

Erklärbarkeit und Visualisierung von Graph Codes mittels generativer KI

Bachelorarbeit
im Studiengang B.Sc. Informatik

eingereicht von
Jens Nathan Andreß
(Matrikelnummer 9763180)

angefertigt am
**Lehrgebiet Multimedia
und Internetanwendungen**

Betreuer:
Prof. Dr.-Ing. Stefan Wagenpfeil

30. Dezember 2023



Inhaltsverzeichnis

Inhaltsverzeichnis	i
Abbildungsverzeichnis	iv
Tabellenverzeichnis	vi
Listingsverzeichnis	vi
1 Einleitung	1
1.1 Motivation	3
1.2 Problembeschreibung	5
1.2.1 Probleme in Bezug auf die Erklärbarkeit	5
1.2.2 Probleme in Bezug auf die Integration	8
1.2.3 Allgemeine Zielsetzung	8
1.3 Forschungsfragen	9
1.4 Methodik und Ziele	9
1.5 Ansatz und Aufbau der Arbeit	11
1.6 Zusammenfassung	13
2 Stand der Wissenschaft und Technik	14
2.1 FZ 1.1/O Erklärbarkeit von MMIR mittels generativer KI	15
2.1.1 GMAF	15
2.1.2 MMFG	16
2.1.3 Graph Codes	21
2.1.4 Generative KI	28
2.1.5 Diskussion	36
2.2 FZ 2.1/O Integration generativer KI in das GMAF	47
2.2.1 Integrationsmöglichkeiten von Graph Codes	47
2.2.2 Integrationsmöglichkeiten von Systemen generativer KI	49
2.2.3 Diskussion	52
2.3 Evaluierungsmethodiken	54
2.4 Zusammenfassung	57
2.4.1 Welche offenen Herausforderungen (OH) verbleiben?	57
2.4.2 Gewonnene Erkenntnisse	58
2.4.3 Getroffene Entscheidungen	59
2.4.4 Behandlung offener Herausforderungen	59
3 Modellierung	62
3.1 Konzeptions- und Modellierungsmethodik	62
3.1.1 Rational Unified Process	63

3.1.2	Unified Modeling Language	64
3.1.3	Informations- und Datenmodell	65
3.2	FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI	66
3.2.1	Erklärbarkeit durch generative KI	66
3.2.2	Anwendungsfälle	66
3.2.3	Wireframe für die Interaktion mit Graph Codes	72
3.2.4	Mechanismen	74
3.2.5	UML-Sequenzdiagramme	80
3.2.6	Diskussion	84
3.3	FZ 2.2/TB Integration generativer KI in das GMAF	86
3.3.1	Transformation von Graph Codes	86
3.3.2	Einbindung von Systemen generativer KI	90
3.3.3	Diskussion	95
3.4	Zusammenfassung	96
3.4.1	Gewonnene Erkenntnisse	96
3.4.2	Weiteres Vorgehen	97
4	Implementierung	100
4.1	Grundlagen der Implementierung	100
4.1.1	Architekturmuster	101
4.1.2	Abhängigkeiten (Maven)	101
4.2	FZ 1.3/I Erklärbarkeit von MMIR mittels generativer KI	104
4.2.1	Benutzungsschnittstelle	104
4.2.2	Diskussion	135
4.3	FZ 2.3/I Integration generativer KI in das GMAF	136
4.3.1	Transformation von Graph Codes	136
4.3.2	Integration der Endpunkte	140
4.3.3	Diskussion	143
4.4	Zusammenfassung	144
4.4.1	Gewonnene Erkenntnisse	144
4.4.2	Weiteres Vorgehen	151
5	Evaluierung	154
5.1	Ziele der Evaluierung und geeignete Evaluierungsmethodik	154
5.2	Wahl und Zuweisung durchzuführender Experimente	156
5.3	FZ 1.4/E Erklärbarkeit von MMIR mittels generativer KI	157
5.3.1	Vorbereitung	157
5.3.2	Durchführung	158
5.3.3	Ergebnisse und Diskussion	160
5.4	FZ 2.4/E Integration generativer KI in das GMAF	162
5.4.1	Vorbereitung	162
5.4.2	Durchführung	163

5.4.3	Ergebnisse und Diskussion	165
5.5	Anpassung der prototypischen Proof-of-Concept Implementierung . .	167
5.6	Zusammenfassung	173
5.6.1	Gewonnene Erkenntnisse	173
6	Zusammenfassung und Diskussion	177
6.1	Einordnung der Arbeitsergebnisse	177
6.2	Verbleibende Herausforderungen und „Future Work“	180
6.3	Persönliches Fazit	181

Abbildungsverzeichnis

1.1	Begriffe im Rahmen von künstlicher Intelligenz.	2
1.2	Beispiel für eine generierte Erklärung	4
1.3	Bauanleitung für ein Lego-Set [20].	6
1.4	Beispiel für ein generiertes Vorschaubild	6
1.5	HeadUp-Display in einem Kraftfahrzeug [24].	7
1.6	Benutzungsschnittstelle im <i>GMAF</i> zum Anpassen von Erklärungen. .	7
1.7	Phasen der Problemlösung nach Nunamaker [25].	10
1.8	Aufbau der Arbeit und Ansatz der Problemlösung nach Nunamaker [25].	12
2.1	Vereinfachte Übersicht des GMAFs (ohne UI / API) nach [22].	16
2.2	Visualisierung eines MMFGs durch yEd.	18
2.3	Beispiel für den Multimedia Feature Graphen $MMFG_{ex}$	19
2.4	Wertungsmatrix des Beispiels $MMFG_{ex}$ in a) Tabellenform und b) Form einer Matrix.	22
2.5	Codierte Wertungsmatrix bzw. Graph Code GC_{ex} zum entsprechenden Beispiel $MMFG_{ex}$ in a) Tabellenform und b) in Form einer Matrix. c) Vom GMAF generierte Graph Code GC_{ex}	23
2.6	Übersicht ausgewählter aktueller Systeme generativer KI nach [84]. .	35
2.7	Beispiel für ein von Midjourney generiertes Bild anhand oben genannter Textbeschreibung [88].	37
2.8	Beispiel für ein von Stable Diffusion generiertes Bild anhand oben genannter Textbeschreibung.	39
2.9	Beispiel für ein von DALL · E 2 generiertes Bild anhand oben genannter Textbeschreibung.	40
3.1	Modellierung zum Vorgehen des Rational Unified Process nach [126].	63
3.2	UML Use-Context (System) und UML Use-Cases (Anwendungsfälle). .	67
3.3	Wireframe für die Grundbereiche der Benutzungsschnittstelle.	72
3.4	Wireframe für den linken Arbeitsbereich (links) und den rechten Arbeitsbereich (rechts) der Benutzungsschnittstelle.	73
3.5	Komplettes Wireframe für die Benutzungsschnittstelle.	74
3.6	Mechanismus für den Anwendungsfall UC-1.1.	75
3.7	Mechanismus für den Anwendungsfall UC-1.2.	76
3.8	Mechanismus für den Anwendungsfall UC-1.3.	76
3.9	Mechanismus für den Anwendungsfall UC-1.5.	77
3.10	Beispielhafte Visualisierung für die Berechnung der Gemeinsamkeiten zweier Graph Codes GC_1 und GC_2	78
3.11	Beispielhafte Visualisierung für die Berechnung der symmetrische Differenz bzw. Unterschiede zweier Graph Codes GC_1 und GC_2	80

3.12	(i) Synchrone Nachricht, (ii) Asynchrone Nachricht, (iii) Antwort auf Nachricht, (iv) Selbstdelegation / Selbstaufruf.	80
3.13	UML-Sequenzdiagramm für den Anwendungsfall UC-1.1.	81
3.14	UML-Sequenzdiagramm für den Anwendungsfall UC-1.2.	82
3.15	UML-Sequenzdiagramm für den Anwendungsfall UC-1.3.	83
3.16	UML-Sequenzdiagramm für den Anwendungsfall UC-1.5.	84
3.17	Einfaches Klassendiagramm für die Erklärungspanelee.	87
3.18	Endpunkte der OpenAI Schnittstelle.	91
3.19	Erweiterung zum UML-Sequenzdiagramm für den Anwendungsfall UC-1.3.	93
3.20	Mechanismus für den Anwendungsfall UC-1.8.	94
3.21	UML-Sequenzdiagramm für den Anwendungsfall UC-1.8.	94
3.22	Klassendiagramm.	97
4.1	Hierarchie des Quellcodes.	101
4.2	Allgemeine Übersicht über das Mockup der Benutzungsschnittstelle.	104
4.3	Grundbereich <i>EditorGraphCode</i>	106
4.4	Aus der Implementierung resultierende Schnittstelle zum Auswählen und Ausführen von Aktionen.	109
4.5	Aus der Implementierung resultierende Liste mit beispielhaften Einträgen für Graph Codes.	110
4.6	Aus der Implementierung resultierende Tabelle zur Darstellung eines ausgewählten Graph Codes.	112
4.7	Vollständige Oberfläche des aus der Implementierung resultierenden Grundbereichs <i>EditorGraphCode</i>	113
4.8	Grundbereich <i>ExplanationPanel</i> zur Generierung von visuellen Erklärungen (links) und textuellen Erklärungen (rechts).	114
4.9	Textfeld für die Prompt und erweiterte Optionen.	117
4.10	Vollständige Schnittstelle für die Klasse <i>TextPanel</i>	118
4.11	Vollständige Schnittstelle für die Klasse <i>ImagePanel</i>	121
4.12	Benutzungsschnittstelle zum Generieren einer visuellen Erklärung.	146
4.13	Benutzungsschnittstelle zum Generieren einer textuellen Erklärung.	147
4.14	Liste mit einer Reihe an importieren Graph Codes.	148
4.15	Auswahl eines Graph Codes und eine generierte, visuellen Erklärung bzw. ein Bild.	149
4.16	Auswahl eines Graph Codes und eine generierte, textuelle Erklärung.	150
5.1	Umordnung der Layouts von <i>ImagePanel</i> und <i>TextPanel</i>	168
5.2	Registerkartenpanel in der Benutzungsschnittstelle <i>ImagePanel</i> mit zwei Registerkarten.	169
5.3	Ausgewählte Registerkarte mit der Darstellung der mit einem Graph Code ursprünglich assoziierten Datei.	170

5.4	Gegenüberstellung eines Graph Codes, der originalen Datei und der erzeugten visuellen Erklärung.	171
5.5	Informationsoberfläche für Tokens in einem generierten Text.	171
5.6	Benutzungsschnittstelle für den Erklärungstyp Audio.	172

Tabellenverzeichnis

1.1	Aufschlüsselung der Forschungsfragen auf Forschungsziele	11
2.1	Struktur dieses Kapitels.	14
2.2	$dict_{ex}$ vom Graph Code GC_{ex}	24
2.3	Offene Herausforderungen	57
2.4	Tabelle zur Übersicht des aktuellen Arbeitsstands.	60
3.1	Struktur dieses Kapitels.	62
3.2	Graph Code zum Beispiel der folgenden Transformationen.	87
3.3	Tabelle zur Übersicht des aktuellen Arbeitsstands.	98
4.1	Struktur des Kapitels 4 „Implementierung“	100
4.2	Tabelle zur Übersicht des aktuellen Arbeitsstands.	151
5.1	Tabelle zur Übersicht des aktuellen Arbeitsstands.	174

Algorithmenverzeichnis

3.1	Berechne Gemeinsamkeiten	78
3.2	Berechne Unterschiede	79
3.3	Anfrage an den Endpunkt Text (<i>ChatCompletionRequest</i>).	92

Listingsverzeichnis

2.1	Beispiel des Graph Codes GC_{ex} im JSON-Format.	47
4.1	ExplainerFrame-Klasse	105
4.2	EditorGraphCode-Klasse	106
4.3	EditorGraphCode-Klasse (Zweiter Teil)	107
4.4	EditorGraphCode-Klasse (Dritter Teil)	109
4.5	GraphCodeListElement-Klasse	110
4.6	GraphCodeTable-Klasse	110
4.7	EditorGraphCode-Klasse (Letzter Teil)	112
4.8	ExplanationPanel-Klasse	114
4.9	TextPanel-Klasse	116
4.10	TextPanel-Klasse (Zweiter Teil)	117
4.11	TextPanel-Klasse (Letzter Teil)	118
4.12	ImagePanel-Klasse	119
4.13	ExplainerConsole-Klasse	121

4.14	Zusammenschluss der Grundbereiche	123
4.15	ImportGraphCodesController-Klasse	124
4.16	RemoveSelectedGraphCodesController-Klasse	126
4.17	GraphCodeSelectionListener-Klasse	127
4.18	GraphCodeListMouseAdapter-Klasse	128
4.19	GraphCodeCalculationController-Klasse	129
4.20	GraphCodeCalculationController-Klasse	130
4.21	GraphCodeCalculationController-Klasse	131
4.22	GraphCodeCalculationController-Klasse	131
4.23	Algorithmus zur Bestimmung der Gemeinsamkeiten von Graph Codes	131
4.24	GraphCodeCalculationController-Klasse	133
4.25	Algorithmus zur Bestimmung der Unterschiede von Graph Codes . .	133
4.26	Quellcode für Methode <i>setUpPrompt</i>	136
4.27	Quellcode für Methode <i>getFormattedTerms</i>	138
4.28	Anfrage an den Endpunkt Text.	140
4.29	Anfrage an den Endpunkt Bild.	141

1 Einleitung

Die Digitalisierung der Welt schreitet immer rasanter voran. Nicht zuletzt wird im Kontext der Digitalisierung auch von dem „Informationszeitalter“ und „Computerisierung“ gesprochen [1]. In einer zunehmend digitalen Welt nimmt die Anzahl internetfähiger digitaler Geräte, den sogenannten *Internet Of Things* (IoT)-Geräten und Anwendern derselben immer mehr zu [2]. Laut Statistiken belief sich die Zahl der Internetnutzer, den sogenannten „Onlinern“ [3], im Jahr 2022 auf rund 5.3 Milliarden. Im Vergleich zum letzten Jahrzehnt ist somit die Anzahl der Internetanwender innerhalb eines Jahrzehnts um rund 2.9 Milliarden auf beinahe das Doppelte angestiegen [3]. Auch die weltweite *Covid-19*-Pandemie hatte keinen hindernden Effekt auf diese Entwicklung [4]. Aufgrund der Pandemie waren viele Menschen dazu gezwungen, mehr Zeit zu Hause zu verbringen, oder von zu Hause aus zu arbeiten. Dies hatte zur Folge, dass mehr Menschen sich digitalen Technologien und Lösungen zuwandten, um mittels Videokonferenzsystemen von zu Hause aus zu arbeiten, oder mittels sozialer Medien mit Freunden und Bekannten in Kontakt zu bleiben [4, 5]. Anhand der weltweiten Reaktionen auf die Pandemie wurde die Umstellung auf digitale Technologien um mehrere Jahre beschleunigt [4]. Die Verwendung dieser Technologien, wie z.B. den sozialen Medien, wie Instagram [6], Facebook [7], Snapchat [8] und weiteren, führt zu enormen Datenmengen, die ohne Hilfsmittel für Menschen nicht mehr greifbar und überschaubar sind. So betrug die Datennutzung im Jahr 2022 schätzungsweise 100 000 Petabytes, von welchem Mobiltelefone einen Großteil ausmachten [9].

Sollen, unabhängig vom Anwendungsbereich, diese enormen Datenmengen und die daraus extrahierten Merkmale effektiv verarbeitet, und gewonnene Erkenntnisse effektiv kommuniziert werden, so gilt es, die Beziehung zwischen der Darstellung der identifizierten Merkmale und der für Menschen verständlichen Bedeutung zu berücksichtigen. Zwischen der Darstellung der identifizierten Merkmale und der für Menschen verständlichen Bedeutung besteht eine sogenannte semantische Lücke, die es zu überbrücken gilt [10]. Die semantische Lücke bezeichnet hierbei den Unterschied zwischen den jeweiligen Darstellungsformen. Die Darstellungsform der identifizierten Merkmale ist ihre maschinelle Darstellung in Form von Nullen und Einsen, auch binäre Darstellung genannt, und die Darstellung der für Menschen verständliche Bedeutung kann in natürlicher Sprache erfolgen. Zwischen diesen Darstellungsformen liegt eine semantische Lücke, dessen Überbrückung in dieser Arbeit untersucht werden soll. Das Überbrücken dieser semantischen Lücke ist eine große Herausforderung und wird seit Jahren intensiv erforscht und ist, aufgrund immer breiterer Anwendung, Gegenstand zahlreicher Untersuchungen im Bereich des *Multimedia Information Retrieval*, kurz *MMIR* [10].

Das Thema dieser Arbeit ist im Bereich *MMIR* und künstlicher Intelligenz (KI) verortet. Das Themengebiet der künstlichen Intelligenz ist nicht neu und schon seit Jahrzehnten Gegenstand der Forschung und zahlreicher Untersuchungen [11]. Die Anwendungsmöglichkeiten von künstlicher Intelligenz sind weitreichend und haben zunehmend Einfluss auf unseren Alltag. Viele Aufgaben, die früher von Menschen erledigt wurden, oder nicht zu erledigen waren, werden bereits durch künstliche Intelligenz übernommen und automatisiert [12]. Weiterhin stellt künstliche Intelligenz für viele Unternehmen und Organisationen schon heute effektiv einen wirtschaftlichen Mehrwert dar [12]. Abb. 1.1 zeigt eine Übersicht wichtiger Begriffe im Bereich der künstlichen Intelligenz, sowie die Beziehung dieser Begriffe zueinander.

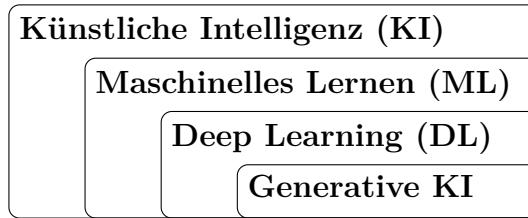


Abb. 1.1: Begriffe im Rahmen von künstlicher Intelligenz.

Künstliche Intelligenz ist ein Überbegriff und bezeichnet den Ansatz, mit Maschinen intelligentes menschliches Verhalten nachzuahmen, um dadurch Probleme zu lösen [13]. Maschinelles Lernen ist ein Überbegriff für die Technologien, die eingesetzt werden, um künstliche Intelligenz zu erreichen [13]. Deep Learning ist eine Weiterentwicklung des maschinellen Lernens und macht Gebrauch von neuronalen Netzwerken, welche dem menschlichen Gehirn nachempfunden sind [13]. Generative KI ist ein besonderer Bereich der künstlichen Intelligenz und ein Sammelbegriff für auf künstlicher Intelligenz basierender Systeme, mit welcher neue und originelle Inhalte erstellt werden können [14].

Die enormen und komplexen Datenmengen, die mittlerweile im Internet generiert werden [9], verstärken das Potenzial von maschinellem Lernen, sowie aber auch ihre Notwendigkeit [15]. Noch bis vor Kurzem war maschinelles Lernen auf vorausschauende Modelle beschränkt, die nur in der Lage waren, Beobachtungen vorzunehmen und Muster zu erkennen. Durch generative künstliche Intelligenz, wie ChatGPT-3.5 [16] oder Dall-E 2 [17], konnte ein Durchbruch erzielt werden, welcher ermöglicht, neue Inhalte zu erzeugen [15].

In dieser Arbeit wird untersucht, ob und inwiefern generative KI verwendet werden kann, um die semantische Lücke zu schließen und inwieweit generative KI geeignet ist, präzise Erklärungen bzw. geeignete semantische Darstellungen aus identifizierten Merkmalen zu generieren.

1.1 Motivation

Bei dieser Arbeit handelt es sich um eine Bachelorarbeit im Forschungsumfeld von Prof. Dr.-Ing. Stefan Wagenpfeil am Lehrgebiet Multimedia und Internetanwendungen (MMIA) von Prof. Dr.-Ing. Matthias Hemmje. Schon vor der Bachelorarbeit bin ich über das von diesem Lehrgebiet angebotene Fachpraktikum Multimedia Information Retrieval mit dem Bereich *MMIR* in Kontakt gekommen. Da mich das Fachpraktikum und das Themengebiet des *MMIR* sehr interessiert und begeistert hat, schreibe ich nun in diesem Lehrgebiet meine Bachelorarbeit, um meine Fähigkeiten auszubauen und mein Wissen zu vertiefen.

Das Lehrgebiet MMIA hat zum Zwecke der Forschung im Bereich des *MMIR* das Forschungsprojekt *Generic Multimedia Analysis Framework*, kurz *GMAF* ins Leben gerufen. Dieses Forschungsprojekt wurde entwickelt, um aktuelle Missstände und Probleme des einfachen *MMIR* auf intelligente Art und Weise zu adressieren. Das *GMAF* hat dabei zum Ziel, durch eine intelligente Erweiterung von *MMIR* semantische, erklärbare, für Menschen verständliche, effektive, effiziente, kompatible und integrierte Lösungen anzubieten [18, S. 20]. Diese Erweiterung wird *Smart Multimedia Information Retrieval*, kurz *Smart MMIR* genannt. *Smart MMIR* führt eine Reihe von Konzepten ein, die über klassisches *MMIR* hinaus gehen. Hierzu gehören erweiterte Metriken für die Berechnung von Ähnlichkeit oder Empfehlungen, Möglichkeiten der Merkmalsintegration und -fusion, sowie die Erklärbarkeit von Prozessen des *MMIR* und deren Inhalte. Unter Erklärbarkeit versteht man in diesem Zusammenhang für Menschen verständliche Antworten auf Fragestellungen, wie z.B. die Frage „Was ist die wichtigste Eigenschaft des Elements?“ oder „Was ist der Unterschied zwischen diesen zwei Bildern?“. Sollen Antworten auf diese Fragen auf für Menschen verständliche Art und Weise ausgedrückt werden, so müssen diese Antworten auf natürlicher Sprache aufbauen, oder prägnant visuell dargestellt werden. Ein Beispiel für eine generierte textuelle Erklärung, die auf natürlicher Sprache aufbaut, ist in Abb. 1.2 zu sehen. Das *GMAF* bietet hierfür ein umfangreiches Konzept zur Generierung von textuellen Beschreibungen von *MMIR*-Inhalten in für Menschen lesbarer und verständlicher Form. Es ist allerdings wichtig anzumerken, dass dieses Konzept statistisch und rein mathematische Ansätze zur Generierung von natürlicher Sprache verfolgt, ohne dabei Methoden maschinellen Lernens zu verwenden. Diese Ansätze generieren jedoch nur textuelle Beschreibungen auf eine statische Art und Weise. Zudem können mit diesen Ansätzen keine anderen Formen, wie z.B. visuelle Beschreibungen, außer Text generiert werden. Es ist also erstrebenswert nach einer Lösung zu suchen, mit welcher man sich von diesen statischen Ansätzen lösen kann. Ein anderer Ansatz, der in den letzten Monaten stark an Popularität gewonnen hat, ist die generative KI. Generative KI scheint eine vielversprechende Alternative zu einem statischen, statistischen und rein mathematischen Ansatz zu sein.



The picture shows two primary objects. The first object is a dog, the second object is a woman. The dog is white and has black eyes. The woman is smiling and has blonde hair. The woman is wearing shoes, shorts, a top, and a watch. The shoes are blue. The shorts are blue. The top is black. The watch is an Apple Watch. In the background there is grass.

Abb. 1.2: Beispiel für eine generierte textuelle Beschreibung eines Bildes im GMAF.

Ließen sich mittels generativer KI tatsächlich *MMIR*-Prozesse erklären, so kann dies einen echten Mehrwert für Benutzer darstellen, da diese von präzisen Erklärungen oder visuellen Darstellungen profitieren würden. Beispiele für Mehrwert für Benutzer können sein:

- **Zeitersparnis.** Generative KI könnte anhand identifizierter Merkmale Datei(en) zusammenfassen und in kürzester Zeit präzise Beschreibungen der Datei(en) erstellen. Ein Benutzer müsste für diese Arbeit erheblich mehr Zeit aufwenden.
- **Überblick / Ordnung.** Generative KI könnte anhand identifizierter Merkmale in einer großen Kollektion an Dateien Zusammenhänge erkennen und dem Benutzer eine Beschreibung dieser Zusammenhänge bieten. Zusammenhänge können in diesem Fall Muster oder eine für den Benutzer versteckte bzw. nicht ersichtliche Ordnung innerhalb der Dateien sein.
- **Barrierefreiheit.** Es gibt bereits Programme, wie z.B. Screen Readers, die blinden Personen Beschreibungen von Bildern vorlesen. Nachteil ist, dass diese Programme abhängig von den im Bild eingebetteten Beschreibungen sind, und viele Bilder im Internet über keine eingebetteten Beschreibungen verfügen. Zusätzlich sind diese Beschreibungen meistens von Menschen verfasst [19]. Generative KI könnte anhand identifizierter Merkmale Beschreibungen solcher Bilder generieren. Diese Beschreibungen könnten in weiteren Schritten von anderen Programmen, oder selbst wieder generativer KI verwendet werden, um diese vorzulesen. Auf diese Weise könnte echte Barrierefreiheit erreicht werden.

1.2 Problembeschreibung

Das *GMAF* bietet bereits ein umfangreiches Konzept zur Erklärbarkeit von Prozessen des *MMIR*. Die Ansätze, die in diesem Konzept verfolgt werden, sind statisch, in ihrer Anwendung begrenzt und ihre Erklärbarkeit vergleichsweise rudimentär. In diesem Abschnitt werden diese Probleme angesprochen, umrissen und subsequent Problembeschreibungen definiert.

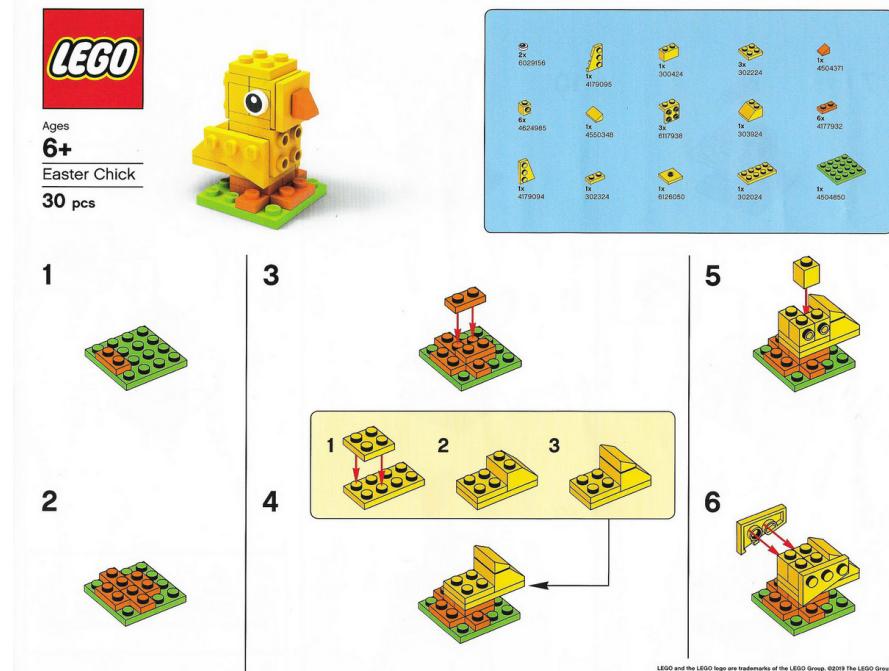
1.2.1 Probleme in Bezug auf die Erklärbarkeit

Das *GMAF* bietet bereits Verfahren zur Erklärbarkeit von *MMIR*-Prozessen. Die Ansätze, die mit diesen Verfahren umgesetzt werden, sind allerdings sehr statisch und basieren wiederum auf statistischen und rein mathematischen Ansätzen. Noch dazu bieten diese Verfahren nur die Erklärbarkeit von *MMIR*-Prozessen anhand von Text. Es ist allerdings wichtig anzumerken, dass Text nicht die einzige Form der Erklärbarkeit ist. Eine andere Form der Erklärbarkeit kann mittels Bilder erzielt werden. So lautet ein bekanntes Sprichwort: „*Ein Bild sagt mehr als 1000 Worte*“. Dieses Sprichwort will zum Ausdruck bringen, dass Bilder, im Vergleich zu Text, ebenfalls komplexe Sachverhalte und Wissen prägnant und in für Menschen verständlicher Form zum Ausdruck bringen können. Ein Beispiel hierfür wäre eine Bauanleitung zu einem Lego-Set. Ein Beispiel für eine Bauanleitung zum Bauen eines Osterkükens ist in Abb. 1.3 zu sehen. Die Bauanleitung veranschaulicht dabei die einzelnen Aufbauschritte durch fortwährende Nummerierung und Abbildungen.

Ein konkretes Anwendungsbeispiel für die Erklärbarkeit in Form von Bildern wäre im Rahmen des *MMIR* die Generierung von Vorschaubildern für eine Textdatei. Abb. 1.4 zeigt ein Beispiel für ein von Dall-E 2 [17] generiertes Vorschaubild für ein Gedicht über einen Schmetterling [21]. Anhand der in der Textdatei identifizierten Merkmale könnte eine generative KI, wie z.B Dall-E 2, ein Vorschaubild generieren, um somit den Inhalt des Textes visuell zu erklären. Ein Vorschaubild kann sich dann als sinnvoll erweisen, wenn viele Dateien zu verwalten sind und Benutzer nicht die Zeit haben diese einzeln zu durchforsten. Durch einen kurzen Blick auf das Vorschaubild könnte ein Benutzer die Datei einordnen und sich besser zurechtfinden.

Ein weiteres Anwendungsbeispiel wäre die Gemeinsamkeit/Differenz aus einer Kollektion an Bildern. Die als Gemeinsamkeit/Differenz identifizierten Merkmale könnten in diesem Beispiel ebenfalls konkret mittels eines Bildes kurz und prägnant visuell dargestellt werden. Diese Art der visuellen Erklärung kann sich dann als sinnvoll erweisen, wenn viele Bilder auf ihre Merkmale hin untersucht werden sollen. Ein Benutzer kann hier schnell den Überblick verlieren.

Statistische und rein mathematische Ansätze reichen für diese Beispiele nicht mehr aus. Ein anderer, vielversprechender Ansatz, um die Erklärbarkeit von Prozessen des



Haftungsausschluss: LEGO ® und das LEGO Logo sind Marken der LEGO Unternehmensgruppe.

© 2023 Lego Gruppe. Diese Arbeit ist unabhängig und wurde von der LEGO Gruppe weder genehmigt, noch gesponsort.

Abb. 1.3: Bauanleitung für ein Lego-Set [20].

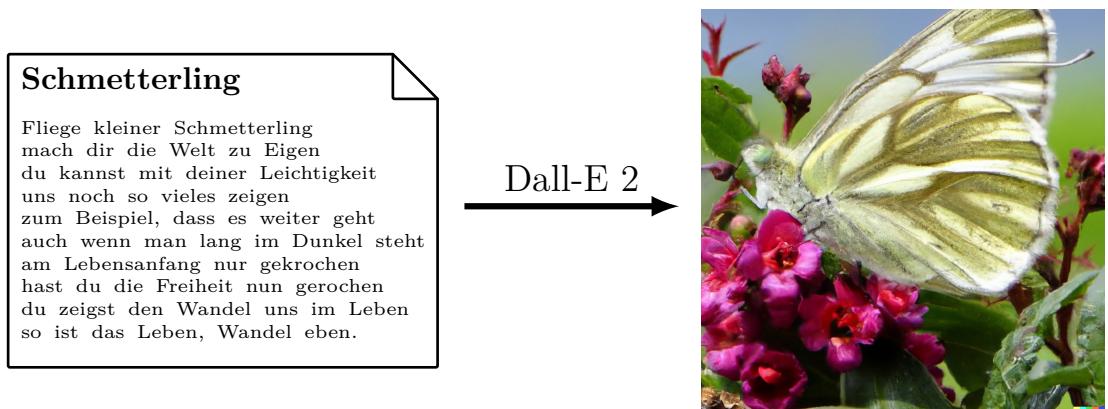


Abb. 1.4: Ein von Dall-E 2 generiertes Vorschaubild.

MMIR zu erreichen, liegt in generativer KI. Mittels generativer KI könnten anhand identifizierter Merkmale eigenständig neuartige Multimedia-Inhalte, wie Texte oder Bilder, erzeugt werden, die zur Erklärbarkeit von MMIR-Prozessen dienen könnten.

Je nach Anwendungsbereich variieren die Anforderungen an Erklärungen allerdings erheblich. Somit ist ein weiterer wichtiger Aspekt die Anpassungsmöglichkeiten von Erklärungen in Abhängigkeit ihrer Anwendungsfälle. So wird *Smart MMIR* für die Erkennung sicherheitsrelevanter Zustände im Bereich der Automobilindustrie, auch Automotive genannt, eingesetzt [22]. Sicherheitsrelevante Zustände in einem Kraftfahrzeug sind Zustände, die für eine reibungslose Fahrt und Funktion sorgen, oder unterstützend und im Sinne des Autofahrers eingreifen oder im Ernstfall zum Schutz von Insassen und Außenstehenden beitragen [23]. Ein Beispiel für eine unterstützende Funktion kann in Form von HeadUp-Displays (HUD) gefunden werden und ist in Abb. 1.5 zu sehen. Das HeadUp-Display projiziert sicherheitsrelevante Informationen, wie z.B. die Geschwindigkeit auf die Windschutzscheibe. Da im Bereich



Abb. 1.5: HeadUp-Display in einem Kraftfahrzeug [24].

Automotive schnelle Entscheidungen getroffen werden müssen, werden hier kurze, präzise Erklärungen benötigt. Es ist also in Bezug auf die Erklärbarkeit notwendig Eigenschaften, wie z.B. die Länge der Erklärung, anpassen zu können. Das *GMAF* bietet bereits eine Benutzungsschnittstelle, die es ermöglicht Anpassungen von textuellen Erklärungen vorzunehmen. Diese Benutzungsschnittstelle ist in Abb. 1.6 zu sehen und zeigt die Elemente, die zur Anpassung von textuellen Erklärungen vorgesehen sind. Diese Benutzungsschnittstelle bietet allerdings nur grobe Möglichkeiten

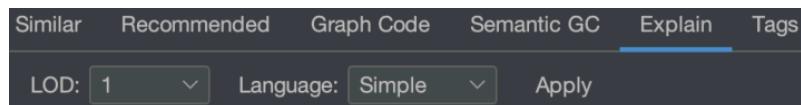


Abb. 1.6: Benutzungsschnittstelle im *GMAF* zum Anpassen von Erklärungen.

zur Anpassung von textuellen Erläuterungen, wie die Anpassung der Komplexität der textuellen Beschreibung. Möglichkeiten zur Anpassung der Länge der textuellen Beschreibungen ermöglicht diese Schnittstelle zum Beispiel jedoch nicht. Schlussendlich

kann in Bezug auf die Erklärbarkeit von *MMIR*-Prozessen folgende Problembeschreibung formuliert werden:

PB1: Generative KI ist aktuell nicht für die Erklärbarkeit von *MMIR*-Prozessen nutzbar.

1.2.2 Probleme in Bezug auf die Integration

Damit generative KI-Systeme aus identifizierten Merkmalen präzise und für Menschen verständliche Erklärungen generieren können, müssen die identifizierten Merkmale als Eingabedaten in die jeweiligen Systeme überführt werden. Das *GMAF* bietet keine Möglichkeiten zur Anbindung von Systemen, die generative KI anbieten, sowie kein Modell zur Übersetzung von identifizierten Merkmalen in Eingabedaten für entsprechende Systeme. Folglich gibt es auch keine Modellierung, die die Anpassung der Eingabedaten ermöglicht, um die in Abschnitt 1.2.1 benannten Modifikationen von Erklärungen zu ermöglichen. Gelingt eine Integration von generativen KI-Systemen, so kann dies einen echten Mehrwert für Benutzer darstellen, sofern dadurch verbesserte und originellere Erklärungen von *MMIR*-Prozessen generiert werden. Besonders die Möglichkeit, je nach Anforderung die Eingabedaten anzupassen, kann für Benutzer von Nutzen sein. Schlussendlich kann in Bezug auf die Integration folgende Problembeschreibung formuliert werden:

PB2: Das *GMAF* bietet keine Integration generativer KI.

1.2.3 Allgemeine Zielsetzung

Aus den in Abschnitt 1.2.1 und Abschnitt 1.2.2 beschriebenen Problembereichen lassen sich zwei direkte Probleme erfassen: Die im *GMAF* implementierten Konzepte zur Erklärung von *MMIR*-Prozessen sind noch sehr statisch und bieten nur Erklärungen in Form von Text, sowie die Integration besserer Lösungen zur Erklärbarkeit von *MMIR*-Prozessen. Die Motivation dieser Arbeit besteht darin, dass die bereits implementierten Konzepte im *GMAF* in Bezug auf die Erklärbarkeit von *MMIR*-Prozessen noch großen Raum für Verbesserung aufweisen. Verbesserung wird sich durch das Anbinden von generativer KI versprochen, da generative KI in der Lage zu sein scheint auf kreative Weise neuartige Inhalte zu generieren.

Werden nun die Problembeschreibungen zusammengefasst, so wird das übergeordnete Ziel sichtbar:

Die im GMAF implementierten Konzepte zur Erklärung von MMIR-Prozessen sollen durch generative KI abgelöst werden, um bessere Erklärungen, sowie andere Formen von Erklärungen zu ermöglichen.

Anhand der in diesem Abschnitt beschriebenen Problembeschreibungen und des allgemeinen Ziels können nun im nächsten Abschnitt 1.3 folgende Forschungsfragen formuliert werden. Im darauffolgenden Abschnitt 1.4 wird genauer auf die Methodik dieser Arbeit, sowie die konkreteren Ziele eingegangen.

1.3 Forschungsfragen

Um die im vorigen Abschnitt definierten Problembeschreibungen zu adressieren, werden folgende Forschungsfragen definiert:

- **FF1:** Kann generative KI für die Erklärbarkeit von *MMIR*-Prozessen genutzt werden?
- **FF2:** Wie lässt sich generative KI in das *GMAF* integrieren?

1.4 Methodik und Ziele

Um die in Abschnitt 1.2 identifizierten Probleme und subsequent die im vorigen Abschnitt 1.3 formulierten Forschungsfragen strukturiert beantworten zu können, wird diese Arbeit auf der vielfach bewährten Methodik von Nunamaker [25] aufbauen. Die Methodik nach Nunamaker teilt ein zu lösendes Problem anhand von Recherche und Entwicklung in vier Phasen der Problemlösung auf.

Diese vier Phasen lauten:

1. Beobachtungsphase
2. Theoriebildungsphase
3. Systementwicklungs - bzw. Implementierungsphase
4. Experimentphase

In der Beobachtungsphase werden im Rahmen einer Recherche Informationen gesammelt. In der Theoriebildungsphase werden Konzepte für Lösungen konzipiert und modelliert. Die Systementwicklungs bzw. Implementierungsphase beschäftigt sich mit der Entwicklung von Lösungen, basierend auf den Theorien aus der Theoriebildungsphase bzw. Beobachtungen aus der Beobachtungsphase, und welche in der letzten Phase, der Experimentphase evaluiert werden können. Diese Phasen sind eng miteinander verwoben, und jede Phase mündet mit seinem Ergebnis in den anderen Phasen und trägt zu diesen bei. Durch die Methodik von Nunamaker wird sicher gestellt, dass jedes Forschungsziel eindeutig einer Problembeschreibung zugeordnet werden kann.

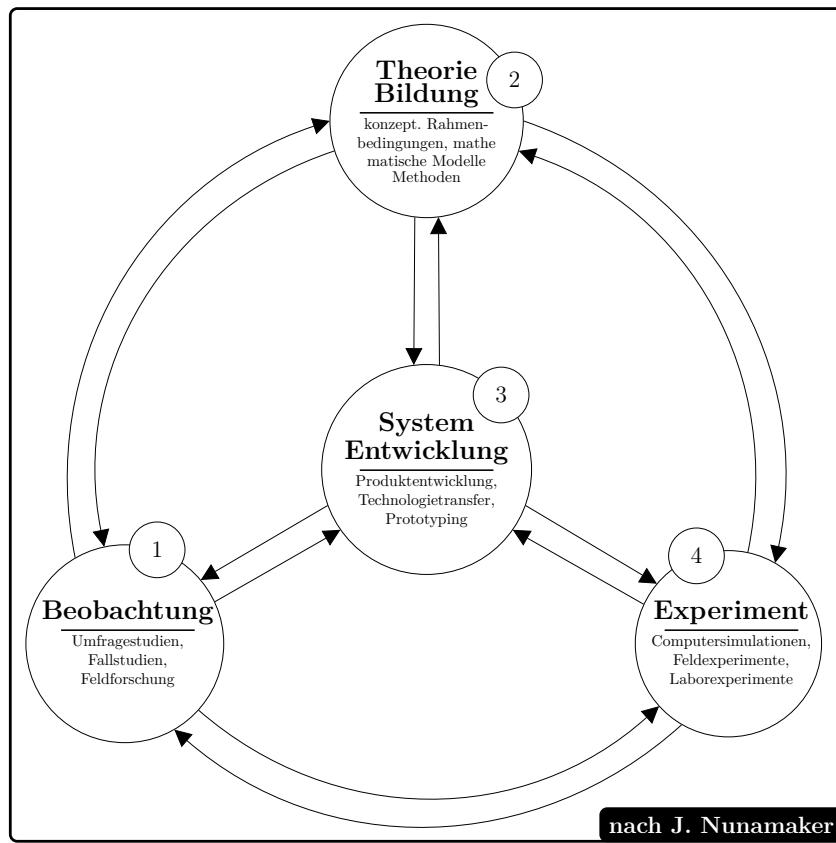


Abb. 1.7: Phasen der Problemlösung nach Nunamaker [25].

Die Anwendung der Methodik nach Nunamaker führt für jede Forschungsfrage zu jeweils vier Forschungszielen, die in jeweils einem Unterkapitel behandelt werden. Dies führt insgesamt zu acht Forschungsaufgaben, die mit dieser Arbeit behandelt werden. Die anfängliche Planung der Forschungsziele ist in Tab. 1.1 einsehbar. Die Benennung der Forschungsaufgaben folgt einem einheitlichen Schema und ergibt sich aus dem Namen des jeweiligen Forschungsziels, erweitert durch einen Buchstaben als Anhängsel, z.B. „O“ für *Observation/Beobachtung*, „TB“ für *Theory Building/Theoriebildung*, „I“ für *Implementation/Implementierung* und „E“ für *Experiment*.

Tabelle 1.1: Aufschlüsselung der Forschungsfragen auf Forschungsziele

Forschungsfragen		
FZ	Beschreibung	PB
Erklärbarkeit von MMIR mittels generativer KI		
FZ 1.1/O	Recherche zur Erklärbarkeit von MMIR mittels generativer KI	PB1
FZ 1.2/TB	Modellierung zur Erklärbarkeit von MMIR mittels generativer KI	PB1
FZ 1.3/I	Implementierung zur Erklärbarkeit von MMIR mittels generativer KI	PB1
FZ 1.4/E	Experiment zur Erklärbarkeit von MMIR mittels generativer KI	PB1
Integration generativer KI in das GMAF		
FZ 2.1/O	Recherche zur Integration generativer KI in das GMAF	PB2
FZ 2.2/TB	Modellierung zur Integration generativer KI in das GMAF	PB2
FZ 2.3/I	Implementierung zur Integration generativer KI in das GMAF	PB2
FZ 2.4/E	Experiment zur Integration generativer KI in das GMAF	PB2

Um sicherzustellen, dass alle Forschungsziele durch den Inhalt dieser Arbeit abgedeckt werden, entspricht die Struktur dieses Dokuments direkt dem Ansatz nach Nunamaker. Der Ansatz und Aufbau der Arbeit wird im nächsten Abschnitt behandelt.

1.5 Ansatz und Aufbau der Arbeit

Die Struktur dieser Arbeit ergibt sich 1:1 aus der Methodik von Nunamaker [25] und folgt seiner Modellierung. Ein Überblick auf die Struktur dieser Arbeit und Nunamakers Ansatz ist in Abb. 1.8. Die entsprechenden Forschungsziele werden, je nach Phase der Modellierung nach Nunamaker, umgeordnet und in die jeweiligen Kapitel dieser Arbeit umgruppiert.

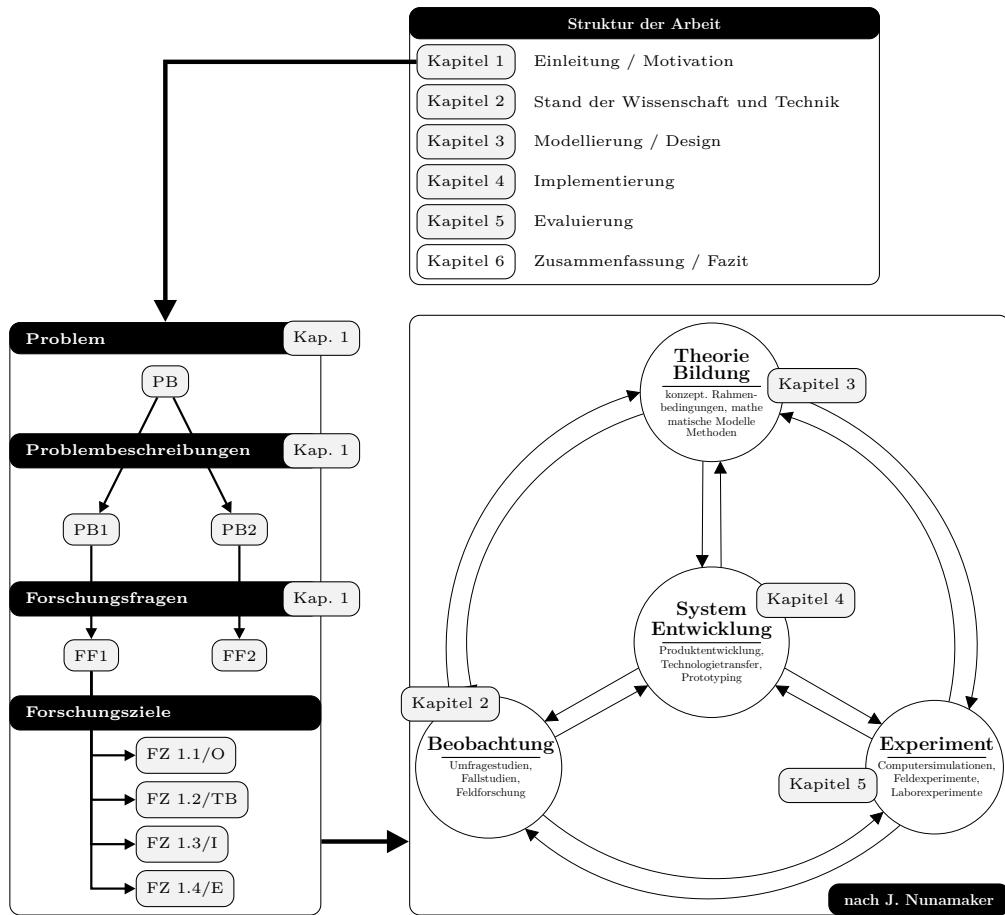


Abb. 1.8: Aufbau der Arbeit und Ansatz der Problemlösung nach Nunamaker [25].

Im Nachfolgenden werden die Kapitel jeweils einzelnen kurz beschrieben. Es wird genannt, welche Forschungsziele von welchem Typ abgearbeitet werden, und welche Aufgaben das entsprechende Kapitel jeweils übernimmt.

- „**Kapitel 2 - Stand der Wissenschaft und Technik**“ deckt alle Forschungsziele vom Typ *Beobachtung* ab, und wird der Reihenfolge nach dem aktuellen Stand der Wissenschaft und Technik zusammenfassen, einführen und darstellen.
- „**Kapitel 3 - Modellierung**“ deckt alle Forschungsziele vom Typ *Theoriebildung* ab und beschreibt die Modellierung und das Design von Konzepten und Algorithmen zu vorgeschlagenen Problemlösungen.
- „**Kapitel 4 - Implementierung**“ beschreibt die Implementierung von Modellen und Konzepten und deckt alle Forschungsziele vom Typ *Systementwicklung bzw. Implementierung* ab.

