: Web information fusion: A review of the state of the art. 9 (2008), Nr. 4, S. 446–449. – URL https://www.sciencedirect.com/science/article/pii/S1566253508000316. – Zugriffsdatum: 2023-03-18. – ISSN 1566-2535

[17] Zhao, Xiaochuan ; Luo, Qingsheng ; Han, Baoling: Survey on robot multi-sensor information fusion technology. In: *2008 7th World Congress on Intelligent Control and Automation*, 2008, S. 5019–5023

A Anhang

A.1 Bericht der Integrationstests

DataFusionResourceTest

all > de.haw.dafne.datafusion.api > DataFusionResourceTest

13	0	0	7.431s
tests	failures	ignored	duration

100% successful

Tests | Standard output | Standard error

Test	Duration	Result
failed-1 valid params (identifiers)()	0.577s	passed
failed-2 valid params (lat_column_name,lon_column_name)()	0.565s	passed
failed-3 valid params (strategy)()	0.023s	passed
failed-4 valid params (start_date,end_date)()	0.009s	passed
failed-5 valid params (max_distance kleiner als 10000)()	0.017s	passed
success-1 zwei csv mit inner join()	2.240s	passed
success-2 zwei csv mit right join()	0.561s	passed
success-3 zwei csv mit left join()	0.567s	passed
success-4 zwei csv mit full outer join()	0.573s	passed
success-5 zwei csv mit Wetter mit inner join()	0.563s	passed
success-6 zwei csv mit Wetter mit right join()	0.583s	passed
success-7 zwei csv mit Wetter mit left join()	0.568s	passed
success-8 zwei csv mit Wetter mit full outer join()	0.585s	passed

Wrap lines ☐
Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

A.2 Bericht der Unit-tests

ApplicationServiceImplTest

all > de.haw.dafne.datafusion.application > ApplicationServiceImplTest

2	0	0	1.650s
tests	failures	ignored	duration

100%
successful

Tests

Test	Duration	Result
success-1 (fuseDataToCsv) fuse bridge data with traffic and Weather()	0.587s	passed
success-2 (fuseDataToCsv) fuse bridge data with traffic()	1.063s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

CsvHandlingServiceImplTest

all > de.haw.dafne.datafusion.domain > CsvHandlingServiceImplTest

6	0	0	0.020s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
fail-1 read throw exception for empty file()	0.003s	passed
fail-2 write throw FusionException if data is empty()	0.004s	passed
fail-3 write throw CsvWriteException if header is empty()	0.002s	passed
fail-4 write throw CsvWriteException if header and data size are different()	0.001s	passed
success-1 read return list of maps()	0.009s	passed
success-2 write return file csv()	0.001s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

FusionServiceImplTest

all > de.haw.dafne.datafusion.domain > FusionServiceImplTest

10	0	0	0.006s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
fail-1 (fuseDataSets) throw FusionException when fuse empty list()	0s	passed
fail-2 (fuseDataSets) throw FusionException when fuse empty datasets()	0.001s	passed
fail-3 (fuseDataSets) throw FusionException when fuse tow empty datasets()	0s	passed
fail-4 (fuseDataSets) throw FusionException when fuse tow empty datasets()	0.001s	passed
success-1 (fuseDataSets) return fused dataset when list of datasets given()	0.001s	passed
success-2 (fuseDataSets) return fused dataset when two datasets given()	0.001s	passed
success-3 (fuseDataSets) return fused dataset when list of datasets given (fuseDataSets) one of them is empty()	0s	passed
success-4 (fuseDataSets) (fuseDataSets) return fused dataset when two datasets one of them is empty()	0.001s	passed
success-5 (fuseDataSets) return fused dataset when two datasets one of them is empty()	0.001s	passed
success-6 (fuseDataSets) return fused dataset when two datasets one of them is empty()	0s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

FusionStrategyFactoryTest

all > de.haw.dafne.datafusion.domain.strategies > FusionStrategyFactoryTest

5	0	0	0.001s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
getFusionStrategy should return FullJoin strategy when type is FULL_JOIN()	0s	passed
getFusionStrategy should return InnerJoin strategy when type is INNER_JOIN()	0s	passed
getFusionStrategy should return LeftJoin strategy when type is LEFT_JOIN()	0s	passed
getFusionStrategy should return RightJoin strategy when type is RIGHT_JOIN()	0.001s	passed
getFusionStrategy should throw IllegalArgumentException when type is unknown()	0s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

FusionStrategyTest

all > de.haw.dafne.datafusion.domain.strategies > FusionStrategyTest

4	0	0	0s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
inner should return a list of fused maps with Inner Join strategy()	0s	passed
left should return a list of fused maps with Left Join strategy()	0s	passed
outer should return a list of fused maps with outer Join strategy()	0s	passed
right should return a list of fused maps with Right Join strategy()	0s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

WeatherServiceImplTest

all > de.haw.dafne.datafusion.domain > WeatherServiceImplTest

1	0	0	0.010s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
getWeather should return a list of weather records with coordinates()	0.010s	passed

Generated by Gradle 7.5.1 at 3 Sep 2023, 02:33:28

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original