CAMINO

ein (Daten-)Adapterkonzept zur Kombination der KI-Vertikalen mit Machine Learning Interfaces in der Software Open60

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Sabrina Göllner

[redacted]

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüferin: Prof. Dr.-Ing. Sabine Schumann

Zweitprüferin: Dr. Vera Kamp

Hamburg, 24. 02. 2020

Inhaltsverzeichnis

Abstract

The aim of the present bachelor thesis was to develop a so called „data-adapter“ that allows the connection of a machine learning service to the AI-Vertical of the decision support system Open60 developed by the company data42. Therefore, the concept named CAMINO was developed. This concept consists of a data-oriented API and important internal functionalities, which enables the task of successfully transferring data between the components. This concept was then implemented successfully through a prototype and presented using a case study.

Zusammenfassung

Das Ziel der vorliegenden Bachelorarbeit war es einen „Daten-Adapter“ zu entwickeln, der die Anbindung eines Machine Learning Services an die KI-Vertikale an das von der Firma data42 entwickelte Entscheidungsunterstützungssystem Open60 ermöglicht. Dazu wurde das Konzept CAMINO entwickelt. Dieses Konzept besteht aus einer datenorientierten API und internen Funktionalitäten, welche die Aufgabe des erfolgreichen Datentransfers zwischen den Komponenten ermöglicht. Anschließend wurde dieses Konzept anhand einer prototypischen Implementierung erfolgreich umgesetzt und mittels eines Leitbeispiels vorgestellt.

1 Einleitung

Die Erfolge von Unternehmen gehen mit einer schnellen und genauen Entscheidungsfindung einher. Aufbauend auf einer guten Datenbasis und deren Aufbereitung, stellen interaktive, softwarebasierte Entscheidungsunterstützungssysteme ein wichtiges Utensil in diesem Kontext dar. Diese helfen Entscheidungsträgern dabei aus Daten nützliche Informationen zusammenzutragen, um Probleme zu ermitteln und Lösungen aufzuzeigen. Einsetzbar ist dieses beispielsweise im Risikomanagement, zur Krisenfrüherkennung, um aussagekräftige Vorhersagen zu erstellen.

1.1 Problemstellung und Motivation

Betrachten wir beispielsweise den Bereich der Krisenprävention, so werden Softwarelösungen benötigt, welche die zur Verfügung stehenden Daten vor der Nutzung verifizieren, sortieren und anschließend die Möglichkeit bieten Analysen darauf durchzuführen. Eine solche Software soll auch insbesondere mit „Big Data“ umgehen können. Damit sind jene Datenmengen gemeint, welche die gewöhnliche Verarbeitungskapazität überschreiten und zu groß für herkömmliche Datenbanksysteme sind. Insbesondere muss eine Möglichkeit gefunden werden, auch komplexe und unstrukturierte Daten, die immer in unterschiedlicher Form vorliegen zu verarbeiten (Dumbill, Croll, Steele & Loukides 2012). Diese lassen sich auch mit den „4 Vs“(Volume, Velocity, Variety & Veracity) beschreiben (Schroeck et al. 2012). Um einen Wert aus Ihnen zu ziehen, müssen sie also sortiert, strukturiert, qualitativ hochwertig und in entsprechend hoher Geschwindigkeit verarbeitet und verwaltet werden.
Es existieren viele sogenannte „Citizen Data Scientists“, die sich mit Daten auseinandersetzen können, jedoch mangelt es diesen im Allgemeinen an Expertenwissen, um Data-Mining-Algorithmen[1] anwenden zu können. Diese können durch „Smart Data Discovery Software“ unterstützt werden. Ziel dieser Methode ist es die Komplexität bei der Interaktion des Benutzers mit der Anwendung zu verringern, indem interaktive und intuitive Analysemöglichkeiten bereitgestellt werden (Sallam & Parenteau 2015).
Zusätzlich ist Machine Learning (ML) eine sehr nützliche Technologie für Entscheidungsunterstützungssysteme und gewinnt in Forschung und Praxis zunehmend an Bedeutung. Das Ziel der in ein intelligentes Entscheidungsunterstützungssystem eingebetteten Machine-Learning-Techniken besteht darin, den Anwender zusätzlich be-

[1] Data Mining stellt eine Sammlung von Techniken, Methoden und Algorithmen für die Analyse von Daten dar (Cleve & Lämmel 2014: S.3).

ratend zu unterstützen. Viele Muster werden durch menschliche Wahrnehmung gar nicht erst erkannt, können aber mit computergestützter Hilfe gefunden werden. Entscheidungsträger werden dabei unterstützt, neue Informationen zu sammeln und zu analysieren, Probleme zu identifizieren und zu diagnostizieren, mögliche Vorgehensweisen vorzuschlagen und jene Maßnahmen zu bewerten. Es soll aufgezeigt werden, wie sich Situationen möglicherweise in naher Zukunft entwickeln könnten. Daraufhin lassen sich Entscheidungen zielgerichteter planen.

1.2 Zielsetzung der Arbeit

Die Anwendung Open60 der Firma data42 stellt eine solche Anwendung zur Entscheidungsunterstützung dar. Diese wurde unter der Analysemethode Smart Data Discovery konzipiert. Über diese können visuelle Korrelationen erkannt und dargestellt werden. Zum einen wird dies für den Nutzer auf Basis von visuellen Objekten möglich. Auf der anderen Seite spielt auch ein interaktiv eingebettetes Machine Learning zur Unterstützung bei Entscheidungen eine wichtige Rolle. Bei diesem Aspekt setzt das zu entwickelnde Konzept **CAMINO** (**C**onnecting the **A**I-Vertical to **M**achine Learning **In**terfaces in **O**pen60) an.
Die Daten können vom Fachanwender direkt zusammen mit dem Problemtyp und dem Machine-Learning-Algorithmus auf der Benutzeroberfläche ausgewählt werden. Die Eingabe wird entsprechend der gewählten Machine-Learning-Plattform vorbereitet, an diese weitergereicht und das Modell mit den Testdaten und dem Algorithmus trainiert. Die daraus resultierenden Daten werden nun so weiterverarbeitet, dass sie problemlos in der Anwendung Open60 kompatibel sind.
Ziel der Arbeit ist es daher einen plattformspezifischen Adapter, der speziell für die Daten zuständig ist, zu entwerfen. Dieser soll die Kommunikation der Machine-Learning-Plattform mit der KI-Vertikalen der Software Open60 (siehe Kapitel 2.5) ermöglichen. In Kapitel 3 wird daher ein Konzept entwickelt, welches auf dieses spezielle Problem eingeht und ein Lösungsweg dafür erarbeitet. Die grobe Veranschaulichung von CAMINO ist in folgender Abbildung zu sehen:

Abbildung 1.1: Abstrakte Veranschaulichung von CAMINO
Quelle: eigene Darstellung

1.3 Aufbau der Arbeit

Zunächst wird in Kapitel 2 ein Überblick über die Grundlagen dieser Arbeit gegeben. In Kapitel 3 wird das Konzept CAMINO sowie die zugrundeliegende Architektur vorgestellt. Kapitel 4 behandelt die prototypische Implementierung und Einbettung von CAMINO in die Software Open60. Auf Basis der prototypischen Implementierung findet die Bewertung des Konzeptes und der verwendeten Technologien statt. Abgeschlossen wird die Arbeit durch eine Zusammenfassung und einen Ausblick in Kapitel 5.

2 Grundlagen

Dieses Kapitel behandelt die Grundlagen, die für die Entwicklung des Konzepts CAMINO notwendig sind. Zunächst wird dabei in Abschnitt 2.1.1 auf die für das Konzept relevante Datenbank und deren zugehörigen Bibliotheken eingegangen. Der Abschnitt 2.2 beinhaltet grundlegende Machine Learning-Verfahren. Diese ermöglichen es Modelle zu erstellen, zu trainieren und zur Ableitung von Prognosen auf Basis von Eingabedaten zu verwenden. Der darauffolgende Abschnitt 2.3 behandelt die Programmiersprache Clojure, mit welcher die Umsetzung des Konzepts CAMINO erfolgen wird. Danach wird in Abschnitt 2.4 das der CAMINO-Architektur zugrundeliegende Software-Architekturpattern vorgestellt. Abschließend soll im Abschnitt 2.5 auf das Prinzip der Vertikalen Softwarearchitektur eingegangen werden, um auf den grundlegenden Aufbau der Software Open60 einzugehen.

2.1 SAP HANA & PAL

2.1.1 Historie und Entwicklung der SAP HANA

SAP HANA ist ein Akronym für „High Performance Analytic Appliance“ (Dt. „Hochleistungsanalyseanwendung“) und stellt eine Technologie- und Entwicklungsplattform des Softwareherstellers SAP dar, mit der Unternehmen große Datenmengen performant auswerten können. In der Literatur existieren diverse Definitionen für den Begriff Plattform. Zusammengefasst lässt sich eine Plattform so charakterisieren, dass sie eine einheitliche Basis darstellt, die einzelne Komponenten innerhalb eines gemeinsamen technologischen Rahmens vereint. Auf Grundlage dessen können neue Anwendungen, Prozesse und Technologien entwickelt werden (Prassol 2016: S.196-197).
Herzstück der Plattform ist die SAP-HANA-Datenbank, ein spaltenorientiertes In-Memory-Datenbanksystem (siehe Abschnitt 2.1.2).
Die Architektur von HANA wurde ursprünglich im Jahr 2008 von SAP in Kooperation mit dem Hasso-Plattner-Institut und der Stanford University für die Analysen von großen Daten in Echtzeit entwickelt. Bevor sich der Name „HANA“ etabliert hat, war die Anwendung in verschiedenen Publikationen auch unter den Bezeichnungen „SanssouciDB“ oder „NewDB“ anzutreffen (Plattner 2011).
SAP HANA wurde erstmals 2011 als Datenbank vorgestellt. Seit 2011 ist die In-Memory-Technologie in der SAP HANA-Datenbank am Markt verfügbar. Das Ziel der Technologie ist es analytische Anwendungen, die sich in hoher Geschwindigkeit umsetzen lassen, zu ermöglichen. Da sämtliche Daten im Arbeitsspeicher gehalten

werden, macht es das System sehr leistungsfähig.
Ab 2013 entwickelte sich SAP HANA zur Plattform weiter und handelt sich heute um eine komplette Lösung für den Betrieb und die Entwicklung von SAP-Geschäftsanwendungen. Eine Übersicht der SAP HANA als Big-Data-Anwendungsplattform ist in folgender Abbildung zu sehen.

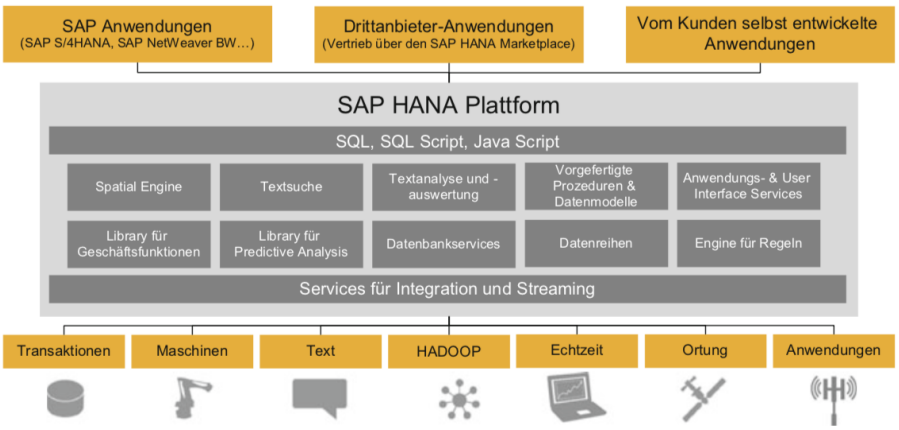

Abbildung 2.1: Architektur der SAP-HANA-Plattform
Quelle: (Prassol 2016: S.199)

In obiger Abbildung wird deutlich, dass die Datenbank nur einen Teil der verfügbaren Dienste darstellt, die mit SAP HANA möglich sein sollen. SAP ist überzeugt, dass eine Plattform außer der Persistenz der Daten auch Business-Logik und -Prozesse verarbeiten können sollte. Dies beginnt bei einer einfachen Währungsumrechnung, geht über Graph- und Spatial-Funktionalitäten bis hin zu einer Bibliothek für Predictive Analytics. Letztere wird im Abschnitt 2.1.3 näher erläutert.

2.1.2 Architektur der SAP HANA-In-Memory-Datenbank

Wie im vorherigen Abschnitt bereits erwähnt, ist der Kern der SAP HANA eine spaltenbasierte relationale In-Memory-Datenbank. Das bedeutet, die Verarbeitung der Daten geschieht im Arbeitsspeicher, dem RAM. Ein laufendes SAP-HANA-System besteht aus mehreren kommunizierenden Prozessen (Services). Im Folgenden werden die wichtigsten SAP-HANA-Datenbankservices in einem klassischen Anwendungskontext dargestellt.
Da der Speichertyp RAM ein flüchtiges Speichermedium ist, werden Daten unter Verlust der Stromzufuhr nicht gespeichert und sind nicht wiederherstellbar. Der „Persistence Layer“ (siehe Abbildung 2.2) ist für diesen speziellen Fall gedacht. Werden Transaktionen angestoßen und Daten verändert, werden die betroffenen Seiten im RAM markiert und es findet ein Schreibvorgang in das nichtflüchtige Speichermedium statt.

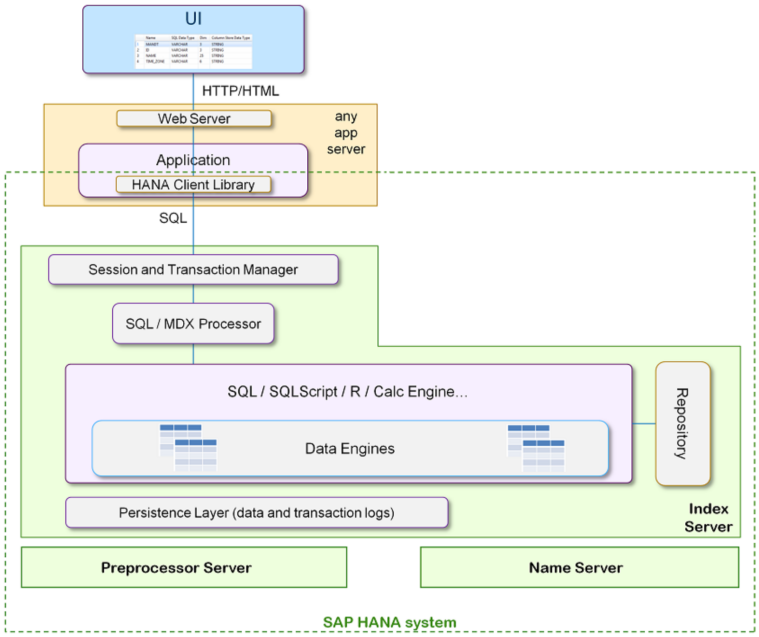

Abbildung 2.2: SAP-HANA-Datenbank-Architektur
Quelle: (SAP SE 2018a: S.11)

Des Weiteren existieren Logdateien, die die Dauerhaftigkeit der Transaktionen sicherstellen und beinhalten alle Änderungen, die an der Datenbank gemacht werden (Berg, Silvia & Frye 2017: S.25).
„Auf Daten im Hauptspeicher (RAM) kann 100.000-mal schneller zugegriffen werden als auf Daten auf einer Festplatte“ (Berg, Silvia & Frye 2017: S.23).
Durch die volle Ausrichtung auf den Arbeitsspeicher lassen sich sehr große Datenmengen wesentlich schneller durchsuchen, aggregieren und analysieren als dies mit klassischen Disk-basierten Datenbanksystemen möglich ist. Diese müssen die Daten bei der Suche immer erst von der Festplatte in den Speicher kopieren und sind dadurch langsamer.
Die Hauptkomponente der SAP-HANA-Datenbankverwaltung ist der Indexserver, der die eigentlichen Datenspeicher und die Engines zur Verarbeitung der Daten enthält. Dieser verarbeitet eingehende SQL- oder MDX-Anweisungen im Kontext authentifizierter Sitzungen und Transaktionen. Der Indexserver verwendet den Präprozessorserver zum Analysieren von Textdaten und zum Extrahieren der Informationen, auf denen die Textsuchfunktionen basieren. Der Nameserver besitzt die Informationen zur Topologie des SAP-HANA-Systems. In einem verteilten System ist auf dem Nameserver hinterlegt, wo die Komponenten ausgeführt werden und welche Daten sich auf welchem Server befinden.
Die SAP-HANA-Datenbank verfügt über eine eigene Skriptsprache namens SQLScript. Sie bettet datenintensive Anwendungslogik in die Datenbank ein. Klassische

Anwendungen tendieren dazu, mit SQL nur sehr eingeschränkte Funktionen in die Datenbank zu verlagern. Dies führt zu einem umfangreichen Kopieren von Daten von und in die Datenbank sowie zu Programmen, die langsam über große Datenschleifen iterieren und sich nur schwer optimieren und parallelisieren lassen. (SAP SE 2018a: S.12).

Spaltenorientierung

SAP HANA organisiert Daten nicht wie in klassischen Ansätzen zeilenbasiert, sondern in Spalten. Durch reduzierte Festplattenzugriffe können Daten mit sehr hoher Geschwindigkeit gelesen werden, was zur beschleunigten Datenbereitstellung beiträgt. Der Grund hierfür ist, dass die Spalten sequentiell gespeichert werden. Die Informationen liegen in einem Block oder in nahen zusammenliegenden Blöcken auf der Festplatte. Bei der Auswertung der Daten muss nur ein Block mit einem Festplattenzugriff geladen werden. Außerdem nutzt die SAP HANA Kompressionsverfahren und kann die Datenverarbeitung noch schneller ausführen. Diese Kompressionsmethoden sind beispielsweise „run-length encoding“, „cluster coding“ und „dictionary coding“. Das spaltenorientierte Modell bringt jedoch auch Nachteile mit sich. Da beim Einfügen von Daten die Indizes neu organisiert werden müssen, sind Änderungen der Daten zeitaufwändiger, jedoch werden diese Nachteile durch interne Verfahren auf technischer Ebene minimiert. (SAP SE 2018b, Berg, Silvia & Frye 2017: S.32-40).

Parallelverarbeitung

Durch das Nutzen der spaltenbasierten Speichermethode ergibt sich für die SAP HANA die Gelegenheit effiziente Parallelverarbeitungen von Daten auszuführen. In Kombination mit den oben erwähnten Kompressionsverfahren werden zusätzlich höhere Geschwindigkeiten erzielt. Operationen wie Joins, Aggregationen und Lesezugriffe können parallel ausgeführt werden. Der spaltenbasierte Speicher ermöglicht auch die parallele Ausführung von Vorgängen mit mehreren Prozessorkernen.
In einem Spaltenspeicher sind die Daten bereits vertikal partitioniert. Dies bedeutet, dass Operationen an verschiedenen Spalten problemlos parallel verarbeitet werden können. Wenn mehrere Spalten durchsucht oder aggregiert werden müssen, kann jede dieser Operationen einem anderen Prozessorkern zugewiesen werden (SAP SE 2018a: S.10).

2.1.3 SAP HANA Predictive Analytics Library

In der SAP HANA können für abgegrenzte Anwendungsgebiete spezielle funktionale Bibliotheken installiert werden. Diese werden, je nach anzuwendender Operation, in den verschiedenen Engines des Index Servers verarbeitet. (SAP SE 2018a: S.14). Die Application Function Library (AFL) von SAP beinhaltet zwei Bibliotheken, die eine

Gruppe von Funktionen mitliefern, welche für abgegrenzte Einsatzzwecke genutzt werden können:

- Die Business Function Library (BFL) ist eine Anwendungsbibliothek, die viele Geschäftsfunktionen im finanziellen Bereich abdeckt.
- Die Predictive Analytics Library (PAL) ist eine Bibliothek für Machine Learning, die als Möglichkeit dient, analytische Algorithmen auf Daten auszuführen. Hierdurch wird es möglich Modelle zu erstellen, zu trainieren und zur Ableitung von Prognosen auf Basis von Eingabedaten zu verwenden.

Die Anwendungsfunktionen der PAL können über das SQL-Skript als Prozedur aufgerufen werden (SAP SE 2018b: S.8). Im Prozeduraufruf müssen als Parameter bestimmte Tabellen angegeben werden. Diese können sich, je nachdem welcher Algorithmus verwendet wurde, unterscheiden. Die Ergebnisse werden in Ausgabetabellen bereitgestellt. In Kapitel 3 wird darauf noch einmal genauer eingegangen. Zur Veranschaulichung folgt ein Überblick über die möglichen Eingabetabellen beim Training:

- Data Table - die Spalten, welche die Trainingsdaten enthalten,
- Control Table - der Parameter, der die Arbeitsweise des Algorithmus bestimmt.

Die Ausgabetabellen dazu können folgendermaßen aussehen:

- Result Significance Table - die Koeffizienten des Modells und ihre Werte,
- Result Fitted Table - die angepassten Werte des Ziels,
- Result Table - die Modellgenauigkeit,
- Result Model - das Modell, das für zukünftige Prognosen verwendet werden kann.

Nach der Ausführung der Prozedur werden die Resultate in Form von physischen Tabellen generiert. PAL enthält klassische und universelle Algorithmen für „Predictive Analytics“ in diversen Data-Mining-Kategorien: Clustering, Klassifizierung, Regression, Assoziation, Zeitreihen, Vorverarbeitung, Statistiken, Analyse sozialer Netzwerke, Empfehlungssystem.