- „**Kapitel 5 - Experiment**“ deckt alle Forschungsziele vom Typ *Experiment* ab und gibt eine detaillierte Beschreibung aller Ergebnisse.

Um in dieser Arbeit einen wissenschaftlich formalen und korrekten Prozess zur Problemlösung anzuwenden, wurde die Modellierung nach Nunamaker eingeführt. Dieser Modellierung folgend, wird eine klare Beziehung zwischen den Forschungszielen und den Forschungsfragen etabliert. Dadurch wird sichergestellt, dass jedes Ergebnis, das in dieser Arbeit dargestellt wird, in direktem Bezug zu einer entsprechenden Forschungsfrage und einer Problembeschreibung steht und umgekehrt. Des Weiteren folgt das Kapitel 3 „Modellierung“ dieser Arbeit dem Ansatz *User Centered System Design* nach Draper [26], welches den Benutzer in den Mittelpunkt sämtlicher Modellierung und Planung stellt. Die Modellierung und Planung wird auf der *Unified Modeling Language (UML)* [27] aufbauen.

1.6 Zusammenfassung

Das erste Kapitel dieser Arbeit hat allgemein in die Motivation des Themas eingeführt. Eine kurze Beschreibung der aktuellen Entwicklung im Bereich Multimedia hat gezeigt, dass *MMIR* eine zunehmend wichtige Rolle spielt. In diesem Zuge spielt auch die Erklärbarkeit von *MMIR*-Prozessen eine immer wichtigere Rolle. Zwei Problembereiche konnten identifiziert werden, die mit dieser Arbeit adressiert werden. Im Bereich der Erklärbarkeit bietet das *GMAF* bereits umfangreiche Konzepte zur Erklärbarkeit von *MMIR*-Prozessen. Diese Konzepte sind aber sehr statisch und in ihrer Anwendung stark eingeschränkt. Hier gibt es allgemein großes Potenzial für Verbesserungen, die man sich durch das Anbinden generativer KI erhofft / verspricht. Im Bereich der Integration konnte festgestellt werden, dass das *GMAF* noch keine Anbindung für generative KI-Systeme bietet und hierfür eine geeignete Modellierung entwickelt werden muss.

Das Definieren von Problembeschreibungen, Forschungsfragen und Forschungszielen dient als formelle Grundlage für die Strukturierung dieser Arbeit. Basierend auf der Modellierung nach Nunamaker folgt diese Arbeit einem wissenschaftlich anerkannten Ansatz bei der Darstellung der Forschungsergebnisse. Um ein starkes Fundament für die Arbeit zu legen, wird im nächsten Kapitel ein detaillierter Überblick zum aktuellen Stand der Wissenschaft und Technik gegeben, in welchem die Ergebnisse der Forschungsziele vom Typ Beobachtung behandelt werden.

2 Stand der Wissenschaft und Technik

Die voranschreitende Digitalisierung, die Entwicklung des Internets, Mobiltelefone, sowie die sozialen Medien und die damit einhergehenden enormen Datenmengen haben zu einer enormen Zunahme an Wissen, Standards und auch Forschung im Bereich des *MMIR* geführt [18]. Im Bereich der Forschung spielt auch die Erklärbarkeit von *MMIR*-Prozessen eine zunehmend wichtige Rolle. In einschlägiger Literatur, wie z.B [18] wurde die Erklärbarkeit von *MMIR*-Prozessen bereits adressiert und es wurden Lösungsvorschläge und Konzepte entwickelt.

In diesem Kapitel wird, den Forschungszielen der Beobachtungsphase entsprechend, der aktuelle Stand der Wissenschaft und Technik bezogen auf ausgewählte Bereiche der Erklärbarkeit von Prozessen des *MMIR* beschrieben. Die in diesem Kapitel dargestellten Informationen wurden im Rahmen einer Literaturrecherche, speziell Publikationen, bezogen auf das GMAF und seine Technologien, sowie durch eine extensive Internetrecherche mit entsprechenden Stichworten zusammengetragen. Das Bezugssystem für diese Arbeit stellt das *GMAF* dar. Die für diese Arbeit relevanten Konzepte werden detailliert beschrieben, um so eine konkrete Vorlage, ein starkes Fundament, sowie eine scharfe Trennung zu den Phasen Modellierung, Implementierung und Experiment zu schaffen, die im weiteren Verlauf dieser Arbeit in den nachfolgenden Kapiteln Modellierung, Implementierung und Evaluierung folgen werden. Somit enthält dieses Kapitel alle nötigen Rechercheergebnisse, Beobachtungsergebnisse und damit Erkenntnisse, die der Beantwortung der Forschungsfragen laut Ansatz dienlich sein sollen. Der Methodik nach Nunamaker [25] folgend, behandelt dieses Kapitel alle Forschungsziele vom Typ Beobachtung, entsprechend der identifizierten Problembeschreibungen aus dem Abschnitt 1.2. In Abschnitt 2.1 wird auf die Erklärbarkeit von MMIR mittels generativer KI eingegangen, welches vorhandene Technologien im Bereich MMIR, sowie generativer KI beinhaltet. In Abschnitt 2.2 wird auf die Integration von generativer KI in das GMAF eingegangen, welches Technologien und technische Aspekte zur Integration beinhaltet. In Abschnitt 2.3 werden Methodiken zur Evaluierung vorgestellt, die später in Kapitel 5 im Rahmen von Experimenten Verwendung finden könnten. Abschließend wird in Abschnitt 2.4 eine Zusammenfassung der in diesem Kapitel erfassten Erkenntnisse und Ergebnisse gegeben. Dieser Strukturierung folgend, wird im nächsten Abschnitt die erste Problembeschreibung adressiert. Die Struktur dieses Kapitels ist in Tab. 2.1 abgebildet.

Tabelle 2.1: Struktur dieses Kapitels.

FZ	Kurze Beschreibung	Abschnitt
FZ 1.1/O	Recherche: Erklärbarkeit & generativer KI	Abschnitt 2.1

FZ	Kurze Beschreibung	Abschnitt
FZ 2.1/O	Recherche: Integration generativer KI	Abschnitt 2.2
Evaluierungsmeth. Zusammenfassung	Methodiken zur Evaluierung von Software Ergebnisse / Erkenntnisse	Abschnitt 2.3 Abschnitt 2.4

2.1 FZ 1.1/O Erklärbarkeit von MMIR mittels generativer KI

Dieser Abschnitt gibt einen Überblick über ausgewählte Ergebnisse und Erkenntnisse aus der Forschung im Bereich der Erklärbarkeit von MMIR und deckt verschiedene Aspekte ab, die sich auf die erste Problembeschreibung beziehen. Die Struktur dieses Abschnittes folgt einer logischen Abfolge. Zuerst werden bereits vorhandene Technologien im Bereich der Erklärbarkeit von MMIR vorgestellt und erläutert. Dies trifft besonders auf Technologien in Bezug auf das GMAF zu, da das GMAF das Bezugssystem dieser Arbeit darstellt. Im weiteren Verlauf wird dann auf generative KI eingegangen, Grundlagen werden erläutert, eine Auswahl an aktuellen Systemen generativer KI vorgestellt, sowie die Einschränkungen von generativer KI hervorgehoben.

2.1.1 GMAF

Das Generic Multimedia Analysis Framework (GMAF) [28] ist ein Framework, welches ausgewählte Technologien zum Verarbeiten verschiedener multimedialer Inhalte, wie Bilder, Videos oder Text nutzt und miteinander kombiniert [29, 30]. Beispiele für solche Technologien sind Verarbeitungssysteme zur Merkmalserkennung in Bildern (z.B zur Objekterkennung oder Erkennung von räumlichen Beziehungen mittels maschinellen Lernens). Das GMAF bietet eine flexible und erweiterbare Architektur, mit welcher auf einfache Art und Weise neue Algorithmen oder Schnittstellen zur Merkmalserkennung im Form von Plugins in das GMAF integriert werden können [30]. Die von den Plugins extrahierten Merkmalen werden vom GMAF in einer zentralen Datenstruktur zusammengefasst und vereint [29, 22]. Abb. 2.1 zeigt eine vereinfachte Übersicht der Architektur des GMAFs und die grundlegenden Komponenten. Die Plugins im GMAF können als ein Zusammenschluss aus Anbietern von semantischen Informationen betrachtet werden [29]. Dieser Zusammenschluss produziert alle Daten, die notwendig sind, um ein Verzeichnis bzw. einen Index aus den semantischen Informationen zu generieren [29]. Allgemein kann jedes durch ein Plugin identifizierte Merkmal einen Indizierungsbegriff darstellen und somit Teil des Indexes sein [31]. Da für gewöhnlich diese Begriffe strukturelle und semantische Be-

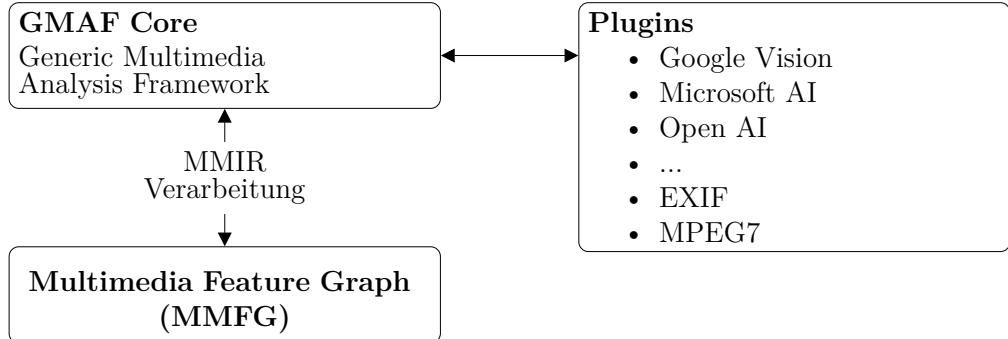


Abb. 2.1: Vereinfachte Übersicht des GMAFs (ohne UI / API) nach [22].

ziehungen zueinander aufweisen, bietet sich zur Darstellung dieser Merkmale und ihren Beziehungen zueinander eine auf Graphen basierende Struktur an [31]. Diese Struktur wird durch den Multimedia Feature Graphen (MMFG) realisiert, welcher die in ihm gespeicherten semantischen Informationen in einem einheitlichen Modell repräsentiert [31].

Der nächste Abschnitt beschreibt die Datenstruktur der MMFGs, welche im Rahmen des GMAF eine zentrale Rolle einnehmen. Zuerst wird die Grundstruktur der MMFGs beschrieben. Im weiteren Verlauf wird dann in Abschnitt 2.1.2.1 die Erklärbarkeit von MMFGs thematisiert. Darauffolgend wird in Abschnitt 2.1.2.2 die Verarbeitung von MMFGs angesprochen, welcher den Übergang zu einer weiteren wichtigen Technologie, den Graph Codes, schafft.

2.1.2 MMFG

Der Multimedia Feature Graph (MMFG) ist ein Resultat des GMAFs und stellt für Prozesse innerhalb des GMAFs eine zentrale gemeinsame Ressource dar [29]. Das grundlegende Konzept der MMFGs ist dabei eng verknüpft mit der Graphentheorie [29]. Genauer lässt sich ein MMFG als ein gewichteter und gerichteter Graph mit attribuierten Knoten und Kanten verstehen, der die aus Plugins identifizierten und extrahierten Merkmale multimedialer Inhalte in einer Graphenstruktur repräsentiert bzw. abbildet [29]. Des Weiteren ist ein MMFG in der Lage Verbindungen zu Teilen anderer MMFGs aufzubauen, wodurch eine rekursive Struktur aus MMFGs entsteht, die Schleifen und Teilgraphen besitzen kann [29]. Ein MMFG kann wie folgt definiert werden:

$$MMFG = (V, E)$$

Die Menge V repräsentiert die Knoten in einem MMFG und stellt die identifizierten Merkmale dar. Die Menge E stellt wiederum die Verbindungen bzw. Beziehungen zwischen den Knoten bzw. Merkmalen dar. Beide Mengen können anhand ihrer Attribuierung in Unterklassen aufgeteilt werden. Die Knotenmenge V besteht aus fol-

genden Unterklassen: $V \in \{n, r, e, w, cn, t, m, s, l, p\}$. Die folgende Aufzählung wurde aus [29] entnommen und erläutert die Bedeutung dieser Unterklassen:

- **Nodes** n_1, \dots, n_n stellt wichtige Objekte, Aktivitäten oder Regionen in einer Datei dar. r ist die Wurzel des MMFGs und selbst wiederum ein Knoten. e bezeichnet externe Knoten, die externe Informationen darstellen. Externe Informationen können aus anderen MMFGs oder anderen externen Systemen, wie dem Semantic Web stammen und so in den MMFG eingebunden werden.
- **Weight** w_1, \dots, w_n stellt die Gewichtung bzw. Relevanz eines Knoten in einem besonderen Kontext dar. So können Objekte, abhängig vom Kontext und Gewichtung, unterschiedliche Bedeutungen in Bezug auf die gesamte Datei besitzen.
- **Childnodes** cn_1, \dots, cn_n sind Objekte, die in einer untergeordneten hierarchischen Beziehung zu anderen Objekten stehen. Diese Objekte entstehen durch wiederholte bzw. rekursive Anwendung des GMAFs auf bereits identifizierte Merkmale. Ein Beispiel für diese rekursive Anwendung ist $Person \rightarrow Body \rightarrow Arm \rightarrow Watch$ und kann in Abb. 2.3 eingesehen werden.
- **TechnicalAttributes** t_1, \dots, t_n stellen nicht semantische Informationen eines Knoten dar. Beispiel für ein technisches Attribut kann die Abgrenzung eines Merkmals in einem Bild sein. Diese Abgrenzung wird in Form eines Rechtecks notiert ($x: 468, y: 176, width: 28, height: 12$).
- **GeneralMetadata** m stellt die Metadaten der Multimedia-Datei dar. Mit Metadaten bezeichnet man Daten, die andere Informationen beschreiben und können durch EXIF oder MPEG7 identifiziert werden. Die Metadaten eines Buches könnten so z.B. der Name des Autors, Auflage oder das Erscheinungsjahr sein.
- **Location** l_1, \dots, l_n stellt Speicherorte dar. Dies kann der Speicherort der ursprünglichen Multimedia-Datei, aber auch von Kopien, unterschiedlichen Auflösungen oder Teilen der Datei sein.
- **SynonymInformation** s_1, \dots, s_n verweist auf Synonyme eines Begriffes.
- **Security and Privacy** p stellt die Sicherheitseigenschaften einer Datei dar. Beispiel für so eine Sicherheitseigenschaft ist eine Zugriffskontrollliste (engl. access control list). Auf diese Weise kann der Umgang mit vertraulichen Daten sichergestellt werden.

Auch die Kanten in E besitzen Unterklassen und können in folgende Unterklassen unterteilt werden: $E \in \{cr, sr, sn, mm, al\}$. Die folgende Aufzählung wurde aus [29] entnommen und erläutert, analog zu den Knoten, die Bedeutung der Unterklassen der Kanten:

- **Composition Relationship** cr_1, \dots, cr_n drückt das Verhältnis in dem zwei Objekte miteinander stehen aus. Beispiele für solche Beziehungen können „verbunden mit (engl. attached to)“, „ein Teil von (engl. part of)“ oder „bezogen auf (engl. related to)“ sein. Diese Beziehungen werden vom GMFA durch rekursives Berechnen der Abgrenzungen von Merkmalen und deren Distanzen zueinander berechnet.
- **Sematic Relationship** sr_1, \dots, sr_n verbindet einen Knoten mit einem externen Knoten aus einem externen System, wie z.B. dem Semantic Web.
- **SynonymLinks** sn_1, \dots, sn_n verweist auf Informationen zu Synonymen.
- **MMFGLinks** mm_1, \dots, mm_n stellt Verbindungen zu anderen MMFGs dar.
- **AssetLinks** al_1, \dots, al_n zeigt auf Knoten, die Speicherorte darstellen.

Abb. 2.2 zeigt die Visualisierung eines MMFGs durch yEd. Dieser MMFG besitzt 185 Knoten und 222 Kanten. Aufgrund des steigenden Levels Of Details (LOD) vieler multimedialer Inhalte, wie z.B. durch höhere Auflösung von Bildern, steigt die Anzahl der Knoten und Kanten in MMFGs drastisch [29]. In der Praxis können MMFGs tausende Knoten und Kanten aufweisen [22]. In diesen Maßstäben ist eine Visualisierung eines MMFGs auf sinnvolle Weise kaum noch möglich. Das Hauptaugenmerk der MMFGs liegt nicht darauf für Menschen lesbar, darstellbar oder verständlich zu sein, sondern anhand ihrer Graphenstruktur geeignete Verarbeitung der Merkmale zu ermöglichen [29].

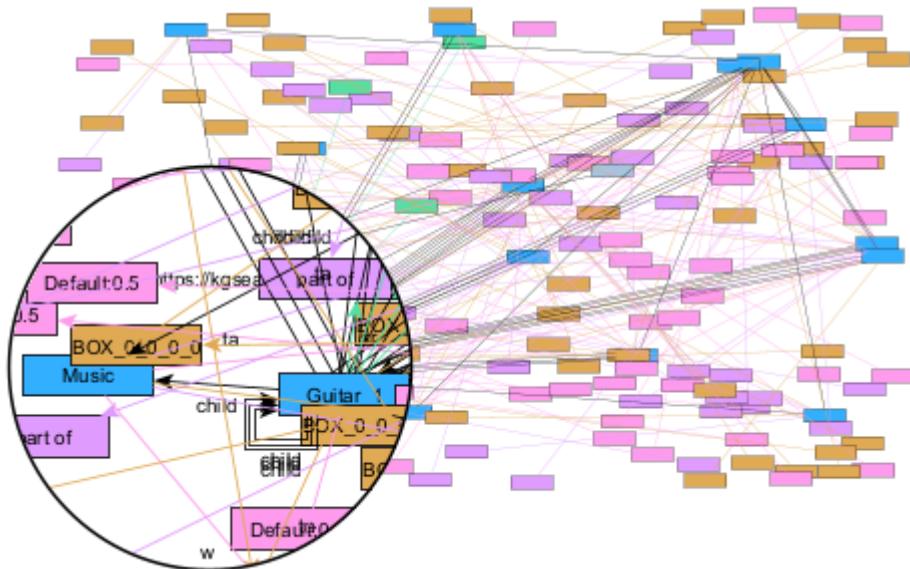


Abb. 2.2: Visualisierung eines MMFGs durch yEd.

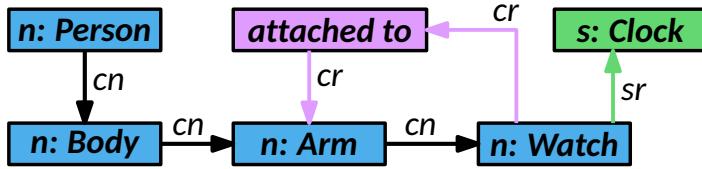


Abb. 2.3: Beispiel für den Multimedia Feature Graphen $MMFG_{ex}$.

Abb. 2.3 zeigt einen simplen $MMFG_{ex}$. Die Knoten *Person*, *Body*, *Arm* und *Watch* repräsentieren identifizierte Merkmale, *Clock* ist ein Synonym für *Watch* und *attached to* verweist auf eine Beziehung zwischen *Arm* und *Watch*.

Erklärbarkeit von MMFGs

Um Erklärbarkeit von MMFGs zu erreichen, muss die technische Darstellung der Merkmale in eine für Menschen verständliche Bedeutung überführt werden. Das Überbrücken dieser Lücke ist nicht in einem einzigen Schritt machbar. Es benötigt mehrere Schritte, um diese Lücke zu schließen. In einem ersten Zwischenschritt muss die Lücke zwischen der technischen Darstellung der Merkmale und ihrer semantischen Darstellung überbrückt werden. Hierzu wird ein formaler und für Maschinen lesbarer Ansatz zur Erweiterung von MMFGs benötigt [32]. In verwandten Arbeiten, wie [32], wurden umfangreiche Konzepte entwickelt und beschrieben, die in der Lage sind durch Erweiterungen von MMFGs Erklärbarkeit zu erreichen. Das Überbrücken der Lücke zwischen der technischen Darstellung der Merkmale und ihrer semantischen Darstellung gelingt mit einer semantischen Erweiterung. Eine semantische Erweiterung von MMFGs kann durch das Anbinden eines externen Informationssystems, wie z.B. dem Semantic Web [33], erreicht werden. Durch das Definieren und Anwenden einer formalen kontextfreien Phrasenstruktur-Grammatik [34] in Kombination mit einer semantischen Erweiterung sind die Darstellungen von MMFGs nicht nur für Menschen lesbar, sondern es können auch für Menschen verständliche, sowie lesbare und auf natürlicher Sprache basierende Ausdrücke generiert werden. Diese Erweiterungen ermöglichen formale semantische Darstellungen und schaffen die Grundlage für auf natürlicher Sprache basierende Erklärungen. Diese Erweiterungen von MMFGs zu formalen semantischen Darstellungen führen zu Semantischen Multimedia Feature Graphen (SMMFGs) [32], sowie zu Erklärbaren Semantischen Multimedia Feature Graphen (ESMMFGs) [32].

Jeder auf natürlicher Sprache basierende Ausdruck, der mit solch einer Grammatik generiert wird, ist inhaltlich *korrekt*, da es nur die ursprünglichen identifizierten Merkmale in einer formalen und für Menschen lesbaren Form darstellt [32]. Auf diese Weise kann jedes Element eines MMFG bzw. identifizierte Merkmal struktu-

rell und semantisch dargestellt werden [32]. Des Weiteren bieten sich anhand der Grammatik und ihrer Produktionsregeln unbegrenzte Möglichkeiten zur Konstruktion natürlicher Ausdrücke [32]. Durch Anpassungen der Produktionsregeln können Applikationen individuell die Konstruktion natürlicher Ausdrücke bestimmen [32]. Anhand dieser Techniken können bereits gute Ergebnisse bzw. Erklärungen erzielt werden. Besonders textbasierte Dokumente können mit dieser Herangehensweise erklärt werden, da die Struktur des MMFGs der Struktur des Dokuments folgt [32]. Dies gilt allerdings nicht für andere Multimedia-Dateien, wie z.B. Bildern. Für diese Dateitypen bleibt die Reihenfolge der konstruierten Sätze zufällig [32].

Wichtig anzumerken ist, dass diese Konzepte ohne maschinelles Lernen auskommen und einen rein mathematischen und statistischen Ansatz verfolgen, um Erklärungen anhand identifizierter Merkmale zu generieren [32].

Verarbeitung von MMFGs

Der steigende Detaillierungsgrad (LOD) vieler multimedialer Inhalte führt zu einem Anstieg in der Anzahl an Elementen in einem MMFG [30]. So können in der Praxis MMFGs tausende Knoten und Kanten enthalten [29]. Ein höherer Detaillierungsgrad und eine höhere Anzahl an Elementen in einem MMFG stärkt zwar die Wirksamkeit der Ergebnisse von Berechnungen eines MMFGs, wie z.B der Berechnung der Ähnlichkeit zweier MMFGs, sorgt aber allgemein auch für eine komplexere Graphenstruktur [31]. Diese Eigenschaft von MMFGs birgt eine Reihe von Problemen. Zum einen ist die Umsetzung aktueller Algorithmen, wie z.B. zur Bestimmung der Ähnlichkeit von MMFGs, auf rechenschwachen Systemen, wie z.B Smartphones, mit dieser Anzahl an Knoten und Kanten aufgrund Speicherplatz und Laufzeitbegrenzungen nicht einfach zu handhaben und zum anderen sind aktuelle Algorithmen, die auf dem Durchlaufen eines Graphen basieren, wie z.B die Bestimmung der Ähnlichkeit zweier MMFGs, bei solch einer Anzahl an Knoten und Kanten nicht mehr effizient und zudem sehr rechenaufwendig [31].

Mathematisch betrachtet können Graphen durch ihre Adjazenzmatrix dargestellt werden [31]. Basierend auf einer Adjazenzmatrix können weitere Umformungen und Berechnungen durchgeführt werden [31]. Ein Beispiel hierfür ist die Umwandlung eines Graphen in einen n-dimensionalen Vektorraum. Auf diesem Vektorraum können dann wiederum weitere Berechnungen durchgeführt werden. Speziell für gewichtete Graphen, zu welchen auch per Definition MMFGs zählen, werden in der Regel Matrizen angewandt [31]. Diese Matrizen erweitern eine Adjazenzmatrix um die Gewichtung der Kanten. Im weiteren Verlauf dieser Arbeit werden diese Matrizen auch Wertungsmatrizen (engl. valuation matrices) genannt. Basierend auf diesen Darstellungen können mathematische Konzepte angewandt werden. Ein Beispiel für so ein mathematisches Konzept ist die Anwendung der Eigenwert-Methode, durch

welche eine auf Merkmale bezogene Ähnlichkeitsberechnung von Adjazenzmatrizen durchgeführt werden kann [31]. Die Eigenwert-Methode reduziert eine quadratische Matrix M auf eine einfache rationale Zahl λ unter der Annahme $\exists \vec{v} : M \cdot \vec{v} = \lambda \cdot \vec{v}$ [31]. Auf diese Weise können Ähnlichkeitsberechnungen auf λ anstatt der gesamten Matrix M durchgeführt werden [31]. Allerdings weisen viele dieser Algorithmen eine polynomiale Laufzeit von $\mathcal{O}((n + e)^2)$ oder gar exponentielle Laufzeit auf (hierbei verweist n auf die Anzahl der Knoten und e auf die Anzahl der Kanten) [31]. Um effektive und effiziente Verarbeitungen, wie z.B. Abfragen, für große Datenstrukturen und Sammlungen multimedialer Inhalte, wie einem MMFG, zu ermöglichen, müssen effizientere Algorithmen, wie zur Bestimmung der Ähnlichkeit von MMFGs, die mit komplexeren Graphenstrukturen umgehen können, eingesetzt werden [31].

Im nächsten Abschnitt wird eine Projektion eines Graphen in einen zweidimensionalen Vektorraum, sowie entsprechende Algorithmen vorgestellt. Diese Projektion stellt eine Erweiterung der Wertungsmatrix eines Graphen dar und ermöglicht statt einer exponentiellen Komplexität der Laufzeit eine lineare Laufzeitkomplexität von $\mathcal{O}(n + e)$ und unterstützt zudem ein höheren Detaillierungsgrad von MMFGs [31]. Das Konzept dieser Projektion wird im weiteren Verlauf der Arbeit als Graph Code [35] bezeichnet.

2.1.3 Graph Codes

In diesem Abschnitt werden die mathematischen und algorithmischen Konzepte der Graph Codes beschrieben. Zuerst werden die Grundlagen von Graph Codes beschrieben. Diese Grundlagen umfassen des Weiteren die in Abschnitt 2.1.3.1 beschriebene Codierung von Graph Codes, sowie die in Abschnitt 2.1.3.2 beschriebenen technischen Eigenschaften, wie Vokabular und Wörterbücher von Graph Codes. In Abschnitt 2.1.3.3 wird dann die Ähnlichkeit von Graph Codes behandelt.

Graph Codes sind eine Projektion eines MMFG in einen zweidimensionalen Raum [35]. Dabei basieren Graph Codes auf Operationen auf der Adjazenzmatrix eines Graphen und können allgemein, per Prinzip, auf jede Art von Graphen angewendet werden. Allerdings wurden Graph Codes spezifisch für MMIR entwickelt. Graph Codes werden eingesetzt, um MMFGs zu transformieren und in einer Form darzustellen, die schnelleres und effizienteres MMIR ermöglicht [31]. Grundlage dieser Umformung ist ein Algorithmus zur Codierung eines MMFG. Dieser Algorithmus nimmt die Wertungsmatrix WM eines MMFGs als Eingabe [35]. Um die grundlegenden Konzepte von Graph Codes vorzustellen, wird das Beispiel $MMFG_{ex}$ aus Abb. 2.3 als Basis benutzt. Eine Wertungsmatrix enthält jeweils eine Zeile und Spalte für einen Knoten in einem MMFG. Aufgrund dessen ist eine Wertungsmatrix auch eine quadratische Matrix. Eine gerichtete Kante zwischen zwei Knoten n_1 und n_2 wird in der Wertungsmatrix durch ihre Gewichtung oder durch den Wert 1 im Eintrag (n_1, n_2) der Wertungsmatrix WM dargestellt.

Die Knoten *Person*, *Body*, *Arm*, *Watch*, *Clock*, *attached to* des Beispiels $MMFG_{ex}$ werden in der Diagonalen der Wertungsmatrix ebenfalls durch den Wert 1 dargestellt. Dies führt zu der in Abb. 2.4 dargestellten Wertungsmatrix.

	Person	Body	Arm	Watch	Clock	attached to	b)
Person	1	1	0	0	0	0	$WM(MMFG_{ex}) =$
Body	0	1	1	0	0	0	$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$
Arm	0	0	1	1	0	0	
Watch	0	0	0	1	1	1	
Clock	0	0	0	0	1	0	
attached to	0	0	1	0	0	1	

Abb. 2.4: Wertungsmatrix des Beispiels $MMFG_{ex}$ in a) Tabellenform und b) Form einer Matrix.

Codierung von Graph Codes

Graph Codes wenden eine Funktion f_{enc} zur Codierung von Informationen in einem MMFG an. Diese Funktion codiert die Informationen zu den Typen der Knoten oder Kanten und ihren entsprechenden Attributen. Hierzu werden für alle Einträge bzw. Felder in der Wertungsmatrix WM , die nicht null sind, numerische Werte berechnet. Die Funktion f_{enc} zur Codierung dieser Informationen kann von Applikationen je nach ihren jeweiligen Anforderungen angepasst werden [35]. Eine beispielhafte Codierung kann wie folgt lauten: Knoten = 1, Blatt = 2, Kindbeziehung = 3, Synonymbeziehung = 4, Beziehung = 5, Knoten (räumliche Beziehung) = 6, Knoten (Synonym) = 7. Angewandt auf das Beispiel $MMFG_{ex}$ führt diese Codierung zu folgender in Abb. 2.5 a-b) dargestellten codierten Wertungsmatrix WM_{enc} bzw. Graph Code GC_{ex} . Die Farben der Einträge in Abb. 2.5 entsprechen der Färbung der Knoten aus Abb. 2.3. Die dazugehörigen Beziehungen sind ebenfalls entsprechend gefärbt. Abb. 2.5 c) hingegen zeigt den durch das GMAF generierten Graph Code GC_{ex} des Beispiels $MMFG_{ex}$. Es sei angemerkt, dass die Unterschiede zwischen dem Graph Code GC_{ex} aus Abb. 2.5 a-b) und Abb. 2.5 c) durch die vom GMAF eigens definierte Funktion f_{enc} zustande kommen. Des Weiteren inkludiert das GMAF die Synonym-Knoten *Clock* und *attached to* nicht in seiner Abbildung. Die Beziehung *attached to* wird im GMAF durch den Eintrag (*Arm*, *Watch*) im Graph Code

GC_{ex} dargestellt. Hier wird eine feinere Differenzierung der räumlichen Beziehung vorgenommen und durch den Wert 17 dargestellt. Durch komplexere Funktionen zur Codierung können Attribute, Gewichtung oder auch andere Informationen in einem MMFG codiert und durch beliebige Zahlen dargestellt werden, welche im weiteren Verlauf einer Verarbeitung gezielt genutzt werden können.

	Person	Body	Arm	Watch	Clock	attached to	
Person	1	3	0	0	0	0	$GC_{ex} =$
Body	0	1	3	0	0	0	$WM_{enc}(MMFG_{ex}) =$
Arm	0	0	1	3	0	0	$\begin{pmatrix} 1 & 3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 & 5 \\ 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 5 & 0 & 0 & 6 \end{pmatrix}$
Watch	0	0	0	2	4	5	
Clock	0	0	0	0	7	0	
attached to	0	0	5	0	0	6	

Abb. 2.5: Codierte Wertungsmatrix bzw. Graph Code GC_{ex} zum entsprechenden Beispiel $MMFG_{ex}$ in a) Tabellenform und b) in Form einer Matrix. c) Vom GMAF generierte Graph Code GC_{ex} .

Bei der Berechnung der Ähnlichkeit von Graph Codes müssen diese miteinander verglichen werden. Damit Graph Codes verglichen werden können, muss der Vergleich dieser Graph Codes auf einer genaustens definierten Metrik für Ähnlichkeit beruhen. Diese Metrik muss die mathematischen Aspekte eines Vergleichs von Matrizen und auch die semantischen Aspekte von Graph Codes berücksichtigen [35]. Im nächsten Abschnitt 2.1.3.2 werden weitere grundlegende Definitionen eingeführt und definiert, die zur Definition einer Metrik zur Ähnlichkeit von Graph Codes notwendig sind und eine Berechnung der Ähnlichkeit von Graph Codes unabhängig von den Typen der Knoten und Kanten ermöglicht.

Vokabular und Wörterbuch eines Graph Codes

Die Menge der Knoten V in einem MMFG, welche die identifizierten Merkmale darstellen, können als eindeutige Bezeichner für das Vokabular bzw. die Gesamtheit aller Wörter, welche einen MMFG ausmachen, betrachtet werden. Die Menge dieser Bezeichner wird mit $FVT_{MMFG} = \{fvt_1, \dots, fvt_n\}$ notiert und stellt somit die Elemente des „Wörterbuchs“ eines Graph Codes dar. Damit ein Bezeichner fvt_i eines Eintrages in einem Graph Code eindeutig identifiziert werden kann, wird ein Vektor

$dict_{GC}$ definiert. Dieser Vektor bietet eine geordnete Reihenfolge für die Menge an Bezeichner FVT_{MMFG} und weist jedem Bezeichner eine eindeutig definierte Position zu. Die Ordnung dieser Bezeichner im Vektor $dict_{GC}$ kann durch eine beliebige Strategie, wie z.B. durch Breitensuche, Tiefensuche oder manuell, erfolgen. Durch den Vektor $dict_{GC}$ kann jedes Feld in der Diagonalen des entsprechenden Graph Codes einem Eintrag im Wörterbuch, dargestellt durch $dict_{GC} = (fvt_1, \dots, fvt_n)$, zugeordnet werden [35]. Angewandt auf den Graph Code GC_{ex} ergibt sich für die Menge an Bezeichnern $FVT_{ex} = \{Person, Body, Arm, Watch, Clock, attached to\}$, sowie der Vektor $dict_{ex} = (Person, Body, Arm, Watch, Clock, attached to)$. Die Reihenfolge der Bezeichner in FVT_{ex} ist unerheblich und ist für gewöhnlich zufällig.

Anders als FVT_{ex} weist $dict_{ex}$ eine spezifische Reihenfolge auf, die in Tab. 2.2 dargestellt wird.

Index i	1	2	3	4	5	6
fvt_i	Person	Body	Arm	Watch	Clock	attached to

Tabelle 2.2: $dict_{ex}$ vom Graph Code GC_{ex} .

Beim Vergleichen von Graph Codes ist es wichtig nur die Einträge äquivalenter Merkmale in den Diagonalen der Graph Codes miteinander zu vergleichen. Da jeder Graph Code einen individuellen Vektor $dict_{GC}$ mit individueller Ordnung besitzt, unterscheiden sich die Wörterbücher der Graph Codes für gewöhnlich voneinander $dict_{GC_1} \neq dict_{GC_2}$. Äquivalente Merkmale können durch die Schnittmenge der Bezeichner bestimmt werden: $FVT_{1 \cap 2} = \{fvt_1, \dots, fvt_n\} = V_{MMFG_1} \cap V_{MMFG_2}$. Analog kann dieses Prinzip auch auf die Wörterbücher von Graph Codes angewendet werden: $dict_{1 \cap 2} = dict_{GC_1} \cap dict_{GC_2}$. Aufgrund der unterschiedlichen Ordnungen der Wörterbücher von Graph Codes können diese nicht einfach miteinander verglichen werden, da die Einträge in den zu vergleichenden Graph Codes unterschiedliche Bezeichner darstellen. Um diesem Problem zu begegnen und die Ordnung von Graph Codes anzupassen, gibt es eine Äquivalenzfunktion $f_{equ}(M_{\cap})$, die die Matrix der Schnittmenge der Wertematrizen bzw. das Wörterbuch so transformiert, dass das entsprechende Wörterbuch nach dem Vektor bzw. Wörterbuch $dict_{\cup}$ geordnet ist [35].

Konträr zur Schnittmenge kann auch eine Vereinigung von Graph Codes erfolgen. Eine Vereinigung kann dann erfolgen, wenn eine Applikation eine Sammlung an MMFGs enthält. Im Falle der Vereinigung von Graph Codes werden die Menge der Bezeichner, sowie die Wörterbücher der Graph Codes miteinander vereint. Die Gesamtheit aller Bezeichner FVT_{Coll} kann als Vereinigung aller Bezeichner in der Sammlung an MMFGs definiert werden und als Vereinigung aller Wörterbücher dargestellt werden $dict_{\cup} : FVT_{Coll} = \bigcup_{i=1}^n FVT_{MMFG_i}$ mit $\forall i, j < n : dict_{\cup} = dict_i \times dict_j$. Um die Vereinigung der Wörterbücher zu erstellen, wird jedes Wörter-

buch $dict_i$ durchlaufen und, sofern noch nicht im Wörterbuch $dict_{\cup}$ vorhanden, jeder individueller Bezeichner hinzugefügt. Wird ein $dict_{\cup}$ aus der vollständigen Sammlung an Graph Codes berechnet, so kann $dict_{\cup}$ als ein globales Wörterbuch betrachtet werden, in welchem jeder einzigartige Bezeichner eine einzigartige Position besitzt [35].

Die in diesem Abschnitt beschriebenen Definitionen ermöglichen eine effiziente und effektive Verarbeitung von Informationen eines Graph Codes. Ein Beispiel für so eine Verarbeitung ist die Berechnung der Ähnlichkeit zweier Graph Codes, die auf einem Vergleich dieser beiden basiert. Wie bereits erwähnt, muss der Vergleich dieser Graph Codes auf einer genaustens definierten Metrik beruhen. Im nächsten Abschnitt werden diese Metriken vorgestellt.

Ähnlichkeit von Graph Codes

In diesem Abschnitt wird eine Metrik zur Berechnung der Ähnlichkeit bzw. zum Vergleich von Graph Codes beschrieben. Die in diesem Abschnitt vorgestellten Metriken wurden aus [35] entnommen und ermöglichen die Quantifizierung der Eigenschaften von Graph Codes. Diese Metrik muss die Zeilen, Spalten und die Einträge, die die Knoten und Kanten darstellen, berücksichtigen. D.h. diese Metrik zur Berechnung der Ähnlichkeit von Graph Codes muss auf Matrizen anwendbar sein, dessen Zeilen und Spalten die Knoten und Kanten eines MMFGs darstellen. Damit ein Vergleich dieser Einträge möglich ist, müssen die Einträge vom selben Typ sein. Da die Wörterbücher von Graph Codes für gewöhnlich unterschiedlich sind, ist es zudem wichtig die korrekten Einträge in einer Matrix zu vergleichen [35]. Es kann zwischen drei Typen an Einträgen differenziert werden:

- Knoten (stellt Merkmale dar)
- Kanten (Beziehungen zwischen Knoten bzw. Merkmalen)
- Kantentyp (Typ der Beziehung zwischen Knoten bzw. Merkmalen)

Für jeden Typ eines Eintrages kann jeweils eine eigene Metrik definiert werden:

- Metrik für Merkmale M_F
- Metrik für Merkmalsbeziehungen M_{FR}
- Metrik für Beziehungstypen M_{RT}

Diese Metriken können wiederum als Tripel in einer Metrik $M_{GC} = (M_F, M_{FR}, M_{RT})$ zusammengefasst werden und bilden die gesamte Metrik, die die Ähnlichkeit von Graph Codes darstellen soll. Diese Metriken werden im Folgenden vorgestellt und beschrieben.

Die **Metrik für Merkmale** M_F berechnet die Ähnlichkeit von Graph Codes anhand der Schnittmenge der Bezeichner in einem Wörterbuch. Die Metrik M_F ist definiert als das Verhältnis zwischen der Kardinalität der Schnittmenge von Bezeichnern $dict_{\cap}$ der Wörterbücher $dict_i$ und $dict_j$ der Graph Codes GC_i und GC_j und der Kardinalität des Wörterbuchs $dict_i$ vom Graph Code GC_i . Die Formel für diese Metrik lautet $M_F(GC_i, GC_j) = \frac{|dict_{\cap}|}{|dict_i|}$. Hierbei bezeichnet $|\vec{v}|$ die Kardinalität eines Vektors, sprich die Anzahl der Elemente in einem Vektor. Anders formuliert, ist der Wert der Ähnlichkeit höher, desto mehr Merkmale zwei Graph Codes gemeinsam haben. Es ist wichtig anzumerken, dass diese Metrik nur Knoten bzw. Merkmale betrachtet, unabhängig von den Beziehungen zwischen diesen Merkmalen [35].

Die **Metrik für Merkmalsbeziehungen** M_{FR} ist die Grudlage für die Berechnung der Ähnlichkeit von Beziehungen zwischen Merkmalen. Beziehungen in einem Graph Code werden durch Einträge dargestellt, die nicht in der Diagonalen der Matrix vorkommen und deren Wert größer null ist. Dabei bezieht die Berechnung dieser Metrik nur Einträge aus der Schnittmenge M_{\cap} der Matrizen ein, die Beziehungen mit demselben Ausgangs- und Zielknoten darstellen. Die Berechnung dieser Metrik basiert auf den nicht in der Diagonalen vorkommenden Einträgen der Adjazenzmatrix $AM(M_{\cap})$. Die Metrik M_{FR} ist definiert als das Verhältnis zwischen der Summe aller nicht in der Diagonalen vorkommenden Einträge und der Kardinalität aller nicht in der Diagonalen vorkommenden Einträge. Die Formel für diese Metrik lautet $M_{FR}(GC_i, GC_j) = \frac{\sum_{i,j} AM(M_i \cap M_j) - n}{|AM(M_{\cap})| - n}$. Die Metrik M_{FR} stellt somit das Verhältnis zwischen der Zahl der Einträge, die Beziehungen darstellen und der Anzahl der äquivalenten und in der Schnittmenge vorkommenden Einträge, die Beziehungen darstellen, von Graph Codes. Somit zählt die Metrik M_{FR} die Anzahl aller Beziehungen zwischen Ausgangs- und Zielknoten. Die Metrik berücksichtigt jedoch nicht die Äquivalenz der Typen der Beziehungen. Die Äquivalenz der Typen von Beziehungen wird von der nächsten Metrik, der Metrik für Beziehungstypen berücksichtigt [35].

Die **Metrik für Beziehungstypen** M_{RT} basiert auf der Metrik für Merkmalsbeziehungen. Die Codierungsfunktion f_{enc} codiert für verschiedene Typen von Beziehungen unterschiedliche Werte. Eine Ähnlichkeit von Typen von Beziehungen setzt aber voraus, dass die Einträge, die die Beziehungen darstellen, vom selben Typ sind, damit sie verglichen werden können. Diese Metrik führt die Berechnungen nicht mehr anhand der Adjazenzmatrix der zu vergleichenden Graph Codes aus, sondern anhand der Schnittmenge M_{\cap} der codierten Wertungsmatrizen. Bei dieser Berechnung wird die Subtraktion auf äquivalenten Einträgen der zu vergleichenden Graph Codes durchgeführt. Ist das Ergebnis dieser Subtraktion gleich null, so ist der Typ der Beziehung äquivalent. Sind alle äquivalenten Einträge gleich, so ist die Summe der Subtraktionen dieser Einträge zueinander gleich null. Im Vergleich zur Metrik M_{FR} , die nur Berechnungen anhand einer Adjazenzmatrix durchführt, verwendet

diese Metrik zusätzlich den Wert der Codierung, welche den Typ der Beziehung darstellt, um festzustellen, ob die verglichenen Typen äquivalent sind. Die Metrik M_{RT} ist definiert als das Verhältnis zwischen der Summe aller nicht in der Diagonalen vorkommenden Einträge und der Kardinalität dieser Einträge. Die Formel für diese Metrik lautet $M_{RT}(GC_i, GC_j) = \frac{\sum_{i,j}^{n,i \neq j} (|M_{\cap i} - M_{\cap j}|)}{|M_{\cap i} - n|}$ [35].

Erklärbarkeit von Graph Codes

Analog zur Erklärbarkeit von MMFGs in Abschnitt 2.1.2.1, muss die technische Darstellung der Merkmale in eine für Menschen verständliche Bedeutung überführt werden. Auch hier gilt es in einem ersten Schritt die Lücke zwischen der technischen Darstellung und ihrer semantischen Darstellung zu überbrücken. Die semantische Darstellung von Merkmalen wurde in Abschnitt 2.1.2.1 bereits durch das Anbinden von externen Informationssystemen erreicht und resultierte in SMMFGs. Das Anwenden der Algorithmen von Graph Codes auf SMMFGs ermöglicht die semantische Darstellung von Merkmalen in Graph Codes und führt zu Semantischen Graph Codes (SGCs) [32]. Während das Wörterbuch eines Graph Codes die Menge der Bezeichner der Merkmale darstellt, stellt das Wörterbuch eines Semantischen Graph Codes die semantische Darstellung der Bedeutung dieser Bezeichner dar. Analog zu MMFGs können auch für Graph Codes, durch das Definieren und Anwenden einer formalen kontextfreien Phrasenstruktur-Grammatik in Kombination mit Semantischen Graph Codes, für Menschen lesbare, verständliche und auf natürlicher Sprache basierende Ausdrücke generiert werden. Die Erweiterung von Graph Codes um eine formale kontextfreie Grammatik führt zu Erklärbaren Semantischen Graph Codes (ESGCs) [30].

In Bezug auf Graph Codes ist wichtig anzumerken, dass Graph Codes nicht einen allgemeinen Ersatz bzw. eine allgemeine Alternative zu MMFGs darstellen und nur in bestimmten Fällen, wie z.B. beim Vergleich von MMFGs zur Bestimmung der Ähnlichkeit, eine Verbesserung bieten [35]. Des Weiteren werden Erklärungen von Graph Codes bislang nur anhand statistischer und rein mathematischer Vorgehensweisen erzeugt. Es existiert keine Möglichkeit oder Untersuchung Systeme generativer KI zur Erklärbarkeit von Graph Codes zu nutzen. Dies führt zur ersten offenen Herausforderung.

Offene Herausforderung OH 1.1 - Erklärbarkeit durch generative KI

Einsatz von generativer KI als Ersatz bzw. Alternative zur statistischen und rein mathematischen Vorgehensweise, um Erklärungen zu generieren.

Die offene Herausforderung **OH 1.1** hat in Bezug auf dieses Forschungsziel einen konkretisierenden Charakter. Das Forschungsziel bezieht sich in erster Linie auf Er-

klärbarkeit von MMIR im Allgemeinen und untersucht hierfür bereits bestehende Verfahren, mit welchen Erklärungen zu Datenstrukturen, wie den MMFGs oder Graph Codes, erzeugt werden können. Die offene Herausforderung bezieht sich hingegen direkt auf die Datenstruktur der Graph Codes und verdeutlicht, dass es keine Möglichkeit gibt, mittels generativer KI Erklärungen für Graph Codes zu erzeugen.

Die offene Herausforderung **OH 1.1** wird im Rahmen einer Modellierung, sowie Implementierung in jeweils FZ 1.2/TB und FZ 1.3/I angesprochen und untersucht.