2.2 Machine Learning

Wie bereits im letzten Abschnitt erwähnt arbeitet PAL mit Machine-Learning-Algorithmen. Dieses Thema soll in diesem Abschnitt näher erläutert werden.
Machine Learning (Dt. Maschinelles Lernen) ist ein Teilbereich der Künstlichen Intelligenz (KI) und wird oftmals auch treffend als Mustererkennung bezeichnet, da der Algorithmus Muster aus Daten extrahiert und auf der Grundlage des identifizierten Musters eine Entscheidung trifft (Essinger & Rosen 2011, Cleve & Lämmel 2014: S.14).

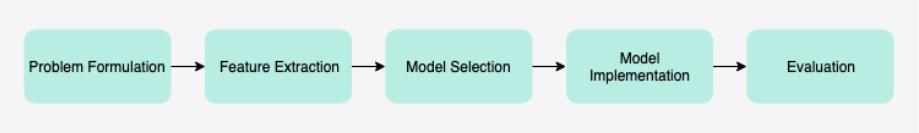

Abbildung 2.3: Grundablauf Machine Learning
Quelle: (nach Essinger & Rosen 2011: S.244)

2.2.1 Algorithmen

Die Lernalgorithmen werden in der Literatur in verschiedene Kategorien eingeteilt. Im Allgemeinen lassen diese sich in „Supervised Learning“ (Dt. überwachtes Lernen) und „Unsupervised Learning“ (Dt. unüberwachtes Lernen) sowie „Reinforcement Learning“ (Dt. bestärkendes Lernen) unterteilen.
Beim Supervised Learning wird dem Algorithmus eine hinreichend große Menge von Ein- und Ausgaben zur Verfügung gestellt, die bereits über das korrekte Ergebnis verfügen. Bei Datensätzen spricht man von „gelabelten“ oder markierten Datensätzen. Ein Beispiel stellt ein großer Datenbestand von Bildern mit Hunden und Katzen dar. Für jedes wurde die Information hinterlegt, ob auf dem Bild ein Hund oder eine Katze abgebildet ist. Diese Information wird genutzt, um den Algorithmus zu trainieren, auf Bildern Hunde und Katzen zu erkennen. Bei neuen Bildern, zu denen diese Information nicht vorliegt, muss der Algorithmus diese selbstständig einordnen (Frochte 2019: S.20).

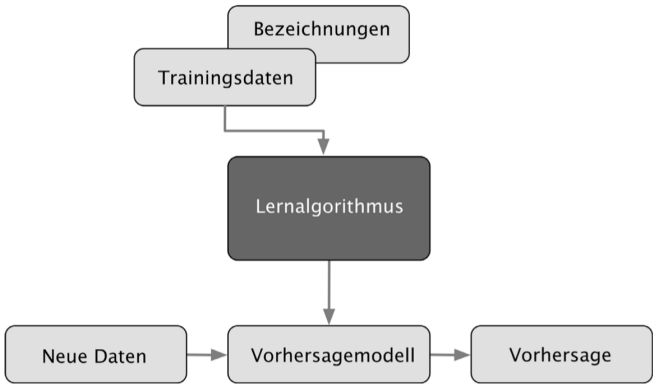

Abbildung 2.4: Supervised Learning
Quelle: (Raschka 2017: S.27)

Wenn der Wahrheitswert nicht bekannt ist, kann man die Daten nicht entsprechend labeln. Für solche Problemstellen sind Techniken aus dem Bereich des Reinforcement Learnings eine Lösungsmöglichkeit. Hierbei muss ein Agent selbständig, auf Basis eines „Trial-and-Error-Verfahrens“, eine Strategie erlernen, um erhaltene Belohnungen zu maximieren. Dadurch entstehen Folgen von Aktionen und Strategien, die zu einer Lösung der Problemstellung führen. Diese Methode ist dem menschlichen Lernen

sehr ähnlich. Hierzu werden oft Techniken aus dem Bereich des Supervised Learnings eingesetzt (Frochte 2019: S.23).Beispielsweise wird ein Schachcomputer mittels Reinforcement Learning trainiert. Jedem Zustand kann eine positive (oder negative) Bewertung zugeordnet werden, und die Belohnung kann dadurch definiert werden, dass ein Gesamtziel erreicht wird, wie z.B. das Gewinnen oder das Verlieren einer Schachpartie. (Raschka 2017: S.30).

Während beim Supervised Learning dem Algorithmus eine Sammlung von Zielwerten zur Verfügung gestellt werden und beim Reinforcement Learning immerhin noch die Ergebnisse eines Verhaltens positiv bzw. negativ bewerten können, gibt es Fälle, in denen beides nicht möglich ist. Dort existiert nur eine Menge an Daten und es wird mittels Unsupervised Learning versucht, versteckte Strukturen in unmarkierten Daten zu finden (Frochte 2019: S.24).

In einigen Quellen findet sich auch eine vierte Kategorie, dem „Semi-supervised Learning“ (Dt. Teilüberwachtes Lernen). Dies ist eine Technik, bei denen ein Muster auf der Grundlage von teilweise gekennzeichneten Trainingsdaten abgeleitet wird. Es kombiniert Elemente aus Supervised- und Unsupervised Learning. Ziel des Semi-supervised Learnings ist es, die markierten und nicht markierten Daten zu nutzen, um bessere Lernmodelle zu erhalten (Grus 2016: S.150).

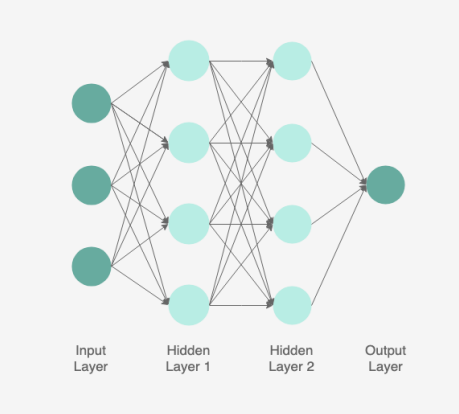

Abbildung 2.5: Künstliches neuronales Netz
Quelle: eigene Darstellung

Deep Learning (Dt. tiefgehendes Lernen) ist eine Disziplin des Machine Learning, das „künstliche neuronale Netze“ (siehe Abbildung 2.5) einsetzt. Der Aufbau eines solchen Netzes lehnt sich strukturell an biologische neuronale Netze aus der Natur an. Die erste Schicht (Input Layer) des Netzes wird durch einen Eingabe-Vektor gebildet. Daran schließen zahlreiche Zwischenschichten (Hidden Layer) an, welche aus sogenannten Neuronen bestehen. Diese verarbeiten eingehende Informationen und leiten

sie entsprechend der Gewichtungen an die darauffolgende Schicht weiter. Dadurch entsteht eine umfangreiche innere Struktur. Dieser Prozess findet solange statt, bis eine sogenannte Ausgabeschicht (Output Layer) erreicht und ein Ausgabe-Vektor erzeugt wird. Dieser stellt im Grunde eine Art Ergebnis-Schlüssel dar, der beispielsweise eine bestimmte Klasse ausweist: z.B. Katze oder Hund. Durch das Training werden die Gewichtungen zwischen den Neuronen angepasst und verändern sich währenddessen so, dass ein bestimmtes Eingabemuster (z.B. Bilder von Haustieren) immer zu einem bestimmten Ausgabemuster (z.B. „Das Foto zeigt eine Katze“) führt. Der Vorteil dessen ist die tiefgehende Abstraktion von Zusammenhängen zwischen Eingabe-Daten, den abstrahierten Neuronen-Werten und Ausgabe-Daten. Daraus leitet sich auch der Name „Deep Learning“ ab und kommt hauptsächlich dann zum Einsatz, wenn andere Machine Learning-Verfahren an Grenzen stoßen. (Rashid 2017, Frochte 2019: S.161-172).

Es existieren bisweilen sehr viele Machine Learning Algorithmen, daher werden in nachstehender Tabelle nur einige bekannte davon aufgeführt und einer Kategorie sowie einem möglichen Problemtyp zugeordnet:

Einordnung von Machine Learning-Algorithmen

Kategorie	Problemtyp	Ziel	Algorithmus	PAL
Supervised Learning	Regression	Beziehung zwischen abhängigen (Ziel) und unabhängigen Variablen (Prädiktor)	Linear	x
			Decision Trees	x
			Bayesian Networks	
			Fuzzy Classification	
			Neural Network	x
	Classification	Neue Daten klassifizieren lernen	Logistic Regression	x
			SVM	x
			Random Forest	x
			Classification Trees	x
			Neural Network	x
Unsupervised Learning	Clustering	Aufteilung der Daten in Cluster, die bestimmte Kriterien erfüllen	K-means	x
			Hierarchical Clustering	x
			Gaussian Mixture Models	x
			Genetic Algorithms	
			Neural Network	x
	Dimension Reduction	Projektion der Daten auf weniger Dimensionen, zur Erfassung fundamentaler Aspekte	PCA	x
			Tensor decomposition	
			Multidimensional statistics	x
			Random Projection	x
			Neural Network	x

Tabelle 2.1: Tabellarische Einordnung diverser bekannter Algorithmen und der Verfügbarkeit in PAL: (auf Basis von (Louridas & Ebert 2016: S.113) und (SAP SE 2018c))

Man kann auch Techniken aus der Regression dazu verwenden um Probleme der

Klassifikation zu lösen. Dies ist darin begründet, dass es sowohl bei der Regression als auch bei der Klassifikation darum geht, eine (mathematische) Funktion f(x) aus Beispielen zu lernen. Daher soll die obige Tabelle nicht als unbedingt geltende Zuordnung zwischen Algorithmus und Problemtyp angesehen werden, sondern nur als mögliche Zuordnung. In der Tabelle soll ebenfalls deutlich gemacht werden, dass man neuronale Netze sowohl im Supervised Learning, im Unsupervised Learning als auch im Reinforcement Learning einsetzen kann. Die Spalte „PAL“ wurde hinzugefügt, um einen Überblick zu schaffen, welche Algorithmen in der Bibliothek PAL zur Anwendung zur Verfügung stehen.

2.2.2 Begriffe im Kontext des Modellierens

Wenn man mit einer Machine Learning-Plattform wie zum Beispiel SAP HANA PAL arbeitet, wird man unter anderem schnell dem gerade erwähnten Begriff „Modell“ begegnen.
Beispielsweise geht es in der Theorie beim Klassifizierungsproblem als auch beim Regressionsproblem darum, die hypothetisch existierende „perfekte Funktion“ zu finden bzw. zu konstruieren. Rein praktisch ist das nicht möglich, weil entweder zu wenige Daten, ein zu schlechtes Modell und/oder die Daten eine zu schlechte Qualität vorliegen. Daher sollte man sich den Lernprozess eher als eine Funktionsannäherung bzw. Funktionsapproximation vorstellen, also etwas, was kontinuierlich verbessert werden kann. Dieser soeben erwähnten Lernprozess wird im Machine Learning „Training“ genannt. Die Spezifikation eines mathematischen Zusammenhangs, der zwischen den Variablen besteht, nennt man „Modell“ .
Eine häufige Gefahr beim Machine Learning ist die Überanpassung, auch Overfitting genannt. Also ein Modell zu bilden, das auf den Trainingsdaten gut funktioniert, aber schlecht auf jeglichen neuen Daten verallgemeinert. Eine Ursache dafür kann sein, dass das „Rauschen“ in den Daten „auswendig“ gelernt wird. Das Gegenteil dazu ist Underfitting, also ein Modell, das nicht einmal auf den Trainingsdaten gut funktioniert. Meistens hat man dann einen ungeeigneten Algorithmus für das Problem gewählt oder die Trainingsdaten sind einfach nicht gut genug oder zu wenig.
Wenn ein Modell trainiert wird, wird dabei die Treffergenauigkeit (Accuracy) ermittelt. Also der Anteil der beim Training richtig erkannten Datenpunkte. Dies bestimmt demnach den „Score“ des Modells, dieser gibt an wie gut ein Modell für Vorhersagen auf Datensätzen geeignet wäre (Grus 2016: S.149-153).

Um Probleme im Kontext der KI lösen zu können bietet sich die funktionale Programmierung besonders gut an. Hierbei ist eine „datennahe“ Programmierung ein vorteilhafter Aspekt. Die Eigenschaften und Vorteile dieses Paradigmas werden im nächsten Abschnitt behandelt.

2.3 Clojure als funktionale Programmiersprache

Da die Software Open60 in der Programmiersprache Clojure entwickelt wird und diese Sprache im Rahmen dieser Arbeit ebenfalls Anwendung findet, soll nun in diesem Abschnitt näher auf deren fundamentalen Aspekte eingegangen werden. Der Fokus wird daher zunächst auf die funktionale Programmierung gelegt, da Clojure den funktionalen Stil fördert.

2.3.1 Funktionale Programmierung

Die funktionale Programmierung hat ihren Ursprung im Lambda-Kalkül, einem formalen System, das in den 1930er Jahren entwickelt wurde, um die Berechenbarkeit, das Entscheidungsproblem, die Funktionsdefinition, die Funktionsanwendung und die Rekursion zu untersuchen. Viele funktionale Programmiersprachen können als Ausarbeitungen des Lambda-Kalkül angesehen werden (Hudak 1989: S.359-411).

In der Informatik ist die funktionale Programmierung ein Programmierparadigma, ein Stil zum Aufbau der Struktur und der Elemente von Computerprogrammen. Das primäre Ziel funktionaler Sprachen ist die Minimierung von unerwartetem Programmverhalten. Dies wird durch die Reduktion von Seiteneffekten welche den Zustand einer Anwendung beeinflussen erreicht.
Die grundlegenden Konzepte, die diesen Programmierstil und somit auch die Sprache Clojure auszeichnen, sind:

1. Referenzielle Transparenz,
2. Reine Funktionen,
3. Rekursion,
4. Funktionen sind „First-Class“ und können „High-Order“ sein,
5. Variablen sind unveränderlich.

„Reine Funktionen“ erfüllen zwei Qualifikationen:

1. Sie sind referenziell transparent.
2. Sie können keine Seiteneffekte hervorrufen.

Als „referenzielle Transparenz“ bezeichnet man, wenn eine Funktion bei der Eingabe der gleichen Argumente immer das gleiche Ergebnis zurückgibt. Dafür stützen sich reine Funktionen nur auf ihre eigenen Argumente und auf unveränderliche Werte, um ihren Rückgabewert zu bestimmen. Mathematische Funktionen sind beispielsweise referenziell transparent.

Eine „Reine Funktion“ kann keine „Seiteneffekte“ verursachen. Als Seiteneffekte werden in der Literatur beispielsweise Änderungen, wie die sich verändernde Zuordnung zwischen einem Namen und seinem Wert innerhalb eines bestimmten Programmteils gemeint. Diese Seiteneffekte können dazu führen, dass nicht mehr nachvollziehbar ist warum und wie ein Name mit einem Wert in Verbindung gebracht wurde, was die Fehlerbehandlung des Programms erschwert. Wenn eine Funktion aufgerufen wird, die keine Seiteneffekte hat, muss lediglich die Beziehung zwischen der Eingabe und der Ausgabe berücksichtigt werden. Sorgen über mögliche Änderungen, die sich auf das System auswirken könnten entfallen. Die Kerndatenstrukturen in der funktionalen Programmierung sind also „immutable“ (Dt. unveränderlich).
In Clojure wird beispielsweise die Rekursion anstelle von Schleifen (die von Seiteneffekten behaftet sind) verwendet. Rekursive Funktionen rufen sich immer wieder selbst auf bis ihre Abbruchbedingung erfüllt ist.

Funktionen sind „First-Class“ und können „High-Order“ sein.
Die Funktion stellt in funktionalen Sprachen das wichtigste Konstrukt dar. Im Gegensatz zu anderen Sprachparadigmen ist die Funktion anderen Datentypen gleichgestellt und kann somit in Variablen gespeichert und als Ein- sowie Rückgabeparameter verwendet werden. Diese Eigenschaft wird als First-Class-Sprachelement bezeichnet.
In Clojure wird die Special Form „fn“ genutzt, um eine Funktion zu erzeugen, sowie „defn“ um eine erzeugte Funktion einer Variablen im aktuellen Namespace zuzuordnen. Funktionen, welche Funktionen produzieren oder konsumieren werden als High-Order-Function bzw. höherrangige Funktionen bezeichnet. Die wichtigsten vordefinierten High-Order-Functions in Clojure sind „map“, „reduce“, „apply“, „partial“ und „comp“.

Dass Variablen „unveränderlich“ sind bedeutet, dass es in der funktionalen Programmierung nicht möglich ist, eine Variable nach ihrer Initialisierung zu ändern. Man kann neue Variablen erstellen, aber vorhandene Variablen können nicht geändert werden. Dies soll dabei helfen, den Zustand während der Laufzeit eines Programms aufrechtzuerhalten.

2.3.2 Eigenschaften von Clojure und LISP

Clojure hat sich aus der Sprache LISP heraus als moderner Dialekt entwickelt. Sie fördert den funktionalen Stil (siehe Abschnitt 2.3.1) und wird auf der Java Virtual Machine ausgeführt, der Laufzeitumgebung, in der auch die Programmiersprache Java ausgeführt wird. Clojure Code wird nach JVM-Bytecode kompiliert, bietet jedoch alle Features auch zur Laufzeit an und bleibt damit vollständig dynamisch.
LISP ist eine der ersten Programmiersprachen überhaupt, die auch heute noch Verwendung findet. Clojure gehört dabei genau wie der heute noch verwendete Dialekt „Scheme“ zur Familie der „lisp-1“ Dialekte. Der Name LISP ist ein Akronym für

„**LIS**t **P**rocessing“ und hat seinen Ursprung daher, dass der gesamte Code aus Listen besteht. Funktionsaufrufe verwenden das erste Element einer Liste als Funktionsnamen und die restlichen Elemente als deren Argumente. Da LISP seine eigenen Datenstrukturen verwendet um Programme auszudrücken lautet eine wichtige Strategie der Sprache „Daten als Code“ (code-as-data) (Tate 2010: S.238).

Clojure schlägt verglichen mit anderen Dialekten eine neue Richtung ein. Diese äußert sich unter anderem in der Erweiterung der verfügbaren Datenstrukturen um Vektoren und Maps und der dazugehörigen Einführung von zusätzlichen Klammer-Typen, die die Lesbarkeit erhöhen sollen, sowie der Einführung der standardmäßigen Unveränderlichkeit der Datenstrukturen (ClojureOrg 2019). Die Vielseitigkeit von Clojure wird noch durch einen existierenden Clojure-nach-JavaScript-Compiler namens ClojureScript, der die Ausführung in JavaScript-Umgebungen erlaubt. Dies ermöglicht es Clojure nicht nur Serverseitig sondern auch Clientseitig einzusetzen.

Clojure-Spezifische-Sprachmerkmale:

Beispielsweise durch „Makros“, „Protokolle“ und „Multimethoden“ hebt sich Clojure von anderen funktionalen Programmiersprachen ab.
Die Sprache besitzt nur wenige eingebaute Operatoren, welche als „Special Forms“ bezeichnet werden. Die gesamte weitere Funktionalität wird durch Funktionen und Makros gebildet. Mit Hilfe von Makros ist der Entwickler in der Lage neue Operationen zu implementieren, welche nicht im Sprachumfang enthalten sind.

Protokolle sind für Clojure, was Interfaces für Java oder C# repräsentieren: Container für Methodensignaturen, welche eine Schnittstelle kennzeichnen. Ein Protokoll wird mit Hilfe der Form „defprotocol“ definiert:

```
1 (defprotocol ProtocolName
2   "documentation"
3   (a-method [this arg1 arg2] "method docstring")
4   (another-method [x] [x arg] "docstring"))
```

Listing 2.1: Clojure Protocol
Quelle: (Emerick 2012: S.264)

Multimethoden bieten die Möglichkeit „Laufzeitpolymorphismus“ zu implementieren. Eine Multimethode ist eine Methode, die einen Aufruf an eine bestimmte Funktions-Implementierung auf der Grundlage einer Eigenschaft ihrer Argumente sendet. Eine Dispatching-Funktion leitet den Aufruf an die korrekte Funktionsimplementierung weiter. Mit anderen Worten: Sie wertet seine Argumente aus und erzeugt einen Dispatching-Wert, der zum Auffinden der tatsächlichen Implementierung (defmethod) verwendet wird. Die letzte „defmethod“ ist die Standardimplementierung, die aufgerufen wird, wenn keine der anderen „defmethods“ mit dem Dispatching-Wert übereinstimmt. (Higginbotham 2015).

Aufgrund dieser Eigenschaften eignet sich LISP bzw. Clojure hervorragend für den Einsatz im Kontext der KI und hat sich auch aus diesem Kontext heraus entwickelt. Denn „Ihre Syntax und Semantik unterstützen somit eine präzise und zweckmäßige Abstraktion, Formulierung, Behandlung und Lösung typischer Probleme und Anwendungen in der KI“ (Hofstedt 2012).

Zusammengefasst lassen sich die Vorteile von Clojure so definieren: Man nutzt die Vorteile einer schnellen und dynamischen, sowie daten-orientierten Sprache und muss nicht auf Robustheit, Performance und Skalierung verzichten. Das beste daran aber ist die Einfachheit und Datenorientierung der Sprache die es dadurch ermöglichst mit weniger Code das Ziel zu erreichen. Daher wird die Sprache in der Software Open60, einem hochdynamischen System, eingesetzt und profitiert somit von diesen Vorteilen.

2.4 Adapter als Softwarearchitektur und Design Pattern

In dieser Arbeit wird ein Konzept für einen Daten-Adapter entwickelt. Dieser Abschnitt soll daher der Begriff Adapter zuerst in Zusammenhang mit der Software-Architektur erläutern und anschließend in Zusammenhang mit der Sprache Clojure und der Verwendung im Kontext dieser Arbeit gebracht werden.