2.1.4 Generative KI

Generative künstliche Intelligenz (GenKI, engl. generative AI (GenAI)) ist ein Typ künstlicher Intelligenz, die es ermöglicht, anhand vorhandener Informationen und Vorgaben, eine große Vielfalt an neuen Inhalten zu erstellen. Beispiele für diese Inhalte können Bilder, Videos, Audiodateien, Texte oder anderweitige synthetische Daten sein [36]. Synthetische Daten bezeichnet Daten, die nicht auf natürliche, sondern auf künstliche Art und Weise durch eine KI erstellt wurden, um echte Daten zu erweitern, oder zu ersetzen [37]. Damit eine generative KI synthetische Daten erzeugen kann, wird das Modell der KI mithilfe eines Algorithmus anhand echter Daten so trainiert, dass es in der Lage ist, die ursprüngliche Struktur und die Eigenschaften, wie Muster, Zusammenhänge und statistische Merkmale, der echten Daten zu reproduzieren [38, 39]. Generative KI wird daher als generativ bezeichnet, da sie auf Grundlage bereits vorhandener Datensätze etwas Neues bzw. etwas bisher nicht Dagewesenes erzeugt. Diese Eigenschaft hebt generative KI wesentlich von diskriminativer KI ab, welche einzig und allein zwischen verschiedenen Eingaben differenzieren kann [40].

Ein Anwendungsbeispiel für eine diskriminative KI wäre eine simple binäre Klassifikation von Bildern und ob ein Bild eine Katze darstellt bzw. enthält, oder nicht. Mit genügend Daten kann das diskriminative Modell einer KI so trainiert werden, dass es bestimmte Eigenschaften, wie Formen und Texturen, die einer Katze gleichen, in einem Bild erkennen kann. Anhand erkannter Merkmale kann das Modell eine Vorhersage treffen, ob in einem Bild eine Katze zu sehen ist, oder nicht. Eine wichtige Voraussetzung für das Training dieses Modells ist es, dass jede Beobachtung in den Trainingsdaten manuell mit einer Bezeichnung versehen ist. Im Fall einer binären Klassifikation sind die Bilder, die Katzen darstellen bzw. enthalten, mit der Bezeichnung 1 versehen, alle anderen Bilder mit der Bezeichnung 0. Durch das Training erlernt das diskriminative Modell beide Gruppen zu unterscheiden.

Die diskriminative Modellierung ist daher gleichbedeutend mit dem überwachten maschinellen Lernen (engl. supervised machine learning) [41]. In der Tat wird ein Großteil der diskriminativen Modelle für überwachtes maschinelles Lernen verwendet [42]. Diskriminative KI bildet also die Eingabe eines mit Bezeichnungen versehenen

Datensatzes auf eine Ausgabe ab. Im Vergleich dazu benötigt das Training eines generativen Modells für gewöhnlich keine explizit mit Bezeichnungen versehenen Datensätze und kann somit mit dem nicht überwachten maschinellen Lernen (engl. unsupervised machine learning) gleichgesetzt werden. Das Training auf einem mit Bezeichnungen versehenen Datensatz ist damit allerdings nicht ausgeschlossen und kann auch Verwendung finden [41].

Es ist wichtig anzumerken, dass generative KI, die auf der Methode des Deep Learnings aufbaut, nicht neu ist und bereits in den 60ern in Form von Chatbots vorgestellt wurde [36]. Die erste Instanz eines solchen Chatbots hieß „ELIZA“ [43] und wurde 1966 am MIT entwickelt und hatte zur Aufgabe Gespräche mit einem Psychotherapeuten zu simulieren [40]. Durch das Aufkommen von künstlichen neuronalen Netzwerken (engl. articial neural networks (ANN)) und entsprechender Algorithmen in den 80ern und 90ern, sowie durch bessere Datenlage und Rechenkapazitäten wurde die Methode des Deep Learning praxistauglicher. Die in 2014 von Ian Goodfellow vorgestellte Technik der generativen gegnerischen Netzwerke (engl. generative adversarial networks (GAN)) führte zu einem Durchbruch und ermöglichte es generativer KI überzeugend wirkende neuartige Inhalte zu generieren [44].

Grundlegende Techniken und Konzepte

In diesem Abschnitt wird genauer auf wichtige Techniken und Technologien eingegangen, die in aktuellen generativen KI-Systemen Verwendung finden. Die Techniken und Technologien werden kurz beschrieben und erläutert. Zuerst werden die allgemeinen Techniken, wie Machine Learning oder Deep Learning vorgestellt bzw. rekapituliert und weiter wichtige Konzepte im Bereich der generativen KI angesprochen. Die in diesem Abschnitt vorgestellten Techniken bzw. Technologien sind dabei so geordnet, dass sie einer logischen Abfolge in Bezug auf ihre Abhängigkeiten zueinander folgen.

Machine Learning (z.Dt. maschinelles Lernen) ist ein Teilbereich der KI [45] und hat zum Ziel die Art und Weise, wie Menschen lernen zu imitieren [13]. Maschinelles Lernen ermöglicht es Systemen, Muster und Zusammenhänge automatisch und selbstständig aus Daten und Erfahrungen zu erlernen und zu verbessern, ohne dabei explizit programmiert zu sein [46]. Damit Machine Learning funktioniert und Muster und Zusammenhänge erlernt werden können, muss ein Datensatz zum Training vorhanden sein [46]. Im Bereich der KI spielen (**un**)-supervised machine learning (z.Dt. (un)-überwachtes maschinelles Lernen) eine zentrale Rolle [45]. Überwachtes Lernen zeichnet sich dadurch aus, dass Algorithmen mithilfe von Daten trainiert werden, die im Trainingsdatensatz manuell mit Bezeichnungen versehen wurden [46]. Daher ist überwachtes Lernen mit einem relativ hohen menschlichen Aufwand verbunden [46]. Weit verbreitete Problemstellungen bzw. Aufgaben, in de-

nen überwachtes Lernen eingesetzt werden, sind die Klassifikation und Regression [47]. Unüberwachtes Lernen zeichnet sich hingegen dadurch aus, dass es ohne die Einflussnahme von Menschen Gemeinsamkeiten, Unterschiede, Gruppen und Muster in Daten erkennen kann [46]. Unüberwachtes Lernen hat, aufgrund seiner Fähigkeit Muster eigenständig erlernen zu können, viele Anwendungsfälle und kann besonders für die Bildanalyse, wie Objekterkennung, oder für die Erkennung von Auffälligkeiten in Daten Verwendung finden [48].

Deep Learning (z.Dt. tiefes Lernen) ist eine besondere Form des maschinellen Lernens und ist in der Lage große Datenmengen zu verarbeiten [49]. Im Mittelpunkt des tiefen Lernens stehen **Artificial Neural Networks (ANNs)** (z.Dt. künstliche neurale Netzwerke (KNN)), die ebenfalls ein Teilgebiet des maschinellen Lernens sind [50]. Künstliche neuronale Netzwerke sind Algorithmen, deren Struktur und Verhalten nach dem biologischen Vorbild des menschlichen Gehirns und seiner Neuronen modelliert ist [49, 51]. Neuronale Netzwerke ermöglichen somit das Erlernen von komplexen Mustern und sind in der Lage genaue Vorhersagen zu treffen. Da tiefes Lernen sehr rechenintensiv ist, kann es Monate dauern, bis Vorhersagen und Entscheidungen gute Ergebnisse liefern [51]. Ein sehr bekanntes Beispiel für ein neuronales Netzwerk ist der Suchalgorithmus von Google [50]. Bekannte Modelle tiefen Lernens im Bereich der generativen KI sind **Generative Adversarial Networks (GANs)** [52] und **Variational Auto Encoders (VAEs)** [53].

Large Language Models (LLMs) (z.Dt. großes Sprachmodell) sind große generative Sprachmodelle und stellen einen Teilbereich der KI dar [54]. Eine sehr prominente Architektur von großen Sprachmodellen sind Transformer [54]. Die Architektur von Transformern basiert auf neuronalen Netzwerken und zeichnet sich dadurch aus, dass es Kontext und Bedeutung durch Beobachtung der Beziehungen zwischen Wörtern erlernen kann [54]. Transformer wurden zuerst von Google in „Attention Is All You Need.“ [55] vorgestellt und führen zwei neue Techniken ein: positionelle Kodierung und Selbstaufmerksamkeit [54]. Große Sprachmodelle werden meist durch unüberwachtes Lernen auf riesigen Datenmengen trainiert und weisen in der Praxis Milliarden an Parameter auf [54]. Als solches erlernen LLMs die Muster der Statistiken und Wahrscheinlichkeitsverteilungen innerhalb der Trainingsdaten und optimieren mit diesen die Genauigkeit ihrer Vorhersagen anhand ihrer Anwendungsfälle [56]. Anwendungsfälle von Transformern sind vielfältig und können das z.B. Erzeugen, Übersetzen oder Zusammenfassen von Text umfassen [54].

Explainable AI (xAI) (z.Dt. erklärbare KI) befasst sich mit der Erklärbarkeit von KI. Erklärbare KI beschäftigt sich u. a. mit der Frage „Wie kommt eine KI zu einem Ergebnis“ und versucht die inneren Abläufe in einer KI zu beleuchten [57]. Das generelle Problem mit KI ist, dass die Algorithmen wie neuronale Netzwerke und tiefes Lernen sogenannte „Black-Boxes“ sind [57]. Zwar ist die Funktionsweise dieser Algorithmen, sowie das finale Ergebnis bekannt, aber es ist nicht erkenntlich

wie die KI dieses Ergebnis erreicht hat [57]. Ziel von erklärbarer KI soll also sein, dass ein Mensch nachvollziehen kann, wie ein Algorithmus ein Ergebnis erreicht [57].

Die meisten Systeme generativer KI arbeiten mit einer sogenannten **Prompt** (z.Dt. Eingabeaufforderung) als Eingabe. Im Rahmen der Verarbeitung natürlicher Sprache und des maschinellen Lernens bezeichnet ein Prompt ein Stück Text, das als eine Anweisung an ein System verstanden werden kann und welches den Kontext und alle dazugehörigen und notwendigen Informationen enthält, damit ein Modell relevante Ausgaben erzeugen kann [58, 59]. Anwendungsbeispiele für Prompts können Erklärungen, Fragen oder anderweitige Texteingaben sein. Das Modell eines Systems verarbeitet die Eingabe und produziert in Abhängigkeit der Eingabe und der Fähigkeiten des jeweiligen Modells entsprechende Ausgaben, wie z.B. Vervollständigungen von Text, Erklärungen, Übersetzungen, Zusammenfassungen oder Bilder. Beim Diriģieren des Verhaltens eines Systems spielen Prompts daher eine entscheidende Rolle [60]. Grundlegende Elemente oder Inhalte einer Prompt können Anweisungen, Kontext, Eingabedaten oder Angaben zur gewünschten Ausgabe umfassen [61]. Bereits durch einfache Prompts können viele Aufgaben ausgeführt werden, jedoch hängt die Qualität der Ausgabe von der Qualität der Eingabe ab [61]. Durch das Bereitstellen von mehr Informationen in einer Prompt können die Wahrscheinlichkeiten zum Generieren gewünschter Ausgaben erhöht werden. Dieser Ansatz, Prompts anzupassen, um optimale Ergebnisse zu erzielen, heißt **Prompt Engineering** (z.Dt. Entwicklung von Eingabeaufforderungen) [62]. Prompt Engineering ist, in Essenz, ein iterativer, experimenteller Prozess. Begonnen wird zumeist mit einfachen Prompts, die iterativ und inkrementell weiterentwickelt werden. Einfache Anpassungsmöglichkeiten sind Prompts schlicht, kurz und knapp zu halten und eine möglichst präzise Formulierung zu verwenden. Diese Anpassungsmöglichkeiten können durch bekannte Techniken des Prompt Engineering, wie Zero-Shot [63], Few-Shot [64] und Chain-of-Thought (CoT) [65] erweitert werden.

Zero-Shot ist eine Technik, in der Modelle wie GPT-3.5 aufgrund der großen Datenmengen, mit welchen sie trainiert wurden, in der Lage sind Prompts ohne vorher spezifizierte Informationen oder Beispiele wirksam auszuführen [63]. Ein Anwendungsbeispiel für Zero-Shot Prompting ist die folgende Instruktion einen Text in eine andere Sprache zu übersetzen.

Eingabe: Übersetze „Das Auto hat einen Platten“ in Englisch.

Ausgabe: „The car has a flat tire“.

Few-Shot beschreibt eine Technik, die sich dadurch auszeichnet, dass der Prompt Instruktionen in Form von ergänzenden Informationen zum Ausführen der Prompt beigefügt werden [64]. Ein Anwendungsbeispiel für Few-Shot Prompting ist die folgende Reihe an Prompts, die als Beispiele dienen.

Eingabe: „Hauptstadt der Niederlande ist ...“.

Ausgabe: „Amsterdam“.

Eingabe: „Hauptstadt von Norwegen ist ...“.

Ausgabe: „Oslo“.

Eingabe: „Hauptstadt von Deutschland ist ...“.

Ausgabe: „Berlin“.

In diesem Beispiel wird dem Modell eine Reihe an Beispielen für das gewünschte Verhalten bereitgestellt. Anhand dieser Beispiele soll das Modell das Muster erkennen und erlernen, um ähnliche Fragen beantworten zu können.

Chain-of-Thought hingegen zeichnet sich dadurch aus, dass die Anweisung, die mit einer Prompt vermittelt werden soll, in eine Reihe an Schritten heruntergebrochen wird [66]. Für jeden Schritt in dieser Reihe wird wiederum ein Prompt definiert. Ein Anwendungsbeispiel für eine leicht modifizierte Technik von CoT ist die folgende Prompt.

Aussage: Die ungeraden Zahlen in folgender Zahlenreihe ergeben eine gerade Zahl: 6, 4, 7, 12, 19, 1, 2. Gehe schrittweise vor.

Antwort:

1. Alle ungeraden Zahlen in der Reihe identifizieren: 7, 19, 1.
2. Alle ungeraden Zahlen addieren: $7 + 19 + 1 = 27$.
3. Feststellen, dass 27 eine ungerade Zahl ist.
4. Erkennen, dass die ursprüngliche Aussage „Die ungeraden Zahlen in folgender Zahlenreihe ergeben eine gerade Zahl: [...]“ eine falsche Aussage ist.

In einem ersten Schritt werden die gegebenen Informationen betrachtet und aus der gegebenen Zahlenreihe alle ungeraden Zahlen identifiziert. In einem zweiten Schritt formuliert das Modell eine Gleichung zum Summieren der ungeraden Zahlen. Subsequent wird die aufgestellte Gleichung gelöst und erkannt, dass die Summe der ungeraden Zahlen wiederum eine ungerade Zahl ist. Im letzten Schritt wird die ursprüngliche Aussage mit den neu gewonnenen Fakten verglichen und es wird erkannt, dass die ursprüngliche Aussage falsch ist.

Das Beispiel endet mit dem Satz „Gehe schrittweise vor“ (engl. Lets think step by step). Durch diesen Satz wird die Anweisung in der Prompt in eine Reihe an Schritten zerteilt. Das schrittweise Vorgehen hilft dem Modell das Problem besser zu verstehen und die richtige Antwort zu liefern. Diese Technik wurde zuerst in [67] beschrieben und wird auch Zero-Shot CoT genannt und kombiniert Zero-Shot und CoT miteinander. Es ist eine leistungsstarke Technik, die eine nicht zu unterschätzen.

zende Wirkung auf Schlussfolgerungen des Modells hat und die zur Verbesserung der Leistung von Modellen bei einer Vielzahl von Aufgaben eingesetzt werden kann.

Prompt Engineering ist ein sich rasant entwickelndes Feld im Bereich der generativen KI und stellt mittlerweile auch einen eigenen Beruf dar. Da Prompt Engineering ein iterativer und inkrementeller Prozess ist, werden stetig neue Techniken entwickelt, oder alte Techniken in neuer Weise miteinander kombiniert.

Systeme generativer KI

Systeme generativer KI haben einen enormen Einfluss auf die Weltwirtschaft [68]. So kann, laut Schätzungen, generative KI die Produktivität der weltweiten Wirtschaften erhöhen und, unter Berücksichtigung vieler Anwendungsfälle, pro Jahr einen Mehrwert in Höhe von \$2.6 bis \$4.4 Billionen schaffen [68]. Dieser steigende wirtschaftliche Anreiz führt zu einer explosionsartigen und rasanten Entwicklung im Bereich der generativen KI und weltweit zur Entwicklung, Integration und Anwendung generativer KI [68, 69]. Aus diesem Grund ist eine Aufführung, oder gar eine ausführlichere Liste solcher Systeme im Rahmen dieser Arbeit nicht machbar. Daher werden in diesem Abschnitt nur ausgewählte Systeme generativer KI vorgestellt und kurz beschrieben. Die Auflistung dieser Systeme erfolgt nach den Kategorien Image und Text, sowie Audio und Video.

Die Kategorie **Image** zählt eine Auswahl an Programmen auf, die auf KI basieren und in der Lage sind, aus Ausdrücken natürlicher Sprache, Bilder und Kunst zu erzeugen. Diese Prozesse der Bilderzeugung nennt man Text2Image (z.Dt. Text zu Bild). Ein paar der bekanntesten Programme sind:

- **DALL · E 2** [17] ist eine von OpenAI [70] entwickelte KI, die mittels Text To Image realistische Bilder und Kunst schaffen kann.
- **Stable Diffusion** [71] ist eine von StabilityAI [72] entwickelte KI, die ebenfalls auf Grundlage von Ausdrücken natürlicher Sprache Bilder erzeugen kann. Im Vergleich zu DALL · E 2 ist Stable Diffusion eine Open Source Software.
- **Midjourney** [73] ist eine KI, die auf Grundlage einer Textbeschreibung Bilder und Kunst erstellen kann.

Die Kategorie **Text** zählt eine Auswahl an Programmen auf, die auf KI basieren und anhand von Eingaben eines Nutzers, meist einer sogenannten „Prompt“, die auf Ausdrücken natürlicher Sprache basieren, wiederum Text zu generieren.

- **OpenAI API** bietet Zugang zu OpenAIs Sprachmodellen, wie GPT-3.5 [74] und GPT-4 [74]. Diese Sprachmodelle sind in der Lage anhand Instruktionen in Form von Ausdrücken natürlicher Sprache eine Reihe an Aufgaben durchzuführen.

- **Google Vertex AI** ist eine Plattform für maschinelles Lernen. Die **Vertex AI API** bietet Zugang zu Googles PaLM 2 [75], einer Familie an Sprachmodellen [76]. Spezialisierungen von PaLM 2 umfassen Med-PaLM für medizinische Anwendungsfälle und Sec-PaLM 2 für sicherheitsbezogene Anwendungsfälle [76]. PaLM 2 treibt generative KI-Funktionen, wie die PaLM API, -Tools, sowie Bard [77], an [76].

Im Rahmen dieser Arbeit sind vor allem Systeme zur Text- sowie Bilderzeugung von vorrangigem Interesse. Generative KI ist allerdings nicht auf Text- oder Bilderzeugung beschränkt, sondern kann auch in anderen Bereichen, wie der Audio- oder Videoerzeugung Verwendung finden. Diese Systeme können, wie bereits in Abschnitt 1.1 beschrieben, einen echten Mehrwert für Benutzer darstellen. Daher wird im Folgenden noch ein kurzer Überblick über die Kategorien Audio und Video gegeben.

In der Kategorie der Programme zur Audioerzeugung gibt es bekannte Systeme, wie die von Eleven Labs [78], Resemble AI [79] oder Bark [80]. Anwendungsfälle für diese Systeme können multilinguale Synchronisation, das Klonen von Stimmen, oder die automatische Erzeugung von Untertiteln in Videos sein. In der Kategorie der Programme zur Videoerzeugung sind bekannte Systeme, wie die von RunwayML [81], Synthesia [82] oder Rephrase AI [83]. Anwendungsfälle für diese Art von Systemen können schnelle Videoerzeugung, Videobearbeitung, vor allem im Bereich Marketing oder Nachrichtensendungen sein.

Die nachfolgende Abb. 2.6 zeigt eine Zusammenfassung und kurze Übersicht über ausgewählte aktuelle Systeme generativer KI zusammen mit ihren entsprechenden Anwendungsmöglichkeiten. Es ist wichtig zu betonen, dass diese Übersicht in keiner Weise Anspruch auf Vollständigkeit erhebt, da sich diese Landschaft rapide verändert und ständig neue Techniken, Technologien und Systeme entwickelt werden.

Einschränkungen

Systeme generativer KI, wie z.B. das große Sprachmodell GPT-3 von OpenAI, sind bereits in der Lage realistische und höchst überzeugende Inhalte zu erzeugen. Trotzdem unterliegen Systeme generativer KI einer Reihe an Einschränkungen, die sich potentiell auf ihre Fähigkeiten und Anwendungsfälle auswirken können. Einige der wichtigsten Einschränkungen von Systemen generativer KI sind:

1. **Mangelnde Transparenz und Erklärbarkeit:** Modelle generativer KI sind sog. Black-Boxen und bieten keinen Einblick in ihren Entscheidungsprozess [57]. Aufgrund dieser mangelnder Transparenz ist es nicht ersichtlich, wie ein Modell zu einer Entscheidung und zu einem Ergebnis kommt [57]. Resultat ist eine eingeschränkte Erklärbarkeit von Modellen.

Anwendungsschicht	Schreiben	Design		Synchronisation
	Notizen	Bild erzeugung		Klonen von Stimmen
	Support (Chat / E-Mail)	Medien / Werbung	Video- bearbeitung / erzeugung	Automatische Erzeugung von Untertiteln
Text		Image	Video	Audio
Modell ebene	OpenAI GPT-3.5 / GPT-4	OpenAI Dall-E 2	RunwayML	Eleven Labs
	DeepMind Gopher	Stable Diffusion	Synthesia	Resemble AI
	Anthropic Claude	Midjourney	Rephrase AI	Bark

Abb. 2.6: Übersicht ausgewählter aktueller Systeme generativer KI nach [84].

2. **Abhängigkeit von Trainingsdaten und ethische Bedenken:** Modelle generativer KI sind Produkte ihrer Trainingsdaten und daher stark von ihnen abhängig [85]. Aufgrund dieser Abhängigkeit können sich Ungenauigkeiten aus den Trainingsdaten in den generierten Ergebnissen widerspiegeln [85]. Des Weiteren sind diese Modelle zumeist speziell auf Bereiche angepasst und lassen sich nur schwer auf andere Bereiche anwenden, die nicht von den Trainingsdaten vorgesehen sind [85]. Ein weiteres Problem generativer KI sind ethische Bedenken [85]. Durch Voreingenommenheiten in den Trainingsdaten können Modelle selbst voreingenommene oder gar schädliche Inhalte erzeugen [85].
3. **Mangelndes Verständnis des Kontextes:** Die Länge des Kontextfensters eines Modells begrenzt die Menge an vorausgehendem Text, welches ein Modell bei der Generierung einer Antwort auf eine Eingabeaufforderung berücksichtigt [86]. Informationen, die außerhalb dieses Kontextfensters liegen, werden unter Umständen nicht mehr berücksichtigt und können zu einem unvollständigen Verständnis des Textes oder zu kontextuell verfälschten Antworten führen [86]. In Kombination mit den ausgewählten Trainingsdaten beeinflusst die Länge des Kontextfensters somit maßgeblich die Fähigkeiten des Modells den Kontext zu verstehen, langfristige Abhängigkeiten zu erfassen und kohärente Texte zu generieren [86].

Trotz der bemerkenswerten Fähigkeit ansprechende und überzeugende Inhalte zu generieren, ist es wichtig die Einschränkungen und Grenzen generativer KI im Blick zu behalten, um so einen effektiven, verantwortungsbewussten und informierten Einsatz von generativer KI garantieren zu können. Im Bereich der Forschung gibt es

laufende Bemühungen und Weiterentwicklungen, um diese Einschränkungen und Grenzen zu adressieren, abzustellen, die Zuverlässigkeit von generativer KI zu verbessern, sowie Fähigkeiten von Systemen generativer KI weiter auszubauen. Diese Bestrebungen können allgemein unter dem Begriff „Responsible AI“ zusammengefasst werden, welche zum Ziel haben KI auf verantwortungsvolle Art und Weise zu entwickeln und die oben genannten Einschränkungen zu berücksichtigen [87].

2.1.5 Diskussion

In den ersten Abschnitten 2.1.1 bis 2.1.3 wurden zuerst grundlegende Technologien in Bezug auf das GMAF, welches das Bezugssystem dieser Arbeit darstellt, vorgestellt und detailliert erläutert. Weiter wurden in Abschnitt 2.1.4 grundlegende Techniken und Konzepte im Rahmen der generativen KI genannt und erklärt, sowie aktuelle Systeme generativer KI nach Kategorien gelistet.

Im Rahmen der Diskussion dieses Forschungsziels werden die Anwendungsmöglichkeiten der in Abschnitt 2.1.4.2 vorgestellten Systeme in Bezug auf das GMAF diskutiert. Anhand dieser Diskussion wird dann eine fundierte Entscheidung getroffen, welche Systeme generativer KI für die Umgebung „Erklärbarkeit von MMIR mittels generativer KI“ am besten geeignet sind.

Damit eine fundierte und informierte Entscheidung für die Auswahl von Systemen generativer KI getroffen werden kann, gilt es einige Kriterien zur Auswahl von Systemen zu beachten. Wichtige Kriterien zur Auswahl eines Systems sind:

- **Anpassungsmöglichkeiten:** Wie flexibel ist ein System? Ermöglicht es die Anpassung anhand Anforderungen besonderer Anwendungsfälle?
- **Kompatibilität und Integration:** Ist ein System mit der Programmiersprache oder der Umgebung des GMAF kompatibel? Bietet das System eine Schnittstelle, um eine einfache Integration des Systems in das GMAF zu ermöglichen? Bietet das System eine klare Dokumentation, um die Integration der Schnittstelle zu unterstützen?
- **Skalierbarkeit und Leistung:** Ist das System in der Lage größere Datens Mengen zu verarbeiten? Kann es auf Anfrage zeitnah Ergebnisse generieren?
- **Kosten und Lizenzierung:** Einige Systeme bieten kostenfreie Nutzung oder berechnen die Kosten abhängig von der Nutzung, wiederum erfordern andere Systeme eine kommerzielle Lizenz zur Nutzung. Welche Lizenzbedingung oder Preise sind mit der Nutzung eines Systems verbunden?

Das Berücksichtigen dieser Kriterien in der Auswahl eines Systems generativer KI ermöglicht eine wohl informierte und fundierte Entscheidung über die Integration eines Systems in das GMAF und legt das Fundament für das Erreichen von Anforde-

rungen und Zielen. Daher wird zuerst die Auswahl eines Systems aus der Kategorie Image zur Bildgenerierung diskutiert. Danach folgt eine Diskussion zur Auswahl eines Systems aus der Kategorie Text zur Textgenerierung. Die Diskussion orientiert sich dabei jeweils nach den Systemen und nicht nach den oben genannten Kriterien.

Ausgewählte Systeme zur Bildgenerierung

Zur Auswahl eines Systems zur Bildgenerierung stehen OpenAIs DALL·E 2, StabilityAIs Stable Diffusion und Midjourney. Um einen besseren Eindruck zu den jeweiligen Modellen bzw. Systemen zu ermöglichen, wird für jedes System ein Bild anhand der Textbeschreibung bzw. Prompt „*Beautiful Venice canals with gondolas and bridges, charming*“ generiert.

Midjourney ist eine generative KI, die auf Grundlage einer Textbeschreibung überzeugende und ästhetisch ansprechende Bilder erzeugen kann. Abb. 2.7 zeigt ein Beispiel für ein von Midjourney generiertes Bild. Um auf Midjourney zugreifen zu



Abb. 2.7: Beispiel für ein von Midjourney generiertes Bild anhand oben genannter Textbeschreibung [88].

können, wird eine Anmeldung über den Onlinedienst Discord vorausgesetzt. Weitere Voraussetzungen zum Nutzen Midjourneys sind das Hinzufügen und Beitreten des Discord-Servers von Midjourney und das Akzeptieren der allgemeinen Nutzungsbedingungen. Sind diese Voraussetzungen erfüllt, kann Midjourney über Befehle im Discord-Chat gesteuert werden. Bilder können über den Befehl /imagine und einer Textbeschreibung **prompt** erzeugt werden. Midjourney bietet eine Reihe an Anpassungsmöglichkeiten seiner Prozesse, die gezielt durch Befehle, die in [89] dokumentiert sind, vorgenommen werden können. Midjourney bietet in einem begrenztem Testversuch ein initiales Guthaben von 25 Credits zum Erstellen von Bildern. Sind diese Credits aufgebraucht, kann aus einer Reihe an unterschiedlichen, gestaffelten Mitgliedschaften (siehe [90]), welche jeweils eigene Vorteile bieten, gewählt werden. Eine Integration von Midjourney in eine andere Software ist aufgrund einer fehlenden Schnittstelle oder Anbindung bzw. Bibliothek nicht möglich. Zwar gibt es Drittanbieter, die eine Schnittstelle bereitzustellen scheinen, allerdings verweist Midjourney in seinen allgemeinen Nutzungsbedingungen ausdrücklich, dass es keine Schnittstelle bereitstellt und Automatisierungen nicht duldet [91].

Stable Diffusion von StabilityAI ist eine quelloffene generative KI, die ebenfalls auf Grundlage einer Textbeschreibung Bilder erzeugen kann. Abb. 2.8 zeigt ein Beispiel für ein von Stable Diffusion generiertes Bild. Zugriff auf Stable Diffusion kann durch eine Installation auf einer lokalen Maschine oder durch das WebInterface DreamStudio erfolgen. In beiden Fällen erfolgt der Zugriff durch ein WebInterface, in welchem eine Reihe an Anpassungsmöglichkeiten durch Einstellungen vorgenommen werden können. Aufgrund der hohen Rechenanforderungen benötigt eine lokale Installation von Stable Diffusion eine hohe Mindestanforderung an VRAM. Auch Stable Diffusion bietet in einem begrenztem Testversuch ein initiales Guthaben von 25 Credits. Sind diese Credits aufgebraucht, können für einen Mindestwert von 10 Euro Credits hinzugekauft werden. Die Preise für die Generierung von Bildern sind abhängig von den gewählten Eigenschaften der Bilder und können in [92] eingesehen werden. Eigenschaften können die Bildgröße, aber auch die Anzahl der Iterationen der Bildgenerierung sein. Diese Einstellungen haben einen Einfluss auf die Qualität der generierten Bilder und folglich auch auf die Rechenintensität und beeinflussen dementsprechend die Preise zur Generierung von Bildern. Eine Integration von Stable Diffusion ist durch eine offizielle REST (Representational State Transfer) [93] Schnittstelle möglich.

DALL · E 2 ist eine generative KI, die ebenfalls auf Grundlage einer Textbeschreibung Bilder erzeugen kann. Abb. 2.9 zeigt ein Beispiel für ein von DALL · E 2 generiertes Bild. Zugriff auf DALL · E 2 erfolgt über ein WebInterface. OpenAI bietet monatlich eine freie Anzahl an Credits, mit welchen Bilder erzeugt werden können. Sind diese Credits aufgebraucht, kann für einen Mindestwert von 15 Dollar 115 Credits hinzugekauft werden. Im Vergleich zu Stable Diffusion sind die Preise zur Bild-



Abb. 2.8: Beispiel für ein von Stable Diffusion generiertes Bild anhand oben genannter Textbeschreibung.

generierung von DALL·E 2 nur von der Bildgröße abhängig (siehe [94]). OpenAI bietet über eine Schnittstelle Zugriff auf DALL·E 2, sowie andere Modelle. Diese Schnittstelle verfügt über eine wohl definierte Dokumentation, die in [95] eingesehen werden kann. Des Weiteren unterstützt OpenAI zur Integration seiner Modelle eine Reihe an Community-Bibliotheken für unterschiedliche Sprachen, darunter eine Anbindung der Schnittstelle für Java (siehe [96]). Auch die Dokumentation zur Anbindung der Schnittstelle in Java unterstützt eine klare Dokumentation (siehe [96]).

Alle drei vorgestellten Systeme ermöglichen das Erzeugen von qualitativ hochwertigen Bildern. Hinter allen drei Systemen stehen in Bezug auf die Bepreisung der Generation von Bildern sogenannte Credits. Dabei bieten alle drei Systeme am Anfang eine gewisse Zahl an freien Credits. Midjourney und Stable Diffusion vergeben beim Erstellen eines Kontos eine endliche Anzahl an Credits. OpenAI hingegen vergibt ein anfängliches Guthaben von 5 Dollar und monatlich eine freie Anzahl an Credits, die jeden Monat zurückgesetzt werden. Credits gelten allerdings nicht für die Schnittstelle von OpenAI und werden separat berechnet [97]. Die Anzahl der Credits sind im Rahmen von OpenAI somit nicht von Belang. Die Berechnung der



Abb. 2.9: Beispiel für ein von DALL · E 2 generiertes Bild anhand oben genannter Textbeschreibung.

Preise geschehen im Fall der Schnittstelle von OpenAI nutzungsbasiert und werden am Ende des Monats abgerechnet, anstatt einer monatlichen Zahlung wie im Fall von Midjourney. Die Tatsache, dass Midjourney keine Schnittstelle anbietet, disqualifiziert Midjourney für die Auswahl eines Systems zur Bildgenerierung im Rahmen dieser Arbeit. Offen für die Auswahl bleiben StabilityAIs Stable Diffusion und OpenAIs DALL · E 2. Ein Argument gegen Stable Diffusion ist, dass sobald Credits aufgebraucht sind, auch zum Anwenden der Schnittstelle neue Credits für einen Mindestwert von 10 Euro hinzugekauft werden müssen, was umgerechnet 1000 standardmäßig generierten Bildern entspricht. Stable Diffusion setzt die Credits, im Gegensatz zu OpenAIs DALL · E 2, voraus, um die REST Schnittstelle nutzen zu können. Ein Argument für OpenAIs DALL · E 2 ist die bestehende Java Anbindung für die Schnittstelle, die eine einfache Integration ermöglicht. Im Rahmen dieser Arbeit wird für die folgende Modellierung und Implementierung Dall-E 2 als Vertreter zur Bildgenerierung ausgewählt.

Ausgewählte Systeme zur Textgenerierung

Zur Auswahl eines Systems zur Textgenerierung stehen OpenAIs Sprachmodelle, wie GPT-3.5 und GPT-4, sowie Googles PaLM 2. Die Anwendung der Systeme zur Textgenerierung werden an folgendem Beispiel demonstriert: „Auf meiner Reise durch die Städte Frankreichs ist mir in einem kleinen Dorf in einem Dialog mit einer Bewohnerin ein Satz aufgefallen, dessen Sinn sich mir nicht erschließen will. Sie sagte: ‚Tomber dans les pommes‘. Wollte sie mir Pommes zum Essen anbieten?“

Pathway Language Model 2 (PaLM 2) [76] ist ein von Google entwickeltes und auf enormen Datenmengen trainiertes großes Sprachmodell [75]. PaLM 2 zeichnet sich durch Fähigkeiten, wie das Verstehen, Erzeugen und Übersetzen von natürlicher Sprache, das Erzeugen von Code, sowie aber auch durch logisches Denken aus [76, 75]. Einige der wichtigsten Merkmale von PaLM 2 ist das verbesserte Verarbeiten von Daten, das Unterstützen vieler Sprachen und verbessertes logisches Denken im Vergleich zum Vorgänger PaLM [75]. Ungeachtet dessen kann PaLM 2 von Zeit zu Zeit unvollständige oder gar fälschliche Erklärungen bereitstellen [98]. Die PaLM 2 Familie besteht aus vier verschiedenen Modellen, die sich jeweils in aufsteigender Größe voneinander unterscheiden: *Gecko*, *Otter*, *Bison*, *Unicorn* [76]. Die unterschiedlichen Größen ermöglichen eine große Bandbreite an Anwendungsmöglichkeiten [76]. Kleinere Modelle, wie Gecko könnten z.B. in mobilen Endgeräten Anwendung finden [76]. PaLM 2 befindet sich noch in der Entwicklung und Google bietet zum Zeitpunkt des Verfassens dieser Arbeit keine offizielle öffentliche Schnittstelle für PaLM 2 an [99]. Eine Möglichkeit dennoch Zugriff auf PaLM 2 zu erhalten, ist über Vertex AI oder über eine Warteliste [100]. Eine weitere Möglichkeit PaLM 2 zu nutzen, ist Bard [77], Googles Pendant zu ChatGPT [101]. Die Bepreisung für das Nutzen der Vertex AI Schnittstelle ist abhängig von der mit den Anfragen verbundenen Rechennutzung [102]. Da sich die PaLM Schnittstelle noch in einer Entwicklungsphase bzw. Vorschauphase befindet, ist der Zugriff für Entwickler, sofern freigeschaltet, kostenfrei [103]. Andernfalls steht die Ankündigung der Bepreisung der PaLM API durch Google noch aus [103]. Folgender Textausschnitt zeigt die Antwort auf das oben genannte Beispiel.



Bard (PaLM 2)

Bard: Nein, die Bewohnerin wollte dir keine Pommes anbieten. Die französische Redewendung „tomber dans les pommes“ bedeutet „sich übergeben“. Es ist eine gängige Redensart in Frankreich, die verwendet wird, um auszudrücken, dass jemand sich sehr schlecht fühlt und sich übergeben muss.

Die Redensart hat ihren Ursprung im 18. Jahrhundert und leitet sich von dem französischen Wort „pomme“ ab, das „Apfel“ bedeutet. Im 18. Jahrhundert glaubte man, dass Äpfel eine heilende Wirkung haben und bei Übelkeit und Erbrechen helfen können. Daher wurde die Redewendung „tomber dans les pommes“ verwendet, um auszudrücken, dass jemand sich so schlecht fühlt, dass er Äpfel essen muss.

Heute wird die Redensart „tomber dans les pommes“ immer noch verwendet, um auszudrücken, dass jemand sich sehr schlecht fühlt und sich übergeben muss. Es ist eine gängige Redensart in Frankreich und wird auch in anderen französischsprachigen Ländern verwendet.

GPT-3.5 [74] und **GPT-4** [74] sind von OpenAI entwickelte und ebenfalls auf enormen Datenmengen trainierte große Sprachmodelle. Ähnlich wie PaLM 2 sind GPT-3.5 und GPT-4 in der Lage Text zu verstehen, Text zu erzeugen, Fragen zu beantworten und Übersetzungen oder Zusammenfassungen anzufertigen [104]. GPT-4 stellt die neuste Version der GPT Modellreihe dar und soll laut OpenAI besseres Textverständnis und eine höhere Leistung als die Vorgängermodelle liefern [105]. Da OpenAI keine genauen Spezifikationen zu GPT-3.5 und GPT-4 veröffentlicht hat, ist die genaue Anzahl der Parameter in GPT-3.5 bzw. GPT-4 nicht bekannt, aber laut Gerüchten soll sich die Anzahl auf 175 Milliarden bzw. 1 Billion Parameter belaufen [106]. Im Vergleich dazu umfasst das Vorgängermodell GPT-3, von welchem GPT-3.5 nur eine optimierte Version ist, nur 175 Milliarden Parameter [104]. Eine der neusten Funktionen von GPT-4 ist die Eingabe und Verarbeitung von Bildern, anhand welcher es Erklärungen generieren können soll [105]. Dies macht GPT-4 zu einem echten multimodalen Modell, welches mehrere Arten von Eingaben akzeptieren kann [105]. Bildeingaben sind aber nach wie vor in einer begrenzten Alpha Testphase und nicht unmittelbar verfügbar [105]. OpenAI bietet über eine REST Schnittstelle einfachen Zugriff auf die Sprachmodelle, wie GPT-3.5 und GPT-4, sowie Derivate, wie z.B. GPT-3.5-Turbo-(4k/16k). Unterschiede in diesen Derivaten liegen in der Größe des Kontextes, also der Länge an Textzeichen, die ein Modell bei der Generierung berücksichtigt. Die für die Sprachmodelle angebotene Schnittstelle ist die gleiche Schnittstelle wie die für DALL·E 2 und bietet ebenfalls eine umfassende Dokumentation (siehe [95]). Diese Dokumentation bietet nützliche Information, Leitfaden zur Nutzung, Beispiele für Quellcode und Beschreibungen für effektives Nutzen der Schnittstelle. Seit dem 6. Juli 2023 erlaubt OpenAI den Zu-

griff auf das Sprachmodell GPT-4-8k für alle Anwender der Schnittstelle, die einen Verlauf erfolgreich abgeschlossener Zahlungen aufweisen [107]. Die Bepreisung hängt dabei, sehr ähnlich zu PaLM 2, von der Anzahl an der in der Anfrage gesendeten Zeichen, sowie der in der Antwort enthaltenen Zeichen ab, und kann in [94] eingesehen werden. Folgender Textausschnitt zeigt die Antwort auf das oben genannte Beispiel.

	ChatGPT (GPT-3.5)
ChatGPT:	Nein, der Satz "Tomber dans les pommes" bedeutet nicht, dass die Bewohnerin dir Pommes zum Essen anbieten wollte. Tatsächlich ist es eine französische Redewendung, die einen Zustand beschreibt, in dem jemand ohnmächtig wird oder in Ohnmacht fällt. Die wörtliche Übersetzung ist "in die Äpfel fallen", aber im übertragenen Sinne bedeutet es, dass jemand kollabiert oder bewusstlos wird. Diese Redewendung wird umgangssprachlich verwendet und ist im Französischen sehr geläufig, um einen plötzlichen Zustand der Bewusstlosigkeit oder des Zusammenbruchs zu beschreiben. Also, die Bewohnerin wollte dir keine Pommes anbieten, sondern sprach darüber, dass jemand ohnmächtig wird.
Benutzer:	Sehr interessant, vielen Dank!
ChatGPT:	Gern geschehen!

Verglichen mit GPT-3.5 bietet GPT-4 grundlegend verbesserte Funktionen und Fähigkeiten [105]. Ein weiterer Vergleich verbleibt somit zwischen OpenAIs Sprachmodellen GPT-3.5 und GPT-4, sowie Googles PaLM 2.

Ein grundlegendes Problem von Sprachmodellen ist, dass sie zu Halluzinationen neigen. Es ist wichtig zu verstehen, dass wenn große Sprachmodelle auf enormen Datenmengen trainiert werden, sie die in diesen Daten enthaltenen statistische Muster und Strukturen erlernen [108]. Als solches sind sie in der Lage Sätze oder Texte zu generieren, die kohärent wirken, ohne dabei wirklich ein wahres Verständnis des Textes oder ein Bewusstsein zu besitzen [108]. Dies kann zu faktisch falschen oder unsinnigen Textzeugnissen, auch Halluzinationen genannt, führen und sind Resultat von mangelndem Kontext und Textverständnis. [109]. Dieses Problem wird verstärkt durch veraltetes Wissen. So gilt für die meisten Sprachmodelle von OpenAI, auch GPT-4, dass ihre Trainingsdaten respektive ihr Wissen nach September 2021 abrupt enden [74]. Dies ist für viele Anwendungsfälle dieser Sprachmodelle eine einschneidende Einschränkung, besonders für den Aspekt der Erklärbarkeit. Ungeachtet dessen sind alle vorgestellten Sprachmodelle nach wie vor von Natur aus Black-Boxes und weisen somit nur bedingt Erklärbarkeit auf. Ein Argument

für GPT-4 oder PaLM 2 ist, dass sie im Vergleich zu ihren Vorgängern jeweils eine wesentliche Verbesserung darstellen. Eine besonders interessante Eigenschaft von PaLM 2 ist der Fokus auf Multilingualität, d.h. PaLM 2 wurde mit Daten in vielen Sprachen trainiert [75]. Jedoch ist ihr Zugang entweder zur Zeit nicht vorhanden oder kann nur über eine Warteliste angefragt werden. Seit dem 6. Juli 2023 erlaubt OpenAI den Zugriff auf das Sprachmodell GPT-4-8k für alle Anwender der Schnittstelle, die einen Verlauf erfolgreich abgeschlossener Zahlungen aufweisen [107]. Ein Argument für GPT-3.5 ist hingegen die Schnelligkeit im Vergleich zu GPT-4 und ein weitaus einfacher Zugriff über die Schnittstelle, die bereits über eine Anbindung für Java verfügt. Im Rahmen dieser Arbeit wird für die folgende Modellierung und Implementierung GPT-3.5 / 4 als Vertreter zur Textgenerierung ausgewählt.

Schlussfolgerung

Anhand der Erkenntnisse der beiden Diskussionen zur Wahl von Systemen zur Bild-, sowie Textgenerierung, fällt die Wahl eines Systems zur Bildgenerierung auf OpenAIs DALL · E 2, sowie die Wahl eines Systems zur Textgenerierung auf das Sprachmodelle GPT-3.5 und seiner Derivate, sowie eventuell auf andere von OpenAI angebotenen Sprachmodelle. Diese Entscheidungen liegen in den folgenden Eigenschaften begründet:

- Einfache Integration durch die offiziell von OpenAI unterstützte Community Bibliothek „OpenAI-Java“ [96] für Java.
- Beide Systeme, DALL · E 2 und die GPT-Sprachmodelle, werden bereitgestellt durch eine einzige Plattform: OpenAI.
- Zahlung einfach und flexibel an einen einzigen Anbieter. Zudem muss nur für das gezahlt werden, was wirklich genutzt wird, *Only pay for what you use*, anstatt einer monatlichen Fixzahlung.

Die Wahl der primären Systeme liegt somit bei den oben genannten Systemen, deren Einsatz im weiteren Verlauf dieser Arbeit verfolgt wird. Unabhängig von der Wahl der primären Systeme verbleiben die Systeme PaLM 2 oder Stable Diffusion weiterhin als optionale Kandidaten, die ebenfalls integriert werden könnten.

Die ausgewählten Systeme bieten umfassende Möglichkeiten zur Bild- und Textgenerierung. Jedoch existiert keine Benutzungsschnittstelle im GMAF, die geeignete Interaktionsmöglichkeiten für Systeme generativer KI ermöglicht. Dies führt zu einer weiteren, zweiten Herausforderung im Rahmen dieses Forschungsziels.

Offene Herausforderung OH 1.2 - Benutzungsschnittstelle

Sollen diese Systeme im Rahmen des GMAF sinnvoll eingesetzt werden, um

Erklärungen generieren zu können, so müssen Anwendern des GMAF durch eine Benutzungsschnittstelle geeignete Interaktionsmöglichkeiten mit diesen Systemen geboten werden.

Die offene Herausforderung **OH 1.2** hat in Bezug auf dieses Forschungsziel einen einschränkenden Charakter. Das Forschungsziel bezieht sich in erster Linie auf Erklärbarkeit von MMIR im Allgemeinen und hat hierfür eine Reihe an existierenden Systemen generativer KI untersucht und vorgestellt. Die offene Herausforderung begrenzt hingegen das Augenmerk auf die Problematik, dass Anwender des GMAF die ausgewählten Systeme generativer KI sinnvoll nutzen sollen, hierfür im GMAF aber keine Benutzungsschnittstelle mit geeigneten Interaktionsmöglichkeiten besteht.

Die offene Herausforderung **OH 1.2** wird im Rahmen einer Modellierung, sowie Implementierung ebenfalls in jeweils FZ 1.2/TB und FZ 1.3/I angesprochen und untersucht.

Im nächsten Abschnitt werden die im Rahmen dieses Forschungsziels identifizierten offenen Herausforderungen zusammengefasst.

Offene Herausforderungen

In Abschnitte 2.1.1 bis 2.1.3 wurden Technologien in Bezug auf das GMAF vorgestellt. Es konnte festgestellt werden, dass der Ansatz MMFGs bzw. Graph Codes erklärbar zu machen, sehr statistisch und rein mathematischer Natur ist. Eine Verbesserung bzw. einen alternativen Ansatz, um MMFGs bzw. Graph Codes erklärbar machen zu können, verspricht der Einsatz von Systemen generativer KI. Dies führt zur ersten offenen Herausforderung **OH 1.1** und der Frage „Wie kann generative KI eingesetzt werden, um Erklärbarkeit zu erreichen?“. Es gilt daher im Rahmen der Umgebung „Erklärbarkeit von MMIR mittels generativer KI“ zu ermitteln, wie Systeme generativer KI genutzt werden können, um Erklärbarkeit zu erreichen bzw. Erklärungen zu generieren.