2.4.1 Design Patterns

Architekturen von Softwaresystemen sollen einfach erweiterbar und weitestgehend standardisiert sein, damit die Entwickler sich leicht über Architekturen austauschen können. Für den objektorientierten Entwurf haben sich zahlreiche wertvolle Architektur- und Entwurfsmuster herausgebildet. Diese Muster basieren auf objektorientierten Prinzipien (Goll 2018).
Der Grund für den Einsatz dieser „Design Patterns[1]“ (Dt. Entwurfsmuster) ist das Lösen bekannter wiederkehrender Entwurfsprobleme, die Design- und Architekturwissen in kompakter und wiederverwertbarer Form zusammenfassen. Sie können die Flexibilität, Verständlichkeit und die Performanz vorhandener Entwürfe steigern. In komplexen Software-Projekten kann der angemessene Einsatz von Mustern das Risiko von Entwurfsfehlern deutlich senken.

[1] Das bekannteste Buch hierzu, mit dem Titel „Design Patterns – Elements of Reusable Object-Oriented Software“, beinhaltet 23 detailliert beschriebene Entwurfsmuster. Die vier Autoren sind unter Entwicklern auch unter ihrem Spitznamen „Gang of Four“ (Dt. Viererbande, kurz GoF) bekannt und verhalfen mit ihrem Buch den Entwurfsmustern zu ihrem Durchbruch.

2.4.2 Adapter-Pattern

Eines dieser Design Patterns ist der sogenannte Adapter. Er passt die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstelle an. Das Adaptermuster lässt Klassen zusammenarbeiten, die ansonsten dazu nicht in der Lage wären (GoF 1995).

Der Adapter zählt zu den „Structural Patterns“ (Dt. Strukturmuster) und findet außerdem Anwendung, wenn folgende Probleme bestehen:

- Man möchte eine Komponente verwenden, deren Schnittstelle nicht mit der von Ihnen benötigten Schnittstelle übereinstimmt.
- Man möchte Klassen zusammenarbeiten lassen, die inkompatible Schnittstellen besitzen.
- Eine Komponente besitzt die richtigen Daten und das richtige Verhalten, bietet aber eine unpassende Schnittstelle an.

(Eilebrecht 2019: S.77)

Die Lösung für dieses Problem ist die Definition eines Adapters, der das benötigte Interface implementiert und Anfragen an die vorhandene Komponente weiterleitet (siehe Abb. 2.6). Die vorhandene Klasse (Vorhandene Komponente, Adaptee) wird durch den Adapter an die benötigte Schnittstelle angepasst (adaptiert).
Folgende Abbildung zeigt ein abstraktes Beispiel zur Anwendung des Patterns:

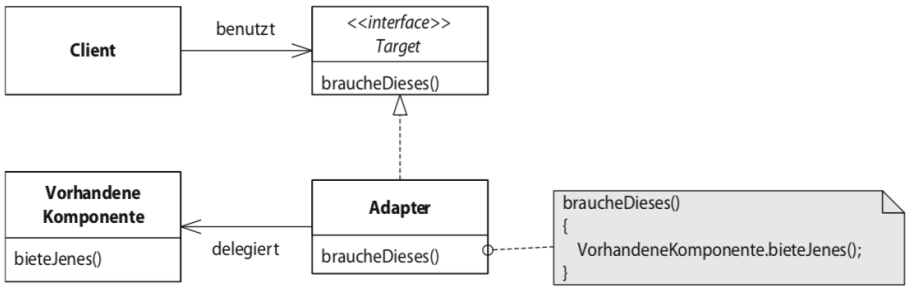

Abbildung 2.6: Adapter Pattern
Quelle: (Eilebrecht 2019: S.78)

2.4.3 Adapter-Pattern in Clojure

In Clojure werden, wie in Kapitel 2.3.1 schon erwähnt, Interfaces durch „Protocols“ definiert. Emerick (2012) beschreibt, dass durch Protocols das gerade beschriebene Problem gelöst werden soll:
„Adapter, Wrapper, Delegate. Made necessary by inflexible type hierarchies elsewhere, these are made unnecessary by protocols, which allow you orthogonally define new behavior for existing types without appeals to inheritance, adaptation, or wrapper“ (Emerick 2012: S.458).
Daraus lässt sich schließen, dass Protocols die Möglichkeit bieten eine Schnittstelle ohne starre Hierarchie zu nutzen. Da diese ein „dynamisches Konstrukt“ darstellen, werden nur deren Spezifikationen (die festlegen, welche Funktionalität ein Modul anbietet) befolgt, jedoch nicht deren Implementierung. Daraus ergibt sich der Vorteil Module austauschen zu können, welche die Schnittstelle verwenden. Mit Protocols wird es ebenso ermöglicht neue Funktionen in vorhandene Typen hinzuzufügen ohne den ursprünglichen Code zu ändern, was für die Programmierung sehr vorteilhaft ist. Dies ist möglich, indem man das Macro „extend-protocol“ verwendet. Bei der Umsetzung in Kapitel 4 werden Protocols für die Adapter-Schnittstelle Anwendung finden.

2.4.4 Adapter im Kontext von CAMINO

Im Kontext CAMINO wird der Begriff „Daten-Adapter“ eingeführt. Dieser soll so verstanden werden, dass durch den Einsatz eines speziell zwischen der KI-Vertikalen (siehe Abschnitt 2.5) in der Software Open60 und einem externen Machine Learning Service ein reibungsloser Datenaustausch stattfinden kann. In Kapitel 3 wird daher ein Konzept entwickelt, welches auf dieses spezielle Problem eingeht und ein Lösungsweg dafür erarbeitet. Das Konzept sieht daher vor, eine passende Schnittstelle und die dazu nötigen internen Aspekte zu definieren. Dieses soll im späteren Verlauf der Arbeit exemplarisch am Beispiel des Machine Learning Services SAP HANA PAL umgesetzt werden, welcher im Abschnitt 2.1.3 vorgestellt wurde.

2.5 Vertikale Softwarearchitektur

Open60 stellt eine Anwendung zur Datenanalyse dar, mit dem Ziel große Datenmengen dynamisch zu verarbeiten. Dabei spielt Skalierbarkeit eine große Rolle. Gemeint ist hier Skalierung im technischen Sinne, d.h. zur Erfüllung der Last- und Performance -Anforderungen des Systems.
Die Softwarearchitektur von Open60 basiert daher auf der Idee das System in mehrere Teilsysteme einzuteilen, die „Vertikalen“ genannt werden.
So ist dessen Makroarchitektur nach dem Prinzip „Divide & Conquer“ gestaltet: „*Teile ein großes Problem in viele kleine Probleme, die sich beherrschen lassen*“(Kraus, Steinacker & Wegner 2013).

Dieses Prinzip wird durch die Einteilung in Vertikalen auf die gesamte Architektur angewandt. Vertikalen sind durch folgende Eigenschaften gekennzeichnet:

- Separat deploybare Einheit,
- Eigenes Frontend,
- Eigene Datenhaltung,
- Klar definierte Zuständigkeit für Daten und Feature,
- Eigene Code-Basis, d.h. kein geteilter Code zwischen den Vertikalen,
- Kein geteilter Zustand zwischen den Vertikalen,
- Kommunikation über definierte Schnittstellen.

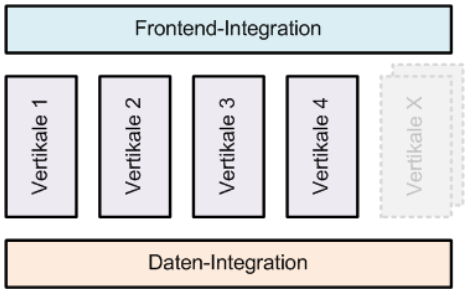

Abbildung 2.7: Vertikale Softwarearchitektur
Quelle: eigene Darstellung

Als Ergebnis erhält man kleine, lose gekoppelte Systeme, die weitgehend unabhängig voneinander lauffähig sind (Kraus, Steinacker & Wegner 2013). Denn das Ziel ist es, eine Architektur zu entwerfen, die es ermöglicht, auf sich ändernde Anforderungen ohne größere Umstrukturierungsmaßnahmen zu reagieren und deren Komplexität so gering wie möglich ist (Tavares de Sousa, Hasselbring, Weber & Kranzlmüller 2018). Somit repräsentieren Vertikalen eine übergeordnete Aufteilung von Microservices. Diese Microservices sind dann innerhalb der Vertikalen selbst zu finden und stellen die sogenannte Mikro-Architektur dar. Sie laufen jeweils in einem Prozess, kommunizieren über einfache Mechanismen, kümmern sich jeweils unabhängig voneinander um eigene kleine Teilaufgaben und sind nach außen hin isoliert (Fowler & Lewis 2015). In Open60 stellt die KI-Vertikale eine dieser Vertikalen dar. Die Abkürzung KI steht dabei für Künstliche Intelligenz. Dort sind alle Funktionalitäten der KI-Services eingebettet, deswegen befindet sich der Schwerpunkt dieser Arbeit in Zusammenhang mit dieser besagten Vertikalen.

2.6 Zusammenfassung

In diesem Kapitel wurden alle notwendigen Grundlagen für die Entwicklung des Konzepts CAMINO behandelt.
Zunächst wurde auf die für das Konzept verwendete Datenbank und deren zugehörigen Bibliotheken eingegangen. Mit SAP HANA lassen sich analytische Anwendungen, die auf eine schnelle Datenbank angewiesen sind ermöglichen. Die Haltung sämtlicher Daten im Arbeitsspeicher und die Organisation der Tabellen in Spalten, machen das System sehr leistungsfähig. Daraus lässt sich schließen, dass SAP HANA gut für Big Data-Anwendungen geeignet ist. Die Bibliothek PAL umfasst dazu viele nutzbare Machine Learning-Algorithmen, die für die Umsetzung von großem Nutzen sein werden. Diese lassen sich mittels Prozeduren aufrufen und berechnen dann aus den angegebenen Daten die Vorhersagen von Ereignissen.
Im Abschnitt 2.2 wurden die grundlegenden Machine Learning-Algorithmen dargestellt und Begriffe genauer erklärt, die für das weitere Bearbeiten des Themas notwendig sind.
Der darauffolgende Abschnitt 2.3 behandelte die Programmiersprache Clojure, mit welcher die Umsetzung des in Kapitel 3 entwickelten Konzepts erfolgt. Clojure unterstützt das Paradigma der funktionalen Programmierung, welches unerwartetes Programmverhalten verhindern soll. Dies wird durch die Reduktion von Seiteneffekten, welche den Zustand einer Anwendung beeinflussen erreicht. Eine problemnahe Ausdrucksweise und ein erheblich kürzerer Programmcode gelten als Vorzüge funktionalen Programmierens. Insbesondere in mathematischen oder technischen Zusammenhängen, wie beispielsweise in der KI, ist dies ein wünschenswerter Aspekt. Diese Eigenschaften vereinfachen wiederum die Wartung und reduzieren die Fehleranfälligkeit.
Im Abschnitt 2.4 wurde das der CAMINO-Architektur zugrundeliegende Software-Architekturpattern Adapter vorgestellt. Mithilfe der in Clojure verfügbaren Protocols, welche Schnittstellen definieren, soll das Problem der inkompatiblen Schnittstellen gelöst werden. Danach wurde nochmals verdeutlicht, dass der Begriff Daten-Adpater so zu verstehen ist, dass zwischen der SAP HANA PAL und Open60 ein durchgängig kompatibler, reibungsloser Austausch von Daten möglich werden soll.
Abschließend wurde im Abschnitt 2.5 auf das Prinzip der Vertikalen Softwarearchitektur eingegangen. Die Software Open60 orientiert sich an diesem Architekturprinzip, welches ein hohes Maß an Flexibilität mit sich bringt. Dies wird durch die Skalierbarkeit des Systems, aufgrund der Einteilung in mehrere weitgehend unabhängige Teilsysteme (Vertikalen), begünstigt. Bei der Verarbeitung großer Datenmengen sind die Erfüllung von Last- und Performance-Anforderungen von hoher Wichtigkeit. In dieses Paradigma bettet sich das Konzept CAMINO ein, welches im nächsten Kapitel behandelt wird.

3 CAMINO Konzept

Dieses Kapitel behandelt das Konzept CAMINO, ein Daten-Adapter zur Kombination der KI-Vertikalen mit Machine Learning Interfaces in der Software Open60.
Dieses Konzept soll am Beispiel des Machine Learning Services SAP HANA PAL entwickelt werden. Der Nutzer soll durch die Ermöglichung der Verwendung von Machine Learning-Techniken in Open60 den Mehrwert haben, schnell und zuverlässig Vorhersagen aus Daten zu generieren. Die Ergebnisse werden visualisiert und ermöglichen dem Nutzer damit zu interagieren und Schlüsse daraus zu ziehen.

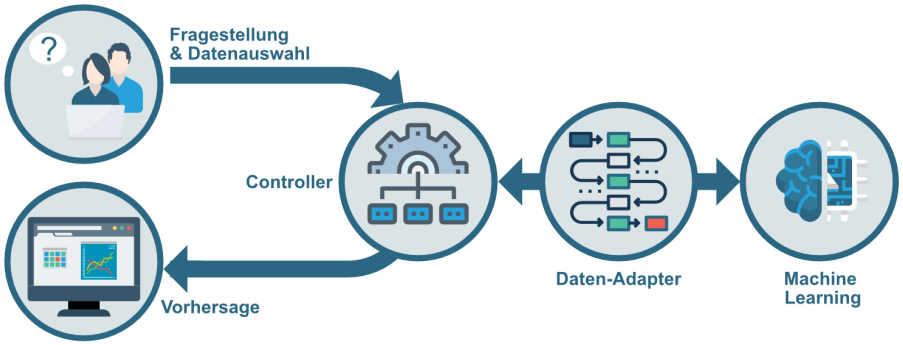

Abbildung 3.1: Kontext für CAMINO
Quelle: eigene Darstellung

Dies erfordert eine entsprechende Kompatibilität zwischen beteiligten Services und soll durch den Einsatz eines speziell zwischen dem Machine Learning-Service und dem Controller der KI-Vertikalen eingebauten Daten-Adapters erreicht werden. Dessen Schnittstellen folgen dem Architekturprinzip „Design By Contract“ und die zugrundeliegende Architektur ist an das Entwurfsmuster „Adapter“ angelehnt (siehe Abschnitt 2.4 der Grundlagen). Abbildung 3.1 dient zur groben Darstellung des Kontexts, in den sich CAMINO einbettet.
Zunächst wird in Abschnitt 3.1.1 ein Anwendungsfall erläutert und anschließend die daraus resultierenden funktionalen Anforderungen an den Adapter CAMINO ermittelt. In Abschnitt 3.2 soll nun detaillierter auf die Einbettung in den übergeordneten Kontext eingegangen werden. Dabei werden zuerst die Schnittstellen der KI-Vertikalen analysiert 3.2.1 und danach speziell der Machine Learning Kontext zu SAP HANA PAL in Abschnitt 3.2.3 dargestellt. Daraufhin folgt die Entwicklung des Adapterkonzepts in Abschnitt 3.3 als Fokus dieser Arbeit. Dabei wird zuerst auf die Daten und Semantikaspekte eingegangen, anschließend die Struktur und das Verhal-

ten des Adapters sowie dessen Architektur. Abgeschlossen wird das Konzept mit einer Zusammenfassung in Abschnitt 3.4.

3.1 Anforderungen

In diesem Abschnitt werden die Anforderungen an die Software und somit auch an den zu entwickelnden Softwareteil CAMINO festgelegt. Dazu wird in einem allgemeinen Anwendungsfall zunächst die Funktionsweise des Systems Open60 erläutert und daran auch die Aufgaben von CAMINO abgeleitet.

3.1.1 Anwendungsfall

Das gesamte System Open60 wird von sogenannten „Citizen Data Scientists“ genutzt. Dies können beispielsweise Fachanwender einer bestimmten Fachabteilung, z.B. für Krisenfrüherkennung, sein oder auch Geschäftskunden, die lernbereit und neugierig darauf sind, ihre Daten mittels Algorithmen für ihre Geschäftsprozesse zu erkunden. Ein idealer Nutzer stammt aus einer Fachabteilung und bringt viel Business-know-how und ein Grundverständnis im Umgang mit Daten mit, muss aber nicht über das gleiche tiefe mathematisch-statistische Wissen verfügen.
Das Ziel der Nutzer ist die Datenanalyse ohne Fachwissen zu Data Mining, Algorithmen und Machine Learning durchführen zu können. Es wird eine bestimmte Problematik oder Fragestellung verfolgt, der man mithilfe von Analysen nachgehen möchte. Eine Fragestellung kann beispielsweise die Entwicklung einer Krise innerhalb eines Landes sein. Unter Einbezug der Machine Learning-Funktionalitäten in der Software können zukünftige Entwicklungen durch Vorhersagen einschätzbar werden.
Das System bezieht seine Daten aus einer Datenbank (aktuell SAP HANA). Da die Daten im gesamten System kompatibel und mit mehreren Datenquellen vergleichbar sein sollen, werden diese in ein Datenmodell überführt und über XML importiert.
Es ist zudem möglich, dass externe Systeme wie zum Beispiel Machine Learning-Services mit dem System interagieren können. Ein solches externes System ist in diesem Fall SAP HANA PAL, eine Bibliothek die Machine Learning-Services bereitstellt, die von Open60 genutzt werden können.
Um den Service zu nutzen, muss der Fachanwender auf der Benutzeroberfläche zunächst die Daten über die Suche auswählen und laden. Die ausgewählten Daten können mittels Drag and Drop des Suchfensters auf das KI-Fenster an die KI-Vertikale übertragen werden. Dort kann der Nutzer nun die Parameter „Problemtyp, Modell, Ziel-Attribut und Zeitraum“ auswählen, und den Button „Vorhersage erstellen“ betätigen. Die Daten aus der Suche und die ausgewählten Machine Learning-Parameter werden an den Server der KI-Vertikale gesendet, um die Vorhersage durchführen zu können. Die Schnittstelle gibt die Daten direkt an den zuständigen Plattform-Adapter weiter. Dort werden alle Bedingungen erfüllt, um die Plattform nutzen zu können.[1]

[1] Im Adapter findet das Training und die Vorhersage mit den übermittelten Daten statt

Die daraus resultierenden Ergebnisse werden dem System danach direkt zur Verfügung gestellt. Der Nutzer kann diese nun auf der Oberfläche visualisieren lassen.
Wenn Probleme bei der Nutzung, wie zum Beispiel die Eingabe ungeeigneter Daten, ein nicht unterstützter Problemtyp oder ein nicht geeignetes Modell auftreten, soll der Nutzer gewarnt werden.
Bei Abbruch des Vorgangs sollte die entsprechende Hintergrundaufgabe umgehend beendet werden und das System direkt wieder für neue Anfragen bereitstehen.
Es ist zukünftig denkbar, dass sogar mehrere Plattformen parallel diese Machine Learning Operationen ausführen, um das Ergebnis vergleichen zu können und das Beste daraus zu ermitteln.

3.1.2 Funktionale Anforderungen

Der Daten-Adapter, welcher die Kompatibilität zwischen den beiden Systemteilen KI-Vertikale und Machine Learning-Service sicherstellen soll, spielt eine zentrale Rolle innerhalb dieses Systemkontexts. Dieser ist in Abbildung 3.2 dargestellt. Der Machine Learning-Controller, innerhalb der KI-Vertikalen, erhält die Daten und die Machine Learning-Parameter. An diesen schließt direkt der Daten-Adapter an, der die erforderlichen Daten für die externen Machine Learning Prozesse verarbeitet. Der Kontext der Problemstellung ist in folgender Abbildung blau dargestellt.

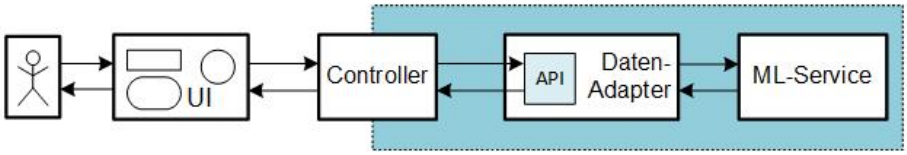

Abbildung 3.2: Systemkontext
Quelle: eigene Darstellung

Aus dem vorgestellten Anwendungsfall in 3.1.1 ergeben sich die funktionalen Anforderungen. Als Teil der Anforderungsanalyse bilden sie die Kernaufgaben des Systemteils ab (Starke 2018: S.38). Diese Anforderungen lassen sich drei Kategorien zuordnen:

Schittstelle

- Trainingsdaten und Trainingsparameter (bestehend aus „Problemtyp“, „Modell“, „Ziel-Attribut“und „Zeitraum“) vom KI-Controller aus der KI-Vertikalen beziehen
- Modell zur Erstellung von Vorhersagen bzw. zum Abspeichern zur Verfügung stellen
- Ergebnisdaten zur Verfügung stellen

Intern

- Parameter, die für die Vorbereitungen des Modells notwendig sind, aus den Input-Daten definieren
- Verbindung zum externen Service aufbauen und das Training bzw. die Vorhersage ausführen
- Ergebnisdaten und Modell aus dem Service empfangen
- Ergebnisdaten in XML konvertieren (siehe Abschnitt 3.2)

Fehlerbehandlung

- Fehler bei nicht implementierten Problemtypen oder Modellen ausgeben

In dieser Arbeit wird das Konzept beispielhaft anhand der SAP HANA PAL in der Programmiersprache Clojure umgesetzt. In SAP HANA PAL wird mit speziellen Tabellen und Funktionen gearbeitet, die näher analysiert und eingebunden werden müssen. Daher fallen diese Anforderungen zusätzlich an. Außerdem erfolgt der Zugriff auf die SAP HANA PAL über die Java Database Connectivity (JDBC)[2]. Da Clojure auf der Java Virtual Machine läuft, ist dies an der Stelle notwendig.

3.1.3 Daten für das Fallbeispiel

Wie im Anwendungsfall bereits erwähnt, wird der Benutzer befähigt, eine Auswahl in der Suche zu treffen und einer Fragestellung nachzugehen. Hierfür eignen sich die Daten aus dem Datensatz des Global Conflict Risk Index (GCRI). Dieser beinhaltet die strukturellen Bedingungen in ausgewählten Ländern und basiert ausschließlich auf quantitativen Indikatoren aus offenen Quellen und ist in mehreren Gruppen aufgeteilt: Politik, Sicherheit, Soziales, Wirtschaft, Geografie und Umwelt. In der Krisenprävention werden solche Daten verwendet, um aussagekräftige Vorhersagen zu treffen. Dies basiert auf der Annahme, dass das Auftreten von Konflikten mit diesen strukturellen Bedingungen zusammenhängt. Somit lassen sich Länder mit hohem Konfliktrisiko identifizieren.

Aufbauend auf seiner Wissensbasis kann der Benutzer nun beispielhaft seine Daten aus diesem Datensatz im Suchfenster auswählen. Nach der Angabe der Traininsgparameter im KI-Fenster kann er den Button für die Vorhersage betätigen und diese Anfrage wird an die KI-Vertikale und darin an den Machine Learning-Controller übergeben. Da nun die Vorhersagen in diesem Beispiel mittels SAP HANA PAL durchgeführt werden sollen, wird dieser Adapter nun aktiv und übernimmt den Datentransfer

[2]Eine Datenbankschnittstelle der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken bietet

und alle internen Prozesse für das Training und die Vorhersage. Das Ergebnis kann nun im Graphical User Interface (GUI) (Dt. Benutzeroberfläche) abgerufen und visualisiert werden. Anschließend ist es dem Benutzer möglich daraus Schlüsse zu ziehen. Beispielsweise könnte dies die Entwicklung der Korruption innerhalb eines Landes in den kommenden zehn Jahren sein.

Die ermittelten funktionalen Anforderungen erfordern eine konzeptuelle Ausarbeitung, welche sich an die Einbettung in den übergeordneten Kontext anschließt.

3.2 Einbettung von CAMINO in den übergeordneten Kontext

Im Folgenden soll nun der übergeordnete Kontext zu Open60 dargestellt und erläutert werden. Dabei werden die Systemelemente betrachtet, die innerhalb von Open60 unmittelbar mit dem Adapter in Verbindung stehen (siehe Abbildung 3.3). Von der Benutzereingabe bis hin zu PAL gibt es einige Systemteile, die unterschiedlichen Aufgaben erledigen und für die Weitergabe der Daten verantwortlich sind. Diese sollen hier kurz vorgestellt werden.

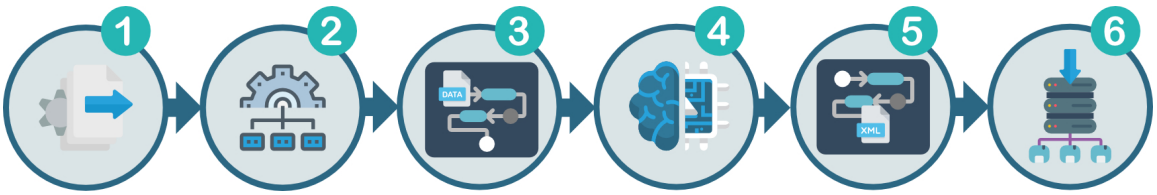

Abbildung 3.3: Schnittstellen zum Datenaustausch innerhalb von Open60
Quelle: eigene Darstellung

Der Ausgangspunkt ist, dass durch die Suche des Nutzers eine neue Dateninstanz[3] erzeugt wird (1). Diese Dateninstanz wird, zur schnellen Verfügbarkeit, lokal in der HANA-Datenbank In-Memory gehalten. Die ID der Dateninstanz wird zusammen mit Modelltyp, Problemtyp und Algorithmus mit dem Request von der Suche an die KI-Vertikale übertragen. Im Anschluss daran wird das aus diesen Elementen bestehende Request im KI-Server angenommen (2), der wiederum den HANA-Daten-Adapter delegiert (3). Dort existiert ein Algorithmus, der dafür sorgt, dass diese Dateninstanz, sowie der gewählte Machine Learning-Algorithmus und Problemtyp auch so verarbeitet werden, dass es von SAP HANA PAL verwendet werden kann (darauf wird im Abschnitt 3.2.3 näher eingegangen). Der Algorithmus wird in SAP HANA PAL nun

[3]Eine Dateninstanz ist die abstrakte Beschreibung einer Datenmenge innerhalb von Open60. Da diese eine systemweite und vereinheitlichte Beschreibung darstellt, kann durch eine eindeutige ID auf eine solche Dateninstanz verwiesen werden.

auf die Daten angewandt 4 und stellt ein Ergebnis bereit, die Vorhersage für den Nutzer. Der Daten-Adapter enthält zudem einen Algorithmus, der die Ergebnisdaten wiederum in eine XML-Struktur[4] überführt 5. Danach erfolgt die Persistenz der Ergebnisdaten in der Datenbank 6.

3.2.1 Allgemeiner Kontext

Vor der Ausarbeitung des Konzepts CAMINO soll zuerst der allgemeine Kontext, mit dem Fokus auf die KI-Vertikale, betrachtet werden.
Abbildung 3.4 zeigt hierzu die Komponentenübersicht, insbesondere die beiden Systemschnittstellen 1 und 2 der KI-Vertikalen zum Systemkontext von Open60. Die Abbildung zeigt auch den geplanten Ablauf und die Richtung des Datenflusses von der Benutzereingabe zur KI-Vertikalen, worin sich ein Handler und ein Controller befinden. Die Weitergabe der XML-Struktur erfolgt vom Controller an die Vertikale Importer. Die ausgegrauten Komponenten werden später erläutert.

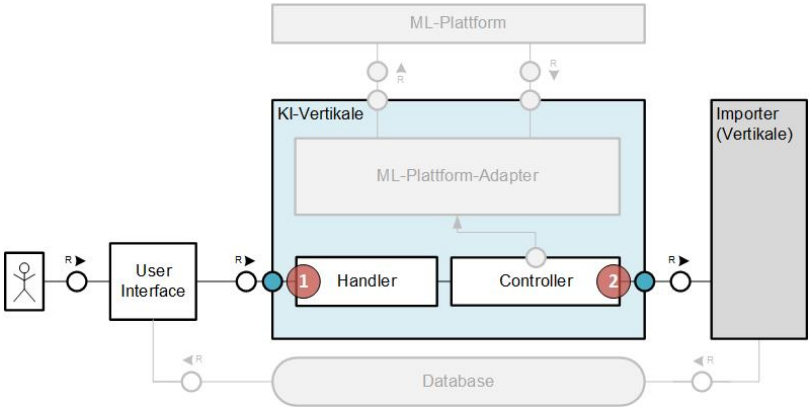

Abbildung 3.4: Komponentenübersicht im allgemeinen Kontext mit Fokus auf die Schnittstellen der KI-Vertikalen zu Open60
Quelle: eigene Darstellung

Für alle Abbildungen dieses Kapitels werden die FMC[5]-Stencils verwendet.

Im nächsten Abschnitt wird auf die Details der Schnittstellen der KI-Vertikalen zu Open60 detaillierter eingegangen.

[4]Da die Dateistruktur der Vorhersage eines externen ML-Service so nicht im System genutzt werden kann, muss sie in eine XML-Struktur konvertiert werden. Alle Daten in Open60 basieren auf einem generischen Datenmodell, welches ein bestimmtes Schema erwartet. Daten sind so systemweit nutzbar, da sie auf der gleichen Struktur basieren. Für den Import wird daher ein XML-Schema erwartet, was dieser Struktur entspricht

[5]Fundamental Modeling Concepts (FMC) sind eine semi-formale Methodik zur Kommunikation über komplexe Softwaresysteme (Knopfel, Grone & Tabeling 2006)

3.2.2 Schnittstellen der KI-Vertikalen zu Open60

Eine Schnittstelle (Application Programming Interface (API)) ermöglicht die Kommunikation mit anderen Systemkomponenten. APIs bilden die Grundlage der „Verträge“, auf deren Basis die Systemkomponenten miteinander arbeiten (design by contract). Erst diese Zusammenarbeit ermöglicht es dem System, seine Aufgaben zu erfüllen (Starke 2018: S.21).

Es folgt zunächst eine Beschreibung der APIs zwischen der KI-Vertikalen und den damit kommunizierenden Komponenten. Diese enthält den technischen Aufbau (Syntax) und welche Informationen und Operationen über die jeweilige API ausgetauscht werden (Semantik).

APIs der KI-Vertikalen zu Open60

Name	Parameter	Semantik	Art	Zweck
websocket-routes 1	Problemtyp, Dateninstanz, Zielattribut, Algorithmus-parameter Modelltyp, Modellname	Für die Vorhersage notwendige Input Daten aus dem UI	API zur KI-Vertikalen	Machine Learning-Parameter
export-data 2	Datum, Land, Notiz, Zielattribut (für jeden Datenpunkt)	Vorhersagedaten von PAL	API zwischen KI-Vertikale und Importer	Transformierte XML-Daten zur Übermittlung an den Importer

Tabelle 3.1: Tabellarische Übersicht der APIs zur KI-Vertikalen in Open60
Quelle: eigene Darstellung

Das Verhalten der APIs ist in Abbildung 3.5 dargestellt. Einige Komponenten, die gerade nicht im Fokus stehen, sind ausgegraut dargestellt, da diese erst im späteren Verlauf der Arbeit genauer betrachtet werden. Die gerade beschriebenen APIs sind rot mit Zahlen versehen.

Die Daten werden über die Funktion „websocket-routes“ 1 des Handlers in der KI-Vertikalen empfangen[6]. Darin wird eine Funktion des Controllers aufgerufen, die dann den Task anstößt, welcher im Adapter ausgeführt werden soll (execute-task). Diese Funktion prüft zuerst, welche Plattform gewählt wurde um den Machine Learning-Prozess auszuführen (current-platform). Nun kommt der Adapter ins Spiel. Dieser

[6]Die Vertikalen in Open60 kommunizieren über Websockets, was durch die Bibliothek „Pneumatic-Tubes“ für Clojure ermöglicht wird

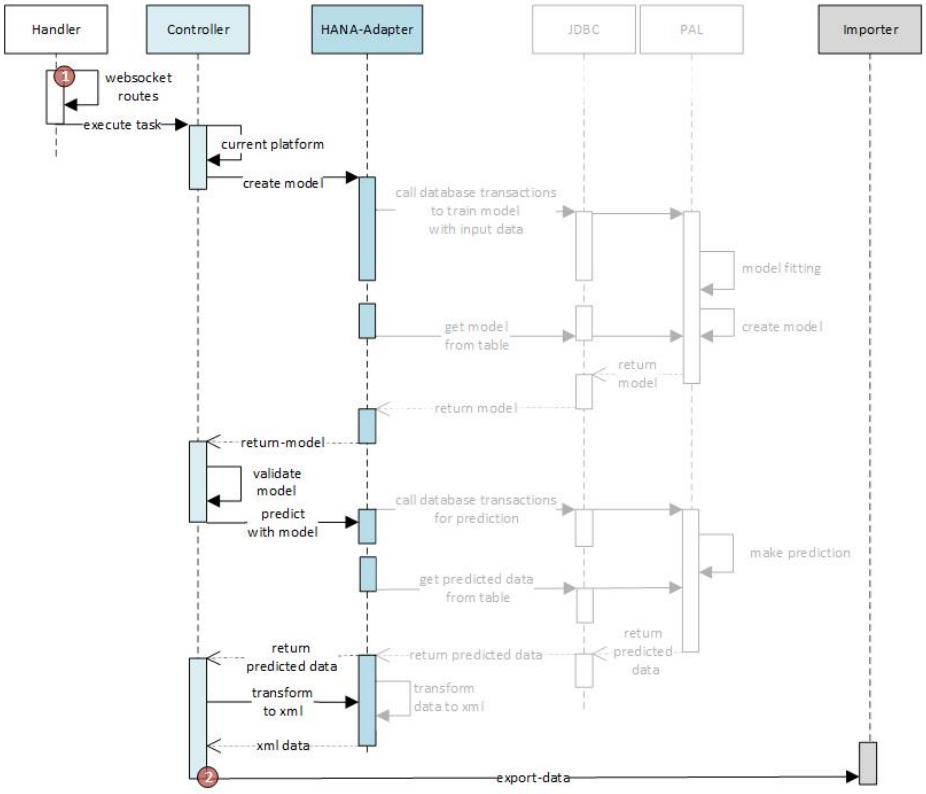

Abbildung 3.5: Ablauf der Datenverarbeitung in der KI-Vertikalen
Quelle: eigene Darstellung

implementiert die Plattform-API und erhält den Aufruf, das Modell erstellen und zu trainieren (create-model). Die Trainingsdaten und die nötigen Verbindungsdaten zur Plattform werden ebenfalls mit übergeben. Der Adapter führt das Training aus und gibt das erstellte Modell zurück (return-model). Im Controller wird dieses auf Konsistenz geprüft (validate-model) und wieder an den Adapter übergeben, um die Vorhersage zu berechnen (predict-with-model). Der Adapter führt die Vorhersage auf Basis der übergebenen Daten und des Modells aus und stellt die Vorhersagedaten zur Verfügung (return-predicted-data). Im Controller werden die Vorhersagedaten angenommen, geprüft und dann wieder an den Adapter übergeben, der diese in die XML-Struktur überführt (transform-to-xml). Im Controller werden letztendlich die Vorhersagedaten als XML-Struktur erwartet (xml-data) und erst dann exportiert (export-data) 2. Dies erfolgt über die REST-API an eine weitere Vertikale, den Importer, über ein HTTP-Request. Dieser importiert die Daten in die Datenbank, sodass sie systemweit zur Verfügung stehen.

Möchte nun ein Benutzer beispielsweise eine Vorhersage für Afghanistan mit dem Ziel-Attribut Korruption der nächsten 10 Jahre mit Hilfe des GCRI Datensatzes (sie-

he Abschnitt 3.1.3) erstellen lassen, kann er dies in der Suche auswählen. Auf Basis der Suche wird der Task an die KI-Vertikale geschickt. „Task“ in Abb. 3.6 entspricht dem in Abbildung 3.5 eingehenden Task 1.

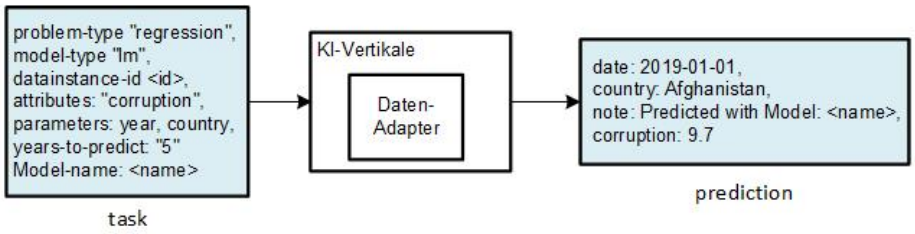

Abbildung 3.6: Beispiel für den (Input-)Task und die daraus resultierende Vorhersage
Quelle: eigene Darstellung

Der Controller ruft den Daten-Adapter CAMINO auf, der mit der Machine Learning-Plattform SAP HANA PAL (current-platform) interagiert. Dann folgen die internen Prozesse im Daten-Adapter, die im nächsten Abschnitt 3.2.3 näher beschrieben werden. Das Ergebnis der Vorhersage besteht nun insgesamt aus 10 Datenpunkten, da der Nutzer 10 Jahre ausgewählt hat. Einer der Datenpunkte ist in Abbildung 3.6 „prediction“ beispielhaft für das Jahr 2019 dargestellt. Der Daten-Adapter sorgt für die Konvertierung ins XML-Format und sendet die Datenpunkte wie beschrieben an den Importer 2. Diese können nun beispielsweise als Chart im GUI visualisiert werden, um die Entwicklung der Korruption in Afghanistan zu analysieren.

3.2.3 Machine Learning-Kontext SAP HANA PAL

Um die Funktionen eines solchen Daten-Adapters zu konzipieren und umzusetzen, müssen zuerst die internen Prozesse eines Machine Learning-Dienstes analysiert werden. Dies soll am Beispiel der SAP HANA PAL (kurz: PAL) gezeigt werden. Ein Machine Learning-Prozess unterteilt sich in Training 3.1 und Vorhersage 3.2.

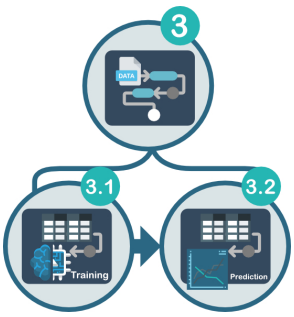

Abbildung 3.7: Unterteilung von Schritt 3 in Training und Vorhersage
Quelle: eigene Darstellung

Für das Training müssen die Daten in PAL-Tabellen überführt werden (siehe Abbildung 3.8, 3.1.1 und 3.1.2). Diese Tabellen enthalten die Trainingsdaten sowie alle Einstellungen, die das Training eines Machine Learning-Modells benötigt. Daraufhin wird eine sogenannte „Wrapper-Procedure“, eine SQL-Prozedur, welche einem internen Funktionsaufruf in PAL entspricht (siehe 3.1.3) erstellt. Danach werden erneut Datentabellen und eine Tabelle für den Algorithmus erzeugt (siehe 3.1.4 und 3.1.5). Schließlich kann diese Wrapper-Prozedur wie eine Funktion ausgeführt werden und gibt das Modell in einer Ausgabetabelle zurück 3.1.6.

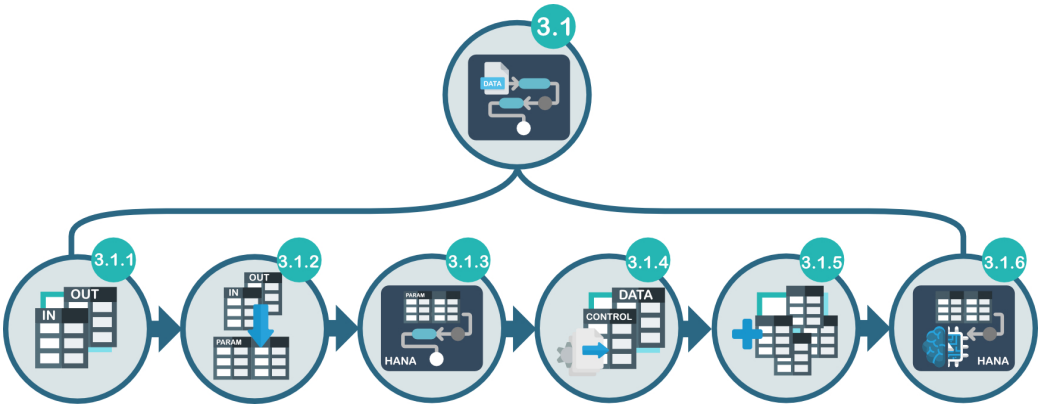

Abbildung 3.8: Prozesse des Trainings
Quelle: eigene Darstellung

Das gleiche Verfahren gilt für die Vorhersage, die ebenfalls mit solchen Tabellen erstellt und durch eine Wrapper-Prozedur generiert wird. Schlussendlich wird das Ergebnis ebenfalls in einer Tabelle zur Verfügung gestellt und lässt sich auslesen.

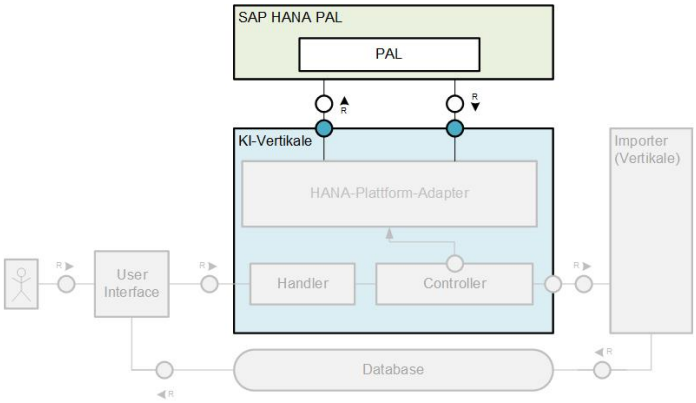

Abbildung 3.9: Komponentendiagramm im Kontext Daten-Adapter und der Kommunikation mit PAL
Quelle: eigene Darstellung

In Abbildung 3.9 wird anhand eines Komponentendiagramms gezeigt, wie sich der Daten-Adapter (in Abbildung 3.9 „HANA-Plattform-Adapter“ genannt) in die KI-Vertikale einbettet. Als Bestandteil dessen, kommuniziert er mit dem externen Machine Learning Service PAL.

Abbildung 3.10 stellt den groben Kommunikationsablauf dar, der zwischen dem Adapter und PAL stattfinden soll.