Im weiteren Verlauf wurden im Abschnitt 2.1.4 grundlegende Technologien, Techniken und Eigenschaften beschrieben und eine Reihe an Systemen generativer KI vorgestellt. Aus dieser Reihe an Systemen wurde in Abschnitt 2.1.5 eine Auswahl an Systemen getroffen, deren Einsatz im weiteren Verlauf dieser Arbeit verfolgt werden. Die in diesem Abschnitt vorgestellten und ausgewählten Systeme bieten umfassende Möglichkeiten zur Bild- und Textgenerierung. Sollen diese Systeme im Rahmen des GMAF sinnvoll eingesetzt werden, um Graph Codes erklärbar zu machen, so müssen Anwendern des GMAF durch eine Benutzungsschnittstelle geeignete Interaktionsmöglichkeiten mit diesen Systemen geboten werden. Es konnte festgestellt werden, dass das GMAF für seine Anwender keine Interaktionsmöglichkeiten mit Systemen

generativer KI bietet. Dies führt zur zweiten offenen Herausforderung **OH 1.2** und der Frage „Wie muss eine Benutzungsschnittstelle beschaffen sein, um geeignete Interaktionsmöglichkeiten mit Systemen generativer KI zu ermöglichen?“. Es gilt daher anhand von Anwendungsfällen geeignete Schnittstellen zur Interaktion mit den in diesem Abschnitt vorgestellten Systemen generativer KI für die Umgebung „Erklärbarkeit von MMIR mittels generativer KI“ im Rahmen des GMAF in einem eigens dafür bestimmten Programm oder als Erweiterung in das GMAF umzusetzen.

Die beiden offenen Herausforderungen **OH 1.1** und **OH 1.2** werden im Rahmen einer Modellierung durch das Forschungsziel FZ 1.2/TB angesprochen und durch das Forschungsziel 1.3/I im Rahmen einer prototypischen Implementierung untersucht. Forschungsziel 1.4/E wird sich der Evaluierung der in diesen Kapiteln erlangten Forschungsergebnissen widmen.

2.2 FZ 2.1/O Integration generativer KI in das GMAF

Dieser Abschnitt gibt einen Überblick über ausgewählte Ergebnisse und Erkenntnisse aus der Forschung im Bereich der Integration von Systemen generativer KI und deckt technische Aspekte ab, die sich auf die zweite Problembeschreibung beziehen. Abschnitt 2.2.1 widmet sich den Integrationsmöglichkeiten von Graph Codes. Weiter befasst sich Abschnitt 2.2.2 mit den Integrationsmöglichkeiten von Systemen generativer KI.

2.2.1 Integrationsmöglichkeiten von Graph Codes

Dieser Abschnitt befasst sich mit den Integrationsmöglichkeiten von Graph Codes. Wie bereits in Abschnitt 2.1.3 beschrieben, sind Graph Codes eine Technologie zur effizienten und effektiven Darstellung und Verarbeitung von Merkmalen im GMAF. Als solches ist eine Integration von Graph Codes im Rahmen des GMAF bereits gegeben. Diese Integration bietet im Rahmen des GMAF folgende Komponenten:

- *GraphCode* bietet eine Implementierung der zweidimensionalen Darstellung von Merkmalen und ermöglicht Zugriff auf das Wörterbuch eines Graph Codes.
- *GraphCodeCollection* bietet eine Reihe an Operationen, wie z.B. die Vereinigung oder Subtraktion von Graph Codes an.
- *GraphCodeGenerator* ist eine Implementierung, die das Erstellen eines Graph Codes aus einem MMFG ermöglicht.
- *GraphCodeIO* bietet eine Reihe an nützlichen Operationen zum Import oder Export von Graph Codes aus oder in Dateien.
- *GraphCodeMetric* bietet eine Implementierung der in Abschnitt 2.1.3.3 beschriebenen Metrik zur Ähnlichkeit von Graph Codes.

Die grundsätzliche Darstellungsform eines Graph Codes erfolgt im textbasierten Dateiübertragungsformat JSON (JavaScript Object Notation) und kann somit leicht von Menschen, aber auch Maschinen gelesen und verarbeitet werden. Listing 2.1 zeigt das Beispiel des Graph Codes GC_{ex} in JSON-Format.

```
1 {  
2   "dictionary": [  
3     "root-image",  
4     "person",  
5     "body",  
6     "arm",  
7     "watch"  
8   ],
```

```

9   "collectionElements": [],
10  "matrix": [
11    [1,1,0,0, 0],
12    [0,1,1,0, 0],
13    [0,0,1,1, 0],
14    [0,0,0,1,17],
15    [0,0,0,0, 2]
16  ]
17 }

```

Listing 2.1: Beispiel des Graph Codes GC_{ex} im JSON-Format.

Dieses Beispiel zeigt, wie die Informationen eines Graph Codes, wie z.B. das Wörterbuch oder die Wertungsmatrix, als ein Objektgraph bestehend aus mehreren Paaren von Schlüssel und Werten im Syntax **Schlüssel : Wert** dargestellt werden. Im Falle der Paare aus Schlüsseln und Werten stellen Schlüssel immer eine Zeichenfolge, wie „dictionary“, „collectionElements“ oder „matrix“ dar, während Werte Zahlen, Strings, Arrays oder wiederum Objekte sein können. Die Elemente „dictionary“ und „collectionElements“ haben jeweils Arrays als Werte, in welchen eine Menge an Bezeichnern bzw. Graph Codes enthalten sein können. Da die gesamte Darstellung eines Graph Codes im JSON-Format wiederum ein Objekt ist, kann „collectionElements“ eine Menge bzw. Sammlung an Darstellungen von Graph Codes enthalten. Auf diese Weise kann eine rekursive Darstellung von Graph Codes erreicht werden. Das Element „matrix“ hat ebenfalls ein Array als Wert. Da allerdings die Adjazenzmatrix eine zweidimensional Matrix ist, sind die Werte im Array wiederum Arrays aus ganzen Zahlen mit einer Länge, die äquivalent zu Größe des Wörterbuchs ist. Das GMAF integriert bereits für die Verarbeitung und Operationen mit JSON die GSON [110] Bibliothek von Google ein.

Auch wenn Systeme wie GPT-3.5 oder GPT-4 und PaLM 2 in der Lage sind Daten im JSON-Format in Textform zu verarbeiten, so bleibt jedoch offen, wie die Informationen aus einem Graph Code und die damit assoziierten Informationen in die Eingabe für ein System generativer KI überführt werden kann, und welche Eigenheiten berücksichtigt werden müssen. Dies führt zu einer ersten offenen Herausforderung im Rahmen dieses Forschungsziels.

Offene Herausforderung OH 2.1 - Transformieren von Graph Codes

Umwandlung von Graph Codes in eine passende Eingabeform für Systeme generativer KI.

Die offene Herausforderung **OH 2.1** hat in Bezug auf dieses Forschungsziel einen einschränkenden Charakter. Das Forschungsziel bezieht sich auf die Integrations-

möglichkeiten von Graph Codes und stellt eine Reihe an Komponenten vor, die mit Graph Codes assoziiert sind und im GMAF implementiert sind. Die offene Herausforderung begrenzt hingegen das Augenmerk auf die Problematik, dass die vorgestellten und bereits im GMAF implementierten Komponenten, keine Möglichkeiten zur Transformation der in Graph Codes gespeicherten Informationen in eine passende Eingabeform für Systeme generativer KI bieten.

Die offene Herausforderung **OH 2.1** wird im Rahmen einer Modellierung, sowie Implementierung in jeweils FZ 2.2/TB und FZ 2.3/I angesprochen und untersucht.

2.2.2 Integrationsmöglichkeiten von Systemen generativer KI

Dieser Abschnitt befasst sich mit den Möglichkeiten der Integration der in Abschnitt 2.1.5 ausgewählten Systeme generativer KI. Dies sind in erster Linie die Systeme, GPT-3.5/4 und Dall-E, die von OpenAI entwickelt wurden. OpenAI bietet zur Übersicht eine Plattform für seine Produkte an. Diese Plattform umfasst neben den Produkten auch Anleitungen, Beispiele und Tipps zu bewährten Praktiken zu diesen Produkten und beschreibt eine Reihe an Möglichkeiten zur Integration seiner Produkte. Diese Integrationsmöglichkeiten umfassen:

- Eine Schnittstelle (API), die es Entwicklern ermöglicht Modelle in ihre eigene entwickelte Software zu integrieren und aufzurufen.
- Unterstützte Systeme zur Softwareentwicklung (SDKs) in vielen Programmiersprachen, die u. a. von OpenAI bereitgestellt werden, oder von Mitgliedern der Community in anderen, nicht offiziell von OpenAI unterstützten Programmiersprachen entwickelt wurden [111].
- Online-Dienste, die es Entwicklern ermöglichen Modelle ohne eigens entwickelte Software zu verwenden. Beispiele für diese Online-Dienste sind ChatGPT, Dall-E, Tokenizer oder ein Textklassifikator für KI.
- Cloud-Dienste, die es Entwicklern ermöglichen Modelle ohne eigene Infrastruktur zu verwenden [112].

Im nächsten Abschnitt wird gesondert auf die Schnittstelle und die Systeme zur Softwareentwicklung eingegangen. Im darauffolgenden Abschnitt 2.2.2 werden dann weitere interessante Werkzeuge im Rahmen der Schnittstelle von OpenAI besprochen und Aspekte hervorgehoben.

API

Eine Schnittstelle (engl. **Application Programming Interface**) ermöglicht Entwicklern die Kommunikation zwischen Softwareanwendungen umzusetzen. Eine API definiert und stellt Funktionen bereit, die Entwickler verwenden können, um Soft-

wareanwendungen miteinander zu verbinden. Funktionen können das Auslesen und Schreiben von Daten, oder Ausführen von Aktionen in einer anderen Anwendung sein. Auf diese Weise ermöglicht eine Schnittstelle den Zugriff auf Funktionen einer Anwendung, ohne dass diese in einer Anwendung selbst neu geschrieben bzw. programmiert werden müssen.

Die API von OpenAI kann praktisch auf jede Aufgabe angewendet werden, die das Verstehen oder Erzeugen von natürlicher Sprache betrifft und erfordert [113]. Des Weiteren kann die API auch verwendet werden, um Bilder zu erzeugen, zu bearbeiten oder Sprache in Text zu konvertieren [113]. Die API von OpenAI bietet Zugriff auf eine Reihe von Modellen, die unterschiedliche Fähigkeiten und Preise aufweisen [113].

Die Schnittstelle von OpenAI wird durch eine RESTful (Representational State Transfer) Schnittstelle bereitgestellt. REST-Schnittstellen sind eine der prominentesten Arten von APIs [114], die eine Reihe an Konventionen verwenden, um die Verarbeitung und Übertragung von Informationen zu gewährleisten. Diese RESTful Schnittstelle wird spezifiziert über eine OpenAPI Spezifikationen [115], die die Funktionen der OpenAI Schnittstelle genaustens dokumentiert. OpenAPI ist eine auf dem Architekturstil REST basierende Spezifikationssprache für die Beschreibung der Struktur, der Reihenfolge und des Verhaltens von APIs. Diese Spezifikation wird für gewöhnlich in YAML oder JSON geschrieben und kann, neben der Dokumentation einer API, auch dazu verwendet werden, um mit spezialisierten Programmen, wie z.B. Swagger Codegen [116], automatisch Quellcode in einer spezifischen Programmiersprache zu generieren. Für die Programmiersprache Java wird diese Spezifikation bereits durch die SDK bzw. Bibliothek OpenAI-Java [96] abgedeckt.

Die API von OpenAI bietet Zugriff auf folgende Endpunkte:

1. /v1/completions
2. /v1/chat/completions
3. /v1/edits
4. /v1/images/generations
5. /v1/images/edits
6. /v1/images/variations
7. /v1/embeddings
8. /v1/audio/transcriptions
9. /v1/audio/translations
10. /v1/files

11. /v1/fine-tunes
12. /v1/moderations

Im Rahmen dieser Arbeit sind nur die Endpunkte 2 und 4 zur Generierung von Text bzw. Bildern von Interesse. In [95] wird eine genaue Dokumentation, mitsamt Beispielen für diese Endpunkte beschrieben.

Eine Anfrage an den Endpunkt **/chat/completions** nimmt als Eingabe eine Liste an Nachrichten in Form einer Unterhaltung zwischen verschiedenen Parteien. Eine Nachricht besteht dabei aus einem Wertepaar **Rolle : Inhalt**. Beide Werte sind notwendige Parameter einer Anfrage an diesen Endpunkt. Der Parameter *Rolle* kann einen von drei Werten annehmen: *System, Assistant, User* [117]. Über den Endpunkt liefert das Modell eine Antwort auf diese Unterhaltung.

Eine Anfrage an den Endpunkt **/images/generations** nimmt als Eingabe eine Eingabeaufforderung bzw. Prompt. Das Modell generiert anhand dieser Prompt ein oder mehrere Bilder und liefert diese über den Endpunkt als Antwort zurück.

Die in diesem Abschnitt vorgestellte Schnittstelle mit all ihren Endpunkten bietet bereits umfassende Möglichkeiten zur Generierung von Text und Bildern. Allerdings gibt es keine bestehende Integration dieser Systeme generativer KI im GMAF. Dies führt zu zweiten, offenen Herausforderung im Rahmen dieses Forschungsziels.

Offene Herausforderung OH 2.2 - Integration generativer KI
Zum Zeitpunkt des Verfassens dieser Arbeit existiert keine Integration der von OpenAI angebotenen Systeme generativer KI im GMAF.

Die offene Herausforderung **OH 2.2** hat in Bezug auf dieses Forschungsziel einen konkretisierenden Charakter. Das Forschungsziel bezieht sich auf die Integration von Systemen generativer KI und stellt hierfür die Schnittstelle und die darin spezifizierten Endpunkte genauer vor. Die offene Herausforderung bezieht sich auf die Integration und Interaktion mit dieser Schnittstelle und ihren Endpunkten und verdeutlicht, dass es im GMAF keine Integration dieser Systeme existiert.

Die offene Herausforderung **OH 2.1** wird im Rahmen einer Modellierung, sowie Implementierung ebenfalls in jeweils FZ 2.2/TB und FZ 2.3/I angesprochen und untersucht.

Tokenizer

Neben der Schnittstelle bietet OpenAI eine Reihe an Werkzeugen bzw. Online-Diensten an. Eines dieser Werkzeuge ist der Tokenizer. Die GPT-Modellreihe benutzt zum Verarbeiten von Text sogenannte Tokens [118]. Tokens stellen häufige im

Text vorkommende Zeichenfolgen dar [118]. Die GPT-Modelle verstehen die statistischen Beziehungen zwischen diesen Token und sind anhand dieser Beziehungen und der bereits bestehenden Folge an Token im Text in der Lage das nächste Token im Text zu erzeugen [118]. Ein abgewandeltes Beispiel für eine Tokenaufteilung ist im folgenden Text einsehbar:

„Multiple models, each with different capabilities and price points. Prices are per 1,000 tokens. You can think of tokens as pieces of words, where 1,000 tokens is about 750 words. This paragraph is 49 tokens.“ Encoding: (*cl100k_base*) [119]

Die Aufteilung der Zeichenfolgen in Tokens ist abhängig vom jeweiligen Modell und kann in [119] eingesehen werden. Die Aufteilung eines Textes in Tokens ist deshalb interessant, da durch dieses Werkzeug die Anzahl der Tokens bestimmt werden kann. In Kombination dieser Information und der Information zur Bepreisung kann ein potentielles Optimierungspotenzial in Bezug auf die Anzahl der Tokens und somit auch der Bepreisung ausgeschöpft werden.

Der Online-Dienst Tokenizer von OpenAI ist offiziell durch die für Python konzipierte Bibliothek „Tiktoken“ [120] oder inoffiziell durch die für Java konzipierte Bibliothek „JTokkit“ [121] als Schnittstelle für die Tokenisierung von Text verfügbar.

2.2.3 Diskussion

In diesem Abschnitt wurden die Möglichkeiten der Integration von Graph Codes, sowie den von OpenAI angebotenen Systemen generativer KI behandelt.

In Bezug auf die Integrationsmöglichkeiten von Graph Codes konnte eine Übersicht über die vom GMAF gebotenen Komponenten, die eine entscheidende Rolle in der Technik der Graph Codes spielen, zusammengestellt werden. Es wurde weiterhin bestimmt, dass die Darstellungsform der in Graph Codes enthaltenen Informationen im JSON-Format erfolgt. Diese Darstellungsform kann insbesondere von aktuellen Systemen generativer KI effizient und effektiv eingelesen werden.

In Bezug auf die Integrationsmöglichkeiten der von OpenAI entwickelten Systemen generativer KI konnte eine Übersicht über die von OpenAI bereitgestellte Schnittstelle gegeben werden. Diese Übersicht gibt auch Einblick über die verfügbaren Endpunkte und Einbindungsmöglichkeiten dieser Endpunkte über eine speziell für Java entwickelte Bibliothek bzw. SDK für eine einfache Integration in das GMAF. Des Weiteren wurde das Tool Tokenizer, welches im Umgang mit den Modellen eine wichtige Rolle spielen kann, genauer erklärt und in eine potentielle Nutzung eingeordnet.

Offene Herausforderungen

In Abschnitt 2.2.1 wurde die Erkenntnis erlangt, dass Graph Codes zur Darstellung im bekannten Dateiübertragungsformat JSON dargestellt werden. Auch wenn die von OpenAI angebotenen Modelle in der Lage sind JSON zu verstehen und analysieren, so bleibt offen, wie eine Umwandlung von Graph Codes in eine geeignete Eingabeform für Systeme generativer KI erfolgen kann. Dies führt zur ersten, offenen Herausforderung **OH 2.1** im Rahmen dieses Forschungsziels. Es gilt daher eine geeignete Überführung der Graph Codes in Systeme generativer KI für die Umgebung „Integration generativer KI in das GMAF“ zu bestimmen.

Im weiteren Verlauf wurde in Abschnitt 2.2.2 eine Übersicht über die von OpenAI bereitgestellte Schnittstelle, mitsamt ihrer verfügbaren Endpunkte, und ihrer Einbindungsmöglichkeiten geschaffen. Es konnte festgestellt werden, dass das GMAF bisher keine Einbindung dieser Endpunkte über die Schnittstelle und folglich auch keinen Systeme generativer KI umsetzt. Dies führt zur zweiten, offenen Herausforderung **OH 2.2** im Rahmen dieses Forschungsziels. Es gilt daher die Funktionen der Schnittstelle zu den Endpunkten der in Abschnitt 2.1.5 ausgewählten Systeme generativer KI für die Umgebung „Integration generativer KI in das GMAF“ zu integrieren.

Die beiden offenen Herausforderungen **OH 2.1** und **OH 2.2** werden im Rahmen einer Modellierung durch das Forschungsziel FZ 2.2/TB angesprochen und durch das Forschungsziel 2.3/I im Rahmen einer prototypischen Implementierung untersucht. Forschungsziel 2.4/E wird sich der Evaluierung der in diesen Kapiteln erlangten Forschungsergebnissen widmen.

2.3 Evaluierungsmethodiken

Die Evaluierung ist ein entscheidender Teil im Zyklus der Softwareentwicklung und dient der Beurteilung der Qualität, Korrektheit und Einhaltung von Standards und Vorschriften. In diesem Abschnitt werden eine Reihe bekannter Evaluierungsmethodiken zur systematischen Evaluierung in der Softwareentwicklung vorgestellt. Mithilfe dieser Evaluierungsmethodiken lassen sich besagte Bewertungen der im Rahmen dieser Arbeit implementierten prototypischen Software vornehmen.

Das IEEE [122] bietet mit der Norm IEEE 1028-2008 [123] Richtlinien zur Durchführung von Softwareüberprüfungen. Diese Softwareüberprüfungen sind, formal betrachtet, Evaluierungen von in der Entwicklungen entstandenen Software-Artifakten, um Defekte festzustellen, die Qualität zu verbessern und sicherzustellen, dass alle Vorschriften eingehalten werden. Die Norm 1028-2008 definiert mehrere Arten von Peer-Review-Verfahren, die im Zyklus der Softwareentwicklung Verwendung finden können. Ein Peer-Review-Verfahren ist ein Prozess, in welchem die Ergebnisse der Softwareentwicklung manuell von einem oder mehreren im selben Feld tätigen Experten bzw. Kollegen überprüft werden. Folgende Peer-Review-Verfahren sind in der Norm IEEE 1028-2008 definiert:

- Informal Review
- Technical Review
- Walkthrough
- Inspection
- Audit

Im Folgenden werden diese einzelnen Verfahren kurz erläutert und ihre Verwendung, der Ablauf, sowie die Vor- und Nachteile näher beschrieben.

Informal Review (z.Dt. informeller Review) ist ein einfaches und flexibles Verfahren, um in erster Instanz nach Defekten zu suchen und Rückmeldungen zu Software-Artifakten zu erhalten, ohne dabei aktiv Programmcode auszuführen. Es werden im Rahmen von informellen Reviews keine Dokumentationen angefertigt. Der Ablauf eines informellen Reviews sieht es vor, dass Prüfer die Dokumentation der Software oder ihren Quellcode individuell untersuchen und anhand ihrer Erfahrungen und Expertise Rückmeldungen liefern. Der Vorteil an informellen Reviews ist, dass sie einfach, schnell und kostengünstig umsetzbar sind. Des Weiteren kann dieses Verfahren bereits in den allerersten Schritten der Entwicklung durchgeführt werden, um frühzeitig Defekten vorzubeugen und Einblicke zu erlangen. Nachteilig an einem informellen Review ist der Mangel an einem strukturierten Vorgehen und mangelnder Gründlichkeit der Untersuchungen.

Technical Review (z.Dt. technischer Review) ist, verglichen mit einem informellen Review, ein detailliertes Verfahren zur Evaluierung von Software-Artifakten, um die Eignung des zu testenden Artifakts für die beabsichtigte Nutzung festzustellen. Schwerpunkte eines technischen Reviews liegen dabei im Einhalten von Standards und Normen, Korrektheit und Vollständigkeit der überprüften Artifakte. Der Ablauf eines technischen Reviews sieht es vor, dass Prüfer die Artifakte gründlich gegen die in den Vorgaben festgelegten Kriterien, wie Spezifikationen und Standards, prüfen und Abweichungen genaustens dokumentieren. Der Vorteil an technischen Reviews ist, dass sie durch eine eingehende Analyse und Bewertung der Software-Artifakte die Einhaltung von Spezifikationen, Normen und Standards garantieren und aufgrund ihrer gründlichen Natur technische Defekte identifizieren können.

Walkthrough (z.Dt. Durchlauf) ist ein Verfahren, in welchem die Vorgehensweise von einem Autor in einer Durchführung durch die Software beschrieben wird. Ziel dieser Durchführung ist es, die Beteiligten mit den Software-Artifakten vertraut zu machen, um Rückmeldungen und Klarstellungen von ihnen zu erhalten. Der oder die Autor(en) der Artifakte demonstrieren diese den Teilnehmern des Durchlaufs, um Rückmeldungen über die vorgestellten Artifakte zu erhalten. Teilnehmer dieses Durchlaufs sind für gewöhnlich eine andere Partei, wie z.B. Arbeitskollegen. Rückmeldungen über die vorgestellten Artifakte enthalten zumeist Fragen, Kommentare oder Verbesserungsvorschläge zu den wahrgenommenen Abweichungen oder Mängeln in der Software im Vergleich zum beschriebenen Ablauf. Der Vorteil an einem Durchlauf ist, dass er die Zusammenarbeit und den Austausch von Wissen zwischen den Beteiligten fördert. Mit Hilfe eines Durchlaufs können Fehler identifiziert, das Verständnis verbessert und unterschiedliche Perspektiven gesammelt werden. Es wird somit die Gültigkeit des beschriebenen Ablaufs bzw. Lösungsvorschlags für die Software geprüft. Eine weit verbreitete Variante des Walkthroughs ist der **Cognitive Walkthrough**. Im Vergleich zu einem simplen Durchlauf ist ein Cognitive Walkthrough speziell auf die Bewertung der Benutzerfreundlichkeit und Benutzererfahrung ausgelegt. Ein simpler Durchlauf kann eher als allgemeines Verfahren zur Prüfung von Software verstanden werden, welches verschiedene Aspekte der Softwareentwicklung umfasst. An einem Cognitive Walkthrough sind für gewöhnlich sogenannte **Usability-Experten** beteiligt. Aufgabe der Usability-Experten ist es, die Perspektive eines potentiellen Nutzers einzunehmen, während in einem simplen Durchlauf in der Regel nur Teammitglieder oder andere Interessensgruppen beteiligt sind, die anhand ihres Fachwissens Rückmeldungen und Erkenntnisse liefern.

Inspection ist ein streng formales Verfahren, welches die Eignung bzw. Beschaffenheit eines Arbeitsergebnisses für die beabsichtigte Verwendung untersucht. Das Verfahren hat zum Ziel Defekte in Software-Artifakten zu identifizieren und die Qualität der Software-Artifakte zu garantieren, sowie zu verbessern [123]. Der Ablauf einer Inspection sieht vor, dass ein Team, welches von einem Moderator geleitet wird, die

Software-Artifakte gründlich und systematisch gegen ihre vorgegebenen Standards und Ansprüche prüft [123]. Defekte und Widersprüche werden festgestellt und klar dokumentiert [123]. Der Vorteil an einer Inspection ist, dass sie einen klaren strukturierten und disziplinierten Ansatz zur Fehlererkennung verfolgen. Weiterhin werden im Rahmen einer Inspection die Untersuchungen von ausgebildeten Inspektoren vorgenommen [123]. Diese Eigenschaften machen eine Inspection zu einem wirksamen Ansatz zur Fehlerentdeckung und Verbesserung der Gesamtqualität von Software-Artifakten.

Audit ist ein Verfahren, das umfassende Bewertungen von Software-Artifakten zum Ziel hat, um Übereinstimmungen bzw. Diskrepanzen mit Anforderungen, Vorschriften oder Standards zu beurteilen. Der Ablauf eines Audits sieht vor, dass Auditoren Software-Artifakte und Dokumentationen prüfen und diese mit den vorgegebenen Anforderungen vergleichen. Abweichungen werden festgestellt und klar dokumentiert. Der Vorteil an einem Audit ist, dass durch umfassende Bewertungen von Software-Artifakten die Einhaltung von Vorschriften garantiert wird und zudem verbessungsfähige Bereiche aufzeigt werden.

Nachteile der genannten Peer-Review-Verfahren sind vor allem zumeist die Zuweisung von Ressourcen, wie z.B. von Zeit, der Definition des genauen Vorgehens, potentielle Voreingenommenheiten bzw. Dynamiken zwischen Prüfern und Teilnehmern, sofern sie aus einer Organisation kommen und nicht unabhängig voneinander arbeiten und begrenzter Fachkenntnisse und Expertise von Prüfern.

Die in diesem Abschnitt vorgestellten Verfahren bzw. Methodiken bieten einen Überblick über eine Bandbreite an Möglichkeiten zur Evaluierung von Software. Eine Auswahl aus diesen Evaluierungsmethodiken wird am Anfang in Kapitel 5 vorgenommen.

2.4 Zusammenfassung

In diesem Kapitel wurde die Recherche zum bisherigen Stand der Wissenschaft und Technik zur Erklärbarkeit und Visualisierung von Graph Codes mittels generativer KI behandelt. In Abschnitt 2.4.1 werden die noch offenen und verbleibenden Herausforderungen zusammengefasst. Weiter werden in Abschnitt 2.4.2 die wesentlichen Ergebnisse und Erkenntnisse dieser Recherche nach den Forschungszielen der Beobachtungsphase, wie nach der Methodik von Nunamaker vorgesehen, aufgeschlüsselt und kurz zusammengefasst. In Abschnitt 2.4.3 werden die Entscheidungen der Diskussionen der behandelten Forschungsziele aufgezählt. Schlussendlich wird in Abschnitt 2.4.4 die Behandlung der noch offenen Herausforderungen angesprochen. Schlussendlich gibt Tab. 2.4 eine Gesamtübersicht über den aktuellen Stand der Arbeit.

2.4.1 Welche offenen Herausforderungen (OH) verbleiben?

In diesem Abschnitt werden die noch offenen und verbleibenden Herausforderungen, die sich im Rahmen der Recherche zum Stand der Wissenschaft und Technik ergeben haben, aufgezählt, kurz beschrieben und jeweils Verortungen der Lösungsansätze genannt.

Tabelle 2.3: Offene Herausforderungen

Nummer der offenen Herausforderungen	Inhalt der offenen Herausforderungen	Ansatz für Lösung
	Es existiert keine Möglichkeit oder Untersuchung Systeme generativer KI zur Erklärbarkeit von Graph Codes zu nutzen. Es bleibt daher offen, ob und wie durch den Einsatz von generativer KI Erklärbarkeit erreicht werden kann.	Modellierung, Implementierung
	Im GMAF ist keine Benutzungsschnittstelle vorhanden, die geeignete Interaktionsmöglichkeiten für Systeme generativer KI zur Bild- oder Textgenerierung ermöglicht. Es bleibt daher offen, wie eine Benutzungsschnittstelle beschaffen sein muss, um geeignete Interaktionsmöglichkeiten mit Systemen generativer KI für Anwender des GMAF zu bieten.	Modellierung, Implementierung

Nummer der offenen Herausforderungen	Inhalt der offenen Herausforderungen	Ansatz für Lösung
	Es existiert kein Algorithmus zum Transformieren der in Graph Codes gespeicherten Informationen in eine passende Eingabeform für Systeme generativer KI. Es bleibt daher offen, wie die Darstellungsform von Graph Codes in eine geeignete Eingabeform für Systeme generativer KI überführt werden kann.	Modellierung, Implementierung
	Es existiert keine Integration der von OpenAI angebotenen Systeme generativer KI im GMAF. Es bleibt daher offen, wie die Funktionen der Schnittstellen und Endpunkte der vorgestellten und ausgewählten Systeme generativer KI in das GMAF integriert werden können.	Modellierung, Implementierung

2.4.2 Gewonnene Erkenntnisse

In diesem Abschnitt werden die Erkenntnisse aus den Forschungszielen „FZ 1.1/O Erklärbarkeit von MMIR mittels generativer KI“ und „FZ 2.1/O Integration generativer KI in das GMAF“ zusammengefasst.

Im ersten Forschungsziel wurde die Erkenntnis erzielt, dass die bisherige Methodik der Erklärbarkeit von MMFGs bzw. Graph Codes sehr statisch, statistisch und rein mathematisch ist. Dies führt zur ersten offenen Herausforderung **OH 1.1**. Ein alternativer Ansatz zur Erklärbarkeit wird im Einsatz generativer KI gesehen. Nach der Auswahl aus vorgestellten Systemen wurde festgestellt, dass es keine Interaktionsmöglichkeiten für Anwender des GMAF mit diesen Systemen gibt. Dies führt zur zweiten offenen Herausforderung **OH 1.2**.

Im Rahmen des zweiten Forschungsziels wurden die Komponenten, die für die Technologie der Graph Codes eine entscheidende Rolle spielen, identifiziert. Weitere Erkenntnisse umfassen die Dateiübertragungsform der Graph Codes. Es wurde festgestellt, dass obwohl aktuelle Modelle diese Dateiübertragungsform einlesen und analysieren können, es offen bleibt, wie Graph Codes in eine geeignete Eingabeform für diese Systeme überführt werden können. Dies führt zur ersten offenen Herausforderung **OH 2.1**. Durch eine genaue Beschreibung der von OpenAI angebotenen Schnittstelle konnte Erkenntnis über die von dieser Schnittstelle bereitgestell-

ten Endpunkte für entsprechende Modelle bzw. Systeme erlangt werden. Nach der Beschreibung der Funktionen dieser Endpunkte konnte festgestellt werden, dass das GMAF keine Funktionen dieser Systeme einbindet. Dies führt zur zweiten offenen Herausforderung **OH 2.2**. Des Weiteren konnte festgestellt werden, dass OpenAI eine Reihe an Tools, darunter auch das Tool „Tokenizer“ anbietet, welches eine wichtige Rolle in der Verarbeitung spielen könnte. Erkenntnis hier ist, dass dieses Tool einen besseren Einblick bzw. mehr Informationen zur Verarbeitung von Tokens bietet.

2.4.3 Getroffene Entscheidungen

In diesem Abschnitt werden die Entscheidungen die in den Forschungszielen „FZ 1.1/O Erklärbarkeit von MMIR mittels generativer KI“ und „FZ 2.1/O Integration generativer KI in das GMAF“ getroffen wurden, rekapituliert.

Im Rahmen des Forschungsziels FZ 1.1/O wurde eine Auswahl aus den vorgestellten Systemen generativer KI getroffen. Die Auswahl fällt auf die von OpenAI angebotenen Modelle bzw. Systeme generativer KI. Diese Entscheidung liegt in der Tatsache begründet, dass diese Modelle beide Aspekte der Text- und Bildgenerierung abdecken und diese durch einen einzigen Anbieter vertreten werden und welcher eine einfache und flexible Zahlung ermöglicht.

Da im Rahmen des Forschungsziels nur die Integrationsmöglichkeiten von Graph Codes, sowie von Systemen generativer KI dargestellt wurden, wurde keine spezielle Entscheidung getroffen. Erwähnenswert bleibt jedoch das in Abschnitt 2.2.2.2 vorgestellte Tool Tokenizer, welches im Rahmen der Verarbeitung von Prompts eine wichtige Rolle spielen kann.

2.4.4 Behandlung offener Herausforderungen

Dieser Abschnitt benennt die Möglichkeiten zur Behandlung der in Tab. 2.3 genannten offenen Herausforderungen. Der Methodik nach Nunamaker folgend, wird in Kapitel 3 die Modellierung und in Kapitel 4 die Implementierung folgen. Hierbei werden die offenen Herausforderungen **OH 1.1** und **OH 1.2** im Rahmen des Forschungsziels „FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI“ adressiert.

Die erste offene Herausforderung **OH 2.1** wird im Rahmen des Forschungsziels „FZ 2.2/TB Integration generativer KI in das GMAF“ adressiert und zuletzt wird die zweite offene Herausforderung **OH 2.2** im Rahmen des Forschungsziels „FZ 2.3/I Integration generativer KI in das GMAF“ adressiert.

Schlussendlich werden in der nachfolgenden Tabelle die Erkenntnisse dieses Kapitels in den aktuellen Stand der Arbeit eingeordnet. Diese Tabelle ist als eine Checkliste

zu verstehen, die sich im weiteren Verlauf dieser Arbeit, schrittweise, Kapitel für Kapitel, füllt und den Fortschritt der Arbeit wiederspiegelt.

Tabelle 2.4: Tabelle zur Übersicht des aktuellen Arbeitsstands.

Forschungsziele		
FZ / OH	Beschreibung	Referenz
Erklärbarkeit von MMIR mittels generativer KI		
FZ 1.1/O	Recherche zur Erklärbarkeit von MMIR mittels generativer KI Grundlegende Technologien: <ul style="list-style-type: none"> • GMAF • MMFG • Graph Code 	Abschnitt 2.1.1 Abschnitt 2.1.2 Abschnitt 2.1.3
OH 1.1	Erste offene Herausforderung Systeme generativer KI und ein Überblick über aktuelle Systeme Diskussion und Auswahl von Systemen	OH 1.1 Abschnitt 2.1.4
OH 1.2	Zweite offene Herausforderung	OH 2.1 Abschnitt 2.1.5
FZ 1.2/TB	Modellierung der Erklärbarkeit von MMIR mittels generativer KI	
FZ 1.3/I	Implementierung der Erklärbarkeit von MMIR mittels generativer KI	
FZ 1.4/E	Evaluierung der Erklärbarkeit von MMIR mittels generativer KI	
Integration generativer KI in das GMAF		
FZ 2.1/O	Recherche zur Integration generativer KI in das GMAF Aufzeigen der Integrationsmöglichkeiten von: <ul style="list-style-type: none"> • Graph Codes 	Abschnitt 2.2.1
	Erste offene Herausforderung <ul style="list-style-type: none"> • Systemen generativer KI 	OH 2.1
	Zweite offene Herausforderung	Abschnitt 2.2.2 OH 2.2
FZ 2.2/TB	Modellierung der Integration generativer KI in das GMAF	

Forschungsziele		
FZ / OH	Beschreibung	Referenz
FZ 2.3/I	Implementierung der Integration generativer KI in das GMAF	
FZ 2.4/E	Evaluierung der Integration generativer KI in das GMAF	

3 Modellierung

Dieses Kapitel widmet sich der konzeptuellen Modellierung und es werden, entsprechend der Methodik nach Nunamaker, alle Forschungsziele der Theoriebildungsphase behandelt. Die Struktur dieses Kapitels folgt der logischen Abfolge der Forschungsziele. Am Anfang wird in Abschnitt 3.1 die Konzeptions- und Modellierungsmeethodik vorgestellt und beschrieben. In Abschnitt 3.2 wird das Forschungsziel „FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI“ behandelt und es werden Möglichkeiten der Erklärbarkeit durch generative KI angesprochen, sowie Anwendungsfälle von Benutzern als Basis für weitere Modellierungsvorgehen identifiziert. In Abschnitt 3.3 wird das Forschungsziel „FZ 2.2/TB Integration generativer KI in das GMAF“ behandelt und es werden die technischen Aspekte der Integration von Systemen generativer KI besprochen. Dies umfasst die Überführung von Graph Codes in eine passende Form der Eingabe für Systeme generativer KI und verwandte Aspekte, sowie die Integration der von der Schnittstelle angebotenen Endpunkte für die ausgewählten Systeme generativer KI. Am Ende jedes Forschungsziels werden die Erkenntnisse aus diesen Bereichen diskutiert und das weitere Vorgehen festgehalten. Abschließend wird in Abschnitt 3.4 eine Zusammenfassung der in diesem Kapitel erarbeiteten Ergebnisse gegeben.

Tabelle 3.1: Struktur dieses Kapitels.

FZ	Kurze Beschreibung	Abschnitt
Konzeptions- und Modellierungsmeethodik	Konzeptuelle Gestaltung der Modellierung	Abschnitt 3.1
FZ 1.2/TB	Modellierung: Erklärbarkeit & gen. KI	Abschnitt 3.2
FZ 2.2/TB	Modellierung: Integration generativer KI	Abschnitt 3.3
Zusammenfassung	Ergebnisse / Erkenntnisse	Abschnitt 3.4

3.1 Konzeptions- und Modellierungsmethodik

Die konzeptuelle Gestaltung und Modellierung in diesem Kapitel folgt dem Paradigma „User Centered System Design“ [26] von Norman & Draper. Ziel dieses Paradigmas ist eine Modellierung aus der Perspektive eines Benutzers oder eines Anwendungsfalls. Die konzeptuelle Gestaltung folgt dem Vorgehen des Rational Unified Process (RUP) [124] und wird mittels der Unified Modeling Language (UML) beschrieben [125].

3.1.1 Rational Unified Process

Der Rational Unified Process (RUP) ist ein für die Nutzung der Modellierungssprache Unified Modeling Language (UML) beschriebenes Vorgehensmodell bzw. Phasenmodell für die Anwendungsentwicklung [125]. Das Vorgehensmodell des RUP wird dabei in UML beschrieben [125]. Der Arbeitsablauf im RUP-Vorgehensmodell leitet sich aus dem evolutionären bzw. inkrementellen Vorgehen ab und lässt sich in vier zeitlich geordnete Phasen und Iterationen jener aufteilen: *Planung*, *Ausarbeitung*, *Errichtung*, *Übergang* [125]. Die Struktur der Phasen und der Iterationen im RUP-Vorgehensmodell werden durch sechs Kernarbeitsprozesse abgedeckt [125]:

- Geschäfts- und Modellierungskonzept
- Anforderungen
- Analyse & Design
- Implementierung
- Test
- Inbetriebnahme

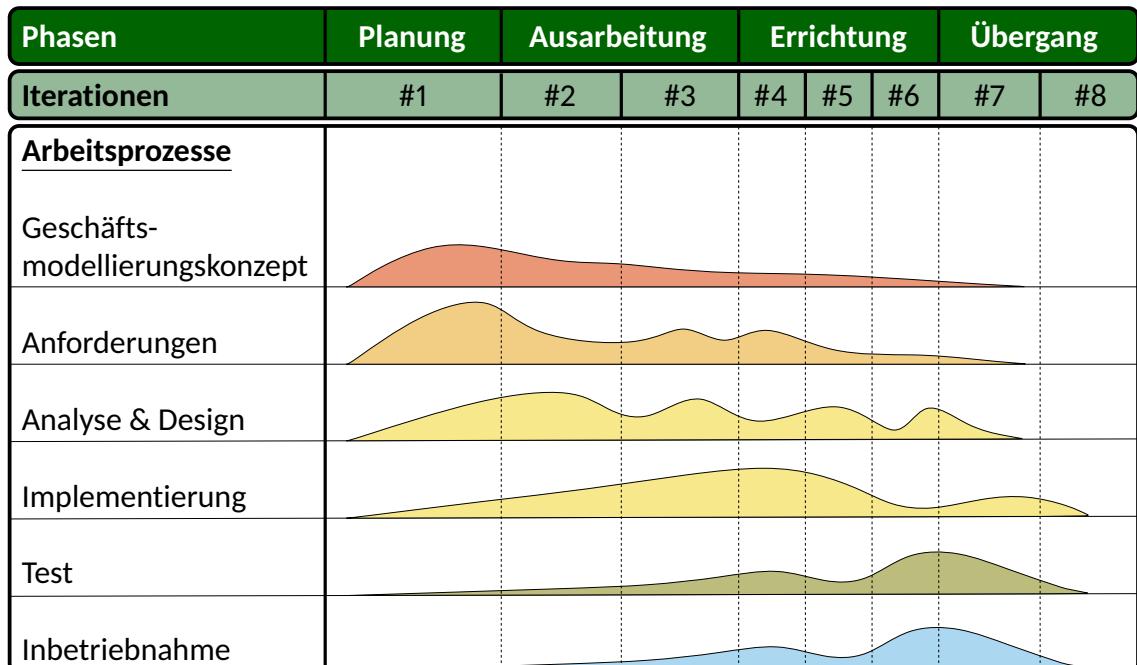


Abb. 3.1: Modellierung zum Vorgehen des Rational Unified Process nach [126].

Die Kern-Arbeitsprozesse „Geschäfts-Modellierungskonzept“, „Anforderungen“ und „Analyse & Gestaltung“ sind Teil dieses Kapitels und setzen die Methodik für die

Bearbeitung der Forschungsziele der Theoriebildungsphase nach Nunamaker um. Der Arbeitsprozess „Implementierung“ des RUP-Vorgehensmodells entspricht den Forschungszielen der Implementierungsphase nach Nunamaker und wird in Kapitel 4 behandelt. Der Arbeitsprozess „Test“ des RUP-Vorgehensmodells entspricht den Forschungszielen der Experimentphase nach Nunamaker und wird in Kapitel 5 behandelt. Der Arbeitsprozess „Inbetriebnahme“ wird in dieser Arbeit nicht behandelt. Somit wird jeder konzeptionelle Entwurf und jede Modellierung aus der Perspektive eines Benutzers oder eines Anwendungsfalls und auf Grundlage der Arbeitsprozesse des RUP-Vorgehensmodells entwickelt und im weiteren Verlauf dieser Arbeit unter Verwendung der UML formalisiert. Dabei ist es Ziel der Modellierung wiederverwendbare Softwarekomponenten zu erstellen, die erweiterbar, konfigurierbar, sowie wartbar sind. Die Auswahl der Konzeptions- und Modellierungsmethodiken erfordert eine spezielle Anordnung der Forschungsziele. Diese Anordnung sieht wie folgt aus:

- FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI
- FZ 2.2/TB Integration generativer KI in das GMAF

Die Notwendigkeit dieser Anordnung ergibt sich nach der Methodik von Norman & Draper, in welcher die Modellierung aus der Perspektive eines Benutzers oder eines Anwendungsfalls beginnt. Des Weiteren bauen die ersten Kern-Arbeitsprozesse „Geschäfts-Modellierungskonzept“ und „Anforderungen“ des RUP auf Anwendungsfällen auf. Somit ist logischer Startpunkt die Erklärbarkeit von MMIR mittels generativer KI.

3.1.2 Unified Modeling Language

Die Unified Modeling Language (UML) [127] ist eine standardisierte, visuelle Modellierungssprache, die ihre Verwendung in der Softwareentwicklung findet [128]. UML kann allerdings auch in anderen technischen Bereichen, oder der Geschäftsmodellierung verwendet werden [129]. Die UML hat zum Ziel, komplexe Systeme und deren Prozesse grafisch darzustellen, zu analysieren, zu spezifizieren und zu dokumentieren [128]. Die UML wurde dazu entwickelt, um die Kommunikation zwischen Softwareentwicklern, Analysten, Designern und anderen involvierten Parteien bzw. Interessensgruppen zu erleichtern [128]. Um diese Kommunikation zu ermöglichen, bietet die UML eine Vielzahl von Diagrammtypen zur einheitlichen, verständlichen und flexiblen Darstellung von Systemstrukturen- und verhalten. Folgende Aufzählung umfasst die bekanntesten und am häufigsten verwendeten Diagrammtypen, sowie ihre jeweiligen Aufgaben.

- UML-Klassendiagramm: Stellt die simple, statische Struktur eines Systems durch seine Klassen, Attribute, Methoden und deren Beziehungen untereinander dar.

- UML-Anwendungsfalldiagramm: Stellt die Interaktion zwischen Akteuren, die Benutzer oder wiederum andere Systeme sein können, und dem System dar. Zweck ist die Darstellung der für die Akteure verfügbaren Funktionalitäten des Systems.
- UML-Sequenzdiagramm: Stellt die Interaktion zwischen Objekten eines Systems in einem zeitlichen Verlauf dar. Interaktion geschieht durch das Senden von Nachrichten zwischen den Objekten.
- UML-Aktivitätsdiagramm: Stellt den Ablauf von Aktivitäten und Prozessen dar, um den Kontrollfluss innerhalb eines Systems zu visualisieren.
- UML-Zustandsdiagramm: Stellt den Lebenszyklus eines Objekts dar und zeigt auf, wie es auf Ereignisse und Zuständsänderungen reagiert.

Die wichtigste Eigenschaft der UML ist seine Unabhängigkeit zu spezifischen Technologien, wie z.B. Programmiersprachen [130]. Diese Eigenschaft ermöglicht die UML zur Modellierung verschiedenster Systeme in vielen unterschiedlichen Bereichen. Bekannte Beispiele sind Softwareanwendungen, Datenbanken und Geschäftsprozesse. Mittels eines wohl definierten, sowie gepflegten Standards, der durch die Object Management Group (OMG) [131] erfolgt und eine einheitliche Sprache für Kommunikation schafft, ermöglicht die UML eine nahtlose Zusammenarbeit zwischen verschiedenen involvierten Gruppen. Angesichts dieser Eigenschaft ist die Unified Modeling Language ein wertvolles Werkzeug für die Softwaremodellierung- und Entwicklung komplexer Systeme.

3.1.3 Informations- und Datenmodell

Die in diesem Abschnitt entwickelten Konzepte zur Interaktion und Verarbeitung von Informationen basieren auf den vom GMAF bereitgestellten Datenstrukturen. Von zentraler Bedeutung für diese Konzepte wird dabei die Datenstruktur *Graph-Code* sein.

3.2 FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI

Dieser Abschnitt befasst sich mit der Modellierung der „Erklärbarkeit von MMIR mittels generativer KI“. Dabei werden die in FZ 1.1/O identifizierten, offenen Herausforderungen **OH 1.1** und **OH 1.2** adressiert. Ziel dieses Abschnitts ist es Konzepte für den Einsatz von Systemen generativer KI für Erklärbarkeit hervorzu bringen und Anwendungsfälle zur Interaktion mit selbigen zu identifizieren. Hierzu wird in Abschnitt 3.2.1 die offene Herausforderung **OH 1.1** und in Abschnitt 3.2.2 die offene Herausforderung **OH 1.2** adressiert.

3.2.1 Erklärbarkeit durch generative KI

In diesem Abschnitt wird die erste offene Herausforderung **OH 1.1** adressiert und es werden mögliche Techniken zum Erzeugen von Erklärungen durch generative KI angesprochen. In Abschnitt 2.1.4.1 wurde das Konzept des Prompt Engineering erklärt und eine Reihe an Techniken, sowie dazugehörige Beispiele präsentiert. Um mithilfe von Systemen generativer KI Erklärungen generieren zu können, müssen die Eingaben in ein System möglichst präzise formuliert sein. Ziel muss es daher sein, mit gezielten und präzise formulierten Eingabeaufforderungen das System so zu instruieren, dass es möglichst genaue Erklärungen generieren kann. In [67] wurde bereits gezeigt, dass das recht simple Textanhängsel „*Lets think step by step.*“ einen großen Einfluss auf die Schlussfolgerungen und somit auch die Ausgabe einer generativen KI haben kann. Es ist daher nicht abwegig anzunehmen, dass eine simple Instruktion, wie z.B. „*Create a coherent textual explanation.*“ oder „*Create a visual explanation.*“ einen positiven Einfluss auf erzeugte Texte bzw. Erklärungen haben könnte. Eine weitere interessante Idee Erklärungen generieren zu lassen, wäre die Methode des Geschichtenerzählens. Eine beispielhafte Instruktion könnte sein: „*Make up a story.*“. Weitere „*elementare Bausteine*“ einer Instruktion können dann beispielhafte Erklärungen umfassen, um dem System zu zeigen, welche Art von Erklärung letztlich gewünscht wird, oder die Vorgabe eines unvollständigen Satzes, welcher dann vom System komplementiert wird.

3.2.2 Anwendungsfälle

Ein sinnvoller Einsatz von Systemen generativer KI im GMAF zur Erklärung von Graph Codes setzt eine Benutzungsschnittstelle mit geeigneten Interaktionsmöglichkeiten voraus. In diesem Abschnitt wird die zweite offene Herausforderung **OH 1.2** adressiert und es werden eine Reihe an Anwendungsfällen aus der Perspektive von Benutzern bzw. Anwendern des GMAF identifiziert und beschrieben. Abb. 3.2 zeigt ein UML-Diagramm zur Übersicht der identifizierten Anwendungsfälle. Diese identifizierten Anwendungsfälle stellen die Basis für weitere Modellierungsvorgehen

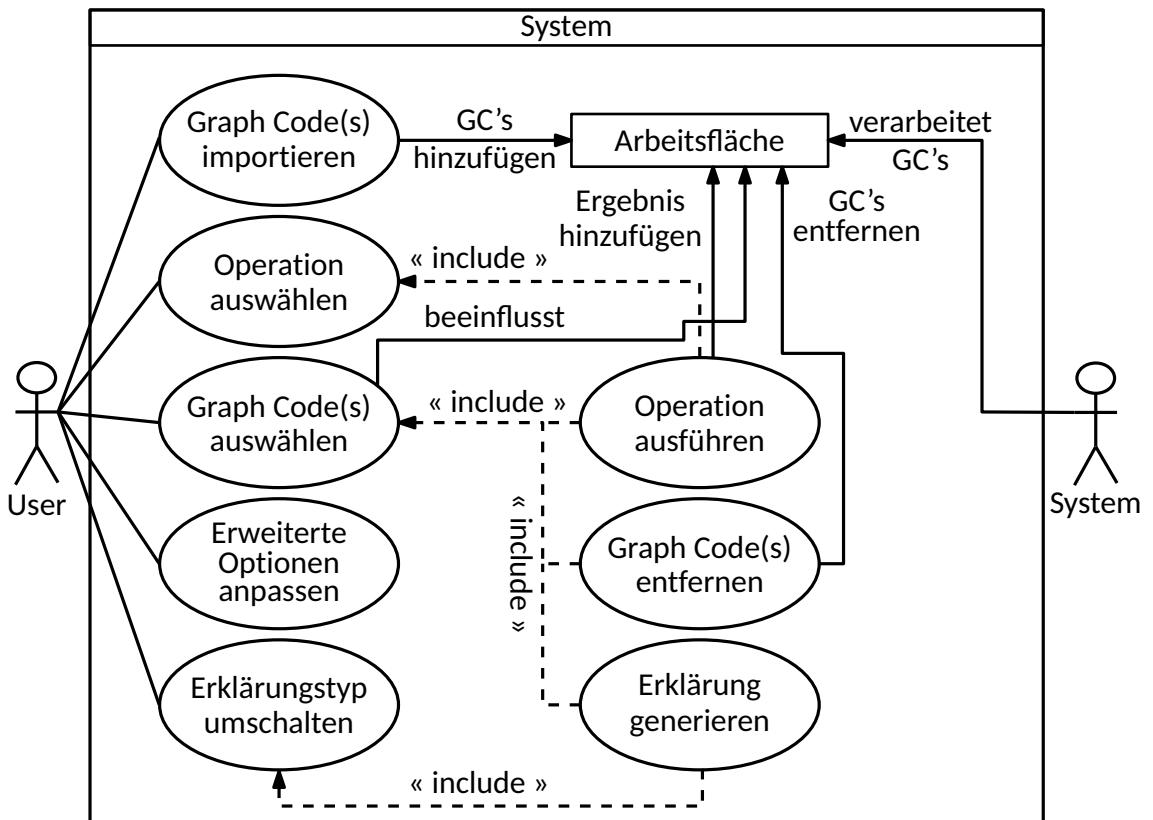


Abb. 3.2: UML Use-Context (System) und UML Use-Cases (Anwendungsfälle).

dar und werden im Laufe dieses Abschnitts weiter ausgeführt. Weitere Ausführungen umfassen textuelle Beschreibungen, erste Wireframes, die das allgemeine Skelett der Anwendung darstellen sollen, Mechanismen, die die an Anwendungsfällen beteiligten Komponenten identifizieren, sowie UML-Sequenzdiagramme, um die Interaktion zwischen den Objekten bzw. Komponenten eines Systems darzustellen. Anhand dieser Modellierungsvorgehen werden die Wireframes weiter ausgeführt bzw. ausgebaut.

Textuelle Beschreibungen

In diesem Abschnitt werden die in Abb. 3.2 dargestellten Anwendungsfälle detailliert beschrieben. Diese detaillierte Beschreibung umfasst jeweils eine Beschreibung der Aufgabe des Anwendungsfalles, die Akteure, die an diesem Anwendungsfall beteiligt sind, die Vorbedingungen, die vor dem Anwendungsfall gelten, einen Ablauf an Schritten zum Durchführen des Anwendungsfalls und schlussendlich die Nachbedingungen, die jeweils nach dem Anwendungsfall gelten. Auf diese Weise wird der Kern-Arbeitsprozess „Anforderungen“ des RUP-Vorgehensmodells abgedeckt.

UC-1.1 Graph Code(s) importieren

Beschreibung: Benutzer klicken in der Arbeitsfläche einen Knopf „Import Graph Code(s)“. Daraufhin öffnet sich ein Filechooser, in welchem Benutzer eine Auswahl von einem oder mehreren Graph Code Datei(en) treffen können. Die ausgewählten Graph Code Datei(en) werden einer Liste in der Arbeitsfläche hinzugefügt und angezeigt.

Akteure: Benutzer, System

Vorbedingung: Keine.

Ablauf:

1. System zeigt eine Liste in der Arbeitsfläche an.
2. Benutzer klickt Knopf „Import Graph Code(s).“
3. System zeigt Filechooser an.
4. Benutzer wählt ein oder mehrere Graph Code Datei(en) aus.
5. System fügt Graph Code Datei(en) der Liste in der Arbeitsfläche zu.

Nachbedingung: Graph Code Datei(en) sind der Liste in der Arbeitsfläche hinzugefügt worden.

UC-1.2 Graph Code(s) entfernen

Beschreibung: Benutzer wählen aus der Liste in der Arbeitsfläche ein oder mehrere Graph Code Datei(en) aus und können über einen Knopf „Remove selected Graph Code(s)“ diese Graph Codes aus der Liste und somit der Arbeitsfläche entfernen.

Akteure: Benutzer, System

Vorbedingung:

- Liste enthält ein oder mehrere Graph Code(s).
- Benutzer haben ein oder mehrere Graph Code(s) ausgewählt.

Ablauf:

1. System zeigt eine Liste an Graph Codes in der Arbeitsfläche an.
2. Benutzer wählt ein oder mehrere Graph Code(s) aus.
3. Benutzer klickt auf Knopf „Remove selected Graph Code(s).“

4. System entfernt ausgewählte Graph Codes aus der Liste in der Arbeitsfläche.

Nachbedingung: Ausgewählte Graph Code Datei(en) sind aus der Liste in der Arbeitsfläche entfernt worden.

UC-1.3 Graph Code(s) auswählen

Beschreibung: Benutzer wählen aus der Liste in der Arbeitsfläche ein oder mehrere Graph Code Datei(en) aus. Die Auswahl von Graph Code Dateien dient als Grundlage für Anwendungsfälle, wie UC-1.2, UC-1.5 oder UC-1.8.

Akteure: Benutzer, System

Vorbedingung: Liste enthält ein oder mehrere Graph Code(s) zum Auswählen.

Ablauf:

1. System zeigt eine Liste an Graph Code(s) in der Arbeitsfläche an.
2. Benutzer wählen ein oder mehrere Graph Code(s) an.

Nachbedingung: Keine.

UC-1.4 Operation auswählen

Beschreibung: Benutzer wählen aus einem Feld eine auszuführende Operation aus. Verfügbare Optionen sind: Vereinigung, Subtraktion, Gemeinsamkeiten, Unterschiede. Das Auswählen einer Operation ist die Vorbedingung für den Anwendungsfall UC-1.5, dem Ausführen einer Operation.

Akteure: Benutzer, System

Vorbedingung: Keine.

Ablauf:

1. System bietet in einem Feld eine Reihe an auszuwählenden Operationen.
2. Benutzer wählt eine Operation aus.

Nachbedingung: Keine.

UC-1.5 Operation ausführen

Beschreibung: Benutzer klicken auf den Knopf „Execute“, um die vorher ausgewählte Operation auf den in der Arbeitsfläche ausgewählten Graph Code Datei(en) auszuführen. Die vorher ausgewählte Operation wird daraufhin auf den ausgewählten Graph Code Datei(en) ausgeführt.

Akteure: Benutzer, System

Vorbedingung:

- Operation in der Arbeitsfläche ausgewählt.
- In der Arbeitsfläche wurden Graph Code Datei(en) ausgewählt.

Ablauf:

1. Benutzer klickt auf Knopf „Execute“.
2. System führt vorher ausgewählte Operation auf den ausgewählten Graph Code Datei(en) aus.

Nachbedingung: Erfolgreich ausgeführte Operation auf ausgewählten Graph Code(s).

UC-1.6 Erklärungstyp umschalten

Beschreibung: Benutzer wählen in der Arbeitsfläche über Knöpfe „Image“ oder „Text“ den Typ der Erklärung aus. Anhand der ausgewählten Erklärung wird die Benutzungsschnittstelle in der Arbeitsfläche für den jeweiligen Erklärungstyp umgeschaltet.

Akteure: Benutzer, System

Vorbedingung: Keine.

Ablauf:

1. Benutzer wählen über die Knöpfe „Image“ oder „Text“ den Typ der Erklärung aus.
2. System schaltet auf für Typ spezifische Benutzungsschnittstelle um.

Nachbedingung: Benutzungsschnittstelle für spezifischen Erklärungstyp umgeschaltet.

UC-1.7 Erweiterte Optionen anpassen

Beschreibung: Benutzer passen in einem dafür vorgesehenen Feld erweiterte Optionen für den Endpunkt, der für die zu generierende Erklärung zuständig ist, an. Die anpassbaren Optionen sind abhängig vom jeweiligen Endpunkt bzw. Erklärungstypen.

Akteure: Benutzer, System

Vorbedingung: Keine.

Ablauf:

1. System bietet, abhängig vom Endpunkt bzw. Erklärungstypen, eine Reihe an anpassbaren Optionen an.
2. Benutzer passen Optionen nach eigenen Bedürfnissen an.

Nachbedingung: Keine.

UC-1.8 Erklärung generieren

Beschreibung: Benutzer klicken auf einen Knopf „Generate ...“, um eine Erklärung für eine zuvor ausgewählte Graph Code Datei zu generieren. Die generierte Erklärung ist abhängig vom zuvor gewählten Endpunkt bzw. Erklärungstypen.

Akteure: Benutzer, System

Vorbedingung: Graph Code Datei wurde ausgewählt.

Ablauf:

1. Benutzer klickt auf Knopf „Generate ...“
2. System lässt durch Endpunkt Erklärung generieren.
3. System zeigt Erklärung in einem dafür vorgesehenen Bereich der Benutzungsschnittstelle an.

Nachbedingung: Keine.

Zweiglauf:

- 2.a Endpunkt für „Image“ generiert eine visuelle Erklärung bzw. ein Bild.
- 2.b Endpunkt für „Text“ generiert eine textuelle Erklärung.

3.2.3 Wireframe für die Interaktion mit Graph Codes

Auf Basis der Anwendungsfälle und den textuellen Beschreibungen dieser, wird in diesem Abschnitt ein Wireframe, sprich ein Konzept für eine Benutzungsschnittstelle, vorgestellt. Abb. 3.3 zeigt eine erste Ausführung eines Wireframes, welches die Grundbereiche und somit das Skelett der Benutzungsschnittstelle darstellen soll.

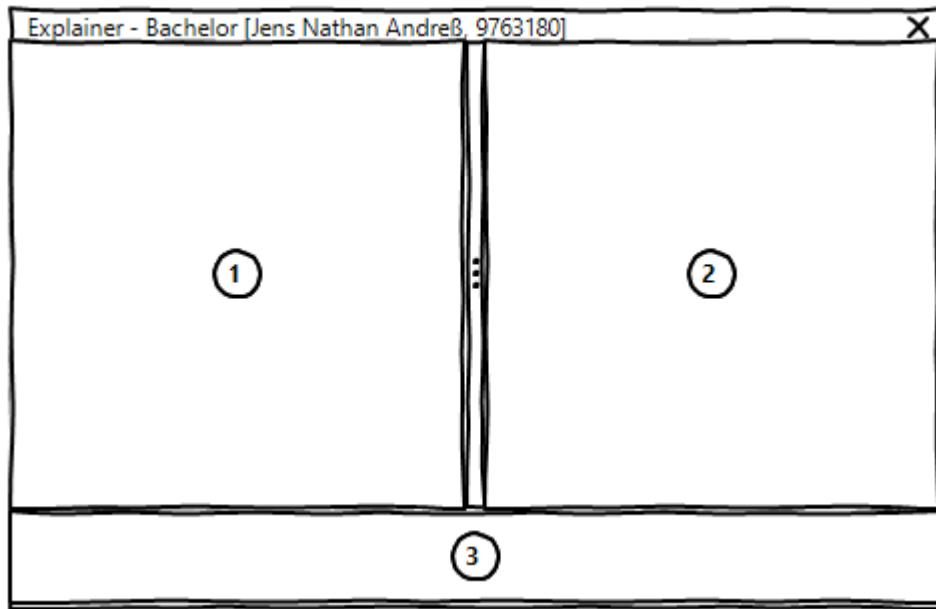


Abb. 3.3: Wireframe für die Grundbereiche der Benutzungsschnittstelle.

In dieser Abbildung sind drei Grundbereiche zu erkennen, die im Folgenden genauer beschrieben werden:

- ① ist der Bereich der Arbeitsfläche, in der die Bearbeitung von Graph Code Dateien stattfindet. Bearbeitungen von Graph Codes umfassen das Importieren, Auswählen und Entfernen von Graph Code Dateien, sowie das Auswählen und Ausführen von Operationen auf Graph Code Dateien.
- ② ist der Bereich der Arbeitsfläche, in der Benutzer Erklärungen generieren können. Benutzer können hier zwischen unterschiedlichen Typen an Erklärungen hin und her schalten.
- ③ ist eine Konsole, in der alle wichtigen Informationen über Aktionen und Prozesse festgehalten werden.

Anhand der Anwendungsfälle UC-1.1 bis UC-1.5 können in der linken Arbeitsfläche, die der Bearbeitung von Graph Codes dienen soll, wichtige Komponenten in der Benutzungsschnittstelle identifiziert werden. Analog können anhand der Anwen-

dungsfälle UC-1.6 bis UC-1.8 in der rechten Arbeitsfläche, die sich der Erklärung von Graph Code Dateien widmet, weitere wichtige Komponenten in der Benutzungsschnittstelle identifiziert werden. Abb. 3.4 zeigt die linke und rechte Arbeitsfläche mit diesen neuen Komponenten in jeweils einem partiellen Wireframe.

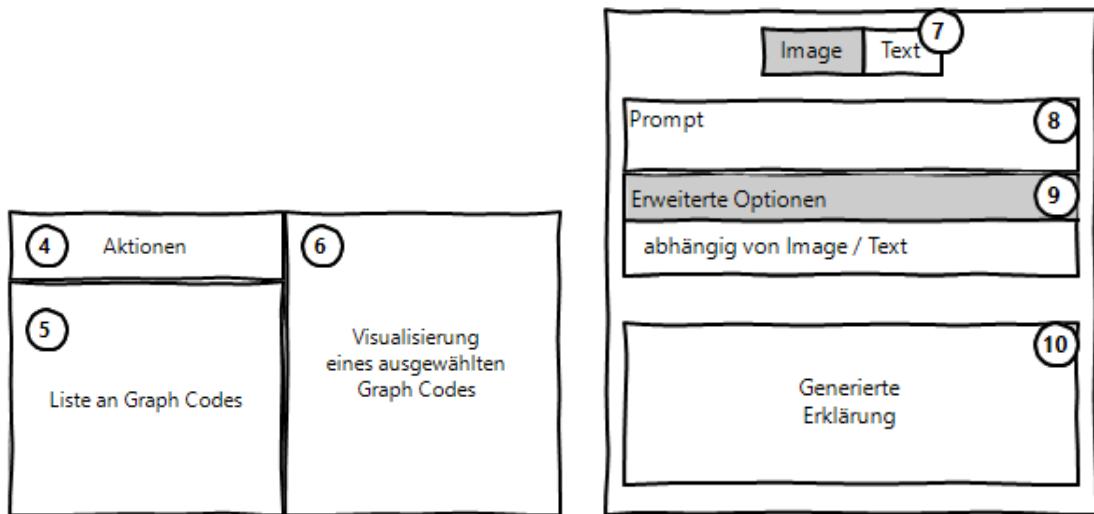


Abb. 3.4: Wireframe für den linken Arbeitsbereich (links) und den rechten Arbeitsbereich (rechts) der Benutzungsschnittstelle.

Diese neuen Komponenten werden im Folgenden genauer beschrieben:

- (4) ist der Aktionsbereich, in welchem Interaktionsmöglichkeiten, wie das Importieren und Entfernen von Graph Codes in die Arbeitsfläche eingebunden werden. Des Weiteren werden in diesem Aktionsbereich auch Interaktionsmöglichkeiten für die Auswahl und das Ausführen von Operationen auf den zuvor in der Arbeitsfläche ausgewählten Graph Code Dateien eingebunden.
- (5) ist der Bereich in der Benutzungsschnittstelle, in der die Graph Code Dateien in einer Liste dargestellt werden. Diese Liste ist die Voraussetzung für Anwendungsfälle UC-1.2, UC-1.3, UC-1.5 und UC-1.8.
- (6) ist der Bereich der Benutzungsschnittstelle, in der ausgewählte Graph Code Datei(en) in einer geeigneten Form visualisiert werden. Da Graph Codes quadratische Matrizen sind, eignet sich für die visuelle Form der Darstellung eine Tabelle.
- (7) sind die Knöpfe, mit welchen Benutzer zwischen den Benutzungsschnittstellen für die jeweiligen Erklärungstypen umschalten können.
- (8) ist die Prompt, die später über den entsprechenden Endpunkt an das System generativer KI übermittelt werden soll.

- (9) zeigt erweiterte anpassbare Optionen für den jeweiligen Endpunkt.
- (10) ist der Bereich, in welchem das Endergebnis der Anfrage an das System generativer KI dargestellt wird.

Zusammengefügt ergeben diese partiellen Wireframes ein ganzes Wireframe, welches in Abb. 3.5 abgebildet ist und die komplette Benutzungsschnittstelle darstellt.

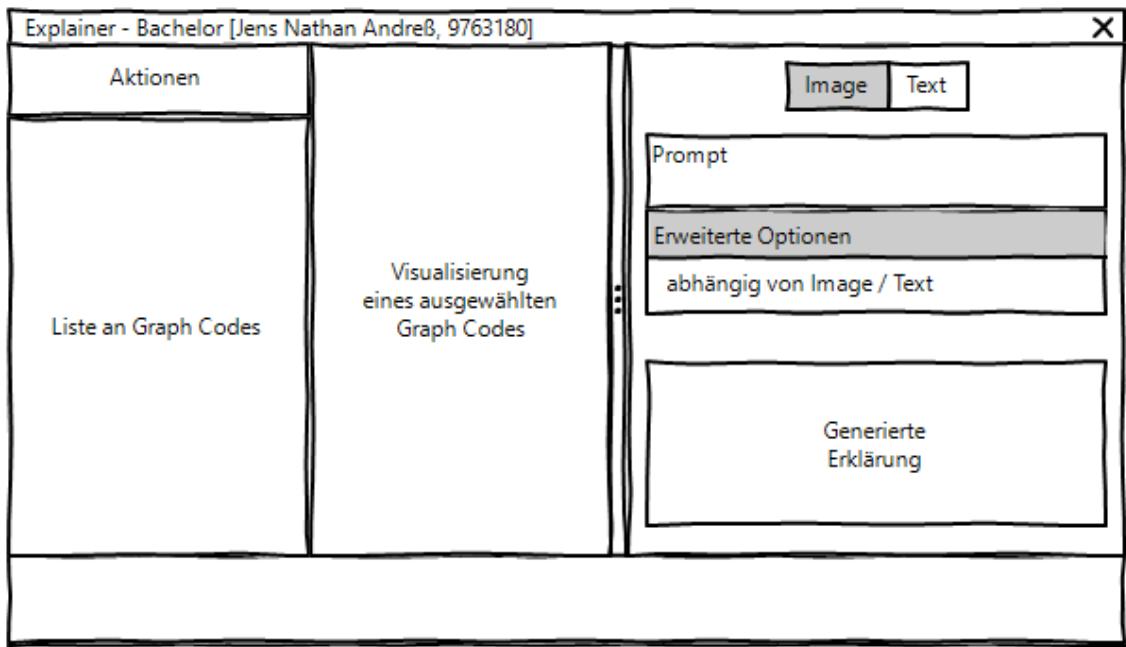


Abb. 3.5: Komplettes Wireframe für die Benutzungsschnittstelle.

Die in diesem Abschnitt dargestellten Wireframes bilden die Struktur der Benutzungsschnittstelle und werden im weiteren Verlauf dieser Arbeit in FZ 1.3/I im Rahmen einer prototypischen Implementierung untersucht. Im weiteren Verlauf dieser Forschungsziels werden nun mittels Mechanismen, die an den Anwendungsfällen beteiligten Komponenten identifiziert, sowie deren Zusammenspiel in einer kurzen Erklärung angerissen. Aufbauend auf diesen Erkenntnissen wird weiter mittels UML-Sequenzdiagrammen das Verhalten zwischen diesen Komponenten untersucht und festgehalten.

3.2.4 Mechanismen

Ein Mechanismus ist ein partielles Klassendiagramm, das die Menge aller Klassen, von denen Instanzen in irgendeinem Objektdiagramm für einen Anwendungsfall vorkommen, darstellt [132]. Mechanismen haben somit zum Ziel, die an einem Anwendungsfall beteiligten Objekte bzw. Klassen zu identifizieren und darzustellen [132].

In einem Mechanismus wird der Anwendungsfall selbst gestrichelt dargestellt und die mit diesem Anwendungsfall assoziierten Klassen werden ebenfalls durch gestrichelte Linien mit dem Anwendungsfall verbunden [132]. Im weiteren Verlauf dieses Kapitels werden die in Mechanismen identifizierten Klassen zusammen mit den in den Anwendungsfällen beschriebenen Abläufen die Basis für UML-Sequenz- und Klassendiagramme bilden. Im weiteren Verlauf dieses Abschnitts werden nun für ausgewählte Anwendungsfälle Mechanismen erstellt.

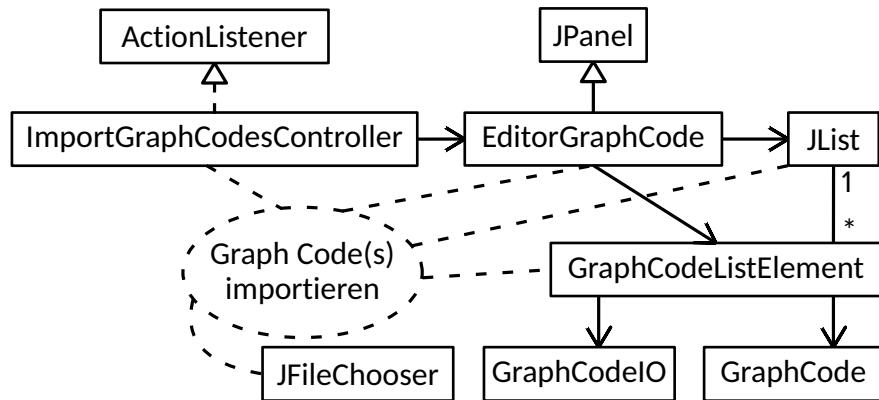


Abb. 3.6: Mechanismus für den Anwendungsfall UC-1.1.

Abb. 3.6 zeigt den Mechanismus für den Anwendungsfall UC-1.1. Sobald der Anwendungsfall UC-1.1 beginnt, benötigt es eine Komponente, die die in diesem Anwendungsfall enthaltenen Subaktionen steuert. Diese Aufgabe übernimmt die Komponente *ImportGraphCodesController*. Da das Importieren von Graph Codes das Auswählen der jeweiligen Graph Code Datei(en) erfordert, benötigt es einen Auswahldialog *JFileChooser*. Die nun durch den Benutzer ausgewählten Dateien werden einer Liste in der Arbeitsfläche hinzugefügt. Genauer ist diese Arbeitsfläche die linke Arbeitsfläche (siehe Abb. 3.3 ①) und wird in diesem Diagramm durch *EditorGraphCode* dargestellt. Diese Arbeitsfläche zeigt eine (1) Liste *JList* (siehe Abb. 3.4 ⑤), welcher beliebig viele (*) Graph Codes *GraphCodeListElement* hinzugefügt werden können.

Abb. 3.7 zeigt den Mechanismus für den Anwendungsfall UC-1.2. Der Mechanismus für den Anwendungsfall UC-1.2 ist sehr ähnlich im Vergleich zum Mechanismus für den Anwendungsfall UC-1.2. Einzig erwähnenswert ist die Komponente *RemoveSelectedGraphCodesController*, die die notwendigen Subaktionen zum Entfernen von Graph Codes steuert.

Abb. 3.8 zeigt den Mechanismus für den Anwendungsfall UC-1.3. Der Anwendungsfall UC-1.3 beginnt mit der Auswahl von Graph Code(s). Das Auswählen von Graph Codes beeinflusst mehrere Komponenten in der gesamten Benutzungsschnittstelle.

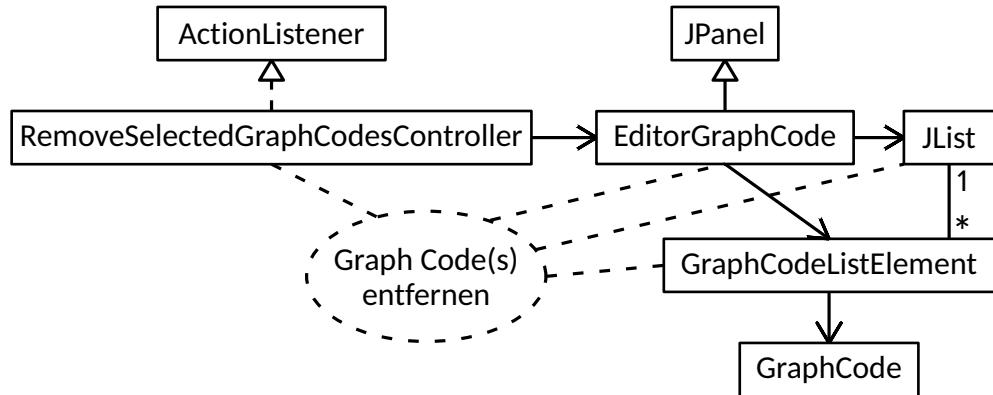


Abb. 3.7: Mechanismus für den Anwendungsfall UC-1.2.

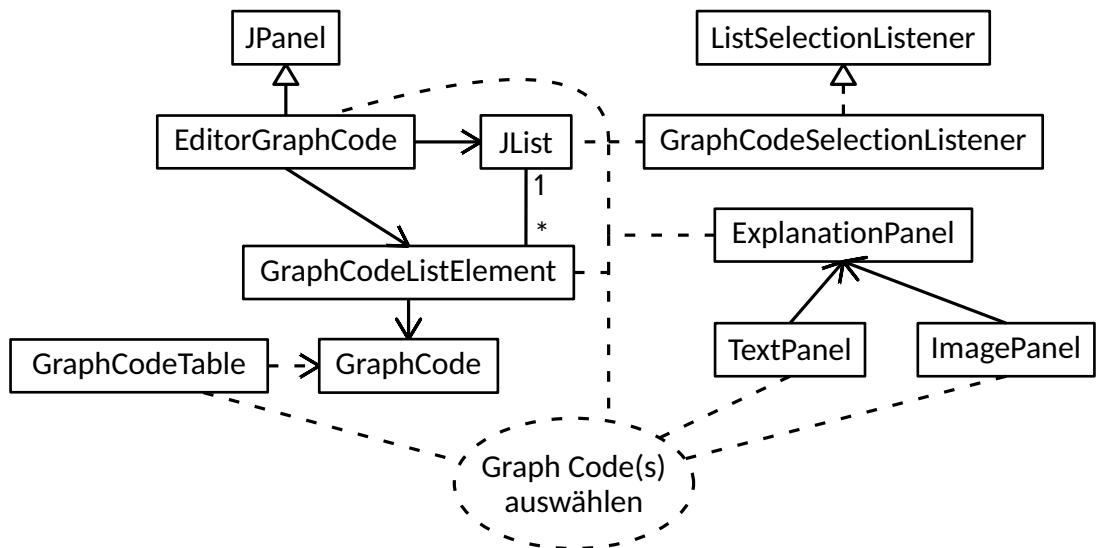


Abb. 3.8: Mechanismus für den Anwendungsfall UC-1.3.

Die Aktionen zur Einflussnahme auf diese Komponenten werden durch die Komponente *GraphCodeSelectionListener* gesteuert. Beeinflusste Komponenten umfassen die Komponente *GraphCodeTable* (siehe Abb. 3.4 (6)), sowie die Komponente *ExplanationPanel* (siehe Abb. 3.4 (2)), in der Benutzer zwischen Erklärungstypen umschalten können. Die Komponenten *ImagePanel* und *TextPanel* sind die für die Erklärungstypen spezifischen Benutzungsschnittstellen und können durch Abb. 3.4 (7) umher geschaltet werden.

Abb. 3.9 zeigt den Mechanismus für den Anwendungsfall UC-1.5. Der Anwendungsfall UC-1.5 „Operation ausführen“ beginnt durch das Klicken des Knopfes „Execute“. Die an diesem Anwendungsfall beteiligten Aktionen werden durch die Komponente *GraphCodeCalculationController* gesteuert und geleitet. Dies beinhaltet auch das

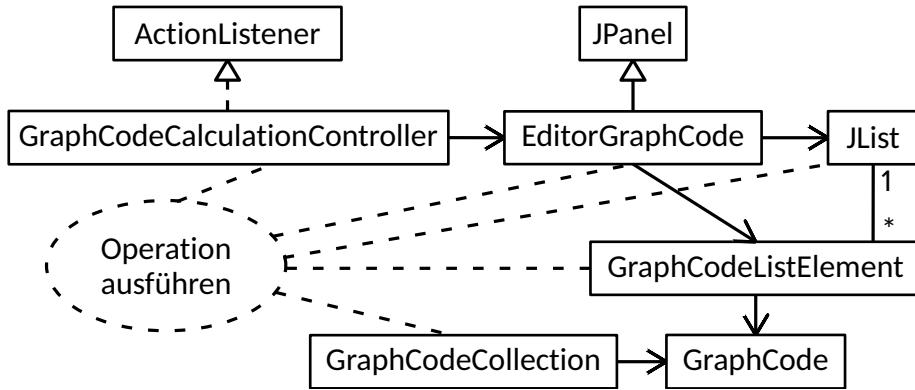


Abb. 3.9: Mechanismus für den Anwendungsfall UC-1.5.

Differenzieren der in Anwendungsfall UC-1.4 zuvor ausgewählten Operation. Diese Operation kann die *Vereinigung (Union)*, *relative Differenz / Unterschied bzw. Subtraktion*, *Gemeinsamkeiten (Similarities)* oder *Unterschiede (Differences)* zwischen mehreren Graph Codes sein. Besonders hierbei ist die *Subtraktion*, die nur auf zwei Graph Codes angewandt werden kann. Die Algorithmen zur Berechnung der Operationen der *Vereinigung* von mehreren Graph Codes, sowie der *Subtraktion* zweier Graph Codes werden bereits durch die Komponente *GraphCodeCollection* bereitgestellt. Im Folgenden werden Konzepte für die weiteren Operationen *Gemeinsamkeiten* und *Unterschiede* in Form von Pseudoalgorithmen vorgestellt, und es werden die an diesen Operationen beteiligten Komponenten, ihre Aufgaben, sowie durchzuführende Schritte beschrieben.

Algorithmus 3.1 zeigt einen Pseudoalgorithmus für die Operation *Gemeinsamkeiten*. Dieser Algorithmus nimmt als Eingabe eine Liste an Graph Codes und gibt als Ergebnis einen Graph Code zurück. Der Algorithmus kann in drei Bereiche unterteilt werden: *Vorbereitung*, *Berechnung* und *Verwertung*. Der Bereich *Vorbereitung* umfasst die Zeilen 2 bis 4. In diesem Bereich wird das Ergebnis als Objekt zum ersten Mal initialisiert. Weiterhin wird die Vereinigung aller Graph Codes aus der Eingabe der Funktion mit der Funktion `getUnion`, welche durch die Komponente *GraphCodeCollection* bereitgestellt wird, berechnet. Ergebnis dieser Vereinigung ist wiederum ein Graph Code, dessen Vokabular zuletzt für die weitere Berechnung in den folgenden Bereichen aufbereitet wird. Der zweite Bereich *Berechnung* umfasst die Zeilen 5 bis 6. In diesem Bereich wird das Vokabular der Vereinigung `unionDic` mit dem Vokabular aller Graph Codes aus der Eingabe miteinander verglichen und es werden alle Merkmale aus `unionDic`, die nicht in irgendeinem Vokabular eines Graph Codes aus der Eingabe vorkommen, entfernt. Übrig bleiben die Merkmale, die in jedem Graph Code aus der Eingabe vorkommen. Der dritte Bereich *Verwertung* umfasst die übrigen Zeilen 7 bis 14. Dem Objekt `sim`, welches das Ergebnis repräsentiert, wird das Vokabular `unionDic` gesetzt. Im weiteren Verlauf wird für jeden Graph

Algorithm 3.1 Berechne Gemeinsamkeiten

Eingabe: Liste an Graph Codes: gcs

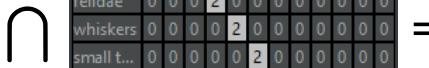
Ausgabe: Graph Code

```

1: function GETSIMILARITIES(gcs)
2:   sim  $\leftarrow$  new GraphCode
3:   union  $\leftarrow$  getUnion(gcs)
4:   unionDic  $\leftarrow$  new Vector(union.getDictionary())
5:   for gc in gcs do
6:     Entferne alle Elemente aus unionDic,
7:     die nicht im Vokabular von gc enthalten sind
8:   dictionary  $\leftarrow$  new Vector(unionDic)
9:   sim.setDictionary(dictionary)
10:  for gci in gcs do
11:    for s in dictionary do
12:      for t in dictionary do
13:        i  $\leftarrow$  gci.getEdgeValueForTerms(s, t)
14:        sim.setValueForTerms(s, t, i)
15:  return sim

```

Code zeilen- und spaltenweise der Wert für ein Paar aus Merkmalen in der Adjazenzmatrix bestimmt und dieser Wert in der Matrix des Ergebnisses eingetragen. Schlussendlich wird das Ergebnis als Ausgabe zurückgegeben. In Abb. 3.10 wird die beispielhafte Berechnung der Gemeinsamkeiten zweier Graph Codes dargestellt.



	GC ₁													
	root-image	cat	eye	window	vertebrate	felidae	carnivore	wood	whiskers	small to m...	fawn	cat_1	grey	snout
root-i...	1	1	1	1	1	1	1	1	1	1	1	1	0	0
cat	0	2	0	0	0	0	0	0	0	0	0	0	0	0
eye	0	0	2	0	0	0	0	0	0	0	0	0	0	0
window	0	0	0	2	0	0	0	0	0	0	0	0	0	0
vertebr...	0	0	0	0	2	0	0	0	0	0	0	0	0	0
felidae	0	0	0	0	0	2	0	0	0	0	0	0	0	0
carnivo...	0	0	0	0	0	0	2	0	0	0	0	0	0	0
wood	0	0	0	0	0	0	0	2	0	0	0	0	0	0
whiskers	0	0	0	0	0	0	0	0	2	0	0	0	0	0
small t...	0	0	0	0	0	0	0	0	0	2	0	0	0	0
fawn	0	0	0	0	0	0	0	0	0	0	2	0	0	0
cat_1	0	1	1	1	0	1	1	1	1	0	1	1	1	1
grey	0	0	0	0	0	0	0	0	0	0	0	2	0	0
snout	0	0	0	0	0	0	0	0	0	0	0	0	2	0

	GC ₂											
	root-image	cat	carnivore	felidae	whiskers	small to m...	snout	fur	paw	domestic_a...	claw	cat_1
root-i...	1	1	1	1	1	1	1	1	1	1	1	1
cat	0	2	0	0	0	0	0	0	0	0	0	0
carnivo...	0	0	2	0	0	0	0	0	0	0	0	0
felidae	0	0	0	2	0	0	0	0	0	0	0	0
whiskers	0	0	0	0	2	0	0	0	0	0	0	0
small t...	0	0	0	0	0	2	0	0	0	0	0	0
snout	0	0	0	0	0	0	2	0	0	0	0	0
fur	0	0	0	0	0	0	0	2	0	0	0	0
paw	0	0	0	0	0	0	0	0	2	0	0	0
domest...	0	0	0	0	0	0	0	0	0	2	0	0
claw	0	0	0	0	0	0	0	0	0	0	2	0
cat_1	0	1	1	1	1	1	1	1	1	1	1	1

	GC ₁ ∩ 2											
	root-image	cat	carnivore	felidae	whiskers	small to m...	snout	fur	paw	domestic_a...	claw	cat_1
root-i...	1	1	1	1	1	1	1	1	1	1	1	1
cat	0	2	0	0	0	0	0	0	0	0	0	0
carnivo...	0	0	2	0	0	0	0	0	0	0	0	0
felidae	0	0	0	2	0	0	0	0	0	0	0	0
whiskers	0	0	0	0	2	0	0	0	0	0	0	0
small t...	0	0	0	0	0	2	0	0	0	0	0	0
snout	0	0	0	0	0	0	2	0	0	0	0	0
fur	0	0	0	0	0	0	0	2	0	0	0	0
paw	0	0	0	0	0	0	0	0	2	0	0	0
domest...	0	0	0	0	0	0	0	0	0	2	0	0
claw	0	0	0	0	0	0	0	0	0	0	2	0
cat_1	0	1	1	1	1	1	1	1	1	1	1	1

Abb. 3.10: Beispielhafte Visualisierung für die Berechnung der Gemeinsamkeiten zweier Graph Codes GC_1 und GC_2 .

Algorithm 3.2 Berechne Unterschiede

```
1: function GETDIFFERENCES(gcs)
2:   diff ← new GraphCode
3:   union ← getUnion(gcs)
4:   unionDic ← new Vector(union.getDictionary())
5:   for gc in gcs do
6:     Entferne alle Elemente aus unionDic,
7:     die nicht im Vokabular von gc enthalten sind
8:     Alle Merkmale bestimmen, die im Vokabular der Vereinigung union,
9:     aber nicht im Vokabular unionDic enthalten sind.
10:    dictionary ← new Vector(unionDic)
11:    diff.setDictionary(dictionary)
12:    for gci in gcs do
13:      for s in dictionary do
14:        for t in dictionary do
15:          i ← gci.getEdgeValueForTerms(s, t)
16:          diff.setValueForTerms(s, t, i)
17:   return diff
```

Algorithmus 3.2 zeigt einen Pseudoalgorithmus für die Operation *Unterschiede*. Dieser Algorithmus ist sehr ähnlich zum Algorithmus 3.1 und nimmt ebenfalls eine Liste an Graph Codes und gibt als Ergebnis einen Graph Code zurück. Zudem kann dieser Algorithmus ebenfalls in die gleichen Bereiche unterteilt werden. Mehr noch sind die Bereiche *Vorbereitung* (Zeile 2 bis 4) und *Verwertung* (Zeile 10 bis 15) identisch und Unterschiede in den Algorithmen bestehen nur im Bereich *Berechnung* (Zeile 6 - 7). Genauer besteht der einzige Unterschied im Vergleich zum Algorithmus 3.1, dass unterschiedliche Merkmale in mehreren Graph Codes das Gegenteil der gemeinsamen Merkmale in mehreren Graph Codes ist. In Abb. 3.11 wird die beispielhafte Berechnung der Unterschiede zweier Graph Codes dargestellt.

Zusammengefasst und einfacher ausgedrückt wird in der Operation *Gemeinsamkeiten* die Schnittmenge aller Vokabulare von Graph Codes, welche wiederum nur Mengen von Merkmalen sind, gebildet. Analog wird in der Operation *Unterschiede* die symmetrische Differenz aller Vokabular von Graph Codes gebildet, welches dem Gegenteil der Schnittmenge aller Vokabulare von Graph Codes entspricht. Dies unterscheidet die Operation *Unterschiede* von der durch die Komponente *GraphCodeCollection* bereitgestellten Operation *Subtraktion*, welche nur die relative Differenz zweier Graph Codes bildet.

In diesem Abschnitt wurden mittels Mechanismen, die an den Anwendungsfällen beteiligten Komponenten identifiziert und erklärt. Mechanismen spielen für dieses Forschungsziel und die Modellierung allgemein eine wichtige Rolle, da anhand dieser

Abb. 3.11: Beispielhafte Visualisierung für die Berechnung der symmetrische Differenz bzw. Unterschiede zweier Graph Codes GC_1 und GC_2 .

identifizierten Komponenten im nächsten Abschnitt die Interaktion zwischen diesen Komponenten in UML-Sequenzdiagramm genauer untersucht werden kann.

3.2.5 UML-Sequenzdiagramme

UML-Sequenzdiagramme modellieren den konkreten zeitlichen Verlauf von mitunter komplexen Operationen in einem Klassenmodell unter Einbeziehung der in diesen Operationen beteiligten Objekten. Die in einer Operation beteiligten Objekte werden dabei am oberen Rand des Diagramms als ein Rechteck mit einer gestrichelten vertikalen Linie dargestellt. Diese gestrichelte vertikale Linie wird als *Lebenslinie* bezeichnet und symbolisiert die Lebenszeit eines Objektes, welche von oben nach unten voranschreitet. Interaktionen zwischen Objekten, wie z.B. dem Aufrufen einer Operation eines Objektes durch ein anderes Objekt, geschehen durch das Senden von Nachrichten zwischen diesen. Die gängigsten Formen von Nachrichten sind:

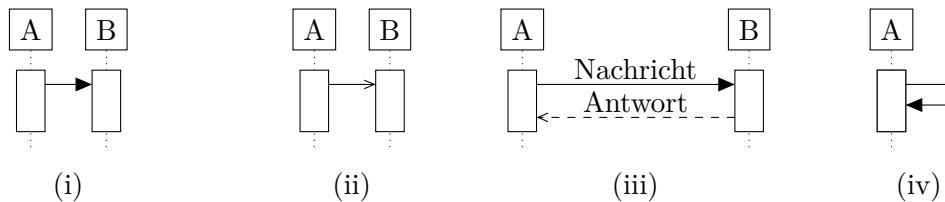


Abb. 3.12: (i) Synchrone Nachricht, (ii) Asynchrone Nachricht, (iii) Antwort auf Nachricht, (iv) Selbstdelegation / Selbstaufruf.

(i) *Synchrone Nachrichten* werden durch einen geschlossenen, gefüllten Pfeil →

dargestellt. Eine synchrone Nachricht repräsentiert einen direkten Aufruf einer Operation vom Dienstnutzenden (Sender) zum Dienstleistenden (Empfänger). Eine wichtige Eigenschaft, die diesem Nachrichtentyp seinen Namen verleiht, ist, dass der Sender auf die Antwort seiner Nachricht wartet, bevor er weitere Nachrichten sendet.

- (ii) *Asynchrone Nachrichten* werden durch einen offenen, spitzen Pfeil → dargestellt. Im Gegensatz zu einer synchronen Nachricht wartet der Sender nach Senden der Nachricht nicht mehr auf eine Antwort des Empfängers, sondern kann unmittelbar weitere Nachrichten senden. Folglich sind bei dieser Art von Nachricht die Sender und Empfänger nicht mehr synchron zueinander.
- (iii) Eine Nachricht vom Typ *Antwort* ist eine Antwort auf eine asynchrone Nachricht und wird durch einen gestrichelten, entgegengesetzten Pfeil <--- dargestellt. Eine Antwort führt den Kontrollfluss zurück auf den Sender der Nachricht nachdem die Operation vom Empfänger abgeschlossen wurde.
- (iv) *Selbstdelegation / Selbstauftrag* ist eine Nachricht eines Objektes auf sich selbst und tritt meist bei rekursiven Methoden auf.

Im Folgenden werden für ausgewählte Anwendungsfälle UML-Sequenzdiagramme vorgestellt und beschrieben.

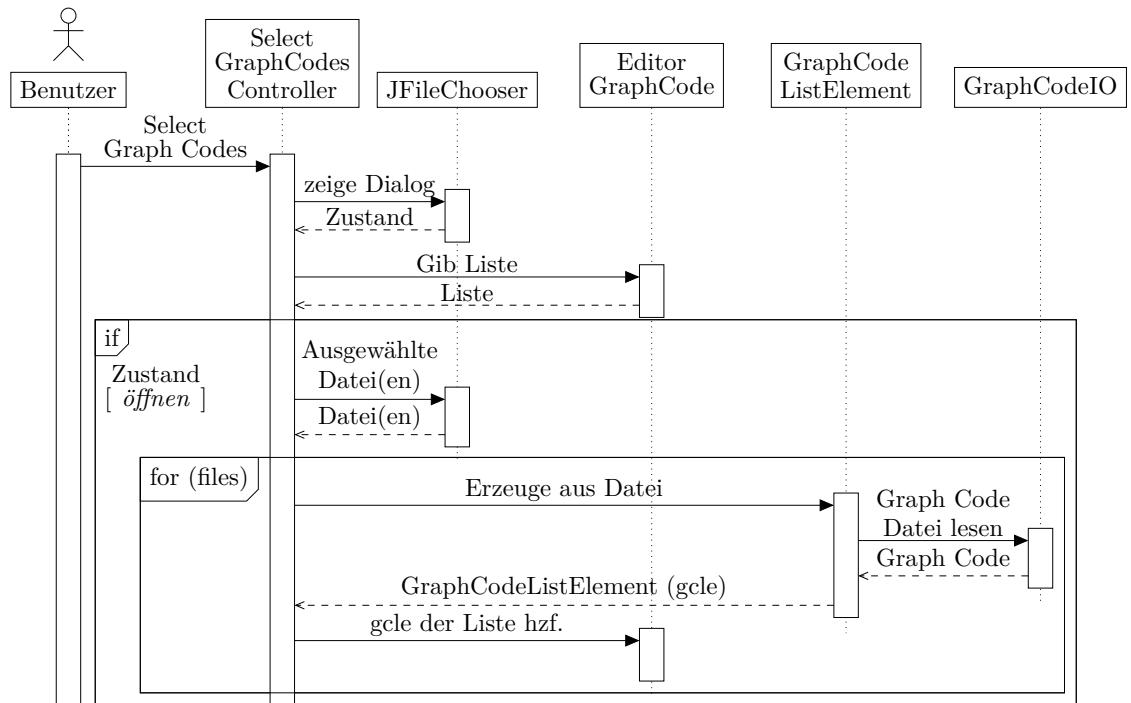


Abb. 3.13: UML-Sequenzdiagramm für den Anwendungsfall UC-1.1.