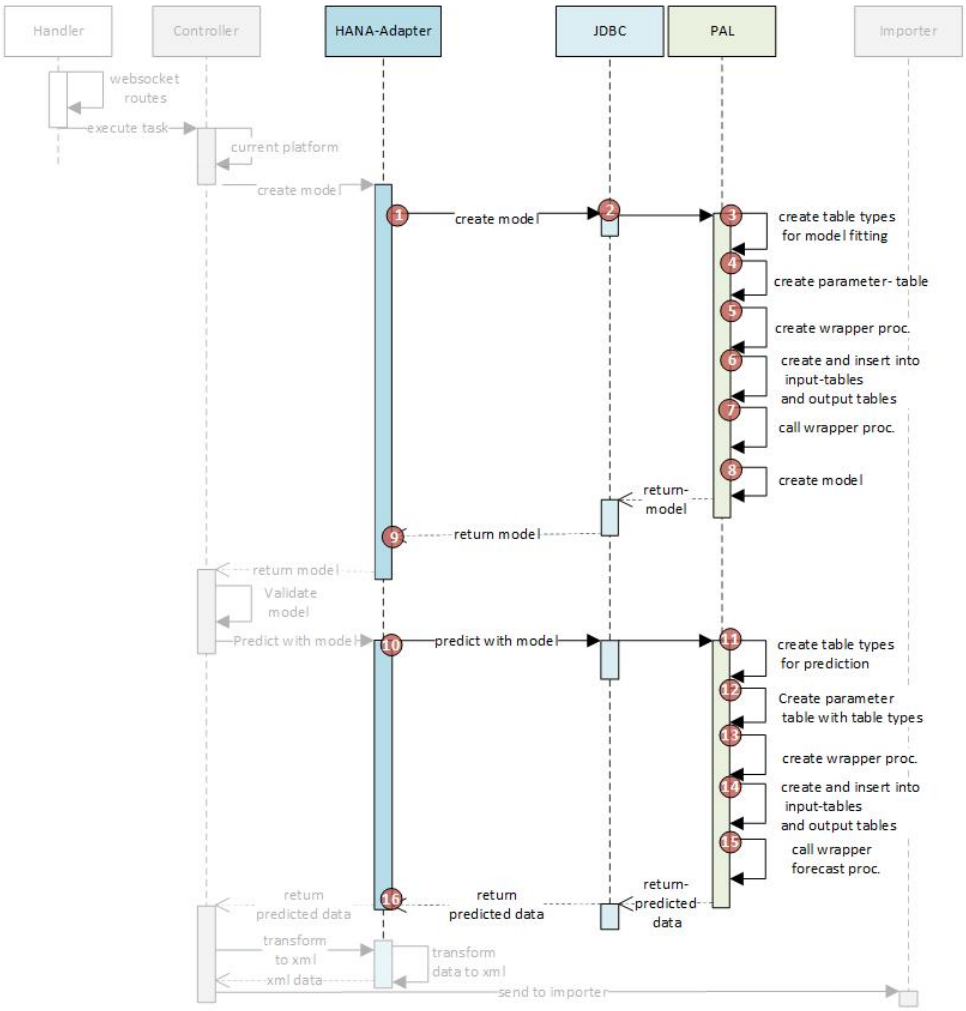

Abbildung 3.10: Verhalten an den Schnitstellen zu PAL
Quelle: eigene Darstellung

Die Funktion „create-model“ ❶ dient als Ausgangspunkt und wird innerhalb des Adapters aufgerufen. Dieser stellt über JDBC ❷ die Anfragen an die PAL, um zuerst die Input- und Output-Tabellen anzulegen ❸, danach die Parametertabelle zu erstellen ❹. Danach wird die Wrapper-Prozedur generiert ❺ und die Input-Tabellen mit Daten befüllt ❻. Anschließend wird die Wrapper-Prozedur ausgeführt ❼ und darauf-

folgend das Modell erstellt 8. Nun kann das Model aus der Result Tabelle abgefragt und zurückgegeben werden 9. Das Modell wird in der Funktion „predict“ wieder verwendet 10. Diese generiert analog zum Trainingsprozess Input- und Output-Tabellen sowie eine eigene Parametertabelle 11 , 12. Anschließend wird hierbei ebenfalls eine Wrapper-Prozedur erstellt 13. Dieser wird nach befüllen der Tabelle 14 ausgeführt 15 und das Ergebnis der Vorhersage in den Output-Tabellen zugänglich gemacht 16. Dieses wird durch den Aufruf der „predict“ Funktion möglich.

Im Folgenden sollen die notwendigen Vorgehensweisen für die Ausführung eines Machine Learning-Prozesses innerhalb von PAL im Detail vorgestellt werden. Der gesamte Prozess findet innerhalb der SAP HANA-Datenbank statt.

Syntax und Semantik der PAL-Funktionen:

Wie bereits im Kapitel 2.1.3 vorgestellt, ist die PAL ein Teil der AFL in SAP HANA. Prozeduren aus „_SYS_AFL“ werden genutzt, um Wrapper-Prozeduren zu generieren, welche dann die PAL-Funktionen verwenden. Diese Wrapper-Prozeduren werden dem Nutzer im Technical-User-Schema zur Verfügung gestellt und lassen sich ausführen. Im Allgemeinen werden Input-Tabellen (z. B. Datentabellen und Parametertabellen) an diese Prozeduren übergeben und die zurückgegebenen Ergebnisse werden in Output-Tabellen gespeichert.

Implementieren eines PAL-Algorithmus:

Die Implementation eines PAL-Algorithmus soll nun anhand der Linearen Regression[7] erläutert werden. PAL-Algorithmen sind immer mit einer spezifischen Abkürzung in Großbuchstaben versehen. Der PAL-Funktionsname für Lineare Regression lautet: „LRREGRESSION“. Zunächst muss das Modell trainiert werden, anschließend steht es für Vorhersagen bereit. Hierfür sind folgende Schritte notwendig:

1. Definieren der Input- und Output-Tabellen:

Zur Erstellung eines Modells muss der Algorithmus über zwei Input-Tabellen (IN) und vier Output-Tabellen (OUT) verfügen. Diese sind von PAL festgelegt und sehen für die Lineare Regression folgendermaßen aus:

IN - Data Table: Die Trainingsdaten für das Modell,

IN - Control Table: Der Parameter, der die Arbeitsweise des Algorithmus festlegt,

OUT - Result Table: Modellgenauigkeit und Vertrauen,

OUT - Result Fitted Table: Die Werte des Ziels,

OUT - Result Significance Table: Koeffizienten des Modells,

OUT - Result PMML Table: Das resultierende Modell für die Vorhersage.

[7] Lineare Regression ist ein statistisches Verfahren, bei dem ein linearer Zusammenhang zwischen zwei Variablen modelliert wird.

1.1 Löschen der Table Types:

Sollten die Tabellen bereits existieren, müssen diese zuerst gelöscht und danach erneut angelegt werden, damit sie verwendet werden können. Dieser Vorgang wird bei jeder der Tabellen ausgeführt.

1.2 Neues Anlegen der Table Types:

Diese Tabellen werden als sogenannter „TYPE“ angelegt. Sie dienen sozusagen als Template für Tabellen. Die in Schritt 4.1 zu erstellenden Tabellen basieren auf diesem Template.

2.1 Anlegen der Parametertabelle:

Als Nächstes muss die Parametertabelle für die Definitionen des Modells angelegt werden.

2.2 Befüllen der Parametertabelle:

In diese Tabelle werden nun die zuvor in Schritt 1. definierten Tabellen als Werte abgelegt. Dazu gehört der Name der Tabelle als Zeichenkette und deren Information, ob es sich um eine Input- oder Output-Tabelle handelt, sowie deren Position und das aktuelle Schema.

3. Erstellen der Wrapper-Prozedur:

Bei der Generierung der Wrapper-Prozedur werden nun der Name des Schemas (DM_PAL), der Prozedurname (PAL_LR_PROC) sowie die zuvor definierte Parametertabelle (PAL_MLR_PDATA_TBL) verwendet.

4.1 Erstellen der Data Table:

Die Tabelle wird auf Basis des in Schritt 1. definierten „TYPE“ angelegt. Das Schlüsselwort hierfür ist „LIKE“.

4.2 Befüllen der Data Table:

Nun kann die Data Table mit den Trainingsdaten befüllt werden.

5.1 Erstellen der Control Table:

Als nächstes wird die Control Table erstellt. Diese dient zum Anpassen der Parameter für den Machine Learning-Algorithmus.

5.2 Werte in die Control Table eintragen:

Hier lassen sich nun optionale Parameter für den Algorithmus, beispielsweise statistische Anpassungen oder Einstellungen zum Modell selbst definieren.

6. Output-Tabellen:

Für die Modellgenauigkeit muss nun die Result-Table erstellt werden sowie die Forecast-Table für das Ergebnis der Vorhersage: Die Tabelle wird ebenfalls auf Basis des in Schritt 1. definierten „TYPE“ angelegt. Dies wird ebenso für Fitted, Significance und PMML-Model durchgeführt.

7. Modell erstellen:

Abschließend muss die Wrapper-Prozedur mit den Tabellen als Übergabeparameter ausgeführt werden, um das Modell zu trainieren.

8. Ergebnis abfragen:

Anschließend ein Zugriff auf das trainierte Modell mittels eines Select-Befehls als SQL-Script möglich. Das Modell wird im Format Predictive Model Markup Language (PMML) zurückgegeben.
(Der Quellcode zu diesen einzelnen Schritten ist im Anhang aufgeführt)

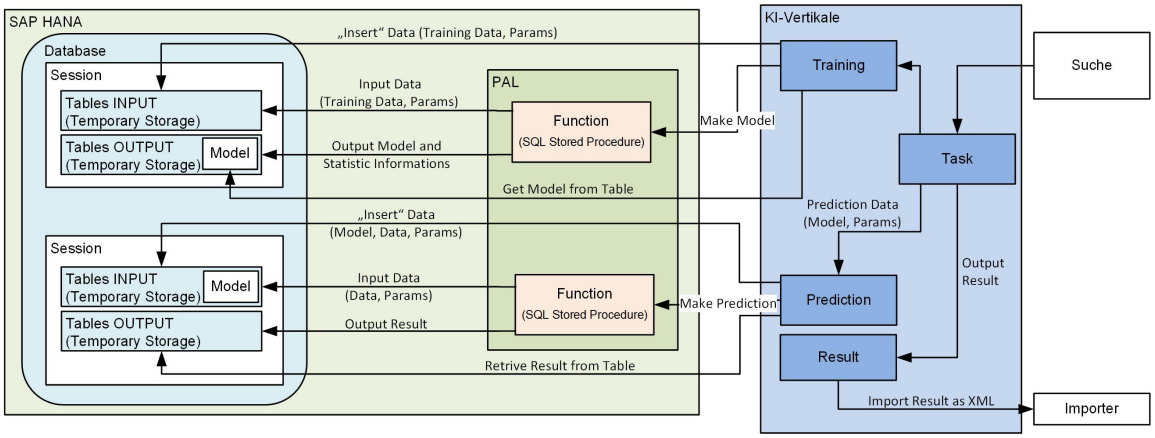

Abbildung 3.11: Funktionale Darstellung der Prozesse
Quelle: eigene Darstellung

Diese funktionale Darstellung zeigt die internen Prozesse im Bezug auf die Funktionen und die dabei zum Einsatz kommenden Tabellen innerhalb der SAP HANA im Zusammenhang mit der KI-Vetikalen. Eine Funktion (SQL Stored Procedure) kann hier beispielsweise die Lineare Regression sein. Eine Session repräsentiert in diesem Fall jeweils Training und Vorhersage.

9. Vorhersage ausführen:

Dann ist es möglich mit dem Modell und ausgewählten Daten eine Vorhersage ausführen zu lassen. Dies geschieht analog zum Training ebenfalls mit entsprechenden Tabellen und einer „Wrapper-Procedure“(siehe Abbildung 3.11).

10. Ergebnis der Vorhersage abrufen:

Im Anschluss werden die Daten der Vorhersage in der Result Table zur Verfügung gestellt. Diese enthält alle Datenpunkte jeweils bestehend aus Datum, Land und dem Ziel-Attribut.

Nach der Analyse der internen Prozesse eines ML-Services am Beispiel zu SAP HANA PAL kann nun ein Konzept entwickelt werden, welches die Kommunikation zwischen den in Abschnitt 3.2.1 vorgestellten Systemmodule von Open60 und diesem Machine Learning-Service in Form eines Daten-Adapters ermöglicht.

3.3 CAMINO API

In Abschnitt 3.2 erfolgte die Einbettung von CAMINO in den übergeordneten Kontext. Aufbauend auf dieser Basis soll nun die Entwicklung des Konzepts des Daten-Adapters CAMINO folgen. Seine Aufgabe ist die Bereitstellung aller notwendigen Funktionalitäten für den reibungslosen Datentransfer zwischen dem Controller in der KI-Vertikalen und einem externen Machine Learning-Service.

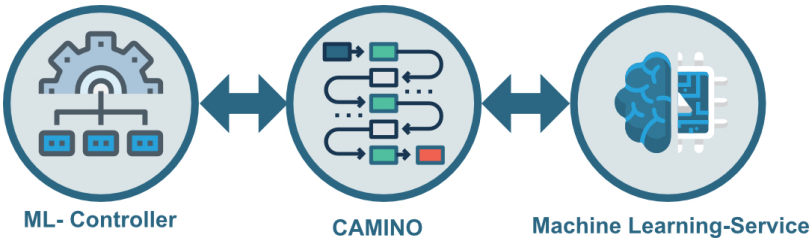

Abbildung 3.12: Komponenten im direkten Kontext zu CAMINO
Quelle: eigene Darstellung

Aus den im Abschnitt 3.2 ermittelten funktionalen Anforderungen an CAMINO ergaben sich im Wesentlichen drei Kategorien: „API“, „Intern“ und „Fehlerbehandlung“. Im Folgenden sollen diese nun ausgearbeitet werden.

Definition der CAMINO-API

Eine der zentralen Aufgaben des Adapters besteht darin, eine API zur Verfügung zu stellen. Diese wird einerseits unter Betrachtung der Interaktion mit dem Controller in der KI-Vertikalen und andererseits für die optimale Anbindung der Dienste des Machine Learning-Services definiert.
Folgenden Fälle dienen daher der API-Definition für CAMINO:

1. Ein Modell mit Input-Daten und einem angegebenen Algorithmus erstellen.
2. Ein bereits trainiertes Modell soll eine Vorhersage mit angegebenen Daten erstellen.
3. Daten, die bereits aus einer Vorhersage stammen, sollen konvertiert werden,
4. Ein vorhandenes Modell speichern,
5. Ein Modell aus der Datenbank abfragen.
6. Ein vorhandenes Modell löschen.

Abbildung 3.13 soll Fall 1 und 2 schematisch darstellen. So wäre im Fall 1 ein Modell auf Basis gegebener Daten und Algorithmus zu generieren und im Fall 2 eine Vorhersage anhand eines vorhandenen Modells zu erstellen. Wenn der Nutzer schon ein trainiertes Modell hat, wird nur Fall 2 unabhängig von Fall 1 benötigt. Daher wird dies auch in der API als getrennte Funktion vorgesehen.

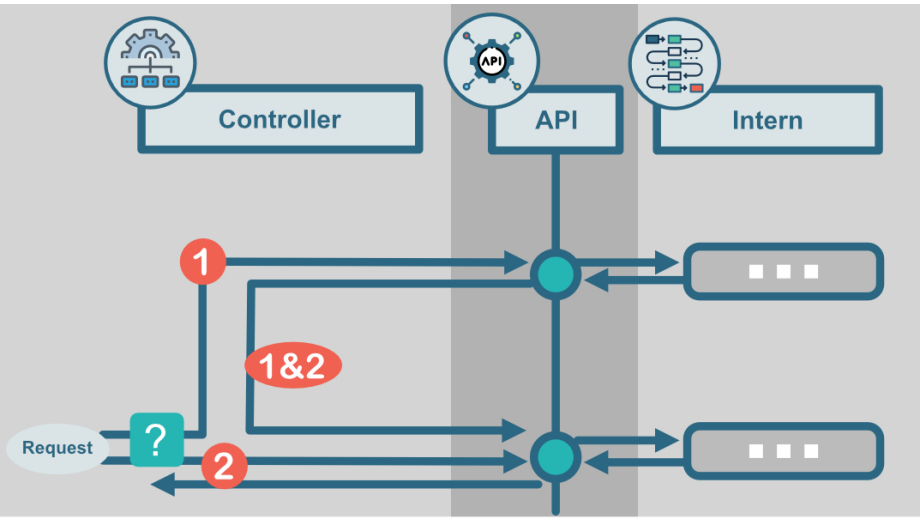

Abbildung 3.13: Definitionen für Fall 1 und 2
Quelle: eigene Darstellung

Daraus ergibt sich, dass es nicht erforderlich ist, zuerst ein Modell zu trainieren, sondern auch direkt eine Vorhersage erstellt werden kann. Da das Modell im PMML-Format zurückgegeben wird, kann es theoretisch auch in einem anderen Machine Learning-Service wiederverwendet werden, der diesen Standard unterstützt[8]. Der in der Abbildung als „intern“ dargestellte Teil, entspricht den internen Verfahren eines Machine Learning-Services. Da in dieser Arbeit beispielhaft mit PAL gearbeitet wird, entspricht dies den im Kapitel 3.2.3 vorgestellten internen Verfahren der PAL.

Da die vorhandene Architektur, die sich an das generische Datenmodell von Open60 anlehnt, eine XML-Struktur für den Import der Daten erwartet, wird hierfür ebenfalls eine Schnittstellenfunktion vorgesehen. Zusätzlich gibt es noch andere Möglichkeiten die Daten nach der Erstellung einer Vorhersage in der Software zur Verfügung zu stellen, was in zukünftigen Arbeiten als Erweiterung ermöglicht werden kann.

Für jeden der genannten Fälle soll eine Schnittstellenfunktion angeboten werden. Durch diese Definitionen unterstützt die API diese Kriterien:

- **Erweiterbarkeit** - Die API ist durch zusätzliche Funktionen erweiterbar.
- **Unabhängigkeit** - Die Funktionen der API sind voneinander unabhängig nutzbar.
- **Kompatibilität** - Die Struktur funktioniert auch mit anderen Machine Learning-Services, denn es ändert sich nur der „interne“ Aspekt, aber nicht die Struktur selbst.

3.3.1 Daten und Semantikaspekte

Nach der Abbildung der Anforderungen für die API werden nun die Funktionen, die beim Aufruf durch den Controller in Aktion treten, genauer definiert.

In Abbildung 3.2 wird der tabellarische Überblick über die Adapter-API und deren Funktionen dargestellt, die man durch den Controller der KI-Vertikalen aufrufen kann. Die Funktionen „save“, „retrieve“ und „delete“ sind hier nur zur Vollständigkeit der API aufgeführt und werden im Rahmen dieser Arbeit nicht näher beschrieben.

[8] PMML ist ein XML-basiertes Standardformat für ML-Modelle, ursprünglich konzipiert von Grossman et al. (2002), wodurch es möglich ist, Modelle mit anderen Anwendungen auszutauschen, die dieses Format unterstützen. Es wurde seitdem stetig von der Data Mining Group, einem Konsortium aus kommerziellen und Open-Source-Data-Mining-Unternehmen, weiterentwickelt. In Anwendungen von IBM oder Microsoft wird dieses Format ebenfalls unterstützt (DMG 2020).

CAMINO-Adapter-API			
Funktionsaufruf	Semantik	Request	Response
create-model (1)	Modell erstellen und trainieren	Connection, Task (Problemtyp, Modelltyp, Dateninstanz-ID, Attribute, Algorithmus-Parameter, Modellname)	Modell, ID, Datensatz, Algorithmus-Parameter
predict-with-model (2)	Modell ausführen und Ergebnisdaten zurückgeben	Connection, Task, Modell	ID, Predicted Data
convert-prediction-to-xml (3)	Daten der Vorhersage in XML konvertieren	Modell, Predicted Data	Ergebnisdaten konvertiert in XML
save-model	Modell in Datenbank persistieren	Connection, Model-Data	Wahrheitswert, über den erfolgreichen Abschluss der Aktion
retrieve-model	Modell aus Datenbank abrufen	Connection, Model-ID	Modell
delete-model	Modell aus Datenbank löschen	Connection, Model-Data	Wahrheitswert, über den erfolgreichen Abschluss der Aktion

Tabelle 3.2: Tabellarische Beschreibung der API

Die Funktionen, ihre Semantik, sowie der jeweilige interne Aspekt und die Fehlerbehandlung werden im Folgenden erläutert:

create-model

- **Semantik:** Um ein Modell erstellen zu können benötigt die Funktion die Verbindungsdaten zur Plattform, um Aktionen darauf ausführen zu können, sowie den Parameter „Task“. Dieser besteht u.a. aus dem Problemtyp („problem-type“), der einen Algorithmus-Typ wie beispielsweise die Regression darstellt und dem Modelltyp („model-type“) d. h. die Art der Regression. Die Dateninstanz-ID ist eine Referenz auf eine Gruppe von Daten, die später intern abgefragt wird, um das Training damit ausführen zu können. Die Eingabeparameter mit denen das Modell trainiert wird, sind in den Eingabeparametern („attribute-input“) festgelegt. In den Algorithmusparametern soll festgelegt werden, welche Attribute zusätzlich als Input dienen, diese werden als Jahr und Land festgelegt. Der Name des Modells wird ebenfalls aus der Benutzereingabe

übernommen. Die Funktion wird mit den Verbindungsparametern („connection“) sowie den Trainingsparametern aufgerufen. Da nur die Dateninstanz-ID mit übergeben wird, muss nun zunächst die Datenmenge hierzu geladen werden. Denn diese ID stellt nur die Referenz dazu dar. Diese werden von einer externen Funktion innerhalb der KI-Vertikalen zur Verfügung gestellt. Nun folgen alle internen Transaktionen mit PAL für das Training, die in Kapitel 3.2.3 erläutert wurden.
Nach dem Training werden die Rückgabeparameter definiert. Diese bestehen u.a aus der Model-ID, einer eindeutigen ID, die für das Modell generiert wurde, damit es nicht zu einer Verwechslung mit einem anderen kommen kann. Der Name des Modells wurde durch den Benutzer festgelegt und bleibt bestehen. Zudem ist die verwendete Plattform, die an die KI-Vertikale angebunden ist, ebenfalls in den Modelldaten hinterlegt. Danach folgt eine Zeichenkette, die das eigentliche Modell im PMML-Format enthält. Ebenfalls wird die Datenquelle im Wert „datasource“ gespeichert sowie das zum Trainieren verwendete Attribut und das damit verbundene Land.
Diese Werte lassen sich nun in einer Datenbank ablegen oder direkt für eine Vorhersage verwenden.