Abb. 3.13 zeigt ein UML-Sequenzdiagramm für den Anwendungsfall UC-1.1. Es zeigt, wie der Benutzer die Aktion „Graph Code(s) importieren“ durch einen Klick auf den Knopf „Import Graph Code(s)“ initiiert. Weitere Aktionen werden dann von der Komponente *SelectGraphCodesController* durchgeführt. Diese Aktionen umfassen das Anzeigen eines Dateiauswahldialogs und der in diesem Dialog ausgewählten Dateien. Für alle ausgewählten Dateien *for (files)* werden dann Listenelemente erzeugt, die der Liste hinzugefügt werden können. Hierfür müssen die Graph Code Dateien durch die Komponente *GraphCodeIO* eingelesen und in ein Objekt umgewandelt werden.

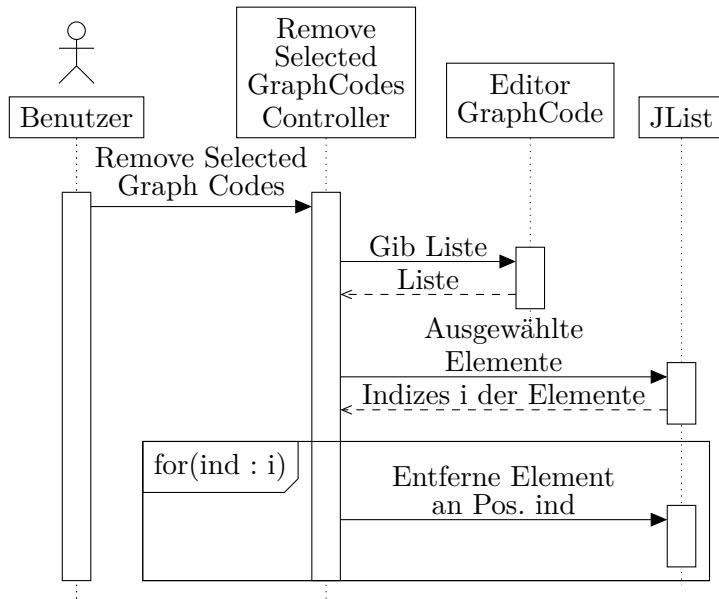


Abb. 3.14: UML-Sequenzdiagramm für den Anwendungsfall UC-1.2.

Abb. 3.14 zeigt ein UML-Sequenzdiagramm für den Anwendungsfall UC-1.2. Es zeigt, wie der Benutzer die Aktion „Graph Code(s) entfernen“ durch einen Klick auf den Knopf „Remove selected Graph Code(s)“ initiiert. Weitere Aktionen werden dann von der Komponente *RemoveSelectedGraphCodesController* durchgeführt. Diese Aktionen umfassen die Abfrage der ausgewählten Elemente in der Liste und die Rückführung der Indizes, die die Positionen dieser Elemente wiedergeben. Anhand dieser Indizes kann der Liste mitgeteilt werden das Element an der Position eines Indexes zu entfernen.

Abb. 3.15 zeigt ein UML-Sequenzdiagramm für den Anwendungsfall UC-1.3. Es zeigt, wie der Benutzer die Aktion „Graph Code(s) auswählen“ durch eine Auswahl auf ein Element in der Liste initiiert. Die Ausführungen der weiteren Aktionen werden dann von der Komponente *GraphCodeSelectionListener* übernommen. Zusammengefasst wird hier das aktuell ausgewählte Element in der Liste abgefragt und von

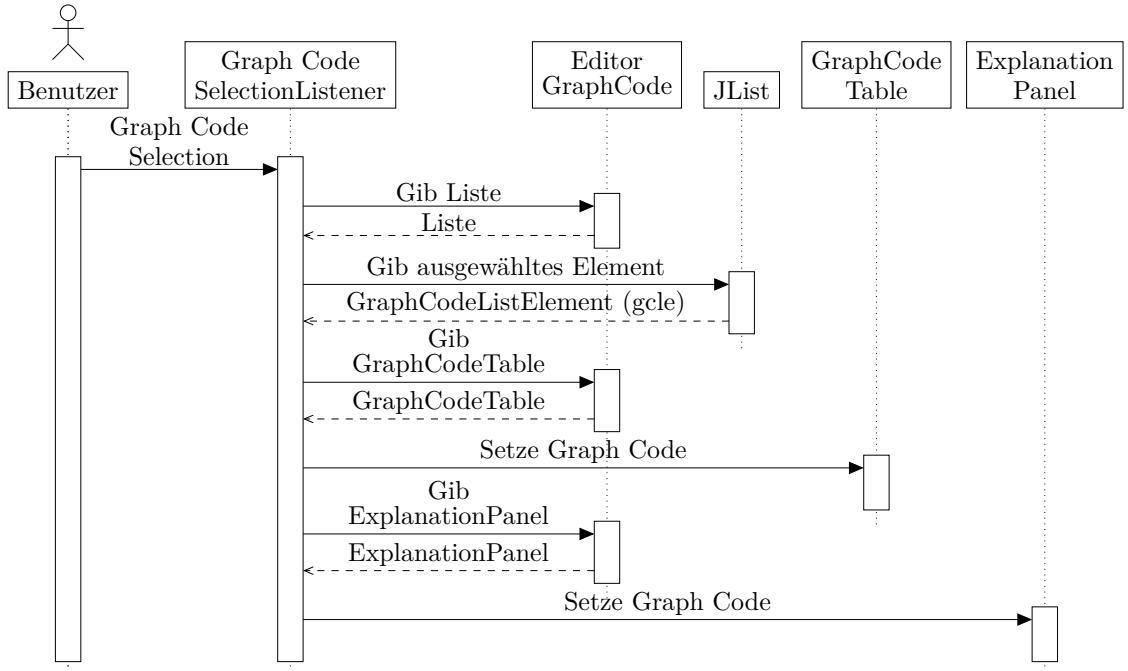


Abb. 3.15: UML-Sequenzdiagramm für den Anwendungsfall UC-1.3.

der Liste zurückgeführt. Die Komponente *GraphCodeListElement* enthält wiederum die Komponente *GraphCode*, welche dann anderen Schnittstellen, wie der Komponente *GraphCodeTable* oder *ExplanationPanel* zur weiteren Verarbeitung mitgeteilt werden kann. Die Komponente *ExplanationPanel* stellt wiederum die Komponenten *ImagePanel* und *TextPanel* zur Verfügung bzw. hält diese Komponenten inne. Die weitere Verarbeitung eines Graph Codes erfolgt dann analog durch diese Komponenten.

Abb. 3.16 zeigt ein UML-Sequenzdiagramm für den Anwendungsfall UC-1.5. Es zeigt, wie der Benutzer die Aktion „Operation ausführen“ durch einen Klick auf den Knopf „Execute“ initiiert. Die weiteren Aktionen werden dann von der Komponente *GraphCodeCalculationController* übernommen. Diese Aktionen umfassen die Abfrage der ausgewählten Elemente aus der Liste und die Rückführung dieser Elemente, damit die in diesen Elementen gespeicherten Graph Codes im weiteren Verlauf verarbeitet werden können. Die Verarbeitung ist dabei abhängig von der in Anwendungsfall UC-1.4 ausgewählten Operation. Die Differenzierung dieser Aktionen wird im UML-Sequenzdiagramm durch die Box *switch* dargestellt. Schlussendlich wird mit dem Ergebnis ein neues Element für die Liste erzeugt und der Liste hinzugefügt.

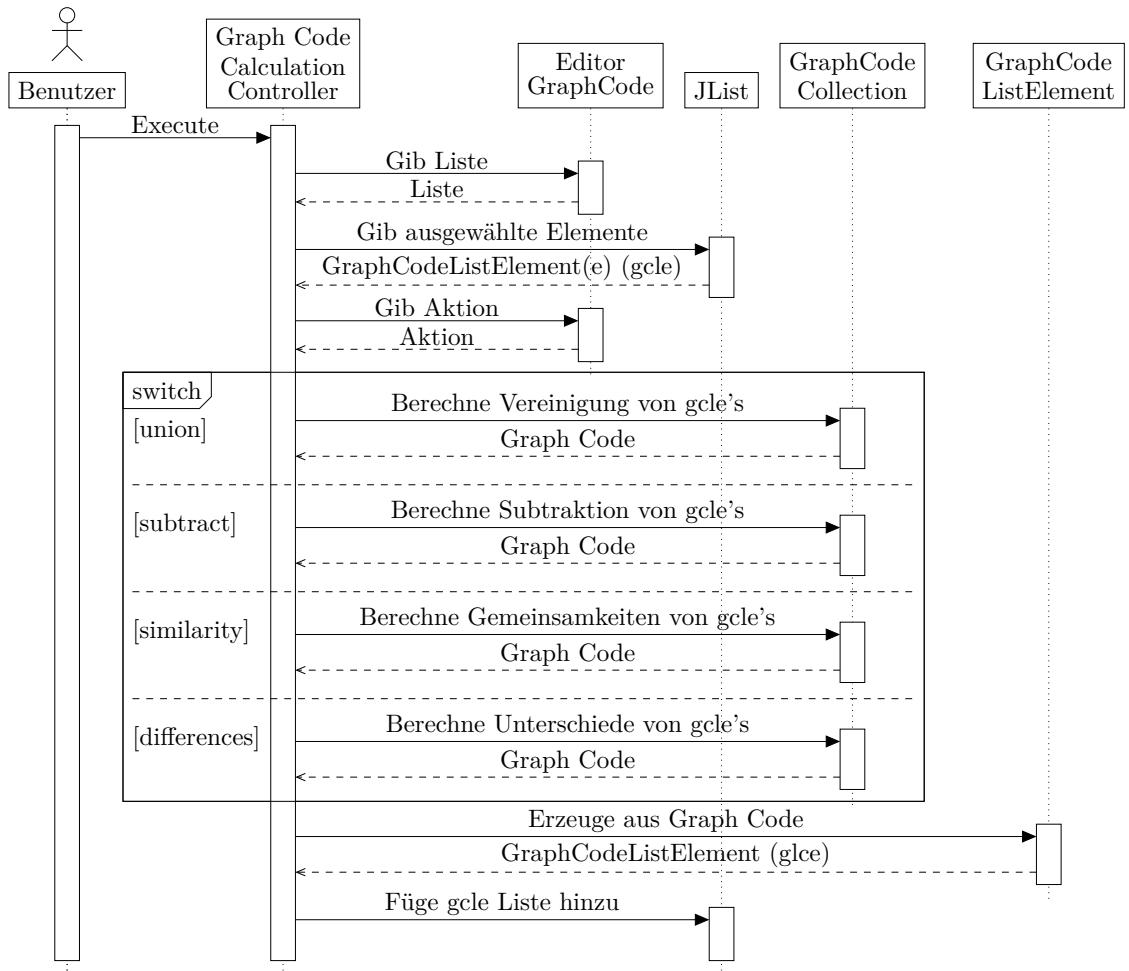


Abb. 3.16: UML-Sequenzdiagramm für den Anwendungsfall UC-1.5.

3.2.6 Diskussion

In diesem Abschnitt wurden die offenen Herausforderungen **OH 1.1** und **OH 1.2** adressiert. In Abschnitt 3.2.1 wurde die offene Herausforderung **OH 1.1** behandelt und Möglichkeiten vorgestellt, wie durch generative KI Erklärbarkeit erreicht werden kann. In Abschnitt 3.2.2 wurde die offene Herausforderung **OH 1.2** behandelt und es wurden Anwendungsfälle aus der Perspektive von Benutzern des GMAF identifiziert und beschrieben. Darüber hinaus wurden Wireframes für eine Benutzungsschnittstelle mit geeigneten Interaktionsmöglichkeiten für den sinnvollen Einsatz von Systemen generativer KI zur Erklärung von Graph Codes konzipiert und weiter ausgeführt. Weiterhin wurden anhand der identifizierten Anwendungsfälle und den dazu entsprechenden textuellen Beschreibungen Mechanismen erstellt, um die an den Anwendungsfällen beteiligten Komponenten zu identifizieren. Anhand

der Anwendungsfälle und der mit diesen Mechanismen identifizierten Komponenten konnten daraufhin UML-Sequenzdiagramme zur genaueren Ablaufbeschreibung erstellt werden.

Die in diesem Abschnitt gewonnenen Erkenntnisse bieten die Grundlage zur Entwicklung einer geeigneten Benutzungsschnittstelle zur Erklärbarkeit von Graph Codes mittels generativer KI.

3.3 FZ 2.2/TB Integration generativer KI in das GMAF

Dieser Abschnitt befasst sich mit der Modellierung der „Integration generativer KI in das GMAF“. Dabei werden die in FZ 2.1/O identifizierten, offenen Herausforderungen **OH 2.1** und **OH 2.2** adressiert. Ziel dieses Abschnitts ist es Konzepte für die Überführung von Graph Codes in eine geeignete Form zur Eingabe in ein System generativer KI zu entwickeln, sowie Konzepte für die Integration generativer KI zu entwickeln. Hierzu wird in Abschnitt 3.3.1 die offene Herausforderung **OH 2.1** und in Abschnitt 3.3.2 die offene Herausforderung **OH 2.2** adressiert.

3.3.1 Transformation von Graph Codes

Dieser Abschnitt widmet sich der Entwicklung von Konzepten zur Überführung der in Graph Codes gespeicherten Informationen, sowie der mit Graph Codes assoziierten Informationen in eine geeignete Form, die als Eingabe in ein System generativer KI verwendet werden kann. Wie bereits in Kapitel 2 recherchiert wurde, erfolgt die Eingabe in ein System generativer KI durch eine Eingabeaufforderung bzw. Prompt, die im Endeffekt eine beliebig lange Zeichenkette sein kann. Die geeignete Form, in die es also Graph Codes zu Überführen gilt, sind somit Prompts. Bevor allerdings mit dem eigentlichen Konzipieren der Transformation von Graph Codes in Prompts begonnen werden kann, ist es sinnvoll zu bestimmen, welche Informationen im Rahmen von Graph Codes überhaupt für eine Erklärung relevant sein können. Von besonderer Relevanz für eine Erklärung sind in erster Linie die Merkmale bzw. das Vokabular, gespeichert im Wörterbuch eines Graph Codes, sowie die Informationen über bestehende Beziehungen zwischen diesen Merkmalen, gespeichert in einer zweidimensionalen, quadratischen Matrix. Daher werden in diesem Abschnitt erst Konzepte für die Grundlagen von Graph Codes entwickelt. Im Anschluss werden dann Konzepte für die mit Graph Codes assoziierten Informationen besprochen. Die in den folgenden Abschnitten genannten Beispiele basieren dabei auf dem in Tab. 3.2 dargestellten Graph Code.

Transformation des Vokabulars

Das Vokabular eines Graph Codes ist eine Menge aus Merkmalen, welche wiederum beliebig lange Zeichenketten sein können. Eine einfache Art das Vokabular eines Graph Codes in eine Prompt zu überführen, ist somit einfach die Merkmale in Textform aufzuzählen. Zusätzlich wird dem System durch eine in der Prompt vorgegebene Instruktion erklärt, welche Bedeutung die Elemente in der Aufzählung besitzen. Ein entsprechendes Beispiel nach Tab. 3.2 hierfür wäre folgende Prompt: „The elements in the following list represent feature vocabulary terms and describe features extracted from multimedia files: root-image, cloud, water, sky, lake, dusk“. Eine solche Transformation des Vokabulars wird durch die Komponente *Gr-*

	root-image	cloud	water	sky	lake	dusk
root-image	1	1	1	1	1	1
cloud	0	2	0	3	0	0
water	0	0	2	0	3	0
sky	0	0	0	2	0	0
lake	0	0	4	0	2	0
dusk	0	0	0	0	0	2

Tabelle 3.2: Graph Code zum Beispiel der folgenden Transformationen.

phCode ermöglicht und kann in der entsprechenden Komponente des Erklärungstyps *ImagePanel* oder *TextPanel* durch eine Funktion *setUpPrompt* eingesetzt werden. Abb. 3.17 zeigt ein einfaches Klassendiagramm mit den Komponenten *ImagePanel*, *TextPanel* und der Funktion *setUpPrompt*.

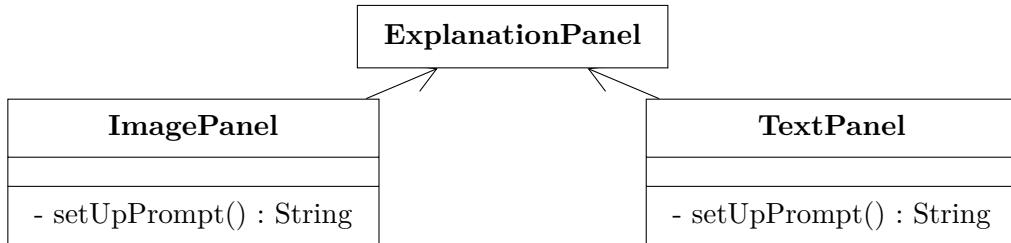


Abb. 3.17: Einfaches Klassendiagramm für die Erklärungspaneele.

Transformation der Matrix

Die Adjazenzmatrix eines Graph Codes ist eine zweidimensionale, quadratische Matrix, dessen Anzahl an Zeilen bzw. Spalten äquivalent zur Größe des Vokabulars ist. Die Werte der Einträge in der Matrix geben Aufschluss über das Vorhandensein von Beziehungen, sowie über die Bedeutung bzw. Art der Beziehung. Bei der Transformation der Matrix müssen daher zwei Aspekte berücksichtigt werden: Das allgemeine Überführen der Informationen aus der Matrix, sowie das Überführen der Werte bzw. der Kodierung der Beziehungen und ihren Bedeutungen.

Eine einfache und naive Transformation kann bereits durch die Funktion *toString* erfolgen, die durch die Komponente *GraphCode* bereitgestellt wird. Diese Funktion überführt die Informationen eines Graph Codes in das JSON-Format. Ein Beispiel für so ein JSON-Format kann in Listing 2.1 eingesehen werden. Die Tatsache, dass die GPT-Modelle in der Lage sind dieses Format zu analysieren und zu verarbeiten,

begünstigt diese Herangehensweise. Ein Beispiel nach Tab. 3.2 hierfür kann folgende Prompt sein: „The following JSON string represents a matrix. The matrix is a square twodimensional matrix where each element represents the relationship between two words: {"matrix":

```
[[1,1,1,1,1,1],[0,2,0,0,0,0],[0,0,2,0,0,0],[0,0,0,2,0,0],[0,0,0,0,2,0],[0,0,0,0,0,2]]}“.
```

Die Umsetzung dieser Transformation findet in der Funktion *toString* in der Komponente *GraphCode* statt und kann in der entsprechenden Komponente des Erklärungstypes *ImagePanel* oder *TextPanel* in der Funktion *setUpPrompt* eingesetzt werden.

Ein Nachteil an dieser Herangehensweise ist das Verhältnis zum Platzverbrauch im Zeichenformat. Aufgrund der quadratischen Natur der Adjazenzmatrix wird überproportional viel Platz verbraucht, was insbesondere anhand des immer weiter steigenden Detaillierungsgrades (LOD) als problematisch betrachtet werden muss. Dies führt auch zu einem zunehmenden Tokenverbrauch, was in Bezug auf einige Modelle, wie z.B. Dall-E 2, die Zeichen- bzw. Tokenbeschränkungen besitzen, ebenfalls ein Problem darstellen könnte. Eine Möglichkeit diesem Problem zu begegnen, wäre durch das Prüfen der Tokenlänge in der entstandenen Prompt. Das Werkzeug *Tokenizer*, vorgestellt in Abschnitt 2.2.2.2, kann hierfür beim Zusammensetzen der Prompt verwendet werden. Da hierdurch das Problem zwar erkannt, allerdings nicht gelöst wird, wird im Folgenden eine verbesserte Herangehensweise zum Transformieren der Informationen eines Graph Codes vorgestellt.

Die verbesserte Herangehensweise zur Überführung der Informationen eines Graph Codes reduziert die zu transformierenden Informationen auf ein Minimum. Anders ausgedrückt, werden in dieser Herangehensweise nur die Informationen betrachtet und transformiert, die von Interesse sind. Dies sind insbesondere die Einträge in einer Matrix, die überhaupt über eine Beziehung verfügen, ungeachtet der Kodierung bzw. des Typs der Beziehung. Da jedes Merkmal n Beziehungen zu anderen Merkmalen aufweisen kann, könnte sich das Format $\langle i_t \rangle - \langle i_{t_1}, \dots, i_{t_n} \rangle$ gut zur Darstellung von Beziehungen zwischen Merkmalen eignen. Der Bezeichner i_t im Format bezeichnet den Index eines Merkmals t im Vokabular des Graph Codes. Effektiv sagt dieses Format aus, dass das Merkmal t mit den anderen Merkmalen t_1, \dots, t_n verbunden ist. Durch das Abbilden des Merkmals auf den entsprechenden Index im Vokabular kann ein Optimierungspotenzial in Bezug auf die Tokenlänge ausgeschöpft werden. Dies ist möglich, da die Merkmale in einem Vokabular in einer fest definierten Reihenfolge geordnet sind. Ein entsprechendes Beispiel nach Tab. 3.2 für so ein Format kann sein: „Some of the dictionary terms are connected through a relationship. These relationships will be noted as $\langle i_t \rangle - \langle i_{t_1}, \dots, i_{t_n} \rangle$, where i_t denotes the index of a feature vocabulary term: $\langle 1 \rangle - \langle 1, 2, 3, 4, 5, 6 \rangle$.“.

Dieses Format trifft allerdings nur Aussagen darüber, ob eine Beziehung zwischen

diesen Merkmalen vorhanden ist, oder nicht. Aussagen über den Typ einer Beziehung erfolgen über die Kodierung bzw. den spezifischen Wert des Eintrages. Durch eine Erweiterung des Formats zu $< i_t > k < i_{t_1}, \dots, i_{t_n} >$ kann die Information der Kodierung in das Format integriert werden. k steht hierbei für den spezifischen Wert bzw. Typ der Beziehung. Ein Beispiel für so ein erweitertes Format kann z.B. „ $< 2 > 3 < 4 > . < 3 > 3 < 5 > .$ “ sein. In diesem Beispiel symbolisiert der Wert 3 die Beziehung „ist Teil von“.

Auf diese Weise kann für jeden Beziehungstyp ein Format erstellt und in die Prompt eingefügt werden. Zusätzlich gilt es im Kontext eines wertespezifischen Formats dem System generativer KI klarzumachen, welche Bedeutung dem Wert k beigemessen wird. Dies kann durch eine Erweiterung der Instruktionen erfolgen. Ein entsprechendes Beispiel nach Tab. 3.2 hierfür ist: „Some of the dictionary terms are connected through a relationship. These relationships will be noted as $< i_t > - < i_{t_1}, \dots, i_{t_n} >$ or $< i_t > k < i_{t_1}, \dots, i_{t_n} >$, where i_t denotes the index of a feature vocabulary term. Additionally – represents an undefined relationship, while k represents a specific relationship. A value of 3 represents the relationship 'is part of' and 4 represents the relationship 'consists of'. The relationships are as follows: $< 1 > - < 1, 2, 3, 4, 5, 6 >, < 2 > 3 < 4 >, < 3 > 3 < 5 >, < 5 > 4 < 3 > .$ “.

Zusammengefasst sieht der gesamte Ablauf vor, dass zuerst die Werte für die Beziehungstypen aus der Adjazenzmatrix gefiltert werden und für die Merkmale, die über diese Beziehung mit anderen Merkmalen verbunden sind, jeweils ein Format erzeugt und in die Prompt integriert wird. Mitsamt Instruktionen kann auf die Weise eine umfassende Prompt erzeugt werden, die die Essenz von Graph Codes erfasst und dem System generativer KI zum Zweck der Generierung einer Erklärung vermittelt.

Die Transformation der in diesem Abschnitt beschriebenen Herangehensweisen erfolgt in der Komponente *GraphCode*. Zu den für die Transformation erforderlichen Schritten, die durch diese Komponente angeboten und durchgeführt werden, gehört das Aufzählen der Wörter aus dem Vokabular, das Filtern der Werte für die Beziehungstypen aus der Adjazenzmatrix und das Überführen dieser in ein entsprechendes, oben aufgeführtes Format. Das Verwerten und Zusammenführen dieser transformierten Informationen geschieht dann in der entsprechenden Komponente des Erklärungstypes *ImagePanel* oder *TextPanel* in der Funktion *setUpPrompt*.

Anwendung der Graph Code Metriken

Metriken von Graph Codes sind mit Graph Codes assoziierte Informationen, die in Abhängigkeit ihrer Aufgabe Aussagen über Graph Codes in Form von Gleitkomma-werten in einem Wertebereich, der meistens zwischen 0 und 1 liegt, ermöglichen. In Abschnitt 2.1.3.3 wurde bereits die Ähnlichkeitsmetrik vorgestellt, die Aussagen über die Ähnlichkeit zwischen zwei Graph Codes ermöglicht. Diese Metrik ist selbst

wieder ein Tripel aus Metriken bezüglich der Merkmale M_F , der Merkmalsbeziehungen M_{FR} und den Beziehungstypen M_{RT} . Im Folgenden wird aufgezählt, welchen Nutzen diese Metriken beim Erreichen des Ziels eine Erklärung zu Graph Codes zu erzielen, beitragen könnten und wie so eine Nutzung bzw. Anwendung dieser Metriken aussehen könnte:

- **Kontrollfluss:** Metriken könnten im Kontrollfluss des Programms verwendet werden, um Fallunterscheidungen zu ermöglichen. In Bezug auf die in Anwendungsfall UC-1.4 genannten Operationen könnten die Metriken in der Komponente *GraphCodeCalculationController* genutzt werden, um vor der Verarbeitung von Graph Codes Prüfungen vorzunehmen. Ein Beispiel für so eine Überprüfung könnte vor der Operation „Subtraktion“ sein, in welcher mit der Merkmalsmetrik M_F geprüft wird, ob die zu verarbeitenden Graph Codes identisch sind. Sind die zu verarbeitenden Graph Codes identisch, müsste die Operation nicht durchgeführt werden, da das Ergebnis eine leere Menge darstellen würde.
- **Prompt / Instruktion:** Die Werte von Metriken könnten, erweitert um Instruktionen, in eine Prompt transformiert bzw. integriert werden, um die Aussagekraft von Graph Codes bzw. in ihnen enthaltenen Merkmalen zu verstärken oder zu gewichten.

Die in diesem Abschnitt konzipierten Herangehensweisen zur Transformation von in Graph Codes gespeicherten Informationen werden im Rahmen einer Implementierung durch das FZ 2.3/I genauer untersucht.

3.3.2 Einbindung von Systemen generativer KI

Dieser Abschnitt widmet sich der Entwicklung von Konzepten zur Integration von Systemen generativer KI in das GMAF. Die Integrationsmöglichkeiten der in Kapitel 2 vorgestellten und ausgewählten Systeme generativer KI wurden bereits in Abschnitt 2.2.2 hervorgehoben. Genauer wurde in Abschnitt 2.2.2 eine Übersicht über die von OpenAI bereit gestellte Schnittstelle und ihre Endpunkte geschaffen. Bevor allerdings mit der Entwicklung von Konzepten zur Integration von Systemen generativer KI begonnen wird, ist es sinnvoll zu bestimmen, welche Komponenten an der Erstellung einer Anfrage an diesen Endpunkten beteiligt sind.

Abb. 3.18 zeigt ein partielles Klassendiagramm, welches die an der Erstellung von Anfragen an die Endpunkte der Schnittstelle beteiligten Komponenten zeigt. Die Komponenten *OpenAiService* und *OpenAiApi* bieten Zugriff auf die Funktionalitäten der Schnittstelle und behandeln zudem die Konfiguration der Schnittstelle. Weitere Komponenten sind *ChatCompletionRequest* und *CreateImageRequest*, die jeweils als Objekt eine Anfrage an einen Endpunkt darstellen. Die Anfragen müssen dann noch über die Komponente *OpenAiService* an die Schnittstelle gesendet

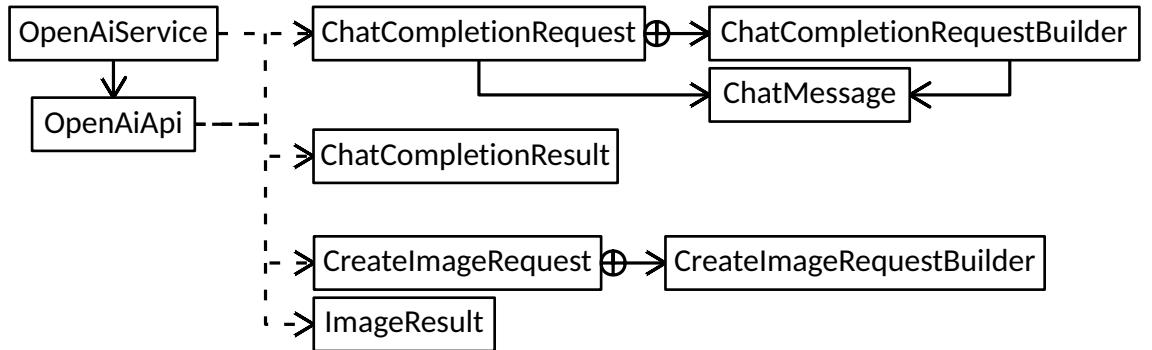


Abb. 3.18: Endpunkte der OpenAI Schnittstelle.

werden. Analog zu den Anfragen sind die Komponenten *ChatCompletionResult* und *ImageResult* Container, die die Ergebnisse der Anfragen an die jeweiligen Endpunkte speichern. Folglich bestehen diese Komponenten wiederum aus kleinen weiteren Komponenten, die die jeweiligen Informationen speichern. Die Komponenten *ChatCompletionRequestBuilder* und *CreateImageRequestBuilder* sind jeweils die Komponenten, die die Parameter der Anfrage spezifizieren. Insbesondere ist in Bezug auf die Anfragen zur Generierung von Text die Komponente *ChatMessage* von Bedeutung. Während bei normalen Anfragen, wie z.B. der Anfrage zur Generierung von Bildern nur einfache Texteingaben benötigt werden, wird bei der Anfrage von Text eine Reihe an Textnachrichten (*ChatMessage*) benötigt, die einen Dialog zwischen mehreren Parteien darstellen. Eine Textnachricht erwartet als Eingabe eine Rolle im Dialog, sowie den Inhalt dieser Partei im Dialog.

Die genannten Komponenten stellen die Grundlage zum Erstellen einer Anfrage an die Endpunkte der Schnittstelle von OpenAI dar. Im Folgenden ist es nun von Interesse herauszufinden, welche Informationen für das Erstellen einer effektiven und sinnvollen Anfrage an einen Endpunkt der Schnittstelle notwendig sind, und welche Komponenten am Erzeugen dieser Informationen beteiligt sind. Speziell für die Textgenerierung bestehen die Informationen für den Dialog, der durch eine Reihe an Textnachrichten *ChatMessage* dargestellt wird, zumeist aus Instruktionen (siehe z.B. Abschnitt 3.2.1), sowie den in Textformat transformierten Informationen aus einem Graph Code (siehe Abschnitt 3.3.1).

Der Prozess der Verarbeitung der Informationen eines Graph Codes wird, wie bereits durch das UML-Sequenzdiagramm für den Anwendungsfall UC-1.3 dargestellt, von einem Benutzer durch die Auswahl eines Graph Codes initiiert. Schlussendlich wird dann der Komponente *ExplanationPanel* ein Graph Code übergeben, welche diesen Graph Code dann an die spezifischen Komponenten *ImagePanel* und *TextPanel* der Erklärungstypen *Image* oder *Text* delegiert. Die Verarbeitung der Informationen erfolgt dann gesondert und abhängig vom Erklärungstyp durch die Komponen-

ten *TextPanel* oder *ImagePanel* in der Methode *setUpPrompt*. Ein Beispiel für die Zusammenstellung der durch die Transformation generierten Informationen in der Methode *setUpPrompt* kann in folgendem Beispiel eingesehen werden:

System: „You are an assistant, who is able to generate cohesive textual explanations based on a collection of words.“

Assistant: „The collection of words represents a dictionary. The dictionary contains so-called feature vocabulary terms. Some of the dictionary terms are connected through a relationship. These relationships will be noted as $< i_t > - < i_{t_1}, \dots, i_{t_n} >$ or $< i_t > k < i_{t_1}, \dots, i_{t_n} >$, where i_t denotes the index of a feature vocabulary term. Additionally – represents an undefined relationship, while k represents a specific relationship.“

User: „The collections of words is as follows: *<Liste der Merkmale aus dem Wörterbuch eines Graph Codes>*. The relationships are as follows: *<Generierte Formate der Beziehungen durch einen Graph Code>*. Generate a coherent textual explanation containing the terms of the dictionary. An example for such an explanation could be: *<Beispiel zur Verdeutlichung...>*.“

Assistant: „Based on the dictionary, here is a cohesive textual explanation containing the terms of the dictionary:“

Die Verarbeitung der durch die Transformation generierten Informationen und in einer Prompt zusammengetragenen Informationen wird im Folgenden durch einen allgemeinen Algorithmus 3.3 beschrieben, der die Sequenz der notwendigen Schritte und Aktionen hervorhebt, um eine Anfrage an den Endpunkt Text (*ChatCompletionRequest*) zur Generierung von Text zu erstellen.

Algorithm 3.3 Anfrage an den Endpunkt Text (*ChatCompletionRequest*).

- 1: **procedure** CREATEREQUEST
 - 2: Service initialisieren... (*OpenAiService*)
 - 3: Anfrage initialisieren... (*ChatCompletionRequest*)
 - 4: Anfrage anpassen, Parameter setzen... (*ChatCompletionRequestBuilder*)
 - 5: Anfrage an Schnittstelle senden (*OpenAiService*)
 - 6: Ergebnis erhalten und entsprechend verwerten (*ChatCompletionResult*, *TextPanel*)
-

Bezüglich der Erstellung einer Anfrage an den Endpunkt Bild (*ImageRequest*) zum Generieren eines Bildes besteht allerdings das Problem der Tokenbegrenzung von Dall-E 2. Im Folgenden wird ein Konzept entwickelt, um diesem Problem zu begegnen.

nen. Dieses Konzept sieht das In-Reihe-Schalten der Endpunkte vor. Genauer wird zuerst eine Anfrage an den Endpunkt Text zum Generieren von Text erstellt und gesendet, um darauffolgend mit dem daraus resultierenden Ergebnis die Eingabe für eine Anfrage an den Endpunkt Bild zu speisen. Dies ist möglich, da die Anfrage an den Endpunkt Text spezifisch dazu parametrisiert werden kann, dass das Ergebnis dieser Anfrage nur eine maximale Länge an Zeichen umfassen darf/soll. Des Weiteren können die Instruktionen der Prompt zur Generierung von Text so angepasst werden, dass dem System generativer KI vermittelt wird, dass das erzeugte Ergebnis dieser Anfrage wiederum als Eingabe für ein Bilderzeugungsprogramm verwendet werden soll (siehe Abschnitt 3.2.1, „Create a visual explanation / description.“). Das Erzeugen einer Anfrage an den Endpunkt Bild verhält sich analog zum Endpunkt Text.

Mit diesen neuen Informationen kann eine Erweiterung bzw. Ergänzung des UML-Sequenzdiagramms für den Anwendungsfall UC-1.3 vorgenommen werden. Abb. 3.19 zeigt die Erweiterung des UML-Sequenzdiagramms um die neuen Abläufe der Verarbeitung von Graph Codes in den Komponenten *ImagePanel* und *TextPanel*.

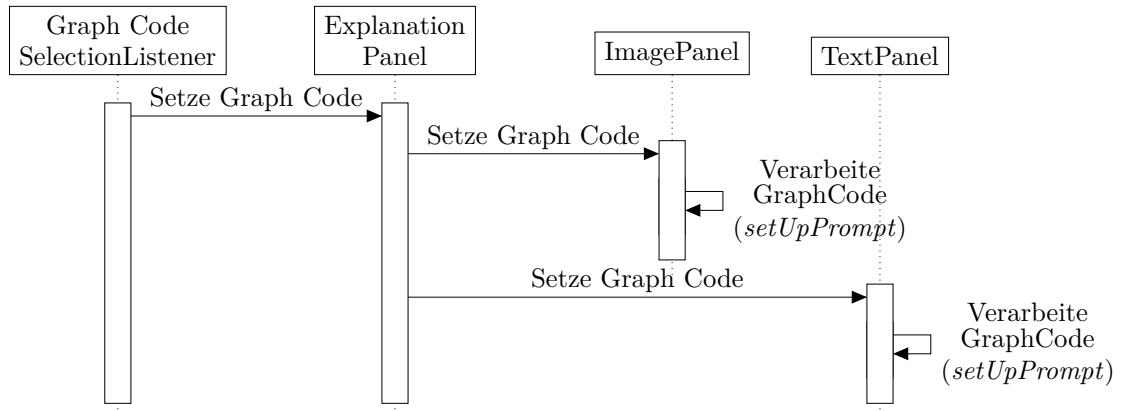


Abb. 3.19: Erweiterung zum UML-Sequenzdiagramm für den Anwendungsfall UC-1.3.

Weitere Parameter zum Anpassen einer Anfrage an einen Endpunkt der Schnittstelle können durch die Benutzungsschnittstellen, dargestellt in Abb. 3.4, generiert werden. Genauer würden die Spezifikationen der Anfragen an die Endpunkte der Schnittstelle durch die gewählten Einstellungen im Rahmen des Anwendungsfall UC-1.7 in den erweiterten Optionen (siehe Abb. 3.4 (9)) zustande kommen.

Anhand dieser Informationen kann nun auch ein Mechanismus für den Anwendungsfall UC-1.8 erstellt werden (siehe Abb. 3.20). Der Anwendungsfall UC-1.8 „Erklärung generieren“ beginnt durch das Klicken des Knopfes „Generate ...“. Im weiteren Verlauf erzeugt das System dann die notwendigen Komponenten für eine Anfrage an

einen Endpunkt der Schnittstelle. Diese Objekte werden dann durch die in den erweiterten Optionen gewählten Einstellungen parametrisiert und die Anfrage finalisiert. Schlussendlich übergibt das System die Anfrage an die Komponente *OpenAiService*, welche die Anfrage dann an die Schnittstelle sendet und auf diese Weise eine Erklärung generieren lässt. Es ist wichtig anzumerken, dass diese Beschreibung für das Generieren von Erklärungen sehr generisch ist und für beide Erklärungstypen gilt.

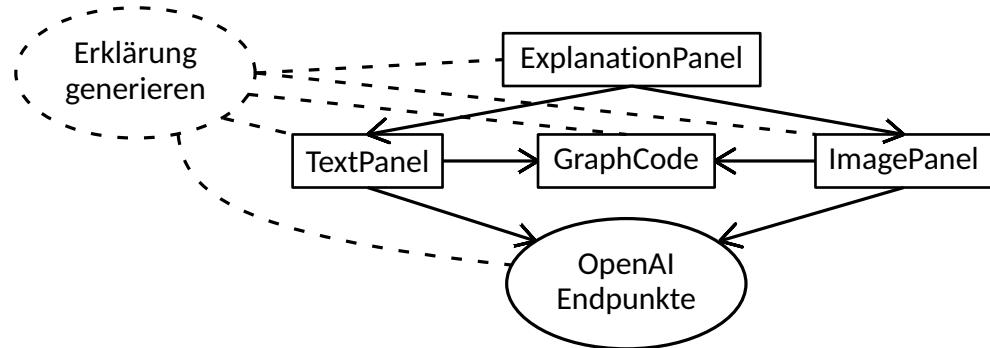


Abb. 3.20: Mechanismus für den Anwendungsfall UC-1.8.

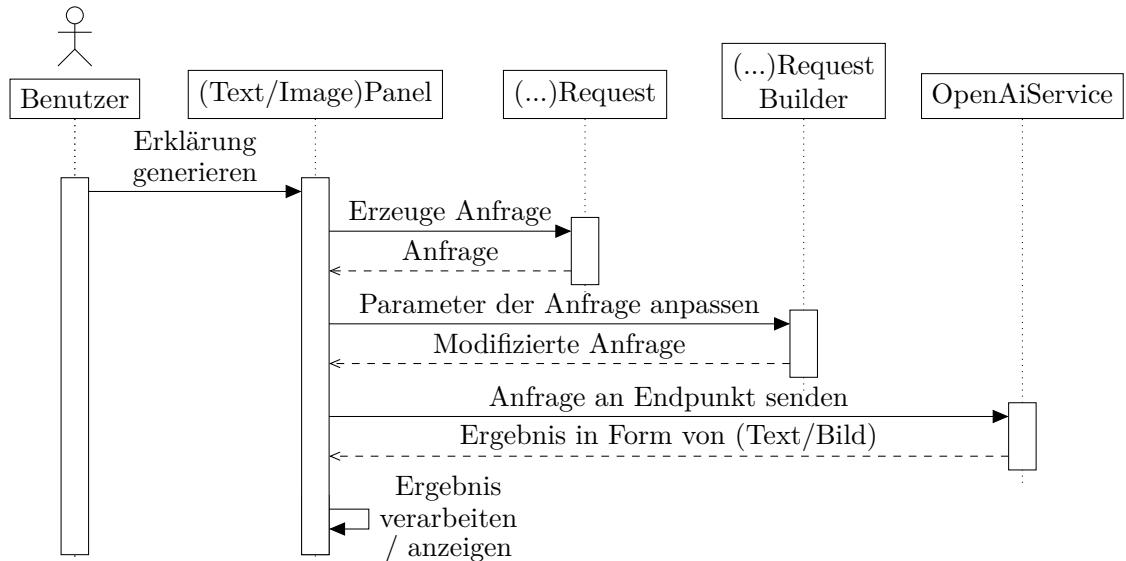


Abb. 3.21: UML-Sequenzdiagramm für den Anwendungsfall UC-1.8.

Abb. 3.21 zeigt ein UML-Sequenzdiagramm für den Anwendungsfall UC-1.8. Es zeigt, wie der Benutzer die Aktion „Erklärung generieren“ durch einen Klick auf den Knopf „Generate ...“ initiiert. Die Abbildung stellt dabei eine generische Darstellung eines Ablaufs zum Generieren einer Erklärung dar und gilt für beide Komponenten *ImagePanel* und *TextPanel*. Im weiteren Verlauf wird dann die notwendigen

Komponenten, die für eine Anfrage benötigt werden, erzeugt und anhand von Benutzereingaben parametrisiert bzw. deren Parameter modifiziert. Hierzu gehören im Übrigen auch die Eingabeaufforderungen / Prompt, die die bereits verarbeiteten Informationen aus einem Graph Code enthalten. Diese angepasste Anfrage kann dann über den Endpunkt an die Schnittstelle gesendet werden. Nach Fertigstellung der Verarbeitung seitens des Systems generativer KI, wird über die Schnittstelle ein Ergebnis, abhängig vom Erklärungstyp, zurückgeliefert. Dieses Ergebnis kann dann schlussendlich von der Komponente *ImagePanel* oder *TextPanel* weiterverarbeitet oder angezeigt werden.

3.3.3 Diskussion

In diesem Abschnitt wurden die offenen Herausforderungen **OH 2.1** und **OH 2.2** adressiert. In Abschnitt 3.3.1 wurde die offene Herausforderung **OH 2.1** behandelt und Konzepte zur Transformation von Graph Codes vorgestellt, mit welchen Graph Codes in eine geeignete Form zur Eingabe in Systeme generativer KI transformiert werden sollen. In Abschnitt Abschnitt 3.3.2 wurde die offene Herausforderung **OH 2.2** behandelt und Konzepte zur Integration von Systemen generativer KI vorgestellt.

3.4 Zusammenfassung

In diesem Kapitel wurde die Modellierung behandelt und es wurden in Abschnitt 3.2 Konzepte für die Erklärbarkeit von MMIR mittels generativer KI, sowie in Abschnitt 3.3 Konzepte für die Integration von generativer KI in das GMAF entwickelt. In Abschnitt 3.4.1 werden die in diesem Kapitel gewonnenen Erkenntnisse zusammengefasst, in Abschnitt 3.4.2 wird das weitere Vorgehen festgehalten und schlussendlich in Tab. 3.3 eine Übersicht des aktuellen Arbeitsstands in einer Tabelle dargestellt. In diesem Kapitel wurde somit der Kern-Arbeitsprozess „Analyse & Design“ des RUP-Vorgehensmodells abgedeckt.

3.4.1 Gewonnene Erkenntnisse

In diesem Abschnitt werden die Erkenntnisse aus den Forschungszielen „FZ 1.2/TB Erklärbarkeit von MMIR mittels generativer KI“ und „FZ 2.2/TB Integration generativer KI in das GMAF“ zusammengefasst.

Im ersten Forschungsziel wurde angesprochen, wie Erklärbarkeit durch generative KI erreicht werden kann. Darüber hinaus wurden in Bezug auf das GMAF eine Reihe an benutzerorientierten Anwendungsfällen identifiziert und textuell beschrieben. Anhand dieser Anwendungsfälle wurden schrittweise Benutzungsschnittstellen mit geeigneten Interaktionsmöglichkeiten konzipiert. Des Weiteren wurden mithilfe von Mechanismen, die an Anwendungsfällen beteiligten Komponenten identifiziert, anhand derer wiederum Sequenz- bzw. Interaktionsdiagramme und Pseudoalgorithmen erstellt werden konnten.

Im zweiten Forschungsziel wurde die Transformation der in Graph Codes gespeicherten, sowie der mit Graph Codes assoziierten Informationen behandelt und es wurden Konzepte zur Überführung dieser Informationen in eine geeignete Form, den Prompts, in Systeme generativer KI entwickelt. Zusätzlich wurde die Einbindung der Systeme generativer KI in das GMAF behandelt und Konzepte zur Integration dieser in das GMAF entwickelt.

Die in den Abschnitte 3.2 und 3.3 gewonnenen Erkenntnisse münden in folgendem Klassendiagramm (siehe Abb. 3.22), in welchem die identifizierten und für die Modellierung wichtigen Klassen, sowie die Beziehungen in denen sie zueinander stehen übersichtlich dargestellt werden.

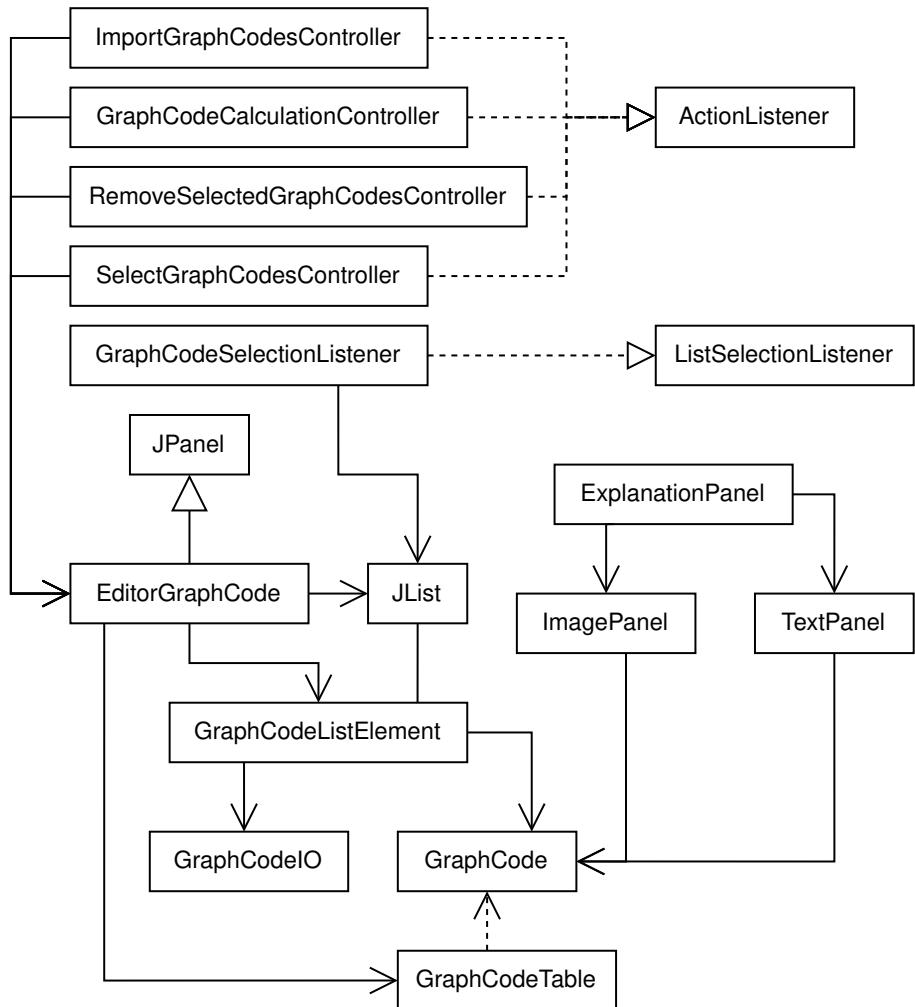


Abb. 3.22: Klassendiagramm.

3.4.2 Weiteres Vorgehen

Im folgenden Kapitel werden die in diesem Kapitel entwickelten Konzepte im Rahmen einer Implementierung durch ihre entsprechenden Forschungsziele untersucht. Genauer werden in Kapitel 4 in FZ 1.3/I Benutzungsschnittstellen und Funktionen implementiert und in FZ 2.3/I die Integration der Systeme generativer KI, sowie die Überführung der Informationen von und um Graph Codes zur Generierung von Erklärungen umgesetzt.

Die nachfolgende Tabelle stellt eine Erweiterung der Tab. 2.4 dar und gibt eine Übersicht über den aktuellen Arbeitsstand nach Abschluss dieses Kapitels.

Tabelle 3.3: Tabelle zur Übersicht des aktuellen Arbeitsstands.

Forschungsziele		
FZ / OH	Beschreibung	Referenz
Erklärbarkeit von MMIR mittels generativer KI		
FZ 1.1/O	Recherche zur Erklärbarkeit von MMIR mittels generativer KI Grundlegende Technologien: <ul style="list-style-type: none"> • GMAF • MMFG • Graph Code 	Abschnitt 2.1.1 Abschnitt 2.1.2 Abschnitt 2.1.3
OH 1.1	Erste offene Herausforderung Systeme generativer KI und ein Überblick über aktuelle Systeme Diskussion und Auswahl von Systemen	OH 1.1 Abschnitt 2.1.4
OH 1.2	Zweite offene Herausforderung	OH 2.1 Abschnitt 2.1.5
FZ 1.2/TB	Modellierung der Erklärbarkeit von MMIR mittels generativer KI Erklärbarkeit durch generative KI → Behandlung der ersten offenen Herausforderung OH 1.1 Anwendungsfälle: <ul style="list-style-type: none"> • Textuelle Beschreibungen Wireframes Mechanismen Sequenzdiagramme → Behandlung der zweiten offenen Herausforderung OH 1.2	Abschnitt 3.2.1 Abschnitt 3.2.2 Abschnitt 3.2.3 Abschnitt 3.2.4 Abschnitt 3.2.5
FZ 1.3/I	Implementierung der Erklärbarkeit von MMIR mittels generativer KI	
FZ 1.4/E	Evaluierung der Erklärbarkeit von MMIR mittels generativer KI	
Integration generativer KI in das GMAF		
FZ 2.1/O	Recherche zur Integration generativer KI in das GMAF	

Forschungsziele		
FZ / OH	Beschreibung	Referenz
	Aufzeigen der Integrationsmöglichkeiten von: <ul style="list-style-type: none"> • Graph Codes Erste offene Herausforderung <ul style="list-style-type: none"> • Systemen generativer KI Zweite offene Herausforderung	Abschnitt 2.2.1 OH 2.1 Abschnitt 2.2.2 OH 2.2
FZ 2.2/TB	Modellierung der Integration generativer KI in das GMAF Transformation von Graph Codes <ul style="list-style-type: none"> • Transformation des Vokabulars • Transformation der Matrix • Anwendung von Graph Code Metriken → Behandlung der ersten offenen Herausforderung OH 2.1 Einbindung generativer KI in das GMAF → Behandlung der zweiten offenen Herausforderung OH 2.2	Abschnitt 3.3.1 Abschnitt 3.3.2
FZ 2.3/I	Implementierung der Integration generativer KI in das GMAF	
FZ 2.4/E	Evaluierung der Integration generativer KI in das GMAF	

4 Implementierung

Dieses Kapitel widmet sich der Implementierung und es werden, entsprechend der Methodik von Nunamaker [25], alle Forschungsziele der Implementierungsphase behandelt. Ziel der Implementierung ist die Untersuchung und Validierung der in Kapitel 3 beschriebenen Modellierung, und die Vorbereitung einer detaillierten Evaluierung in der Experimentphase, sowie Untersuchungen weiterer Beobachtungen, die während der Implementierung gemacht wurden. Die Struktur dieses Kapitels folgt der logischen Abfolge der Forschungsziele der Implementierungsphase. Doch zuvor werden in Abschnitt 4.1 die Grundlagen bzw. Vorbereitungen für die Implementierung getroffen. Dies umfasst die Beschreibung der geplanten Struktur und Architektur der Implementierung, sowie weitere technische Voraussetzungen und die in dieser prototypischen Implementierung genutzten Bibliotheken bzw. Abhängigkeiten. In Abschnitt 4.2 werden ausgewählte Implementierungsdetails der Erklärbarkeit von MMIR mittels generativer KI vorgestellt. Dies umfasst besonders die Benutzeroberfläche, ihre Benutzerelemente und die Interaktion mit diesen. Analog dazu werden in Abschnitt 4.3 ausgewählte Implementierungsdetails der Integration generativer KI in das GMAF vorgestellt. Dies umfasst die Transformation von Graph Codes und die Integration der Schnittstelle von OpenAI und ihre Endpunkte.

Tabelle 4.1: Struktur des Kapitels 4 „Implementierung“.

FZ	Kurze Beschreibung	Abschnitt
Grundlagen der Implementierung	Technische Grundlagen	Abschnitt 4.1
FZ 1.3/I	Implementierung: Benutzungsschnittstelle	Abschnitt 4.2
FZ 2.3/I	Implementierung: Transformation von Graph Codes & Integration gen. KI	Abschnitt 4.3
Zusammenfassung	Ergebnisse / Erkenntnisse	Abschnitt 4.4

4.1 Grundlagen der Implementierung

In diesem Abschnitt werden die Grundlagen für die in diesem Kapitel beschriebene prototypische Proof-of-Concept Implementierung festgehalten. Dies umfasst die allgemeine Architektur, ihr zugrunde liegendes Prinzip, ihr hierarchischer Aufbau, sowie die Abhängigkeiten zu Drittanbieter-Bibliotheken.

4.1.1 Architekturmuster

Die Implementierung der Software folgt dem Muster des *Model-View-Controller*-Prinzips und teilt die Architektur der Software in drei separate Bereiche: *Model* für die Datenstrukturen, *View* für das Frontend bzw. die Benutzungsschnittstellen und *Controller* zum Handhaben von Ereignissen und Interaktionen zwischen den Bereichen *Model* und *View*. Ziel dieser Architektur ist durch das gezielte Trennen dieser Bereiche die Eigenschaften wie Wiederverwendbarkeit, Wartbarkeit und Robustheit der Software zu erhöhen und zu stärken.

Das erste Forschungsziel FZ 1.3/I widmet sich der Benutzungsschnittstelle, ihren Elementen und der Interaktion mit diesen und behandelt somit die Bereiche *View* und *Controller*. Das zweite Forschungsziel FZ 2.3/I widmet sich der Transformation von Graph Codes, sowie der Integration der Schnittstelle von OpenAI, ihrer Endpunkte und dem Erstellen von Anfragen an diese Endpunkte und behandelt somit die Bereiche *Controller* und *Model*.

Abb. 4.1 zeigt die Hierarchie des Quellcodes, der in diesem Kapitel beschriebenen prototypischen Proof-of-Concept Implementierung.

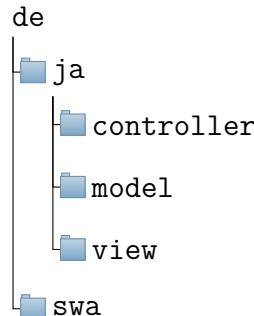


Abb. 4.1: Hierarchie des Quellcodes.

4.1.2 Abhängigkeiten (Maven)

Für die prototypische Proof-of-Concept Implementierung der in Kapitel 3 vorgestellten konzeptuellen Modellierung wird die Programmiersprache Java [133] ausgewählt. Der in diesem Kapitel vorgestellte Quellcode ist auf GitHub verfügbar. Die Implementierung in Java baut hierbei auf Maven [134], einem Build-Tool der Apache Software Foundation für die Projektverwaltung und Management von Abhängigkeiten, auf. Im Folgenden werden die in diesem Projekt verwendet Abhängigkeiten aufgezählt und ihre Verwendungszwecke genannt.

OpenAI-Java

```
<!-- https://mavenrepository.com/artifact/com.theokanning.openai
     -gpt3-java/service -->
<dependency>
  <groupId>com.theokanning.openai-gpt3-java</groupId>
  <artifactId>service</artifactId>
  <version>0.13.0</version>
</dependency>
```

Verwendungszweck: Diese Abhängigkeit stellt die Schnittstelle von OpenAI und ihre Endpunkte in Java bereit.

JTokkit

```
<!-- https://mavenrepository.com/artifact/com.knuddels/jtokkit
     -->
<dependency>
  <groupId>com.knuddels</groupId>
  <artifactId>jtokkit</artifactId>
  <version>0.6.1</version>
</dependency>
```

Verwendungszweck: Diese Abhängigkeit bietet Funktionen zum Verarbeiten der von OpenAI genutzten Zeichenaufteilung in Tokens.

Abhängigkeiten für die Benutzeroberfläche

```
<!-- https://mavenrepository.com/artifact/org.swinglabs/jxlayer
     -->
<dependency>
  <groupId>org.swinglabs</groupId>
  <artifactId>jxlayer</artifactId>
  <version>3.0.4</version>
</dependency>

<!-- https://mavenrepository.com/artifact/org.swinglabs.swingx/
     swingx-all -->
<dependency>
  <groupId>org.swinglabs.swingx</groupId>
  <artifactId>swingx-all</artifactId>
  <version>1.6.5</version>
```

```
</dependency>

<!-- https://mavenrepository.com/artifact/com.miglayout/
     miglayout -->
<dependency>
    <groupId>com.miglayout</groupId>
    <artifactId>miglayout</artifactId>
    <version>3.7.4</version>
</dependency>

<!-- https://mavenrepository.com/artifact/com.fifesoft/
     rsyntaxtextarea -->
<dependency>
    <groupId>com.fifesoft</groupId>
    <artifactId>rsyntaxtextarea</artifactId>
    <version>3.3.1</version>
</dependency>
```

Google Guava

```
<!-- https://mavenrepository.com/artifact/com.google.guava/guava
     -->
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>32.1.2-jre</version>
</dependency>
```

Verwendungszweck: Diese Abhängigkeit bietet Funktionen zur Datenverarbeitung.

4.2 FZ 1.3/I Erklärbarkeit von MMIR mittels generativer KI

In diesem Abschnitt wird die prototypische Proof-of-Concept Implementierung der „Erklärbarkeit von MMIR mittels generativer KI“ beschrieben und behandelt. Die in Kapitel 3 identifizierten Anwendungsfälle beschreiben die Interaktionen eines Benutzers mit dem System. Um eine geschickte Interaktion mit dem System zu ermöglichen, wird eine Benutzungsschnittstelle mit adäquaten Interaktionsmöglichkeiten benötigt. Die Beschreibung der Implementierung solch einer Benutzungsschnittstelle wird im Folgenden in Abschnitt 4.2.1 vorgenommen. Diese Beschreibung umfasst die Elemente der Benutzeroberfläche, sowie die Interaktion zwischen diesen.

4.2.1 Benutzungsschnittstelle

In diesem Abschnitt wird die Implementierung der Benutzeroberfläche beschrieben. Für eine prototypische Proof-of-Concept Implementierung der Benutzeroberfläche wird Java's *Swing* verwendet. *Swing* ist ein Werkzeug für das Erstellen von grafischen Benutzungsschnittstellen bzw. Oberflächen und ist ein grundlegender Bestandteil der Laufzeitumgebung von Java. Das Benutzen von *Swing* hat somit den Vorteil, dass keine zusätzliche Installation notwendig ist und *Swing* somit problemlos auf beliebigen Rechnern, auf welchen Java installiert ist, ausgeführt werden kann.

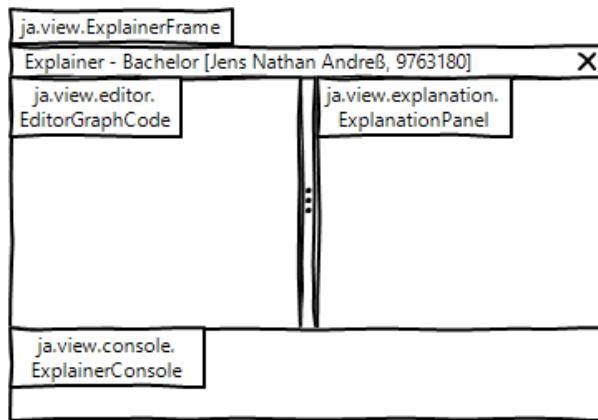


Abb. 4.2: Allgemeine Übersicht über das Mockup der Benutzungsschnittstelle.

Abb. 4.2 zeigt eine allgemeine Übersicht über das bereits in Abb. 3.3 vorgestellte Mockup der Benutzungsschnittstelle, erweitert um Zuweisungen der Komponenten mitsamt implementierungsspezifischen Paketnamen. Die Elemente der in Abb. 4.2 gezeigten Benutzungsschnittstelle werden in Abschnitt 4.2.1.1 genauer beschrieben. Aufbauend auf diesen Elementen wird dann in Abschnitt 4.2.1.2 die Interaktion

zwischen diesen Elementen beschrieben. Auf diese Weise entspricht der Ablauf dieses Abschnitts dem Ablauf der Modellierung der Benutzungsschnittstelle.

Elemente der Benutzeroberfläche

Das Fundament der Benutzungsschnittstelle ist das *ExplainerFrame*, welches durch die Klasse *ja.view.ExplainerFrame* umgesetzt wird. Listing 4.1 zeigt die wichtigsten Aspekte der Implementierung dieser Klasse.

```
1 public class ExplainerFrame extends JFrame {
2
3     // EditorGraphCode
4     private EditorGraphCode editorGraphCode;
5     // ExplainerPanel
6     private ExplanationPanel explanationPanel;
7     // ExplainerConsole
8     private ExplainerConsole explainerConsole;
9     // Datenmodell für ExplainerConsole
10    private ExplainerConsoleModel explainerConsoleModel;
11
12    public ExplainerFrame() {
13        // Frame initialisieren u. konfigurieren
14        initFrame();
15        // Komponenten initialisieren (siehe Listing 4.14)
16        initComponents();
17    }
18
19    /* Getter für EditorGraphCode, ExplanationPanel,
20     * ExplainerConsole u.w. */
21}
```

Listing 4.1: ExplainerFrame-Klasse

Nach der Modellierung enthält dieses Fundament drei Grundbereiche mit folgender Nummerierung (siehe Abb. 3.3): ① *EditorGraphCode*, ② *ExplainerPanel* und ③ *ExplainerConsole*. Diese Grundbereiche werden in Listing 4.1 durch die privaten Variablen *editorGraphCode*, *explanationPanel* und *explainerConsole* dargestellt. Beim Erzeugen der Klasse *ExplainerFrame* wird zuerst durch die Methode *initFrame()* das Frame initialisiert und konfiguriert. Dies umfasst die Dimension des Frames, die Position, sowie den Titel. Darauffolgend werden dann durch die Methode *initComponents()* die Komponenten, welches auch die Grundbereiche umfassen, initialisiert, konfiguriert und dem Frame hinzugefügt. Dies umfasst besonders die notwendigen Schritte zum Hinzufügen und Layouten (z.Dt. Auslegen) der Grundbereiche im *ExplainerFrame*. Im weiteren Verlauf wird zuerst der Grundbereich *EditorGraphCode*, welcher die linke Arbeitsfläche darstellt, dann der Grundbereich *ExplanationPanel*, welcher die rechte Arbeitsfläche darstellt und schlussendlich der Grundbereich

ExplainerConsole, welcher die Konsole darstellt, behandelt. Jeder dieser Grundbereiche wird mitsamt seiner enthaltenen Komponenten ausführlich behandelt, bevor mit dem nächsten Grundbereich fortgefahrene wird.

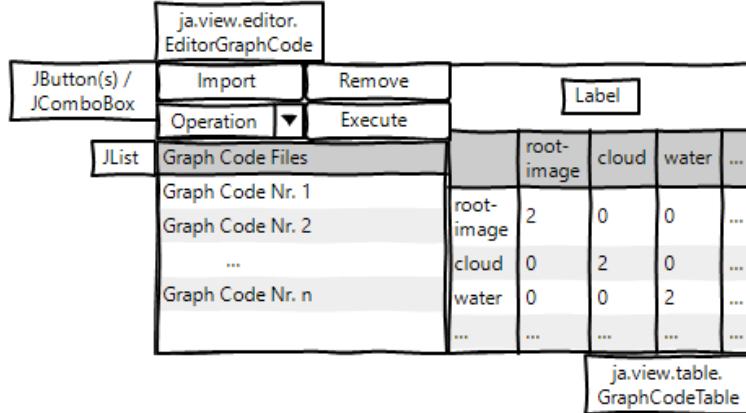


Abb. 4.3: Grundbereich *EditorGraphCode*.

Abb. 4.3 zeigt den Grundbereich *EditorGraphCode*, umgesetzt durch die Klasse *ja.view.editor.EditorGraphCode*, in welchem Benutzer durch geeignete Interaktionsmöglichkeiten die Bearbeitung von Graph Code Dateien durchführen können. Listings 4.2 bis 4.4 und 4.7 zeigen im Folgenden die wichtigsten Aspekte der Implementierung dieser Klasse.

```

1 public class EditorGraphCode extends JPanel {
2
3     // Name des ausgewählten Graph Codes (oberer rechter Teil)
4     private final JLabel graphCodeName;
5     // Darstellung eines Graph Codes in Tabellenform
6     private final GraphCodeTable graphCodeTable;
7     // Operationen nach Anwendungsfall UC-1.4
8     private final JComboBox<String> calculationComboBox;
9     // Liste für und zur Darstellung von Graph Code Dateien
10    private final JList<GraphCodeListElement> graphCodeList;
11
12    public EditorGraphCode(ExplainerFrame reference) {
13        // Eigenschaften konfigurieren ... (Layout, etc.)
14        // Linker Teil der Arbeitsfläche (umfasst Abb. 3.4 ④ + ⑤)
15        JPanel leftPart = new JPanel();
16        // Rechter Teil der Arbeitsfläche (umfasst Abb. 3.4 ⑥)
17        JPanel rightPart = new JPanel();
18        // (siehe Abb. 3.4 ④)
19        JPanel operationsPanel = new JPanel();
20        operationsPanel.setLayout(new GridLayout(0, 2));
21        // (siehe Abb. 3.4 ⑤)
22        graphCodeList = new JList<>(graphCodeListModel);

```

```

23 // Rechter Teil (oben)
24 JPanel topRight = new JPanel();
25 graphCodeName = new JLabel();
26 // (siehe Abb. 3.4 (6))
27 graphCodeTable = new GraphCodeTable();

```

Listing 4.2: EditorGraphCode-Klasse

Der Modellierung nach ist diese Arbeitsfläche in zwei nebeneinanderliegende Arbeitsflächen aufgeteilt, die in Listing 4.2 durch die Variablen *leftPart* und *rightPart* dargestellt werden. Auch diese Arbeitsflächen bestehen wiederum aus mehreren Komponenten. Die linke Arbeitsfläche umfasst eine Schnittstelle zum Auswählen und Ausführen von Aktionen (siehe Abb. 3.4 (4)), dargestellt durch die Variable *operationsPanel* und eine Liste an Graph Code Dateien (siehe Abb. 3.4 (5)), dargestellt durch die Variable *graphCodeList*. Die rechte Arbeitsfläche hingegen umfasst im Wesentlichen eine Tabelle zur Visualisierung eines ausgewählten Graph Codes (siehe Abb. 3.4 (6)), dargestellt durch die Variable *graphCodeTable*, sowie ein Label zum Darstellen des Namens der entsprechenden Graph Code Datei über dieser Tabelle, dargestellt durch die Variable *graphCodeName*.

Im Folgenden wird in Listing 4.3, dem zweiten Teil der EditorGraphCode-Klasse, die Schnittstelle (4) zum Auswählen und Ausführen von Aktionen genauer spezifiziert und im weiteren Verlauf erklärt.

```

29 // Anwendungsfall UC-1.1
30 JButton openGraphCodeChooserButton = new JButton("Import Graph
   Code(s)");
31 // ImportGraphCodesController (siehe Listing 4.15)
32 openGraphCodeChooserButton.addActionListener(new
   ImportGraphCodesController(this));
33 // Anwendungsfall UC-1.2
34 JButton removeSelectedButton = new JButton("Remove selected
   Graph Code(s)");
35 // RemoveSelectedGraphCodesController (siehe Listing 4.16)
36 removeSelectedButton.addActionListener(new
   RemoveSelectedGraphCodesController(this));
37 // Anwendungsfall UC-1.4
38 calculationComboBox = new JComboBox<>();
39 calculationComboBox.addItem("Union");
40 calculationComboBox.addItem("Subtraction");
41 calculationComboBox.addItem("Similarities");
42 calculationComboBox.addItem("Differences");
43 // Anwendungsfall UC-1.5
44 JButton calculationOperationButton = new JButton("Execute");
45 // GraphCodeCalculationController (siehe Listings 4.19 bis 4.22
   und 4.24)
46 calculationOperationButton.addActionListener(new
   GraphCodeCalculationController(this));

```

```

47 // Hinzufügen der Elemente
48 operationsPanel.add(openGraphCodeChooserButton);
49 operationsPanel.add(removeSelectedButton);
50 operationsPanel.add(calculationComboBox);
51 operationsPanel.add(calculationOperationButton);

```

Listing 4.3: EditorGraphCode-Klasse (Zweiter Teil)

Die Schnittstelle zum Auswählen und Ausführen von Aktionen, dargestellt durch *operationsPanel*, besitzt vier Elemente. Diese Elemente sind JButtons bzw. Knöpfe und eine JComboBox bzw. ein Button mit einer aus- und einklappbaren Auswahlliste und werden im Folgenden aufgezählt:

- Knopf „Import Graph Code(s)“. Dieser Knopf ist mit dem Anwendungsfall UC-1.1 „Graph Code(s) importieren“ verbunden und wird im Quellcode als Variable *openGraphCodeChooserButton* eingeführt. Weiterhin wird die Interaktion durch ein Steuerelement *ImportGraphCodesController*, welches in einem Mechanismus (siehe Abb. 3.6) identifiziert werden konnte, gesteuert. Das an diesem Anwendungsfall beteiligte Steuerelement *ImportGraphCodesController* wird in Listing 4.15 genauer behandelt.
- Knopf „Remove selected Graph Code(s)“. Dieser Knopf ist mit dem Anwendungsfall UC-1.2 „Graph Code(s) entfernen“ verbunden und wird im Quellcode als Variable *removeSelectedButton* eingeführt. Weiterhin wird die Interaktion durch ein Steuerelement *RemoveSelectedGraphCodesController*, welches in einem Mechanismus (siehe Abb. 3.7) identifiziert werden konnte, gesteuert. Das an diesem Anwendungsfall beteiligte Steuerelement *RemoveSelectedGraphCodesController* wird in Listing 4.16 genauer behandelt.
- Aus- und einklappbare Auswahlliste. Diese Auswahlliste ist mit dem Anwendungsfall UC-1.4 „Operation auswählen“ verbunden und wird im Quellcode als Variable *calculationComboBox* eingeführt. Die Interaktion mit diesem Element wird durch das Steuerelement *GraphCodeCalculationController*, welches in einem Mechanismus (siehe Abb. 3.9) identifiziert werden konnte, gesteuert. Das an diesem Anwendungsfall beteiligte Steuerelement *GraphCodeCalculationController* wird in Listings 4.19 bis 4.22 und 4.24 genauer behandelt.
- Knopf „Execute“. Dieser Knopf ist mit dem Anwendungsfall UC-1.5 verbunden und wird im Quellcode als Variable *calculationOperationButton* eingeführt. Weiterhin wird die Interaktion ebenfalls durch das Steuerelement *GraphCodeCalculationController* gesteuert.

Schlussendlich ist in Listing 4.3 zu sehen, wie die Elemente der Schnittstelle hinzugefügt werden. Die aus der Implementierung resultierende Schnittstelle zum Auswählen und Ausführen von Aktionen kann in Abb. 4.4 eingesehen werden.

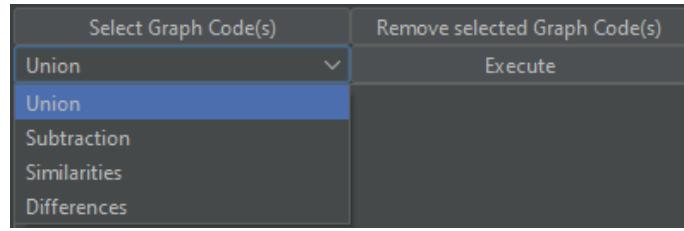


Abb. 4.4: Aus der Implementierung resultierende Schnittstelle zum Auswählen und Ausführen von Aktionen.

Im Folgenden wird in Listing 4.4, dem dritten Teil der EditorGraphCode-Klasse, die Schnittstelle (5) bzw. Liste von Graph Codes genauer spezifiziert und im weiteren Verlauf erklärt.

```

52 // Eigenschaften der Liste konfigurieren
53 // Anwendungsfall UC-1.3, Interaktion (siehe Listing 4.17)
54 graphCodeList.addListSelectionListener(new
55     GraphCodeSelectionListener(reference));
56 // Interaktion (siehe Listing 4.18)
57 graphCodeList.addMouseListener(new
58     GraphCodeListAdapter(this));
59 // Drag und Drop
60 graphCodeList.setTransferHandler(new
61     ListItemTransferHandler<GraphCodeListElement>());

```

Listing 4.4: EditorGraphCode-Klasse (Dritter Teil)

Zuerst werden Eigenschaften der Liste konfiguriert. Eine dieser Eigenschaften ist z.B. der Auswahlmodus *MULTIPLE_INTERVAL_SELECTION*, um eine mehrfache Auswahl in der Liste zu ermöglichen.

Die Liste ist mit dem Anwendungsfall UC-1.3 „Graph Code(s) auswählen“ verbunden. In Bezug auf den Anwendungsfall wird die Interaktion mit der Liste durch das Steuerelement *GraphCodeSelectionListener* gesteuert, welches in einem Mechanismus (siehe Abb. 3.8) identifiziert werden konnte. Das Steuerelement *GraphCodeSelectionListener* wird in Listing 4.17 genauer behandelt. Zusätzlich besitzt die Liste weitere Steuerelemente, wie *GraphCodeListAdapter* und *ListItemTransferHandler*, die ebenfalls in Abschnitt 4.2.1.2 genauer behandelt werden. Des Weiteren konnte in diesem Mechanismus, wie bereits auch schon in den vorherigen Mechanismen, die Komponente *GraphCodeListElement* identifiziert werden. Diese Komponente ist in Bezug auf die Liste von besonderer Bedeutung, da diese ein Element bzw. Eintrag in der Liste darstellt. Der Quellcode für diese Komponente kann in Listing 4.5 eingesehen werden.

```

1 public class GraphCodeListElement implements Serializable {
2
3     // Graph Code
4     private final GraphCode graphCode;
5     // Name der Graph Code Datei
6     private String fileName;
7     // Referenz zur Datei
8     private File file;
9
10    public GraphCodeListElement(File file) {
11        this.file = file;
12        this.fileName = file.getName();
13        this.graphCode = GraphCodeIO.read(file);
14    }
15
16    public GraphCodeListElement(GraphCode graphCode, String name) {
17        this.graphCode = graphCode;
18        this.fileName = name;
19    }
20
21    /* Getter / Setter */
22}

```

Listing 4.5: GraphCodeListElement-Klasse

Abb. 4.5 zeigt ein Beispiel für die aus der Implementierung resultierende Liste mit-samt beispielhaften Einträgen für Graph Codes.

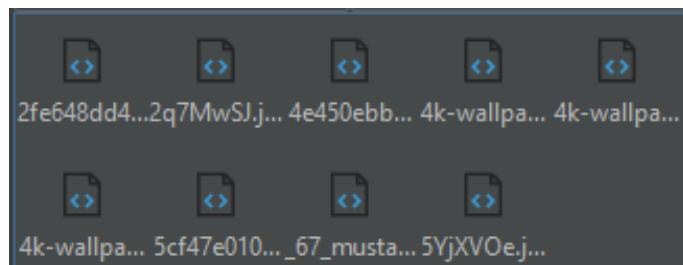


Abb. 4.5: Aus der Implementierung resultierende Liste mit beispielhaften Einträgen für Graph Codes.

Schlussendlich wird in Listing 4.6 die letzte Schnittstelle (6) *GraphCodeTable* des Grundbereichs *EditorGraphCode* behandelt und im weiteren Verlauf erklärt.

```

1 public class GraphCodeTable extends JPanel {
2
3     // Platzhalter
4     private final JLabel jTablePlaceholder;
5     // Model für die Tabelle

```

```

6   private GraphCodeTableModel graphCodeTableModel;
7   // Tabelle für ausgewählten Graph Code
8   private JTable graphCodeTable;
9   // Container für Tabelle
10  private JPanel tablePanel;
11
12  private JScrollPane graphCodeTableSP;
13
14  public GraphCodeTable() {
15      graphCodeTable = new JTable();
16      graphCodeTableModel = new GraphCodeTableModel();
17      graphCodeTable.setModel(graphCodeTableModel);
18      // Tabelle hinzufügen und einbetten
19      tablePanel = new JPanel();
20      tablePanel.add(graphCodeTable);
21      graphCodeTableSP = new JScrollPane(tablePanel);
22      // Platzhalter hinzufügen
23      jTablePlaceHolder = new JLabel("No GraphCode selected!");
24      add(jTablePlaceHolder, BorderLayout.CENTER);
25  }
26
27  public void setGraphCode(GraphCode graphCode) {
28      if(graphCode != null) {
29          remove(jTablePlaceHolder);
30          // Model über Änderung informieren...
31          graphCodeTableModel.setGraphCode(graphCode);
32          graphCodeTableModel.fireTableStructureChanged();
33          add(graphCodeTableSP, BorderLayout.CENTER);
34      } else {
35          // Sofern Tabelle vorhanden ist, entfernen
36          // Model über Änderung informieren
37          if(graphCodeTable != null) {
38              remove(graphCodeTableSP);
39              graphCodeTableModel.setGraphCode(null);
40          }
41          addPlaceHolder();
42      }
43      // UI aktualisieren...
44  }
45
46  private void addPlaceHolder() {...}
47 }

```

Listing 4.6: GraphCodeTable-Klasse

Die Schnittstelle *GraphCodeTable* besitzt zwei Elemente: Eine Tabelle zur Darstellung eines ausgewählten Graph Codes, umgesetzt durch ein *JTable* und ein Platzhalter, zum Signalisieren, dass zu diesem Zeitpunkt noch kein Graph Code ausgewählt ist / wurde. Die Tabelle wird in einem *JScrollPane* eingebettet, um auch größere

re Tabellen darstellen zu können. Des Weiteren ist ein wichtiger Teil dieser Klasse ein Datenmodell für die Tabelle, dargestellt durch die Komponente *GraphCodeTableModel*. Dieses Datenmodell wird der Tabelle zugewiesen und hat zur Aufgabe, die Informationen in einem Graph Code auf die Zeilen und Spalten der Tabelle abzubilden, sodass diese in Tabellenform dargestellt werden können. Die wichtigste Methode der Klasse *GraphCodeTable* ist *setGraphCode*. Diese Methode nimmt als Parameter einen ausgewählten Graph Code. Anhand dieses Graph Codes werden Veränderungen an der Schnittstelle, sowie dem Datenmodell vorgenommen. Sofern kein Graph Code ausgewählt ist, wird ein Platzhalter angezeigt. Andernfalls wird dem Datenmodell der Graph Code zur Verarbeitung zugewiesen.

Abb. 4.6 zeigt ein Beispiel für die aus der Implementierung resultierende Tabelle zur Darstellung eines ausgewählten Graph Codes.

	root-image	water	sky	cloud	building	skyscraper	dusk	body of wa...	lake	atmospheric	horizon
root-i...	1	1	1	1	1	1	1	1	1	1	1
water	0	2	0	0	0	0	0	0	0	0	0
sky	0	0	2	0	0	0	0	0	0	0	0
cloud	0	0	0	2	0	0	0	0	0	0	0
building	0	0	0	0	2	0	0	0	0	0	0
skyscra...	0	0	0	0	0	2	0	0	0	0	0
dusk	0	0	0	0	0	0	2	0	0	0	0
body o...	0	0	0	0	0	0	0	2	0	0	0
lake	0	0	0	0	0	0	0	0	2	0	0
atmosp...	0	0	0	0	0	0	0	0	0	2	0
horizon	0	0	0	0	0	0	0	0	0	0	2

Abb. 4.6: Aus der Implementierung resultierende Tabelle zur Darstellung eines ausgewählten Graph Codes.

Im Folgenden wird in Listing 4.7, der vierte und letzte Teil der *EditorGraphCode*-Klasse, genauer spezifiziert und im weiteren Verlauf erklärt.

```

58     // Hinzufügen von ④ + ⑤
59     leftPart.add(operationsPanel, BorderLayout.NORTH);
60     leftPart.add(new JScrollPane(graphCodeList), BorderLayout.
61                 CENTER);
62     // Hinzufügen von ⑥
63     topRight.add(graphCodeName, BorderLayout.CENTER);
64     rightPart.add(graphCodeTable, BorderLayout.CENTER);

```

```

64     rightPart.add(topRight, BorderLayout.NORTH);
65     // Linke und rechte Arbeitsfläche hinzufügen
66     add(leftPart);
67     add(rightPart);
68 }
69
70 /* Getter */
71 }

```

Listing 4.7: EditorGraphCode-Klasse (Letzter Teil)

Listing 4.7 zeigt das abschließende Zusammenfügen der Schnittstellen ④, ⑤ und ⑥. Zusammengefügt ergeben diese Schnittstellen den Grundbereich *EditorGraphCode*. Damit ist der Grundbereich ①, *EditorGraphCode*, abgeschlossen. Abb. 4.7 zeigt abschließend die vollständige Benutzeroberfläche des aus der Implementierung resultierenden Grundbereichs *EditorGraphCode* mitsamt Beispielen.

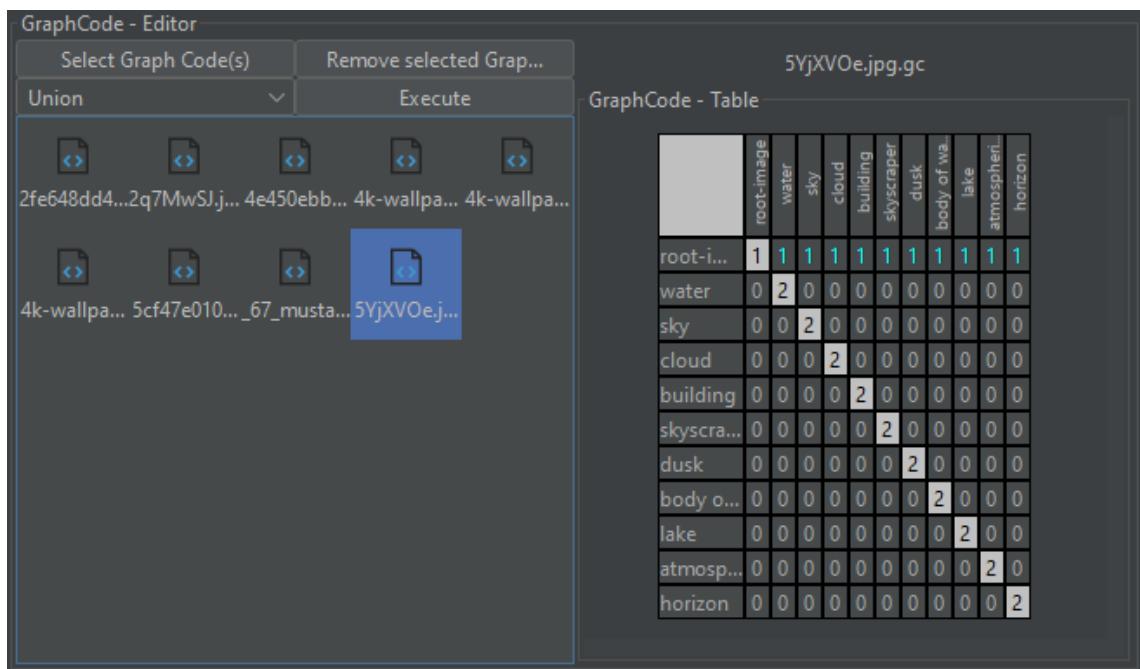


Abb. 4.7: Vollständige Oberfläche des aus der Implementierung resultierenden Grundbereichs *EditorGraphCode*.

Im weiteren Verlauf dieses Abschnitts wird in Listing 4.8 der Grundbereich ②, *ExplanationPanel*, behandelt und die darin enthaltenen Schnittstellen detailliert erläutert.

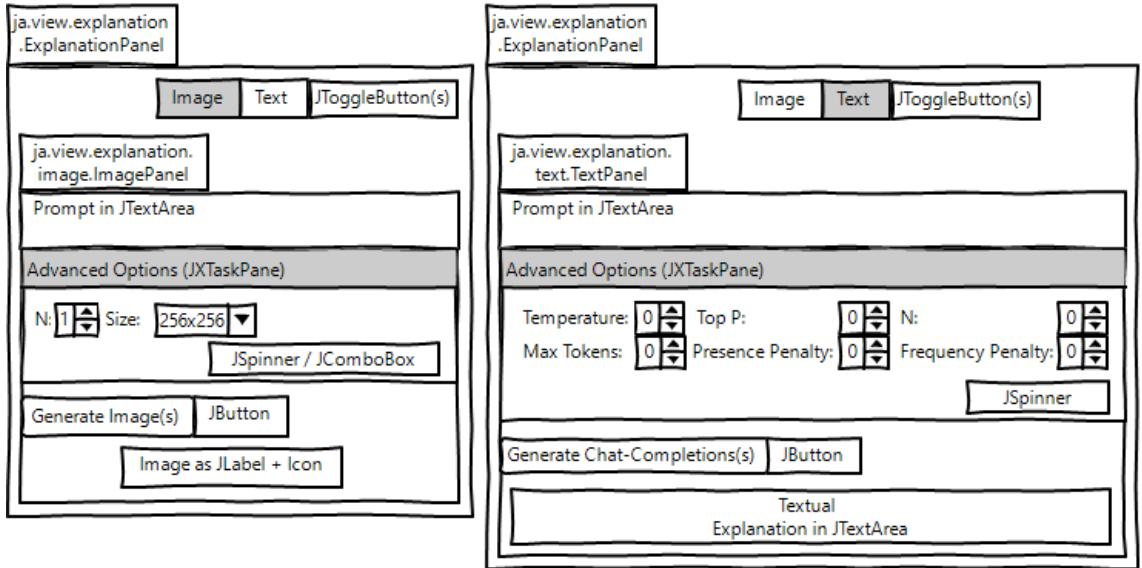


Abb. 4.8: Grundbereich *ExplanationPanel* zur Generierung von visuellen Erklärungen (links) und textuellen Erklärungen (rechts).

Abb. 4.8 zeigt den nächsten Grundbereich ②, *ExplanationPanel*, umgesetzt durch die Klasse `ja.view.explanation.ExplanationPanel`, in welchem Benutzer durch geeignete Interaktionsmöglichkeiten die Generierung von textuellen und visuellen Erklärungen zu Graph Codes durchführen können. Hierzu besitzt der Grundbereich *ExplanationPanel* eine Fläche, in der Komponenten miteinander ausgetauscht werden können. Genauer wird zwischen den zwei Komponenten: *TextPanel* und *ImagePanel*, gewechselt.

```

1 public class ExplanationPanel extends JPanel implements
2   ActionListener {
3
4   // Schnittstelle zur Generierung visueller Erklärungen
5   private final ImagePanel imagePanel;
6   // Schnittstelle zur Generierung textueller Erklärungen
7   private final TextPanel textPanel;
8
9   public ExplanationPanel() {
10     JToggleButton imageButton = new JToggleButton("Image");
11     imageButton.setActionCommand("ImagePanel");
12     imageButton.addActionListener(this);
13
14     JToggleButton textButton = new JToggleButton("Text");
15     textButton.setActionCommand("TextPanel");
16     textButton.addActionListener(this);
17     // Nur ein Button kann zeitgleich ausgewählt sein
18     ButtonGroup group = new ButtonGroup();
19     group.add(imageButton);

```

```

19     group.add(textButton);
20
21     imagePanel = new ImagePanel();
22     textPanel = new TextPanel();
23
24     JPanel tabPanel = new JPanel();
25     tabPanel.add(imageButton);
26     tabPanel.add(textButton);
27     add(tabPanel, BorderLayout.NORTH);
28     add(imagePanel, BorderLayout.CENTER);
29 }
30
31     public void setGraphCode(GraphCode graphCode) {
32         // Graph Code delegieren...
33         imagePanel.setGraphCode(graphCode);
34         textPanel.setGraphCode(graphCode);
35     }
36
37     @Override
38     public void actionPerformed(ActionEvent e) {
39         // Wechseln zwischen ImagePanel und TextPanel.
40         // Ist ImagePanel ausgewählt, wird TextPanel entfernt u. vice
41         // versa.
42     }

```

Listing 4.8: ExplanationPanel-Klasse

Der Grundbereich *ExplanationPanel* besitzt zwei Knöpfe (*JToggleButton*) „Image“ und „Text“, mit welchen zwischen den Schnittstellen *ImagePanel*, zum Generieren von visuellen Erklärungen zu Graph Codes und *TextPanel*, zum Generieren von textuellen Erklärungen zu Graph Codes, hin- und hergeschaltet werden kann. Die Interaktion für diese Knöpfe wird innerhalb der Klasse durch die Methode *actionPerformed* umgesetzt. Hierfür implementiert die Klasse das Interface *ActionListener* und wird den Knöpfen registriert. Damit nur ein Knopf zeitgleich aktiviert sein kann, werden die Knöpfe einer *ButtonGroup* hinzugefügt. Die zwei Schnittstellen *TextPanel* und *ImagePanel* werden beide initialisiert und existieren zum selben Zeitpunkt. Jedoch kann, in Abhängigkeit des gewählten Knopfes, immer nur eine Schnittstelle angezeigt werden. Am Anfang ist die Schnittstelle *ImagePanel* zu sehen.

Des Weiteren umfasst die Klasse zwei Methoden: *setGraphCode* und *actionPerformed*. Die Methode *setGraphCode* delegiert einen Graph Code an die Schnittstellen *ImagePanel* und *TextPanel*, wo diese dann entsprechend weiter verarbeitet werden. Die Methode *actionPerformed* wird durch das Interface *ActionListener* bereitgestellt und durch die Methode *addActionListener* auf die Knöpfe registriert. Diese Methode übernimmt das Hin- und Herschalten zwischen *Text*- oder *ImagePanel* und entfernt *TextPanel*, sofern *ImagePanel* ausgewählt wird und vice versa.

Im Folgenden werden die Schnittstellen *TextPanel* und *ImagePanel* genauer erklärt. Dabei wird zuerst *TextPanel* und dann *ImagePanel* behandelt. Grund für diese Reihenfolge ist die Art wie die Erklärungen, ob textuell oder visuell, generiert werden. Wie bereits in Abschnitt 3.3.2 konzipiert wurde, wird beim Erzeugen einer visuellen Erklärung erst eine mit Tokenbegrenzung versehene, textuelle Erklärung generiert, die eine visuelle Beschreibung darstellen und als Grundlage für eine Bilderzeugung dienen soll. Somit ist das Generieren einer textuellen Erklärung ein wichtiger Zwischenschritt in der Generierung einer visuellen Erklärung und wird als thematisiert.

```

1 public class TextPanel extends JPanel implements ActionListener {
2
3     // API-Schlüssel
4     private static String key;
5     // Textfeld für die Prompt
6     private final JTextArea promptArea;
7     // Textfeld für die textuelle Erklärung
8     private final JTextArea textResponseArea;
9     // Elemente für erweiterte Optionen
10    private final JSpinner temperatureSpinner, topPSpinner,
11        nSpinner, maxTokensSpinner, presencePenaltySpinner,
12        frequencyPenaltySpinner;
13    // Textnachrichten
14    private List<ChatMessage> messages = new ArrayList<>();
15    // Referenz
16    private final ExplainerFrame reference;
17
18    public TextPanel(ExplainerFrame reference) {
19        this.reference = reference;
20        key = System.getenv("OpenAI-Key");
21        // Layout: MigLayout
22        // Textfeld für die Prompt
23        promptArea = new JTextArea();
24        JScrollPane promptSP = new JScrollPane(promptArea);
25        // Erweiterte Optionen
26        JXTaskPane advancedOptions = new JXTaskPane("Advanced Options")
27            ;
28        // Elemente der erweiterten Optionen
29        // umfassen JLabels + JSpinners für
30        // die Optionen: Temperature, TopP, N,
31        // MaxTokens, PresencePenalty,
32        // frequencyPenalty (siehe ...).
33        // Die JSpinners besitzen zudem
34        // Datenmodelle für Wertebereiche (SpinnerNumberModel's).
35        advancedOptions.add( JLabels + JSpinners);

```

Listing 4.9: TextPanel-Klasse

Listing 4.9 zeigt den ersten Teil der Klasse *TextPanel* und die ersten oberen Schnittstellen (8) und (9). Genauer sind dies ein Textfeld für die Prompt, eingeführt durch

die Variable *promptArea* und eine ein- und ausklappbare Fläche für erweiterte Optionen, eingeführt durch die Variable *advancedOptions*. Diese ein- und ausklappbare Fläche wird durch die Klasse *JXTaskPane* umgesetzt und ist Teil der Abhängigkeit *SwingX*. Dieser Fläche werden wiederum mehrere Elemente in Form von *JLabels* und *JSpinner* hinzugefügt, die die erweiterten Optionen darstellen. Mithilfe der erweiterten Optionen können Anfragen an den Endpunkt Text angepasst werden und die Wahrscheinlichkeiten für bestimmte Ausgaben beeinflusst werden. Da diese Optionen für gewöhnlich eingeschränkte Wertebereiche aufweisen, werden den *JSpinner* daher Datenmodelle (*SpinnerNumberModel*) zugewiesen, um Benutzereingaben einzuschränken. Abb. 4.9 zeigt die ersten beiden oberen Schnittstellen (8) und (9) für *TextPanel*.