- **Request:** Verbindungsdaten („connection“) zur Plattform: „user-name“, „password“, „host“, „port“, „user-schema“ und einen zweiten Parameter „task“. Darin sind die Trainingsparameter bestehend aus „problem-type“, „model-type“, „data-
instance-id“, „attributes-input“, „algo-params“ („year“ und „country“) sowie „model-name“ enthalten.

- **Response**: Als Rückgabewert stellt die Funktion eine Hash Map bestehend aus Key-Value-Paaren der Parameter „model-id“, „name“, „platform“, „pmml-model“, „datasource“, „attribute“ und „country“ bereit. Diese werden dem Controller bereitgestellt.

- **Intern:** Durch eine Bedingung wird geprüft, um welchen Problemtyp („problem-type“) es sich handelt, denn das hat Auswirkungen auf die weitere Verarbeitung. Ist der Problemtyp nicht vorhanden, wird ein Fehler ausgegeben, dass dieser Problemtyp noch nicht implementiert wurde. Trifft einer der Problemtypen zu, wird dort die nächste Funktion aufgerufen, die nun prüft, um welchen Modelltyp („model-type“) es sich handelt. Auch hier wird bei nicht Vorhandensein des gefragten Problemtyp eine Fehlermeldung ausgegeben.
Ist ein valider Problemtyp gefunden worden, wird anschließend die Funktion aufgerufen, die das PAL-Model mit den gegebenen Daten nun erstellen wird (siehe Abbildung 3.14).

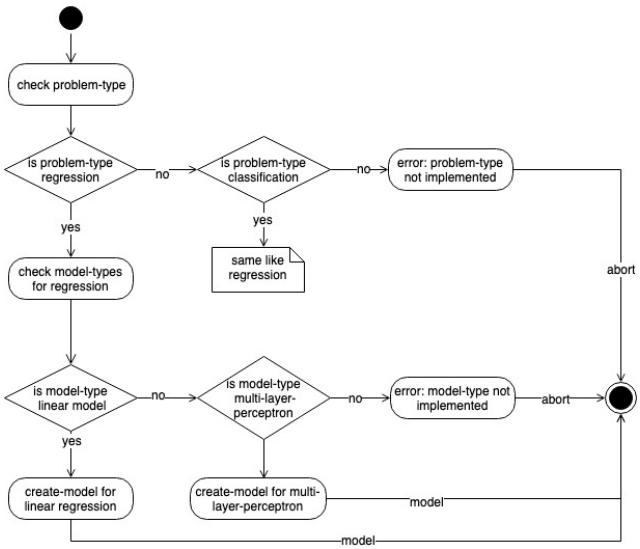

Abbildung 3.14: Aktivitätsdiagramm der Funktion create-model
Quelle: eigene Darstellung

- **Fehlerbehandlung:** Bei Nichtvorhandensein des gefragten Problemtyps oder Modelltyps folgt eine Fehlermeldung.

predict-with-model

- **Semantik:** Diese API-Funktion dient allein zum Erstellen einer Vorhersage durch ein gegebenes Modell im PMML Format und den zugehörigen Daten innerhalb eines „Task“. Dieser definiert sich durch: Den Problemtyp (einen Algorithmus Typ wie die Regression), den Modelltyp (die Art der Regression) sowie die Dateninstanz-ID um die zugehörigen Daten intern abzufragen. Die Eingabeparameter, mit denen das Modell trainiert wird, sind in „attribute-input“ festgelegt. In den Algorithmusparametern soll festgelegt werden durch welche Attribute für die Vorhersage erstellt wird und welche Ausgabeparameter daraus resultieren.

- **Request:** Dafür benötigt die Funktion die Parameter „connection“, die Verbindungsdaten zur Plattform, die bei PAL aus „user-name“, „password“, „host“, „port“ und „user-schema“ bestehen. Ein zweiter Parameter „task“ bestehend aus „problem-type“, „model-type“, „datainstance-id“, „attributes-input“ und das Modell „model“ im PMML-Format und dessen festgelegte eindeutige ID.

- **Response**: Als Rückgabewert stellt die Funktion eine „Hash-Map“ bestehend aus Key-Value-Paaren der Parameter „id“ und „data“ bereit. In „data“ sind alle Datenpunkte in Key-Value Paaren aus Jahr und Land sowie dem Vorhersageattribut angegeben. Außerdem ist hinterlegt, mit welchem Modell die Vorhersage

berechnet wurde. Dies ist durch die Model-ID gekennzeichnet. Somit sind auch alle Daten zum Modell immer mit der Vorhersage verbunden. Diese werden vom Controller empfangen.

- **Intern:**

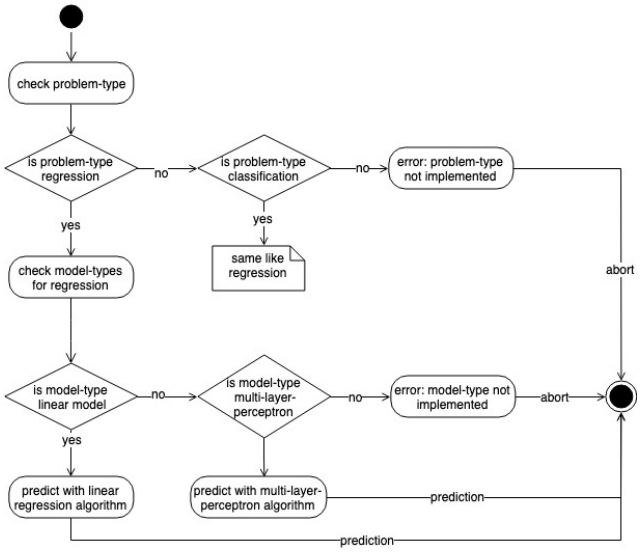

Abbildung 3.15: Aktivitätsdiagramm der Funktion predict-with-model (Intern)
Quelle: eigene Darstellung

Durch eine Bedingung wird geprüft um welchen Problemtyp („problem-type“) es sich handelt, denn das hat Auswirkungen auf die weitere Verarbeitung. Ist der Problemtyp nicht vorhanden wird ein Fehler ausgegeben, dass dieser Problemtyp noch nicht implementiert wurde. Trifft einer der Problemtypen zu wird dort die nächste Funktion aufgerufen, die nun prüft, um welchen Modelltyp („model-type“) es sich handelt. Nun folgen alle internen Transaktionen mit PAL für die Vorhersage, die in Kapitel 3.2.3 erläutert wurden. (siehe Abbildung 3.15)

- **Fehlerbehandlung:** Bei Nichtvorhandensein des gefragten Problemtyps oder Modelltyps folgt eine Fehlermeldung.

convert-to-xml

- **Semantik:** Diese Funktion ist notwendig um Daten in die Software zu importieren, damit sie für den Nutzer zur Verfügung stehen. Da diese Dateistruktur einer Vorhersage so nicht im System genutzt werden kann, muss diese konvertiert werden. Dies geschieht ebenfalls im Adapter, dafür sorgt die Funktion „convert-to-xml“. Da die Vertikale „Importer“ für den Import in die Datenbank ein bestimmtes Format, nämlich XML erwartet, muss die Datenstruktur in diese XML-Struktur überführt werden.

- **Request:** Diese Funktion erwartet zunächst als Eingabeparameter ein Modell, wie es in „create-model“ erstellt wurde und als zweiten Parameter die Ergebnisdaten einer Vorhersage als Key-Value-Paare bestehend aus Jahr und Land mit dem Vorhersageattribut. Zusätzlich sollte ein Datum und der Modell-Name vorhanden sein, um die Daten für den Nutzer sinnvoll visualisieren zu können.

- **Response**: Als Rückgabewert stellt die Funktion eine XML-Struktur[9] bereit. Dieser kann anschließend über ein HTTP-Request an den Importer gesendet werden.

- **Intern:** Innerhalb der Funktion wird folgende XML-Struktur erzeugt:
  - Das umschließende Attribut ist das Data-Set und dieses enthält wiederum drei andere Sets: Context-Set, Datasource-Set und Item-Set.
  - Context-Set Dieses enthält die Information um welches Land es sich handelt. Jedes weitere Land würde wiederum als neuer Context angelegt werden. Jeder Context besitzt eine Global-ID.
  - Datasource-Set Die Dateninstanz der Datenquelle (Abstraktion der gewählten Datenmenge), es enthält außerdem eine Datasource-ID, den Namen des Modells und den Link dazu.
  - Item-Set Pro Datenpunkt der Vorhersage wird dem Item-Set ein Item hinzugefügt. Dieses Item enthält die Attribute, die zu einem Datenpunkt gehören. Global-ID des Items, „date“, „location“, „notes “(Notizen, bsp. der Name des Modells), Interaction und Properties. Properties beinhaltet das Vorhersage-Attribut. In diesem Beispiel „corruption“.

3.3.2 Struktur und Verhalten

Abbildung 3.16 zeigt die Einbettung CAMINO in den Gesamtablauf. Die Funktionsaufrufe der CAMINO-API sind hervorgehoben und numerisch markiert.
Der Controller ruft die Methoden „create-model“ 1 und „predict-with-model“ 2 im Adapter auf um die angebundenen Machine Learning-Dienste zu nutzen. Im Adapter selbst wird nun die Konvertierung für die Vorhersage in PAL vorbereitet. Nachdem diese Informationen erstellt wurden, wird über JDBC eine Verbindung zur PAL aufgebaut. Dort wird mittels der vorgestellten Prozeduren ein Wrapper generiert und die darin enthaltenen Tabellen in PAL erstellt, worauf dann das Trainieren des Modells erfolgt. PAL sendet das Ergebnis zurück über JDBC an den Adapter. Nun wird das Modell verwendet, um eine Vorhersage anzustoßen. Der Adapter überführt die Ergebnisdaten in „convert-data-to-xml“ in das XML-Format 3 bestehend aus Context-Set, Datasource-Set und Item-Set.

[9] Im Anhang befindet sich ein Beispiel für ein valides XML, das aus Vorhersagedaten konvertiert wurde

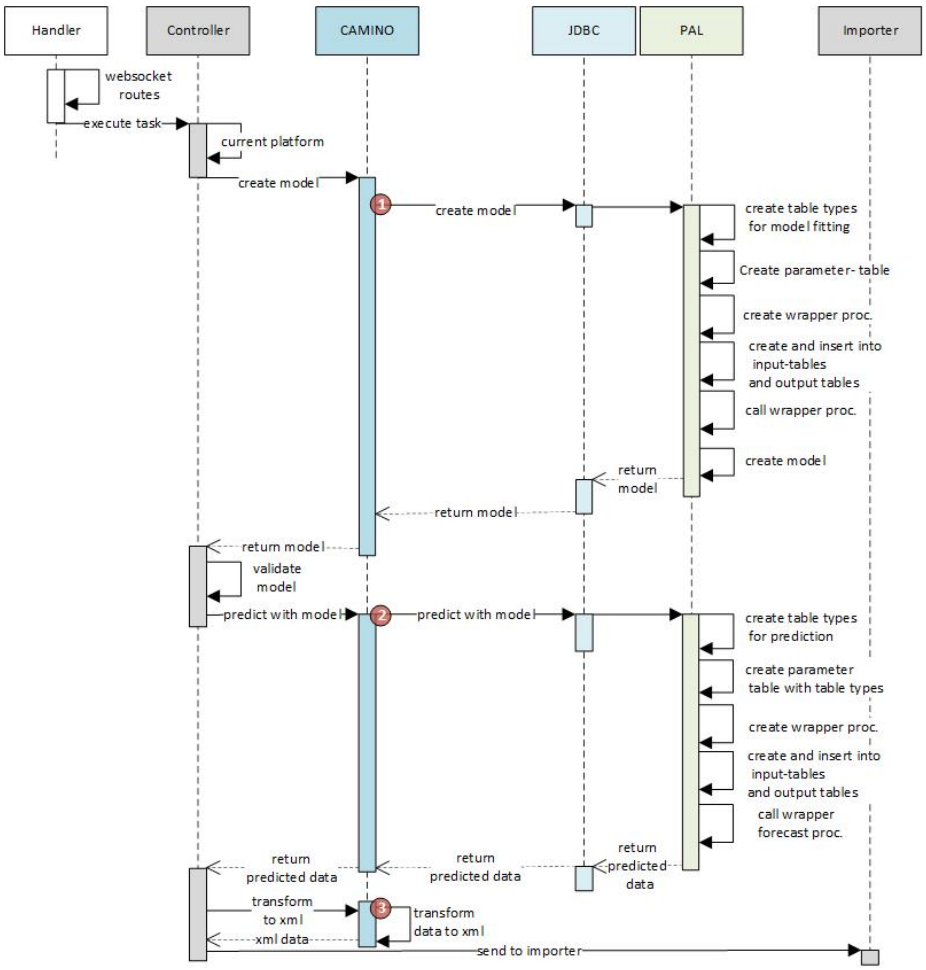

Abbildung 3.16: Einbettung von CAMINO in den Gesamtablauf
Quelle: eigene Darstellung

Am Ende erhält der Controller nun die Ergebnisse in XML-Format und kann diese direkt an den Importer senden. Die Ergebnisse der Vorhersage können nun in der Benutzeroberfläche als neue Datenquelle ausgewählt und visualisiert werden.

3.3.3 Adapterarchitektur

Nach der Beschreibung der Funktionen und deren internen Aspekte soll nun in Abbildung 3.17 die Architektur des Adapters veranschaulicht werden.

- **Platform-Adapter**
  Dient als eine Art Template und enthält die Schnittstellenfunktionen, die von jedem Adapter, hier beispielsweise dem PAL-Platform-Adapter, implementiert werden können (create-model, predict-with-model, convert-to-xml, save-model, retrieve-model, delete-model).

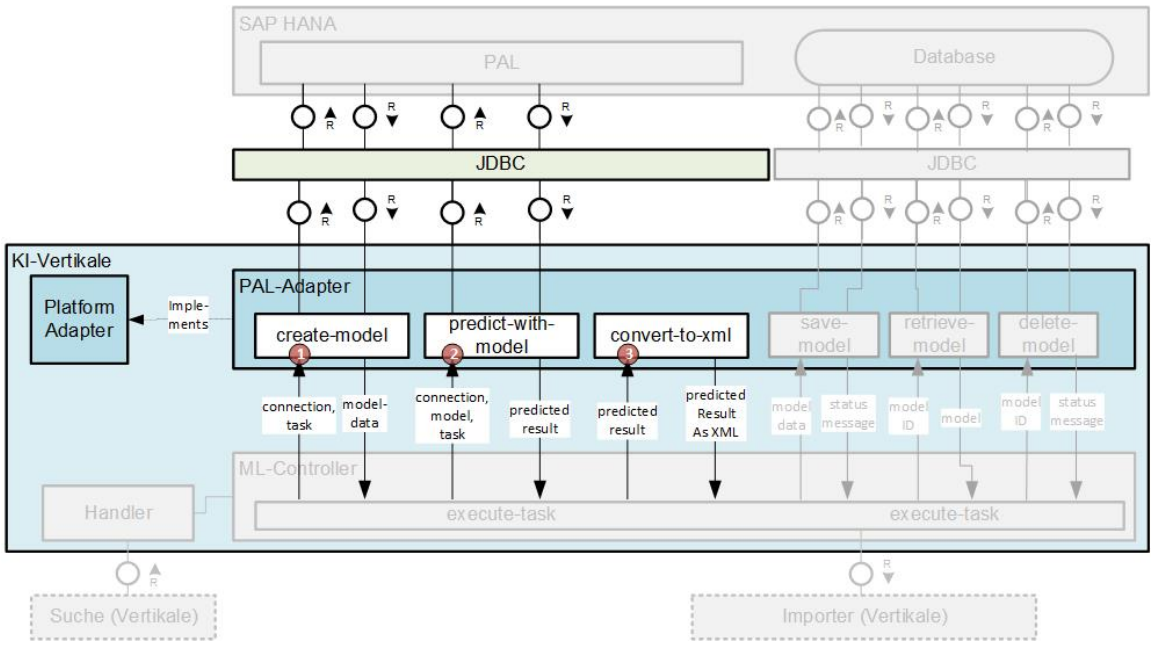

Abbildung 3.17: Adapter-Architektur innerhalb der KI-Vertikalen
Quelle: eigene Darstellung

- **PAL-Adapter**
  Implemetiert die Schnittstellenfunktionen des Platform-Adapter und stellt diese für die Nutzung von PAL bereit

Jeder Platform-Adapter ist nach der Architektur in Abbildung 3.17 aufgebaut und besteht aus diversen Modulen, welche die in 3.3.1 vorgestellten internen Aufgaben ausführen. Diese Module sollen nachstehend visuell dargestellt werden:

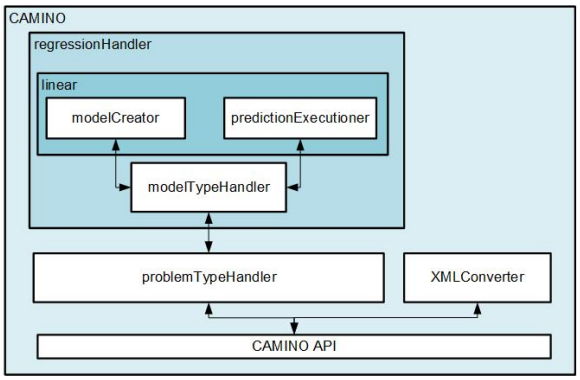

Abbildung 3.18: Module des Adapters
Quelle: eigene Darstellung

Der „problemTypeHandler“ interagiert mit dem „regressionHandler“. Das Modul ist für Regressionsmodelle verantwortlich. Jeder weitere Modelltyp würde als weiteres weitere Modul der Architektur des „regressionHandlers“ folgen, beispielsweise könnte das ein „classificationHandler“ für Problemtypen der Klassifikation sein. Innerhalb des „regressionHandlers“ gibt es den „modelTypeHandler“ der dann die Daten an die jeweiligen Module „modelCreator“, für die Erstellung von Modellen und dem „predictionExecutioner“ für die Erstellung der Vorhersage sendet.

3.4 Zusammenfassung

In diesem Kapitel wurde das Konzept CAMINO entwickelt. Dafür wurden zunächst die Anforderungen für den Adapter aus einem Anwendungsfall abgeleitet. Danach wurden die sich im Systemkontext befindlichen Komponenten näher beleuchtet. In Open60 existiert zum einen eine Datenschnittstelle, die die Input-Daten aus dem User Interface an die KI-Vertikale weiterleitet und zum anderen eine REST-Schnittstelle die Daten zum Importer sendet, damit sie in der Datenbank hinterlegt werden können. Es wurde anschließend der Kontext zu SAP HANA PAL hergestellt und den Aufbau der PAL Tabellen und dessen Funktionalitäten erläutert. Innerhalb der KI-Vertikalen werden die jeweiligen Adapter aufgerufen, die den Machine Learning-Service nutzen und am Ende die Ergebnisdaten bereitstellen. Dafür wurde auf dieser Basis ein Schnittstellendesign entwickelt und die internen Aspekte des Adapters in das Konzept integriert. Dabei wurden Struktur und Verhalten anhand diverser Softwarearchitekturansichten detailliert beschrieben. Im nächsten Kapitel wird anhand einer prototypischen Implementierung ein Proof of Concept erstellt und anhand dessen das Konzept bewertet.

4 Proof of Concept

Nach der Ausarbeitung des Konzepts in Kapitel 3 behandelt dieses Kapitel die Anbindung an die vorhandene Anwendung Open60. Auf Basis dessen folgt im Anschluss die Bewertung des Konzepts und der verwendeten Technologien bei der Umsetzung.

4.1 Anbindung an die vorhandene Anwendung

Mithilfe der Proof of Concept-Implementierung soll die Funktion des Daten-Adapters geprüft werden. Dieser soll die Möglichkeit schaffen, die KI-Vertikale mit einem externen Machine Learning-Service an Open60 anbinden zu können, um dessen Dienste zu nutzen. Dafür wurde das Konzept aus Kapitel 3 exemplarisch in der Programmiersprache Clojure und der Anbindung an die Plattform SAP HANA PAL umgesetzt. Um den Prozess zu testen, wird das Szenario mit dem Ziel: „Die Entwicklung der Korruption im Land Afghanistan in den nächsten 10 Jahren“ betrachtet.

4.1.1 Workflow

1. Der Nutzer stellt eine Suchanfrage. Er wählt den Datensatz GCRI und das Land Afghanistan aus:

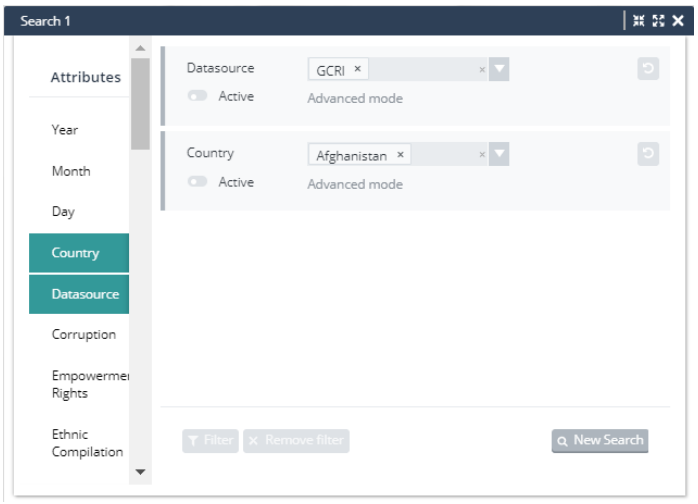

Abbildung 4.1: Suchanfrage im Suche-Fenster
Quelle: Screenshot aus Open60

2. Danach verbindet er das Suche-Fenster mit dem KI-Fenster. Als Algorithmus wird „Lineare Regression“ (Problem-Type: Regression, Target-Model-Type: Linear Model) ausgewählt. Das Attribut ist Korruption (Input-Attribute: corruption) und die Anzahl der Jahre, für die die Vorhersage generiert wird, entspricht 10 (Years To Predict: 10). Der Nutzer betätigt anschließend den Button, um die Vorhersage anzufordern:

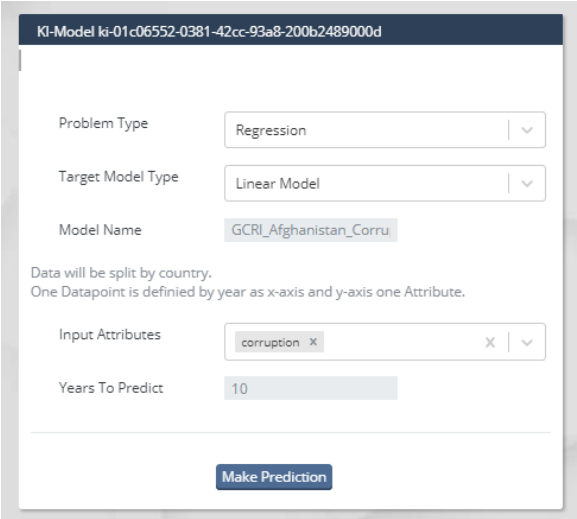

Abbildung 4.2: KI-Fenster
Quelle: Screenshot aus Open60

3. In der prototypischen Implementierung werden in der Konsole des Browsers Statusmeldungen zum aktuellen Stand der Vorhersage ausgegeben[1]. Sobald „prediction done“ ausgegeben wird, stehen die Daten zur Verfügung:

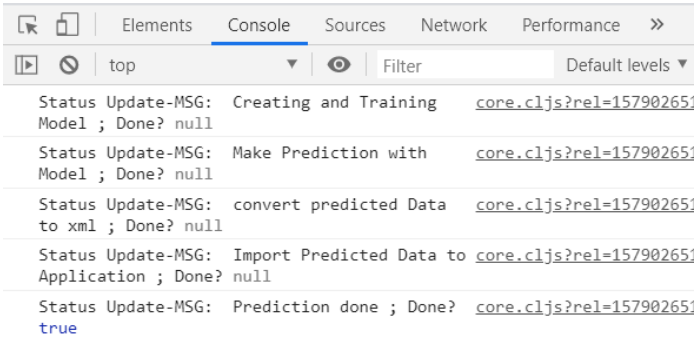

Abbildung 4.3: Konsolenausgabe aus dem Browserfenster mit Statusmeldungen
Quelle: Screenshot aus der Konsole des Browsers

[1]Statusmeldungen werden zukünftig im KI-Fenster für den Nutzer sichtbar sein.

4. Wenn der Nutzer nun eine neue Suche öffnet, sieht er das Ergebnis als neuen Datensatz:

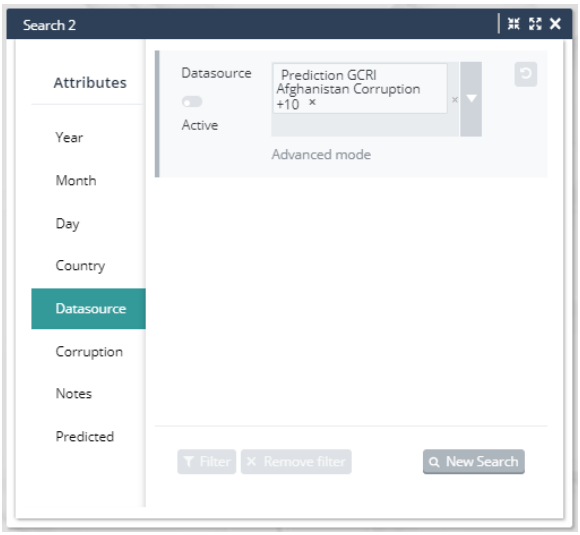

Abbildung 4.4: Daten der Vorhersage im Such-Fenster
Quelle: Screenshot aus Open60

5. Anschließend ist es möglich die Vorhersage beispielsweise durch einen Graphen zu visualisieren. Dabei entspricht die Y-Achse dem Attribut Korruption, die X-Achse zeigt diesen Wert für 10 Jahre.

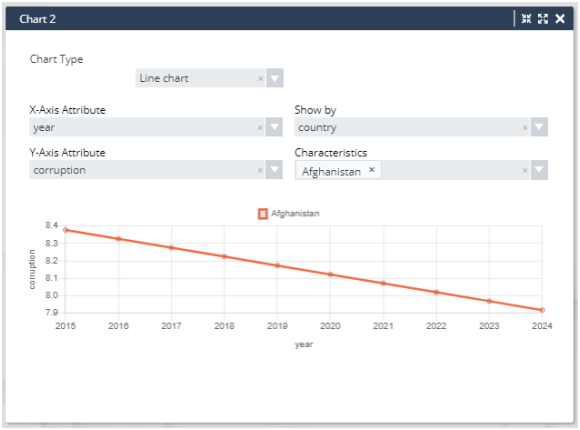

Abbildung 4.5: Daten der Vorhersage in der Visualisierung
Quelle: Screenshot aus Open60

4.1.2 Vergleich zwischen Vorhersage und Echtdaten

Zusätzlich zum gezeigten Workflow des Fallbeispiels kann der Nutzer auch einen Vergleich zwischen Vorhersage und Echtdaten durchführen. Dadurch besteht die Möglichkeit die Ergebnisse der Vorhersage bewerten zu können. In diesem kurzen Beispielszenario sollen die Daten aus Afghanistan für die Jahre 1989-2010 ausgewählt werden und eine Vorhersage für 4 Jahre durchgeführt werden. Das Ergebnis davon entspricht der Prädiktion (rechts). Da die Echtdaten von 2011-2014 (links) schon vorliegen, kann nun ein visueller Vergleich stattfinden:

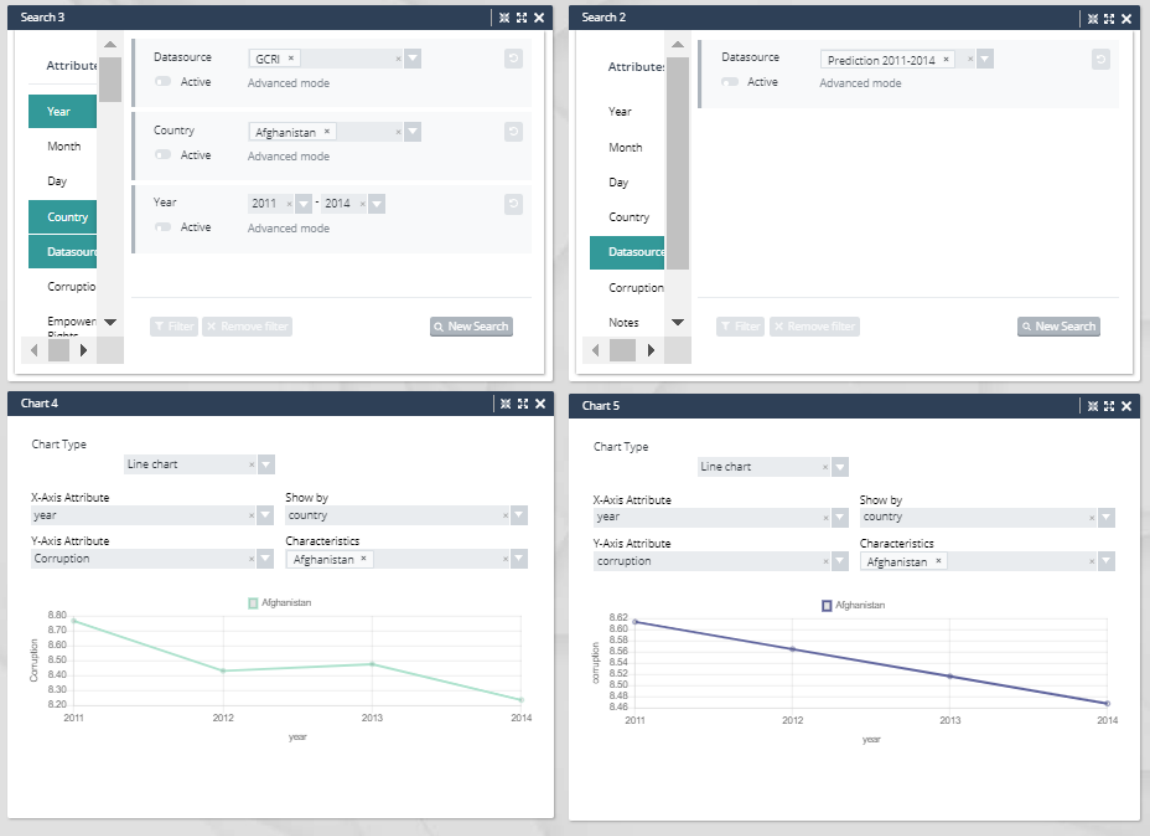

Abbildung 4.6: Echtdaten (links) und die Daten der Vorhersage (rechts) mit zugehöriger Suche im Vergleich
Quelle: Screenshot aus Open60

4.2 Bewertung des Konzepts und der Technologien

In diesem Abschnitt wird das Konzept CAMINO bewertet, welches in Kapitel 3 aufgestellt und anhand einer prototypischen Implementierung umgesetzt wurde. Anschließend sollen die dafür verwendeten Technologien ebenfalls beurteilt werden.

4.2.1 Bewertung des Konzepts

Anhand der Proof of Concept-Implementierung soll begutachtet werden, ob das Ziel, die Nutzung eines externen Machine Learning-Services innerhalb der Software Open60 durch den Einsatz von CAMINO als Daten-Adapter erreicht wurde.
Der Fokus der Implementierung liegt darauf, den Workflow des Nutzers abzubilden und zu zeigen, dass der angebundene Machine Learning-Service SAP HANA PAL durch den Einsatz von CAMINO in der Software Open60 nutzbar ist. Daher wurde in Abschnitt 4.1 ein Leitbeispiel aufgeführt. Dieses soll Schritt für Schritt die Interaktion des Nutzers zeigen.

Die angefragten Daten des Nutzers werden aus der Datenbank geladen und im Suche-Fenster angezeigt. Nach der Auswahl der gewünschten Machine Learning-Kriterien im KI-Fenster können die internen Prozesse (in der Browser-Konsole) verfolgt werden. Es wird schrittweise angezeigt, welcher Prozess gerade ausgeführt wird und der Nutzer somit über den aktuellen Status informiert. Zuerst wird das Modell erstellt und mit den gewählten Trainingsdaten trainiert und direkt für die Vorhersage verwendet. Nachdem die Vorhersage generiert wurde, erfolgt direkt im Anschluss die Konvertierung in das geforderte XML-Format und abschließend der Import als neue Datenquelle. Diese steht nun in der Suche zur Auswahl und lässt sich graphisch visualisieren. Es ist selbstverständlich möglich die Daten auch tabellarisch oder auch in GoOSE[2] als Eventkartenlayout darzustellen. Die Visualisierung eines Graphen bietet an der Stelle jedoch die beste Möglichkeit den Verlauf der Korruption zu begutachten.

Der exemplarische Workflow hat gezeigt, dass das Konzept für die Anbindung externer Machine Learning-Services in Open60 eine gute Basis darstellt. Nachfolgend soll zusätzlich auf die in Kapitel 3 aufgestellten technischen Kriterien bei der Umsetzung eingegangen werden.

Technische Kriterien:

Die Architektur des Adapters hält wurde nach der Vorgabe des Konzepts umgesetzt. Da die Entwicklung in der Programmiersprache Clojure erfolgte, wurden die Module des Adapters, welche in Abbildung 3.18 mittels „Namespaces“ realisiert. Jedes Modul ist einem eindeutigen Namespace zugeordnet, der eine interne Kommunikation innerhalb des Programms ermöglicht. Nach der vorgestellten Architektur können auch weitere Module hinzugefügt werden und somit das angestrebte Kriterium der Erweiterbarkeit erfüllen.

[2] GoOSE, ein Akronym für „Graph oriented Object Set Explorer“, eine Vertikale in Open60, die es ermöglicht Events anhand eines Rasters grafisch darzustellen. Anhand verschiedener Kriterien können diese gefiltert und sortiert werden.

Der Programmablauf folgt ebenfalls dem im Konzept beschrieben Ablauf. Die API-Funktionen „create-model“, „predict-with-model“ und „convert-to-xml“ werden in der prototypischen Umsetzung direkt nacheinander ausgeführt um die Funktionalität zu testen. Am Ende wird dann das Ergebnis via Import in der Anwendung zur Verfügung gestellt.
Das Kriterium „Unabhängigkeit“ im Sinne von unabhängiger Nutzung der API-Funktionen ist aktuell mit der SAP HANA PAL möglich sofern ein Modell im PMML-Format vorliegt. Das wäre beispielsweise bei IBM der Fall. Wenn der Daten-Adapter in späteren Arbeiten Plattformübergreifend für andere Machine Learning-Services wie beispielsweise Google TensorFlow genutzt werden soll, müsste eine Konvertierung in ein anderes Modellformat stattfinden.
Die Kompatibilität mit andern Services sollte in sofern möglich sein, dass die Spezifikation des Interface von diesen übernommen werden kann. Da jeder ML-Service intern einen andere Verarbeitung nach sich zieht, was beispielsweise bei der PAL die Tabellen darstellen, wird sich die Implementierung (die internen Aspekte) aber unterscheiden.

Auffälligkeiten bei der Umsetzung am Beispiel PAL:

Bei der Umsetzung ist deutlich geworden, dass die Nutzung von PAL mit vielen sich wiederholenden Prozessen verbunden ist. Dies bezieht sich besonders auf die in PAL zu verwendeten Prozeduren und zugehörigen Tabellen. Denn bei der Erweiterung der Module (siehe Abbildung 3.18) um weitere Algorithmen ist es notwendig immer neue spezifische Tabellen für den jeweiligen Algorithmus anlegen zu müssen. In zukünftigen Arbeiten sollte daher ein Weg gefunden werden dieses umständliche Verhalten durch einen softwaretechnischen Ansatz zu verbessern.

In Kapitel 3 Abbildung 3.11 ist der gesamte Prozess, der von den Daten durchlaufen wird über alle beteiligten Module hinweg veranschaulicht. In zukünftigen Arbeiten sollte auf eine Verkürzung des Prozesses hingearbeitet werden, da dieser aktuell mit einer hohen Anzahl an Verschiebung der Daten verbunden ist. Dies könnte eine hohe Auswirkungen auf die Performance nach sich ziehen. Durch eine Zwischenspeicherung von Daten oder Modellen in einem zentralen Cache könnte die Performance gesteigert werden und kein erneutes abfragen aus der Datenbank erfordern.

4.2.2 Bewertung der Technologien

Zur Umsetzung des Konzeptes wurden die Technologien Clojure, JDBC und SAP HANA PAL verwendet. Diese sollen im Folgenden bewertet werden.

Technische Umsetzung mit Clojure

Zur Implementierung wurde die Programmiersprache Clojure verwendet. Dabei konnten viele neue Kenntnisse im Bereich der funktionalen Programmierung gewonnen werden. Der Vorteil bei diesen Paradigma ist, dass die Daten im Fokus der Implementierung stehen, was im KI-Kontext einen wünschenswerten Aspekt darstellt. Dies führt zu dem Vorteil „näher“ am Problem zu arbeiten, da kein grundsätzlicher Unterschied zwischen Daten und Programmen existiert.
Das Ziel mit geringem Aufwand fehlerfreie Programme schreiben zu können wird durch die Reduktion von Seiteneffekten, welche den Zustand einer Anwendung beeinflussen, erreicht (siehe Kapitel 2 Abschnitt 2.3.2). Dadurch ist es möglich die Wartung des Programmcodes wesentlich einfacher und schneller zu gestalten.
Des Weiteren ist deutlich geworden, dass Algorithmen in funktionalen Programmen sehr viel kürzer und prägnanter dargestellt werden können, was zu übersichtlicherem und auch eleganterem Code führt.
Diese Aspekte ließen sich auch bei der Implementierung bestätigen und zeigen, dass sie sich für die Entwicklung einer dynamischen und performance-orientierten Software sehr gut eignet. Insbesondere im Kontext der Verarbeitung großer Datenmengen sind die aufgezeigten Eigenschaften von Vorteil.

Die API des Daten-Adapters wurde mit der Clojure-Special-Form „defprotocol“ (siehe Kapitel 2 Abschnitt 2.4.3) umgesetzt. Der Adapter profitiert daher von den im Abschnitt 2.4.3 dargelegten Vorteilen der Protocols, wie beispielsweise der dynamischen Erweiterung von Funktionen.
Folgendes Listing zeigt die Realisierung der CAMINO-API:

```
1  (defprotocol CAMINO-API
2    "Protocol that defines the Adapter-API"
3    (register [this]
4      "Returns false if the registration for the use of this platform has failed.")
5    (create-model [this task]
6      "Returns the created model")
7    (predict-with-model [this task model]
8      "Execute the model and returns the prediction-data")
9    (convert-prediction-to-xml [this model execution-result]
10     "Converts the prediction-data and returns a valid XML structure"))
```

Listing 4.1: CAMINO-API

Um neue ML-Plattformen anzubinden, lässt sich die API durch einen neuen „deftype“ implementieren, was in Listing 4.2 gezeigt werden soll. Durch die Funktion „register“ kann man PAL dort als solche registrieren.

```
(deftype HANA-PAL[conn]
  platform/CAMINO-API
  (register [this]
    (platform/register-platform :HANA-PAL this))
  (create-model [this task]
    (problem-type-handler/create-model (.conn this) task))
  (predict-with-model [this task model]
    (problem-type-handler/predict-with-model (.conn this) task model))
  (convert-prediction-to-xml [this model execution-result]
    (XML-converter/convert-prediction-to-xml (.conn this) model execution-result))
```

Listing 4.2: ML-Platform registriert sich bei der API

Anschließend kann die Schnittstelle mit den PAL-Verbindungsdaten verwendet und aus dem KI-Controller aufgerufen werden. Abbildung 4.7 zeigt die Schnittstelle anhand eines Komponentendiagramms.

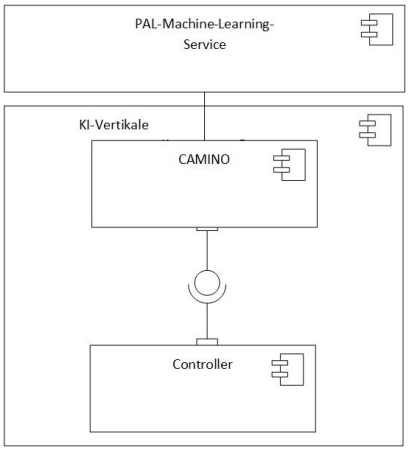

Abbildung 4.7: Komponentendiagramm der API
Quelle: eigene Darstellung

Clojure-JDBC

Um die Funktionen von PAL innerhalb einer Clojure-Umgebung ausführen zu können, wurde JDBC eingesetzt. Dafür existiert „clojure.jdbc“, eine Bibliothek für den JDBC-basierten Datenbankzugriff. Dieser bietet spezielle Funktionen, mit denen es möglich ist, die Datenbankanfragen auszuführen: Es existieren diverse Möglichkeiten von DDL-, DML- und DQL-Befehlen. Es konnten auch „multi-inserts“ verwendet werden, was beim Einfügen großer Datenmengen in Tabellen hilfreich ist. In Listing 4.3 wird gezeigt wie mittels „insert“-Befehl in eine Tabelle Daten eingefügt werden.

```
(jdbc/insert! t-con "PAL_MLR_PDATA_TBL"
["POSITION" "SCHEMA_NAME" "TYPE_NAME" "PARAMETER_TYPE"]
[1 "DM_PAL" "PAL_MLR_DATA_T" "IN"])
```

Listing 4.3: JDBC Insert

SAP HANA PAL

Bei der Einarbeitung in die Technologie wurde festgestellt, dass die Statements für die Tabellen bei vorhandenen Kenntnissen in SQL gut verständlich waren, da sie in einer ähnlichen Sprache, nämlich SQL-Script geschrieben sind. PAL verfügt über eine strukturierte Online-Dokumentation, die man zuverlässig verwenden kann. Auch bei Fehlern wurden entsprechende Exceptions ausgegeben, die meistens aufschlussreich waren, die Fehlerursache zu beheben. Über die gerade vorgestellte Bibliothek Clojure-JDBC konnten die Statements an PAL gesendet werden. In der Web-IDE von SAP HANA kann man diese Tabellen und Prozeduren auch anzeigen lassen, was folgender Screenshot zeigen soll:

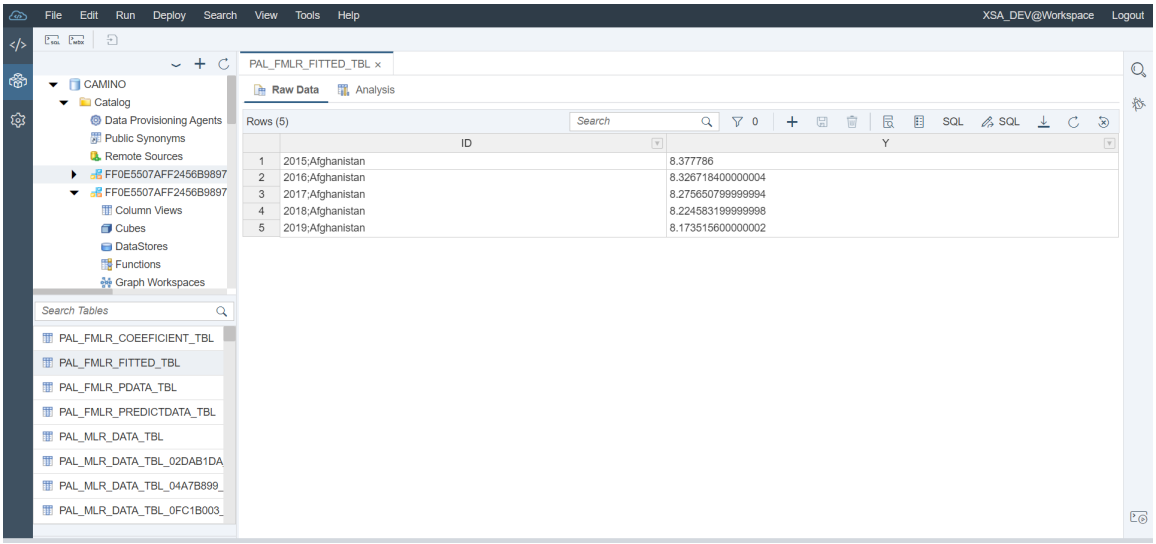

Abbildung 4.8: Ansicht der Fitted-Table in der SAP HANA-Web-IDE
Quelle: eigene Darstellung

5 Zusammenfassung und Ausblick

Machine Learning stellt eine sehr nützliche Technologie für Entscheidungsunterstützungssysteme dar und gewinnt in Forschung und Praxis zunehmend an Bedeutung. Es besteht daher die Herausforderung intelligente Entscheidungsunterstützungssysteme zu entwickeln, die den Anwender zusätzlich beratend unterstützen können. Es sollte aufgezeigt werden, wie sich Situationen möglicherweise in naher Zukunft entwickeln, um daraufhin Entscheidungen zielgerichteter treffen zu können.

5.1 Zusammenfassung

In dieser Arbeit wurde ein Weg entwickelt, Machine Learning im Entscheidungsunterstüzungssystem Open60 einsetzen zu können. Dies wurde durch einen speziell zwischen der KI-Vertikalen und dem Machine Learning-Service eingebauten Daten-Adapters möglich.
In einem ersten Schritt wurde in Kapitel 2 Grundlagen für die Erstellung eines Konzepts behandelt. Dabei wurde zunächst die Technologie SAP HANA & PAL betrachtet. Diese bietet aufgrund ihrer Architektur die Möglichkeit einer schnellen Bereitstellung von Daten. Die Bibliothek PAL, die innerhalb von SAP HANA genutzt werden kann, bietet die Möglichkeit Machine Learning-Technologien zu verwenden. Im darauffolgenden Abschnitt wurde das Thema Machine Learning näher erläutert, eine Möglichkeit, um aus vorhandenen Daten mittels mathematischer Algorithmen Vorhersagen zu generieren. Diese stellen daher eine Möglichkeit dar, computergestützten Systemen die Fähigkeit zu geben anhand von Trainingsdaten zu lernen und eigenständig bestimmte Probleme zu lösen. Mit der Programmiersprache Clojure wurde eine Sprache vorgestellt, die das Paradigma der funktionalen Programmierung unterstützt, was sich in hochdynamischen und datengetriebenen Anwendungen hervorragend eignet. Zudem wurde das Architekturpattern Adapter näher betrachtet, da es als Grundlage für das nachfolgende Konzept diente. Ein Adapter verbindet Softwarekomponenten, die sonst inkompatibel zueinander sind und ermöglicht eine reibungslose Kommunikation dieser. Abschließend wurde kurz auf die in Open60 zugrundeliegende vertikale Softwarearchitektur eingegangen, um zu verdeutlichen in welchem Kontext sich die Arbeit befindet.

Das Konzept CAMINO wurde in Kapitel 3 vorgestellt. Die Anwendung von CAMINO, der den Daten-Adapter darstellt, ermöglicht die Nutzung von Machine Learning-Services in Verbindung mit der KI-Vertikalen. Nach der Erläuterung eines Anwen-

dungsfalls wurden zunächst die funktionalen Anforderungen an den Daten-Adapter ermittelt. Dabei wurden auch spezielle Anforderungen an die Verwendung des Machine Learning Service SAP HANA PAL herausgearbeitet.
Anschließend wurde der übergeordnete Kontext in Open60 betrachtet. Dabei fand eine Analyse der Datenschnittstellen anhand von diversen Softwarearchitektursichten statt. Zusätzlich erfolgte eine ausführliche Analyse von SAP HANA PAL und der internen Aspekte, die von diesem Service zu erwarten sind. Dies wurde am Beispiel des Algorithmus Lineare Regression dargestellt.
Auf dieser Basis folgte die Ausarbeitung des Konzepts wobei auf die API, die internen Aspekte und die Fehlerbehandlung der Funktionen eingegangen wurden. Diese wurden semantisch und strukturell ausgearbeitet. Danach folgte die Erstellung der Architektur und der Module, die den Adapter abbilden.

Im Kapitel Proof of Concept wurde dieses Konzept anhand der Durchführung eines Leitbeispiels bewertet. Dieses ergab, dass die Anbindung an die vorhandene Anwendung erfolgreich war. Es konnte mittels vorgegebener Eingabedaten eine Vorhersage für ein bestimmtes Zielattribut erstellt werden. Dafür wurde exemplarisch der Algorithmus Lineare Regression verwendet. Dem Benutzer ist es damit möglich Vorhersagen zur Entscheidungsunterstützung zu generieren.
Danach wurde im Detail auf programmatische Aspekte in der Umsetzung eingegangen und die Technologien Clojure, JDBC und SAP HANA PAL bewertet.

5.2 Ausblick & Fazit

Mit CAMINO konnte ein Daten-Adapter konzipiert werden, der den Machine Learning-Service PAL an die KI-Vertikale anbindet und durch diese verwendet werden kann. Dies wurde anhand eines Leitbeispiels in Abschnitt 4.1 erfolgreich aufgezeigt. Die konzipierte API ist durch zusätzliche Funktionen erweiterbar, die Funktionen der API sind voneinander unabhängig oder in Kombination nutzbar und deren Struktur auch mit anderen Machine Learning-Services, die zukünftig angebunden werden können möglich.
Die prototypische Implementierung fand anhand eines Leitbeispiels statt und ist in dieser Form daher nur auf den Machine Learning-Service SAP HANA PAL und dem Algorithmus der Linearen Regression beschränkt. Dies ist darin begründet, dass jeder PAL-Algorithmus auf jeweils anderen, dafür spezifischen Tabellen aufbaut. Die Tabellen für die Lineare Regression können beispielsweise nicht für das Training und die Vorhersage mit einem „Multilayer Perceptron“ genutzt werden. Zudem können angelegte Tabellen nicht wiederverwendet werden und existieren daher in einem temporären Speicher. Durch diesen Umstand müssen die dafür zuständigen Programmfunktionen zuerst erstellt werden, was für zukünftige Arbeiten vorgesehen wird. Erst dann ist es auch möglich andere PAL-Algorithmen in Open60 zu nutzen.

Für zukünftige Entwicklungen sind ebenfalls folgende Aspekte interessant:

- **Erweiterung um zusätzliche PAL-Algorithmen:** Softwaretechnischer Ansatz zur Erweiterung um PAL-Algorithmen ohne Redundanzen bei der Erstellung der Tabellen ermöglichen.
- **Persistenz von Modellen:** Dies kann zukünftig innerhalb der API angebunden werden. Dafür wurde bereits eine Funktion entworfen, die diese Aufgabe ermöglichen kann. Damit könnten bereits erstellte Modelle wiederverwendet oder auch gelöscht werden. Hierfür wurden ebenfalls bereits Ansätze in der API entworfen.
- **Verfügbarkeit mehrerer Machine Learning-Services:** Die Anbindung anderer Machine Learning-Plattformen wie beispielsweise „Google Tensor Flow“ oder „Amazon Web Services“.
- **Parallele Nutzung mehrerer ML-Services:** Es ist zukünftig denkbar, dass sogar mehrere Plattformen parallel diese Machine Learning Operationen ausführen, um das Ergebnis vergleichen zu können und das Beste daraus zu ermitteln.
- **Modellbewertung:** Eine Möglichkeit der Modellbewertung anhand der Modellgüte oder eines Vergleiches verschiedener Machine Learning-Services, die parallel genutzt werden.
- **Abbruch des Prädiktionsvorgangs ermöglichen:** Bei Abbruch des Prädiktionsvorgangs sollte die entsprechende Hintergrundaufgabe umgehend beendet werden und das System direkt wieder für neue Anfragen bereitstehen.
- **REST-API:** Umsetzung mit der API mittels REST anstatt der Clojure Protocols: Clojure Protocols bieten zwar eine praktische Verwendung an, jedoch wäre eine unabhängige Einbindung durch HTTP-REST noch vorteilhafter. Denn damit ist kein neues Kompilieren des Service bei jeder Änderung nötig und es fügt sich besser in die Micro-Service-Architektur (vorgestellt in Abschnitt 2.5) ein. Diese basiert auf dem Einsatz kleiner, voneinander unabhängiger Komponenten, die sich dynamisch erweitern lassen. Ein Nachteil von REST-APIs ist jedoch dass sie statuslos sind. Dies bedeutet, dass Aufrufe unabhängig voneinander erfolgen können und jeder Aufruf alle Daten enthält, die für eine erfolgreiche Ausführung erforderlich sind. Diese redundanten Informationen fallen zur Last des Systems und mindern unter umständen die Performance.
- **Alternativer Import der Prädiktionsdaten:** Eine Möglichkeit die Vorhersagedaten nicht über das XML-Schema an den Importer zu senden, sondern direkt aus der KI-Vertikalen in einem entsprechenden Format.
- **Benutzeroberfläche:** Die Darstellung der Benutzeroberfläche sollte durch direkte Statusmeldungen erweitert werden. Es sollte zudem eine Möglichkeit bieten die Plattform auszuwählen, die für das Machine Learning verwendet wird.

CAMINO ist ein tragfähiges Konzept und lässt die gerade beschriebenen Aspekte für zukünftige Entwicklungen zu. Das in Kapitel 1.1 gesetzte Ziel der Anbindung eines Machine Learning-Service an die KI-Vertikale von Open60 wurde erreicht.

Anhang

```
1 DROP TYPE PAL_MLR_DATA_T;
```

Listing 1: Löschen des Data TableType

```
1 CREATE TYPE PAL_MLR_DATA_T AS TABLE
2 ("ID" varchar(50), "Y" DOUBLE, "X1" DOUBLE,
3    "X2" varchar(100)
4 );
```

Listing 2: Anlegen des Data Table Type

```
1 CREATE COLUMN TABLE PAL_MLR_PDATA_TBL
2 ("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
3 "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
```

Listing 3: Parametertabelle PDATA

```
1 INSERT INTO PAL_MLR_PDATA_TBL VALUES (1,'DM_PAL','PAL_MLR_DATA_T','IN');
2 INSERT INTO PAL_MLR_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
3 INSERT INTO PAL_MLR_PDATA_TBL VALUES (3,'DM_PAL','PAL_MLR_RESULT_T','OUT');
4 INSERT INTO PAL_MLR_PDATA_TBL VALUES (4,'DM_PAL','PAL_MLR_FITTED_T','OUT');
5 INSERT INTO PAL_MLR_PDATA_TBL VALUES (5,'DM_PAL','PAL_MLR_SIGNIFICANCE_T','OUT');
6 INSERT INTO PAL_MLR_PDATA_TBL VALUES (6,'DM_PAL','PAL_MLR_PMMLMODEL_T','OUT');
```

Listing 4: Parametertabelle PDATA

```
1 CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE
2 ('AFLPAL','LRREGRESSION','DM_PAL','PAL_LR_PROC',PAL_MLR_PDATA_TBL);
```

Listing 5: Erstellen des Wrappers

```
1 CREATE COLUMN TABLE PAL_MLR_DATA_TBL LIKE PAL_MLR_DATA_T;
```

Listing 6: Erstellen der Data Table

```
1 INSERT INTO PAL_MLR_DATA_TBL VALUES (0, -6.879, 0.00, 'A');
2 ...
```

Listing 7: Befüllen der Data Table

```
1 CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL
```

Listing 8: Control Table

```
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT',0,null,null);
```

Listing 9: Werte in die Control Table einfügen

```
CREATE COLUMN TABLE PAL_MLR_RESULTS_TBL LIKE PAL_MLR_RESULT_T;
```

Listing 10: Result Table erstellen

```
CALL DM_PAL.PAL_LR_PROC
(PAL_MLR_DATA_TBL,
"#PAL_CONTROL_TBL",
PAL_MLR_RESULTS_TBL,
PAL_MLR_FITTED_TBL,
PAL_MLR_SIGNIFICANCE_TBL,
PAL_MLR_PMMLMODEL_TBL)
WITH OVERVIEW;
```

Listing 11: Wrapper-Prozedur ausführen

```
SELECT * FROM PAL_MLR_PMML_TBL;
```

Listing 12: Modell abrufen

Quelle: SAP Online (2019)

```
<data-set>
    <context-set>
        <context type='country'>
            <global-id>5f1d61e113568ea6dff6b301a2fd0808</global-id>
            <name>Afghanistan</name>
        </context>
    </context-set>
    <datasource-set>
        <datasource>
            <global-id>Source;42da0e6e-15e4-4e0d-9836-c6e94fb51750"</global-id>
            <name>testme</name>
            <url>42da0e6e-15e4-4e0d-9836-c6e94fb51750</url>
        </datasource>
    </datasource-set>
    <item-set type='predicted'>
        <global-id>item-set;42da0e6e-15e4-4e0d-9836-c6e94fb51750"</global-id>
        <datasource>
            <link>Source;42da0e6e-15e4-4e0d-9836-c6e94fb51750"</link>
        </datasource>
        <items>
            <item type='predicted'>
                <global-id>f8fc1bfe869fde277ca9b781417e502a12b6eb06f17</global-id>
                <interaction type='predicted'>testme</interaction>
                <date type='iso-8601'>2019-01-01</date>
                <properties>
                    <property name='corruption' type='decimal'>8.377786</property>
                </properties>
                <contexts>
                    <context>
                        <link>5f1d61e113568ea6dff6b301a2fd0808</link>
                    </context>
                </contexts>
                <locations>
                    <location>
                        <position>
                            <link>5f1d61e113568ea6dff6b301a2fd0808</link>
                        </position>
                    </location>
                </locations>
                <notes>Predicted with Model Prediction Afghanistan Corruption +10
                </notes>
            </item>
        </items>
    </item-set>
</data-set>
```

Listing 13: Ein Beispiel für die in XML überführten Ergebnisdaten

Akronyme

AFL Application Function Library

API Application Programming Interface

BFL Business Function Library

CAMINO **C**onnecting the **A**I-Vertical to **M**achine Learning **In**terfaces in **O**pen60

DDL Data Definition Language

DML Data Modeling Language

DQL Data Query Language

GCRI Global Conflict Risk Index

GoOSE Graph oriented Object Set Explorer

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

JDBC Java Database Connectivity

KI Künstliche Intelligenz

MDX Multidimensional Expressions

ML Machine Learning

PAL Predictive Analytics Library

PMML Predictive Model Markup Language

RAM Random Access Memory

REST Representational State Transfer

SAP HANA SAP High Performance Analytic Appliance

SQL Structured Query Language

XML Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

Berg, B., Silvia, P. & Frye, R.: *SAP HANA. Die neue Einführung*, 3. und aktualisierte Auflage, Rheinwerk Publishing 2017

Clojure.org https://clojure.org 2019, letzter Zugriff: 20.02.2020

Cleve, J., Lämmel, U. *Data Mining*, De Gruyter Studium 2014

Data Mining Group http://dmg.org/pmml/products.html 2020, letzter Zugriff: 07.02.2020

Dumbill, E.,Croll, A., Steele, J. & Loukides, M.K. *Planning for Big Data*, Beijing: O'Reilly Media 2012

Eilebrecht, K. & Starke, G.: *Patterns kompakt: Entwurfsmuster für effektive Softwareentwicklung*, Springer-Verlag 2019

Emerick, C., Carper, B. & Grand, C.: *Clojure Programming: Practical Lisp for the Java World*, O'Reilly Media Inc. 2012

Essinger, S. D. & Rosen, G. L.: „An introduction to machine learning for students in secondary education", *Digital Signal Processing and Signal Processing Education Meeting*, 2011, S. 243-248.

Fowler, M., & Lewis, J. : „Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr", *Objektspektrum*, vol. 1(2015), S. 14-20

Frochte, J.: *Maschinelles Lernen, Grundlagen und Algorithmen in Python*, 2. aktualisierte Auflage, Carl Hanser Verlag GmbH & Co. KG 2019

Gamma, E., Helm, R., Johnson, R. & Vlissides, J.: *Elements of Reusable Object-Oriented Software*, Design Patterns. Addison-Wesley Publishing Company 1995

Goll, J.: *Entwurfsprinzipien und Konstruktionskonzepte der Softwaretechnik*, 2. akualisierte Auflage, Springer Vieweg 2018

Grossman, R. & Bailey, S. & Ramu, A. & Malhi, B. & Hallstrom, P. & Pulleyn, I. & Qin, X. „The management and mining of multiple predictive models using the predictive modeling markup language."*Information and Software Technology.* vol. 41, 2002

Grus, J.: *Einführung in Data Science: Grundprinzipien der Datenanalyse mit Python*, O'Reilly 2016

Higginbotham, D. *Clojure for the brave and true: learn the ultimate language and become a better programmer*, No Starch Press 2015

Hofstedt, P.: *Die Evolution der Programmiersprachen der KI*, Springer 2012

Hudak, P. „Conception, evolution, and application of functional programming languages". *ACM Computing Surveys (CSUR)*, vol. 21, 1989

Hunter, K. L.: *Irresistible APIs: designing web APIs that developers will love*, Manning Publications Co. 2017

Knopfel, A., Grone, B. & Tabeling P. *Fundamental Modeling Concepts: Effective Communication of IT Systems*, John Wiley & Sons Inc. USA 2006

Kraus, S., Steinacker, G., & Wegner, O. (2013).: „Teile und Herrsche: Kleine Systeme für große Architekturen", *OBJEKTspektrum*, (5)3013, S. 8-13

Louridas, P. & C. Ebert: „Machine learning", *IEEE Software*, vol. 33, 2016, S. 110 115

Patgiri R. & Ahmed A., „Big Data: The V's of the Game Changer Paradigm", *IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, 2016, S. 17-24

Plattner, H.: „SanssouciDB: an in-memory database for processing enterprise workloads". In: Härder, T., Lehner, W., Mitschang, B., Schöning, H. & Schwarz, H. (Hrsg.), *Datenbanksysteme für Business, Technologie und Web (BTW)* Bonn: Gesellschaft für Informatik e.V. 2011, S. 2-21

Prassol, P., Fasel, D. & Meier, A.: „In-Memory-Platform SAP HANA als Big Data-Anwendungsplattform", In: Fasel D., Meier A. (eds) *Big Data*, Edition HMD. Springer Vieweg 2016

Raschka, S. & Mirjalili, V.: *Machine Learning mit Python: Das Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning*, Packt Publishing Ltd 2017

Rashid, T.: *Neuronale Netze selbst programmieren: ein verständlicher Einstieg mit Python.*, O'Reilly 2017

Sallam, R. L. & Parenteau, J.: *Smart data discovery will enable a new class of citizen data scientist.*, Gartner Research 2015

SAP SE : *SAP HANA Modeling Guide for SAP HANA Studio, SAP HANA Platform 2.0 SPS 03*, https://help.sap.com/doc/fb8f7a9f7860468b84a07eab0a7d0a98/2.0.03/en-US/SAP_HANA_Modeling_Guide_for_SAP_HANA_Studio_en.pdf, 2018, letzter Zugriff: 20.02.2020

SAP SE : *SAP HANA Developer Guide for SAP HANA Studio, SAP HANA Platform 2.0 SPS 03*, https://help.sap.com/doc/fbb802faa34440b39a5b6e3814c6d3b5/2.0.03/en-US/SAP_HANA_Developer_Guide_for_SAP_HANA_Studio_en.pdf, 2018, letzter Zugriff: 20.02.2020

SAP SE : *SAP HANA Predictive Analytics Library (PAL), SAP HANA Platform 2.0 SPS 03*, https://help.sap.com/doc/86fb8d26952748debc8d08db756e6c1f/2.0.03/en-US/SAP_HANA_Predictive_Analysis_Library_PAL_en.pdf, 2018, letzter Zugriff: 20.02.2020

SAP SE: *SAP Online Documentation* https://help.sap.com/viewer/2cfbc5cf2bc14f028cfbe2a2bba60a50/2.0.01/en-US/eedc9094daf04419bc25f6ed097ac03b.html, letzter Zugriff: 20.02.2020

Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., & Tufano, P.: „Analytics: The real-world use of big data.“*IBM Global Business Services*, vol. 12(2012), 2012, S. 1-20

Spichale, K.: *API Design*, dpunkt Verlag 2017

Starke, G.: *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*, 8. aktualisierte Auflage, Carl Hanser Verlag GmbH Co KG 2018

Tate, B. A.: *Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages*, Pragmatic Bookshelf 2010

Tavares de Sousa, N., Hasselbring, W., Weber, T., & Kranzlmüller, D.: „Designing a generic research data infrastructure architecture with continuous software engineering“, *CEUR Workshop Proceedings*, vol. 2066, 2018

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Sabrina Göllner