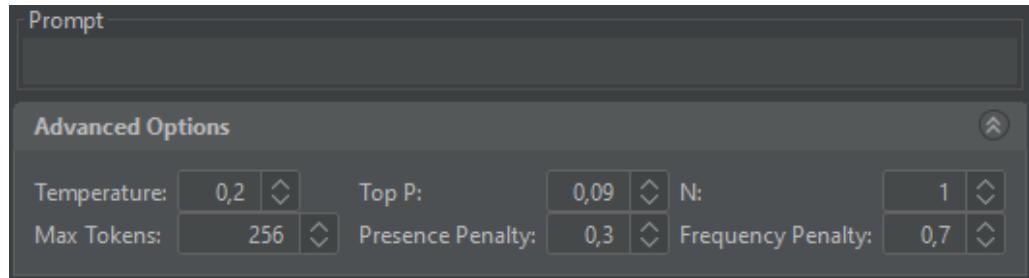


Abb. 4.9: Textfeld für die Prompt und erweiterte Optionen.

```

35     // Knopf zum Generieren einer textuellen Erklärung
36     JButton generateChatCompletions = new JButton("Generate Chat-
37         Completion(s)");
38     // Interaktion (siehe actionPerformed)
39     generateChatCompletions.addActionListener(this);
40     // Textfeld für die erzeugte textuelle Erklärung
41     textResponseArea = new JTextArea();
42     JScrollPane textResponseAreaSP = new JScrollPane(
43         textResponseArea);
44     // ... Constraints fürs Layout
45     add(promptSP, "cell 0 0, ...");
46     add(advancedOptions, "cell 0 1, ...");
47     add(generateChatCompletions, "cell 0 2, ...");
48     add(textResponseAreaSP, "cell 0 3, ...");
49 }
```

Listing 4.10: TextPanel-Klasse (Zweiter Teil)

Listing 4.10 zeigt den zweiten Teil der Klasse *TextPanel* und die untere Schnittstelle (10). Dies ist im Fall der Klasse *TextPanel* ein Textfeld für die generierte textuelle Erklärung und diese wird im Quellcode als Variable *textResponseArea* eingeführt. Zusätzlich wird ein Knopf „Generate Chat-Completion(s)“ hinzugefügt und die Klasse

als *ActionListener* dem Knopf registriert. Hierfür implementiert die Klasse *TextPanel* das Interface *ActionListener*. Aufgabe des Knopfes ist das Erstellen und Verarbeiten einer Anfrage an den Endpunkt Text der OpenAI-Schnittstelle. Abb. 4.10 zeigt die vollständige Schnittstelle für die Klasse *TextPanel*.

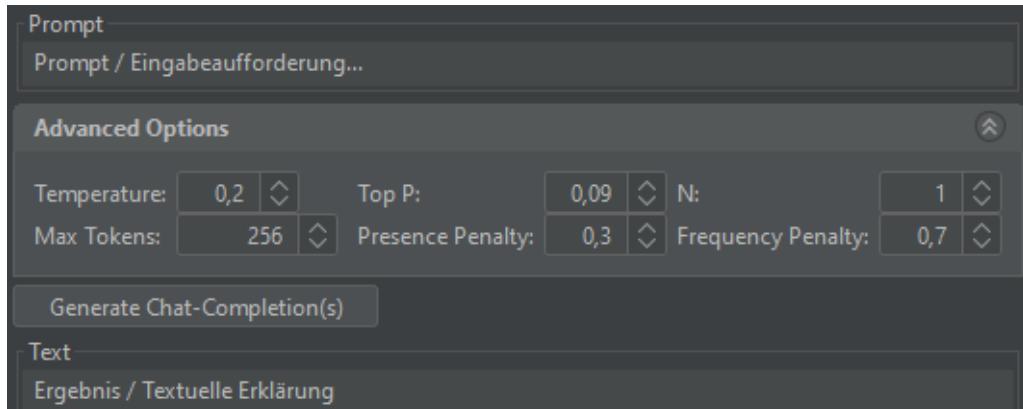


Abb. 4.10: Vollständige Schnittstelle für die Klasse *TextPanel*.

```

49  public void setGraphCode(GraphCode graphCode) {
50      // Graph Code verarbeiten
51      if(graphCode != null) {
52          String prompt = setUpPrompt(graphCode);
53          promptArea.setText(prompt);
54      } else {
55          promptArea.setText(null);
56      }
57  }
58
59  // Aufbereitung der Prompt (siehe FZ 2.3/I Abschnitt 4.3.1)
60  private String setUpPrompt(GraphCode graphCode) {...}
61
62  // Integration der Endpunkte (siehe FZ 2.3/I Abschnitt 4.3.2)
63  @Override
64  public void actionPerformed(ActionEvent e) {...}
65 }
```

Listing 4.11: TextPanel-Klasse (Letzter Teil)

Listing 4.11 zeigt den letzten Teil der Klasse *TextPanel* und die für diese Klasse wichtigen Methoden *setGraphCode*, *setUpPrompt* und *actionPerformed*. Die Methode *setGraphCode* hat die Verarbeitung eines Graph Codes zur Aufgabe und ruft hierfür wiederum die Methode *setUpPrompt* auf. Diese Methode ruft die Informationen in einem Graph Code ab und erstellt aus diesen eine Reihe an Textnachrichten, die zusammen eine Prompt darstellen. Das Ergebnis dieses Aufrufs wird dann in einem dafür vorgesehenen Textfeld angezeigt. In dieser Methode wird somit die

Transformation von Graph Codes vorgenommen. Die genauen Details der Implementierung dieser Methode werden daher in Abschnitt 4.3.1 behandelt. Schlussendlich hat die Methode *actionPerformed* die Erzeugung einer Erklärung und der damit verbundenen Erzeugung einer Anfrage an den Endpunkt Text zur Aufgabe. In dieser Methode wird somit die Integration des Endpunkts Text vorgenommen. Genauere Details dieser Methode zum Erstellen einer Anfrage an den Endpunkt Text werden in Abschnitt 4.3.2 behandelt.

Im Folgenden zeigt Listing 4.12 die Schnittstelle *ImagePanel*. Listing 4.12 verhält sich dabei analog zu den Listings 4.9 bis 4.11. Als solches ist die Schnittstelle *ImagePanel* im Aufbau seiner Benutzeroberfläche, mit Außnahme der erweiterten Optionen und dem *JLabel + Icon* zum Darstellen eines erzeugten Bildes im Vergleich zum Textfeld, identisch (siehe Abb. 4.8). Ein weiterer Unterschied besteht in der Methode *actionPerformed* beim Generieren einer visuellen Erklärung zu einem Graph Code. Hier wird zuerst eine spezielle, textuelle Erklärung generiert, die dann wiederum als Eingabe für den Endpunkt Bild zum Generieren einer visuellen Erklärung genutzt wird. Genauere Details zum Erstellen einer Anfrage an den Endpunkt Bild werden in Abschnitt 4.3.2 behandelt.

```

1 public class ImagePanel extends JPanel implements ActionListener {
2
3     // API-Schlüssel
4     private static String key;
5     // Anzahl an Bildern
6     private final JSpinner nSpinner;
7     // Größe des generierten Bildes
8     private final JComboBox<String> sizeComboBox;
9     // Textfeld für die Prompt
10    private final JTextArea promptArea;
11    // JLabel + Icon (für Bild)
12    private final JLabel picLabel;
13    // Textnachrichten
14    private List<ChatMessage> messages = new ArrayList<>();
15    // Referenz
16    private final ExplainerFrame reference;
17
18    public ImagePanel(ExplainerFrame reference) {
19        this.reference = reference;
20        key = System.getenv("OpenAI-Key");
21        // Layout: MigLayout
22        // Textfeld für die Prompt
23        promptArea = new JTextArea();
24        JScrollPane promptSP = new JScrollPane(promptArea);
25        // Erweiterte Optionen
26        JXTaskPane advancedOptions = new JXTaskPane("Advanced Options")
27        // Elemente der erweiterten Optionen
28        // umfassen JLabels + JSpinner für

```

```

29     // die Optionen: Anzahl an Bildern (N)
30     // und Größe des Bildes (Size) (siehe ...).
31     // Die JSpinner besitzen zudem
32     // Datenmodelle (SpinnerNumberModel's).
33     advancedOptions.add( Labels + JSpinner );
34     // Knopf zum Generieren eines Bildes
35     JButton generateImageButton = new JButton("Generate Image(s)");
36     // Interaktion (siehe actionPerformed)
37     generateImageButton.addActionListener(this);
38     // Label + Icon zum Anzeigen des erzeugten Bildes
39     picLabel = new JLabel();
40     // ... Constraints fürs Layout
41     add(promptSP, "cell 0 0, ...");
42     add(advancedOptions, "cell 0 1, ...");
43     add(generateImageButton, "cell 0 2, ...");
44     add(picLabel, "cell 0 3, ...");
45 }
46
47 public void setGraphCode(GraphCode graphCode) {
48     // Graph Code verarbeiten
49     if(graphCode != null) {
50         String prompt = setUpPrompt(graphCode);
51         promptArea.setText(prompt);
52     } else {
53         promptArea.setText(null);
54     }
55 }
56
57 // Aufbereitung der Prompt (siehe FZ 2.3/I Abschnitt 4.3.1)
58 private String setUpPrompt(GraphCode graphCode) {...}
59
60 // Integration der Endpunkte (siehe FZ 2.3/I Abschnitt 4.3.2)
61 @Override
62 public void actionPerformed(ActionEvent e) {...}
63 }
```

Listing 4.12: ImagePanel-Klasse

Damit sind beide Schnittstellen *TextPanel* und *ImagePanel* abschließend beschrieben und erklärt. Abb. 4.11 zeigt die vollständige Schnittstelle für die Klasse *ImagePanel*. Der Grundbereich (2), *ExplanationPanel*, ist damit abgeschlossen.

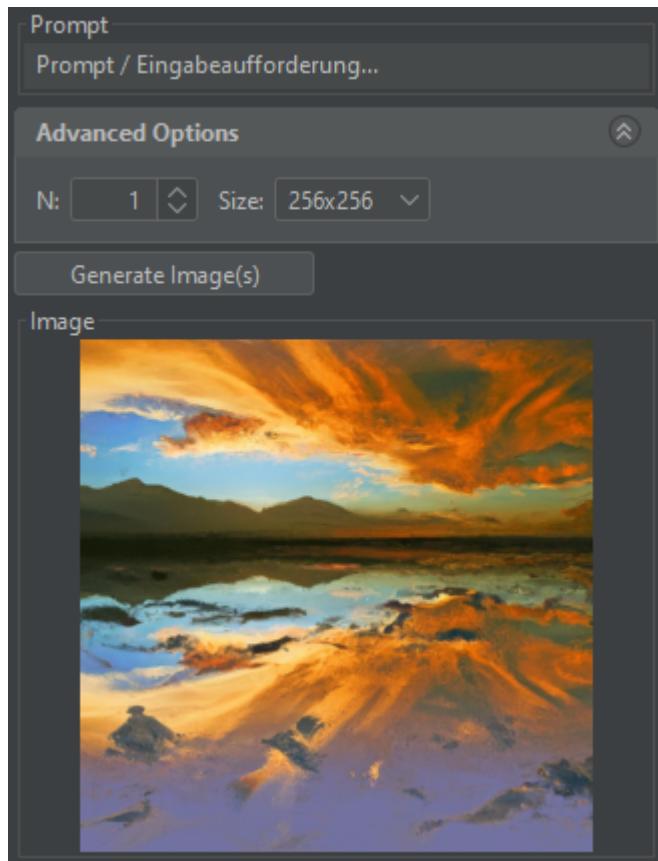


Abb. 4.11: Vollständige Schnittstelle für die Klasse *ImagePanel*.

Im weiteren Verlauf dieses Abschnitts wird in Listing 4.13 schlussendlich die Implementierung des letzten Grundbereichs (3), *ExplainerConsole*, behandelt und erklärt. Die allgemeine Aufgabe des Grundbereichs *ExplainerConsole* ist es wichtige Informationen über Prozesse des Programms zu loggen. Wichtige Informationen können Bestätigungen von Aktionen und Fehlermeldungen umfassen.

```

1 public class ExplainerConsole extends JPanel implements
2   ITextInsertListener {
3
4   // Textbereich für die Ausgabe
5   private final RSyntaxTextArea rSyntaxTextArea;
6
7   public ExplainerConsole(ExplainerConsoleModel model) {
8     // Layout: BorderLayout
9     if(model != null) {
10       model.addListener(this);
11     }
12     // Textbereich
13     rSyntaxTextArea = new RSyntaxTextArea();

```

```

13     RTextScrollPane rTextScrollPane = new RTextScrollPane(
14         rSyntaxTextArea);
15     // ToolBar
16     JToolBar toolBar = new JToolBar();
17     // Knopf zum Leeren des Textbereichs
18     URL url = getClass().getClassLoader().getResource("images/trash
19         .png");
20     ImageIcon trashIcon = new ImageIcon(Objects.requireNonNull(url)
21         );
22     JButton clearButton = new JButton(trashIcon);
23     clearButton.addActionListener(e -> {
24         if(model != null) {
25             model.clear();
26         }
27     });
28     toolBar.add(clearButton);
29     // Hinzufügen der Komponenten
30     add(toolBar, BorderLayout.NORTH);
31     add(rTextScrollPane, BorderLayout.CENTER);
32 }
33 // Einfügen von Text in die Konsole (Textbereich)
34 @Override
35 public void onInsert(String text) {...}
36 // Löschen des Textes in der Konsole (Textbereich "")
37 @Override
38 public void onClear() {...}

```

Listing 4.13: ExplainerConsole-Klasse

Die Benutzeroberfläche des dritten Grundbereichs, *ExplainerConsole*, besteht aus zwei nennenswerten Komponenten: Einem Textfeld für die angegebenen Informationen, sowie eine Toolbar mit Button zum Leeren des Textfeldes. Das Textfeld ist eine Instanz der Klasse *RSyntaxTextArea* und entstammt der gleichnamigen Abhängigkeit. Die beiden Methoden *onInsert* und *onClear* sind Teil des eigens entwickelten Interfaces *ITextInsertListener* und behandeln das Einfügen von Text in die Konsole, sowie das Leeren des Textfeldes der Konsole.

Damit sind alle Grundbereiche der Benutzeroberfläche des Programms abgeschlossen. Der Zusammenschluss dieser Grundbereiche wird, wie bereits am Anfang des Abschnitts angemerkt, durch die Methode *initComponents* in der Klasse *ExplainerFrame* vorgenommen (siehe Listing 4.1). Die genaue Implementierung dieser Methode kann in Listing 4.14 eingesehen werden und verdeutlicht den Zusammenschluss bzw. das Zusammenfügen der einzelnen Grundbereiche in das Fundament der gesamten Benutzungsschnittstelle.

```

1  private void initComponents() {
2      JPanel mainPanel = new JPanel();
3
4      JXMultiSplitPane multiSplitPane = new JXMultiSplitPane();
5      Split main = new Split();
6      main.setRowLayout(false);
7
8      Split top = new Split();
9      top.setWeight(0.8);
10
11     Leaf tL = new Leaf("top.left");
12     Leaf tR = new Leaf("top.right");
13
14     top.setChildren(tL, new Divider(), tR);
15
16     Leaf bottom = new Leaf("bottom");
17     bottom.setWeight(0.2);
18
19     main.setChildren(top, new Divider(), bottom);
20 }
21
22 // Layout
23 MultiSplitLayout layout = new MultiSplitLayout(main);
24 multiSplitPane.setLayout(layout);
25 // Grundbereich EditorGraphCode
26 editorGraphCode = new EditorGraphCode(this);
27 // Grundbereich ExplanationPanel
28 explanationPanel = new ExplanationPanel(this);
29 // Grundbereich ExplainerConsole
30 explainerConsoleModel = new ExplainerConsoleModel();
31 ExplainerConsole explainerConsole = new ExplainerConsole(
32     explainerConsoleModel);
33 // Hinzufügen
34 multiSplitPane.add(editorGraphCode, "top.left");
35 multiSplitPane.add(explanationPanel, "top.right");
36 multiSplitPane.add(explainerConsole, "bottom");
37 }
38 mainPanel.add(multiSplitPane, BorderLayout.CENTER);
39 }

```

Listing 4.14: Zusammenschluss der Grundbereiche

Besonders erwähnenswert für diese Methode ist die Komponente *JXMultiSplitPane*, die Teil der Abhängigkeit *SwingX* ist. Zu dieser gehören ebenfalls die Komponenten *MultiSplitLayout*, *Split*, *Divider* und *Leaf*, die den Aufbau beschreiben. Mithilfe der Komponente *JXMultiSplitPane* können auf einfache Art und Weise und ohne großen Aufwand Schnittstellen angelegt werden, die in ihrer Größe anpassbar sind und die

andernfalls nur durch aufwendige Verschachtelung mehrerer *JSplitPane*'s erzeugt werden könnten.

In diesem Abschnitt wurde die Implementierung der Benutzeroberfläche beschrieben. Hierfür wurden die Quellcodes für die in der Benutzeroberfläche verbauten Komponenten kompakt dargestellt und detailliert erläutert. Eine abschließende Gesamtabbildung der vollständigen Benutzeroberfläche samt Beispielen kann im Abschnitt 4.4 „Zusammenfassung“ in den Abbildg. 4.12 bis 4.14 eingesehen werden. Im folgenden Abschnitt wird die Interaktion mit und zwischen den in diesem Abschnitt beschriebenen Komponenten der Benutzeroberfläche thematisiert.

Interaktion mit der Benutzeroberfläche

In diesem Abschnitt wird die Implementierung der Interaktion zwischen Benutzern und Komponenten der Benutzeroberfläche genauer beschrieben. Darüber hinaus werden auch die Zusammenhänge zwischen den Komponenten genauer betrachtet. Dies umfasst Beschreibungen, wie die Komponenten gegenseitig miteinander interagieren und sich untereinander beeinflussen. Die Beschreibung der Interaktion konzentriert sich dabei im Wesentlichen auf die Anwendungsfälle UC-1.1, UC-1.2, UC-1.3 und UC-1.5. Dies sind die Anwendungsfälle, in welchen Benutzer durch die Interaktion mit Komponenten in der Benutzeroberfläche komplexe und bedeutende Abläufe in Bezug auf Graph Codes in Gang setzen, die wiederum Einfluss auf andere Komponenten haben. Allerdings sind von diesen Beschreibungen die verbleibenden Anwendungsfälle: UC-1.4 „Operation auswählen“, UC-1.7 „Erweiterte Optionen anpassen“ (Interaktionen werden bereits durch jeweilige Komponenten bereitgestellt), UC-1.6 „Erklärungstyp umschalten“ (Interaktion beeinflusst zwar die Komponente *ExplanationPanel*, umfasst im Wesentlichen aber nur das Entfernen und Hinzufügen von *ImagePanel* und *TextPanel*) ausgenommen. Der verbleibende Anwendungsfall UC-1.8 wird explizit in Abschnitt 4.3.2 behandelt.

Listing 4.15 zeigt das Steuerelement *ImportGraphCodesController*, welches mit dem Anwendungsfall UC-1.1 „Graph Code(s) importieren“ assoziiert ist. Das Steuerelement wird durch einen Klick auf den Knopf „Import Graph Code(s)“ initiiert und importiert ein oder mehrere aus einem Auswahldialog ausgewählte Graph Code Dateien in die Liste für Graph Codes (siehe Abb. 3.4 (5)).

```
1 public class ImportGraphCodesController implements ActionListener {
2
3     // Referenz
4     private final EditorGraphCode reference;
5
6     public ImportGraphCodesController(EditorGraphCode reference) {
7         this.reference = reference;
8     }
```

```

9
10    @Override
11    public void actionPerformed(ActionEvent e) {
12        // Auswahldialog vom System
13        JFileChooser fileChooser = new JFileChooser();
14        // Relatives Verzeichnis zum Projekt
15        fileChooser.setCurrentDirectory(new File("./graphcodes"));
16        // Filter initialisieren und konfigurieren
17        GraphCodeFilter graphCodeFilter = new GraphCodeFilter(".gc", ".gc (Graph Code)");
18        fileChooser.setFileFilter(graphCodeFilter);
19        // Datenmodell der Liste für Graph Codes (siehe Abb. 3.4 (5))
20        DefaultListModel<GraphCodeListElement> gclm = reference.
21            getGraphCodeListModel();
22        // Auswahldialog anzeigen
23        int openDialog = fileChooser.showOpenDialog(null);
24        // Ausgewählte Dateien öffnen
25        if(openDialog == JFileChooser.OPEN_DIALOG) {
26            File[] f = fileChooser.getSelectedFiles();
27            // Alle ausgewählten Dateien dem Datenmodell hinzufügen
28            Arrays.asList(f).forEach(file -> {
29                gclm.addElement(new GraphCodeListElement(file));
30            });
31        }
32    }

```

Listing 4.15: ImportGraphCodesController-Klasse

Der Auswahldialog wird, wie auch bereits in Abb. 3.6 identifiziert, durch die Komponente *JFileChooser* bereitgestellt. Durch eine Anpassung des Verzeichnisses zeigt der Dialog relativ auf das Verzeichnis des Projekts, in welchem Graph Code Dateien gespeichert sind. Weitere Anpassungen des Dialogs umfassen einen Dateifilter, sodass Benutzern nur Dateien mit der Endung „.gc“ angezeigt werden bzw. nur diese ausgewählt werden können, sowie eine mögliche Mehrfachauswahl von Dateien. Werden die ausgewählten Dateien geöffnet, so werden für alle ausgewählten Dateien Einträge (*GraphCodeListElement*) für die Liste geschaffen, die dann dem Datenmodell (*DefaultListModel*) der Liste hinzugefügt werden.

Damit entspricht der Ablauf der Aktionen in Listing 4.15 grundsätzlich den modellierten Abläufen im UML-Sequenzdiagramm, dargestellt in Abb. 3.13, für den Anwendungsfall UC-1.1.

Listing 4.16 zeigt das Steuerelement *RemoveSelectedGraphCodesController*, welches mit dem Anwendungsfall UC-1.2 „Graph Code(s) entfernen“ assoziiert ist. Das Steuerelement wird durch einen Klick auf den Knopf „Remove selected Graph Code(s)“ initiiert und entfernt die in der Liste ausgewählten Graph Codes.

```

1 public class RemoveSelectedGraphCodesController implements
2   ActionListener {
3
4   // Referenz
5   private final EditorGraphCode reference;
6
7   public RemoveSelectedGraphCodesController(EditorGraphCode
8     reference) {
9     this.reference = reference;
10 }
11
12 @Override
13 public void actionPerformed(ActionEvent e) {
14   // Liste für Graph Codes (siehe Abb. 3.4 (5))
15   JList<GraphCodeListElement> gcl = reference.getGraphCodeList();
16   // Datenmodell der Liste
17   DefaultListModel<GraphCodeListElement> dlm = reference.
18     getGraphCodeListModel();
19   // Indizes der ausgewählten Einträge in der Liste
20   int[] selIndices = gcl.getSelectedIndices();
21   if(selIndices.length > 0) {
22     // Einträge aus dem Datenmodell entfernen
23     for(int i = selIndices.length - 1; i >= 0; i--) {
24       dlm.removeElementAt(selIndices[i]);
25     }
26   }
27 }

```

Listing 4.16: RemoveSelectedGraphCodesController-Klasse

Voraussetzung für das erfolgreiche Entfernen von ein oder mehreren Graph Code Datei(en) ist, dass die Liste einen oder mehrere Graph Code(s) enthält bzw. diese von einem Benutzer ausgewählt worden sind. Hierfür wird die Liste und das Datenmodell der Liste benötigt. In einem ersten Schritt werden die Indizes von potentiell ausgewählten Einträgen der Liste abgefragt und geprüft, ob die Anzahl der ausgewählten Einträge größer null ist. Sofern dies der Fall ist, werden diese Indizes durchlaufen und dem Datenmodell mitgeteilt, dass der Eintrag am entsprechenden Index entfernt werden soll.

Damit entspricht der Ablauf der Aktionen in Listing 4.16 ebenfalls grundsätzlich den modellierten Abläufen im UML-Sequenzdiagramm, dargestellt in Abb. 3.14, für den Anwendungsfall UC-1.2.

Listing 4.17 zeigt das Steuerelement *GraphCodeSelectionListener*, welches mit dem Anwendungsfall UC-1.3 „Graph Code(s) auswählen“ assoziiert ist. Das Steuerelement wird durch die Auswahl eines oder mehrerer Elemente in der Liste initiiert

und beeinflusst daraufhin durch Initiierung weiterer Aktionen andere Benutzungsschnittstellen, indem Informationen an diese weitergegeben werden.

```
1 public class GraphCodeSelectionListener implements
2     ListSelectionListener {
3
4     // Referenz
5     private final ExplainerFrame reference;
6
7     public GraphCodeSelectionListener(ExplainerFrame reference) {
8         this.reference = reference;
9     }
10
11    @Override
12    public void valueChanged(ListSelectionEvent e) {
13        if (!e.getValueIsAdjusting()) {
14            EditorGraphCode egc = reference.getEditorGraphCode();
15            // Liste für Graph Codes (siehe Abb. 3.4 (5))
16            JList<GraphCodeListElement> gcl = egc.getGraphCodeList();
17            // Tabelle zur Darstellung eines ausgewählten Graph Codes
18            GraphCodeTable gct = egc.getGraphCodeTable();
19            // Ausgewähltes Element
20            GraphCodeListElement gcle = gcl.getSelectedValue();
21            JLabel nameLabel = egc.getGraphCodeName();
22            if (gcle != null) {
23                // Ausgewählter Graph Code
24                GraphCode gc = gcle.getGraphCode();
25                nameLabel.setText(gcle.getFileName());
26                // Graph Code an Tabelle delegieren
27                gct.setGraphCode(gc);
28                // Graph Code an ExplanationPanel delegieren
29                reference.getExplanationPanel().setGraphCode(gc);
30            } else {
31                // Kein ausgewählter Graph Code
32                gct.setGraphCode(null);
33                // Namen zurücksetzen
34                nameLabel.setText(null);
35            }
36        }
37    }
}
```

Listing 4.17: GraphCodeSelectionListener-Klasse

Voraussetzung für das erfolgreiche Auswählen von ein oder mehreren Graph Code Datei(en) ist, dass die Liste einen oder mehrere Graph Code(s) enthält. Ist ein Eintrag aus der Liste ausgewählt, so können andere Benutzungsschnittstellen über die Informationen in diesem Eintrag informiert und angepasst werden. Die Informationen eines Eintrages umfassen den aus der Datei gelesenen Graph Code und den

Dateinamen. Zu den angepassten Benutzungsschnittstellen gehören somit die Tabelle zur Darstellung des Graph Codes *GraphCodeTable* inklusive eines Textfeldes zur Anzeige des Dateinamens, sowie die Schnittstelle *ExplanationPanel* zur Erzeugung von Erklärungen. Die Informationen eines ausgewählten Eintrages werden hierbei jeweils an *GraphCodeTable* und *ExplanationPanel* delegiert, woraufhin diese in den entsprechenden Komponenten weiterverarbeitet werden.

Damit entspricht der Ablauf der Aktionen in Listing 4.17 ebenfalls grundsätzlich den modellierten Abläufen im UML-Sequenzdiagramm, dargestellt in Abb. 3.15, für den Anwendungsfall UC-1.3.

Listing 4.18 zeigt das Steuerelement *GraphCodeListMouseAdapter*. Dieses Steuerelement ist nicht explizit mit einem Anwendungsfall assoziiert und wird durch eine Interaktion der Maus mit der Liste initiiert. Ist diese Interaktion ein Doppelklick, so wird in der Liste geprüft, ob an der Position der Interaktion ein Eintrag vorliegt. Sofern dies der Fall ist, wird der Index dieses Eintrags bestimmt und der Eintrag aus dem Datenmodell *DefaultListModel* extrahiert. Dem Benutzer wird dann ein Dialog angezeigt, mit welchem dieser eine Umbenennung des Eintrags vornehmen kann.

```

1 public class GraphCodeListMouseAdapter extends MouseAdapter {
2
3     // Referenz
4     private final EditorGraphCode reference;
5     // Liste für Graph Codes
6     private final JList<GraphCodeListElement> graphCodeList;
7     // Datenmodell für Liste
8     private final DefaultListModel<GraphCodeListElement> dlm;
9
10    public GraphCodeListMouseAdapter(EditorGraphCode reference) {
11        this.reference = reference;
12        this.graphCodeList = reference.getGraphCodeList();
13        this.dlm = reference.getGraphCodeListModel();
14    }
15
16    @Override
17    public void mouseClicked(MouseEvent e) {
18        // Doppelklick.
19        if(e.getClickCount() == 2) {
20            // Index des gewählten Eintrages
21            int selIdx = graphCodeList.locationToIndex(e.getPoint());
22            if(!dlm.isEmpty()) {
23                // Ausgewähltes Element in Liste
24                GraphCodeListElement element = dlm.getElementAt(selIdx);
25                String rename = JOptionPane.showInputDialog(null,
26                    "New Name", "Rename GraphCode",
27                    JOptionPane.PLAIN_MESSAGE);
28                if(rename != null && !rename.isEmpty()) {
29                    // Graph Code umbenennen

```

```

30         element.setFileName(rename.trim());
31         reference.getGraphCodeName().setText(rename);
32     }
33 }
34 }
35 }
36 }

```

Listing 4.18: GraphCodeListMouseAdapter-Klasse

Neben dem Steuerelement *GraphCodeListMouseAdapter* wird der Liste noch ein weiteres Steuerelement hinzugefügt bzw. registriert. Dieses Steuerelement heißt *ListItemTransferHandler* und ermöglicht das Verschieben von Einträgen mittels Drag and Drop.

Listings 4.19 bis 4.22 und 4.24 zeigen das Steuerelement *GraphCodeCalculationController*, welches mit dem Anwendungsfall UC-1.5 „Operation ausführen“ assoziiert ist. Das Steuerelement wird durch einen Klick auf den Knopf *Execute* initiiert und führt die in Anwendungsfall UC-1.4 ausgewählte Operation auf den ausgewählten Graph Codes aus.

Listing 4.19 zeigt die für die Anwendung von Operationen auf Graph Codes notwendigen Komponenten. Dies umfasst die in Anwendungsfall UC-1.4 ausgewählte Operation *actionItem*, die in den folgenden Listings 4.20 bis 4.22 und 4.24 in einem Switch-Statement zur Differenzierung der auszuführenden Aktionen dienen wird, sowie eine Liste *selGraphCodes* der ausgewählten Einträge aus der Liste.

```

1 public class GraphCodeCalculationController implements
2   ActionListener {
3
4   // Referenz
5   private final ExplainerFrame reference;
6
7   public GraphCodeCalculationController(ExplainerFrame reference) {
8     this.reference = reference;
9   }
10
11  @Override
12  public void actionPerformed(ActionEvent e) {
13    EditorGraphCode egc = reference.getEditorGraphCode();
14    // Liste an Operationen
15    JComboBox<String> calcCmb = egc.getCalculationComboBox();
16    // Ausgewählte Operation
17    String actionItem = (String) calcCmb.getSelectedItem();
18    // Liste für Graph Codes
19    JList<GraphCodeListElement> graphCodeList = egc.
      getGraphCodeList();
20    // Datenmodell der Liste

```

```

20     DefaultListModel<GraphCodeListElement> dlm = egc.
21         getGraphCodeListModel();
22     // Liste an ausgewählten Einträgen aus der Liste
23     List<GraphCodeListElement> selGraphCodes = graphCodeList.
24         getSelectedValuesList();

```

Listing 4.19: GraphCodeCalculationController-Klasse

Listing 4.20 zeigt die Anwendung der Operation „Union“. Die Operation zur Berechnung der Vereinigung von Graph Codes wird bereits vom GMAF durch die Komponente *GraphCodeCollection* bereitgestellt. Um diese Operation nutzen zu können, muss in einem ersten Schritt aus der Liste an Einträgen eine Liste an Graph Codes geschaffen werden. Dies wird durch das Abbilden eines Eintrages auf einen Graph Code erreicht. Weiter wird die Liste in einen Vektor umgewandelt, da die Funktion der Vereinigung einen Vektor als Eingabe erwartet. Die Ausgabe der Vereinigung ist wiederum ein Graph Code, welcher dann in einen neu erstellten Eintrag eingebettet wird. Über einen Dialog kann dann ein Benutzer nach Anwendung der Operation dem Eintrag einen neuen Namen zuweisen. Der Eintrag wird dann dem Datenmodell der Liste hinzugefügt.

```

1     // Differenzierung zwischen den Operationen
2     switch(Objects.requireNonNull(actionItem)) {
3         case "Union":
4             Vector<GraphCode> vGcsUni = selGraphCodes.
5                 stream().map(GraphCodeListElement::getGraphCode).
6                 collect(Collectors.toCollection(Vector::new));
7             GraphCode union = GraphCodeCollection.getUnion(vGcsUni);
8             GraphCodeListElement unionListElement = new
9                 GraphCodeListElement(union, "Union");
10            String unionElementName = JOptionPane.showInputDialog(null,
11                "Set Name", "Union", JOptionPane.PLAIN_MESSAGE);
12            unionListElement.setFileName(unionElementName);
13            dlm.add(0, unionListElement);
14        break;

```

Listing 4.20: GraphCodeCalculationController-Klasse

Listing 4.21 zeigt die Anwendung der Operation „Subtraction“. Auch die Operation zur Berechnung der Subtraktion eines Graph Codes von einem anderen Graph Code wird durch die Komponente *GraphCodeCollection* bereitgestellt. Im Fall dieser Operation wird aus den ausgewählten Eintrag allerdings nur der erste und zweite Eintrag berücksichtigt. Ausgabe der Operation der Subtraktion ist ebenfalls ein Graph Code. Die verbleibenden Aktionen verhalten sich analog zur Operation der Vereinigung von Graph Codes.

```

1   case "Subtraction":
2       GraphCodeListElement gcLe1 = selGraphCodes.get(0);
3       GraphCodeListElement gcLe2 = selGraphCodes.get(1);
4       GraphCode subtract = GraphCodeCollection.subtract(gcLe1.
5           getGraphCode(), gcLe2.getGraphCode());
6       GraphCodeListElement subtractListElement = new
7           GraphCodeListElement(subtract, "Subtract");
8       String subtractElementName = JOptionPane.showInputDialog(
9           null,
10          "Set Name", "Subtraction", JOptionPane.PLAIN_MESSAGE);
11      subtractListElement.setFileName(subtractElementName);
12      dlm.add(0, subtractListElement);
13      break;

```

Listing 4.21: GraphCodeCalculationController-Klasse

Listing 4.22 zeigt die Anwendung der Operation „Similarities“. Die Operation zur Berechnung der Gemeinsamkeiten von Graph Codes wurde bereits in Algorithmus 3.1 als Teil der Modellierung in einem Pseudoalgorithmus vorgestellt. Die Implementierung dieses Algorithmus ist in Listing 4.23 einsehbar und wird ähnlich wie die Operationen zur Vereinigung und Subtraktion in die Klasse *GraphCodeCollection* eingebunden.

```

1   case "Similarities":
2       Vector<GraphCode> vGcsSim = selGraphCodes.
3           stream().map(GraphCodeListElement::getGraphCode).
4           collect(Collectors.toCollection(Vector::new));
5       GraphCode sim = GraphCodeCollection.getSimilarities(vGcsSim
6           );
7       GraphCodeListElement simListElement = new
8           GraphCodeListElement(sim, "Similarities");
9       String simElementName = JOptionPane.showInputDialog(null,
10          "Set Name", "Similarities", JOptionPane.PLAIN_MESSAGE);
11      simListElement.setFileName(simElementName);
12      dlm.add(0, simListElement);
13      break;

```

Listing 4.22: GraphCodeCalculationController-Klasse

```

1 public static GraphCode getSimilarities(Vector<GraphCode> gcs) {
2     // Ergebnis
3     GraphCode similarities = new GraphCode();
4     // Berechnung der Vereinigung
5     GraphCode union = getUnion(gcs);
6     // Kopie des Wörterbuchs
7     Vector<String> simDic = new Vector<>(union.getDictionary());
8     // Gemeinsame Elemente bestimmen
9     for(GraphCode gc : gcs) {

```

```

10     simDic,retainAll(gc.getDictionary());
11 }
12 // Neues Wörterbuch
13 Vector<String> dictionary = new Vector<>(simDic);
14 similarities.setDictionary(dictionary);
15 // Werte für Einträge bestimmen
16 for(GraphCode gci : gcs) {
17     for(String s : dictionary) {
18         for(String t : dictionary) {
19             int i = 0;
20             try {
21                 i = gci.getEdgeValueForTerms(s, t);
22             } catch (Exception x) {
23                 x.printStackTrace();
24             }
25             similarities.setValueForTerms(s, t, i);
26         }
27     }
28 }
29 return similarities;
30 }

```

Listing 4.23: Algorithmus zur Bestimmung der Gemeinsamkeiten von Graph Codes

Die für die Anwendung der Operation „Similarities“ notwendigen Schritte sind äquivalent zu den Schritten der Operation der Vereinigung, mit Ausnahme des Aufrufs der spezifischen Funktion, deren Schritte im Weiteren genauer beschrieben werden.

In einem ersten Schritt wird in Zeile 5 die Vereinigung aller Graph Codes berechnet. Ergebnis ist ein Graph Code mit einem Wörterbuch, welches alle Merkmale aller Graph Codes enthält. In einem weiteren Schritt werden von Zeile 11 bis 13 alle Graph Codes in einer Schleife durchlaufen. Bei jedem Durchlauf werden alle Elemente aus dem Wörterbuch der Vereinigung entfernt, die nicht in irgendeinem Wörterbuch der anderen Graph Codes vorhanden sind. Hierzu wird in Zeile 7 eine Kopie des Wörterbuchs erstellt und später verwendet. Übrig bleiben alle Elemente, die auch in den Wörterbüchern aller anderen Graph Codes vorkommen. Dieses Wörterbuch wird dann dem Ergebnis zugewiesen. Von Zeile 18 bis 30 werden dann in einem letzten Schritt für alle Graph Codes alle Elemente zeilen- und spaltenweise durchlaufen, um die Werte für diese Einträge zu bestimmen und dem Ergebnis zuzuweisen. Schlussendlich ist das Ergebnis dieser Funktion ein Graph Code, dessen Wörterbuch alle gemeinsamen Elemente bzw. Merkmale und Werte aller verarbeiteten Graph Codes enthält. Auf die Modellierung rückblickend ist die Implementierung dieses Algorithmus somit sehr ähnlich zu dem konzipierten Pseudoalgorithmus.

Listing 4.24 zeigt die Anwendung der Operation „Differences“. Die Operation zur Berechnung der Unterschiede von Graph Codes wurde bereits in Algorithmus 3.2 als

Teil der Modellierung in einem Pseudoalgorithmus vorgestellt. Die Implementierung dieses Algorithmus ist in Listing 4.25 einsehbar und wird ebenfalls in die Klasse *GraphCodeCollection* eingebunden.

```

1   case "Differences":
2       Vector<GraphCode> vGcsDiff = selGraphCodes.
3           stream().map(GraphCodeListElement::getGraphCode).
4           collect(Collectors.toCollection(Vector::new));
5       GraphCode difference = GraphCodeCollection.getDifferences(
6           vGcsDiff);
7       GraphCodeListElement differenceListElement = new
8           GraphCodeListElement(difference, "Difference");
9       String diffElementName = JOptionPane.showInputDialog(null,
10           "Set Name", "Differences", JOptionPane.PLAIN_MESSAGE);
11       differenceListElement.setFileName(diffElementName);
12       dlm.add(0, differenceListElement);
13   break;
14 }
```

Listing 4.24: GraphCodeCalculationController-Klasse

```

1 public static GraphCode getDifferences(Vector<GraphCode> gcs) {
2     // Ergebnis
3     GraphCode differences = new GraphCode();
4     // Berechnung der Vereinigung
5     GraphCode union = getUnion(gcs);
6     // Kopie des Wörterbuchs
7     Vector<String> simDic = new Vector<>(union.getDictionary());
8     // Gemeinsame Elemente bestimmen
9     for(GraphCode gc : gcs) {
10         simDic.addAll(gc.getDictionary());
11     }
12     // Differenz zwischen Vereinigung und Gemeinsamkeit
13     Sets.SetView<String> setView = Sets.difference(new LinkedHashSet
14         <>(union.getDictionary()), new LinkedHashSet<>(simDic));
15     // Neues Wörterbuch
16     Vector<String> dictionary = new Vector<>(setView);
17     differences.setDictionary(dictionary);
18     // Werte für Einträge bestimmen
19     for(GraphCode gci : gcs) {
20         for(String s : dictionary) {
21             for(String t : dictionary) {
22                 int i = 0;
23                 try {
24                     i = gci.getEdgeValueForTerms(s, t);
25                 } catch (Exception x) {
26                     x.printStackTrace();
27                 }
28             }
29         }
30     }
31 }
```

```

27         differences.setValueForTerms(s, t, i);
28     }
29 }
30 }
31 return differences;
32 }
```

Listing 4.25: Algorithmus zur Bestimmung der Unterschiede von Graph Codes

Die für die Anwendung der Operation „Differences“ notwendigen Schritte sind ebenfalls äquivalent zu den Schritten der Operation der Vereinigung, mit Ausnahme des Aufrufs der spezifischen Funktion.

Der in Listing 4.25 dargestellte Algorithmus zur Berechnung der Unterschiede von Graph Codes ist dabei nahezu identisch mit dem in Listing 4.23 dargestellten Algorithmus zur Berechnung der Gemeinsamkeiten. Folglich ist die Implementierung dieses Algorithmus ebenfalls sehr ähnlich zu dem entsprechend konzipierten Pseudoalgorithmus. Der bedeutende Unterschied besteht in der Art der Berechnung des Wörterbuchs. Genauer ist die Differenz aller Graph Codes die Vereinigung aller Graph Codes abzüglich der Gemeinsamkeiten aller Graph Codes. Die Berechnung dieser Differenz geschieht in Zeile 17 und nutzt hierfür die Abhängigkeit *Guava* von Google. Die Methode *difference* erwartet als Eingabe zwei Sets. Dies kann ein Problem darstellen, da ein Set nicht die Ordnung bzw. Reihenfolge seiner Elemente beibehält, diese aber für das Wörterbuch eines Graph Codes besonders wichtig ist. Daher werden in diesem Fall die Wörterbücher in ein *LinkedHashSet* übernommen, welche die Reihenfolge ihre Elemente beibehalten. Diese werden dann als Eingabe für die Funktion benutzt.

In diesem Abschnitt wurde die Implementierung der Interaktion zwischen Benutzern des GMAF und den Komponenten der Benutzeroberfläche genauer beschrieben. Es wurden die an dieser Interaktion beteiligten Steuerelemente vorgestellt und beschrieben, welche Aktionen diese Steuerelemente jeweils durchführen und wie diese die Verarbeitung oder die Benutzeroberfläche beeinflussen. Im nächsten Abschnitt werden die Erkenntnisse aus den Abschnitten dieses Forschungsziels zusammengefasst, diskutiert und eingeordnet.

4.2.2 Diskussion

Der Modellierung nach orientiert, wurde in Abschnitt 4.2.1.1 die Implementierung der Benutzungsschnittstelle genauer beschrieben. Dies umfasst eine genauere Beschreibung des in Bereiche aufgeteilten Aufbaus der Benutzeroberfläche. Die Implementierung der in diesen Bereichen enthaltenen Komponenten wurde dabei mit kompaktem Quellcode dokumentiert und beschrieben. Des Weiteren wurden in der Beschreibung dieser Komponenten Vergleiche bzw. Assoziationen zu den Anwendungsfällen und Mechanismen aus der Modellierung gemacht. In Abschnitt 4.2.1.1 wurde somit der Bereich *View* behandelt. Allerdings konnten die Methoden *setUpPrompt* und *actionPerformed* in Listings 4.11 und 4.12.

Anhand dieser Komponenten wurde dann in Abschnitt 4.2.1.2 die Implementierung der Interaktion zwischen den Benutzern des GMAF und den Komponenten der Benutzeroberfläche, sowie die Interaktion zwischen den Komponenten untereinander genauer beschrieben. Hierfür wurden die Komponenten für die Interaktion, auch Steuerelemente genannt, mit kompaktem Quellcode dokumentiert und beschrieben. In Abschnitt 4.2.1.2 wurde somit der Bereich *Controller* behandelt.

In diesem Abschnitt wurden allerdings die Methoden *setUpPrompt* und *actionPerformed*, jeweils aus den Listings 4.11 und 4.12 ausgelassen. Diese Methoden dienen zur Vorbereiten einer Prompt bzw. der Interaktion zur Erstellung einer Anfrage an einen Endpunkt und wurden explizit in diesem Forschungsziel ausgelassen und werden separat im zweiten Forschungsziel „Integration generativer KI in das GMAF“ behandelt.

4.3 FZ 2.3/I Integration generativer KI in das GMAF

In diesem Abschnitt wird die prototypische Proof-of-Concept Implementierung der „Integration der Endpunkte in das GMAF“ beschrieben und behandelt. In Kapitel 3 wurden Konzepte entwickelt und vorgestellt, die mögliche Vorgehensweisen zur Transformation von Graph Codes und Integration der Endpunkte der Schnittstelle von OpenAI beschreiben. Die Beschreibung der Implementierung der Transformation von Graph Codes in eine Prompt wird im Folgenden in Abschnitt 4.3.1 vorgenommen. Des Weiteren wird in Abschnitt 4.3.2 die Integration und Interaktion mit den Endpunkten der Schnittstelle von OpenAI zum Erstellen von Erklärungen behandelt und beschrieben.

4.3.1 Transformation von Graph Codes

In diesem Abschnitt wird die Implementierung der Transformation der in Graph Codes gespeicherten Informationen beschrieben. Im Folgenden zeigt Abb. 3.17 die Methode *setUpPrompt*, welche bereits in Listings 4.11 und 4.12 angedeutet wurde und welche die Transformation von Graph Codes umsetzt und die zur Erstellung von Anfragen notwendige Prompt aufbereitet bzw. erstellt.

```
1 private String setUpPrompt(GraphCode graphCode) {
2     // Aufzählung der Merkmale
3     String terms = graphCode.listTerms();
4     // Formate für Terme generieren
5     String formats = graphCode.getFormattedTerms();
6     // Textnachrichten (Prompt)
7     messages = new ArrayList<>();
8     // Reihe an Textnachrichten
9     // zum Erstellen eines Dialogs (Prompt)
10    // für den Endpunkt Text (siehe Dialog 4.1)
11    messages.add(...);
12    // Textnachrichten vereinen
13    return messages.stream().map(ChatMessage::getContent)
14        .collect(Collectors.joining("\n"));
15 }
```

Listing 4.26: Quellcode für Methode *setUpPrompt*.

Die für eine Anfrage notwendige Prompt wird durch einen Dialog dargestellt, der aus einer Reihe an Textnachricht besteht. Innerhalb der Methode *setUpPrompt* werden nun eine Reihe an Textnachrichten, die diesen Dialog erzeugen sollen, zusammengefügt. Hierbei inkorporieren einige dieser Textnachrichten in einem geeigneten Textformat Informationen des entsprechenden Graph Codes. Diese Informationen werden durch die Methoden *listTerms* und *getFormattedTerms* in ein geeignetes Textformat überführt und in die Textnachrichten eingebunden. Dialog 4.1 zeigt diese Reihe an Textnachrichten.

Textnachrichten für die Prompt

System: You are an assistant, who is able to generate cohesive textual explanations based on a collection of words.

System: The collection of words represents a dictionary. The dictionary contains so-called feature vocabulary terms. Additionally some of these terms are connected through a relationship. These relationships will be noted as $\langle i_t \rangle - \langle i_{t_1}, \dots, i_{t_n} \rangle$, where i_t denotes the index of a feature vocabulary term in the given collection.

System: Using these terms, we can create a coherent explanation that accurately describes the terms and its relations.

An example could be: The image shows water, the sky, and clouds. We can imagine a scene with clouds floating in the sky above.

User: The collections of words is as follows: $\langle \text{terms} \rangle$. Only respect these terms and its relations: $\langle \text{formats} \rangle$, and ignore all others. Do not create an explanation regarding the dictionary. Only generate a text containing the terms of the dictionary like in the example above.

Assistant: Based on the dictionary, here is a cohesive text containing the terms from the dictionary:

Die drei Parteien *System*, *Assistant* und *User* sind Teil des Dialogs. Die ersten drei Nachrichten stammen von der Partei *System* und vermitteln dem GPT-Modell, wie es sich zu verhalten und zu antworten hat. Darüber hinaus erhalten diese Nachrichten wichtige Informationen über die Struktur und Logik von Graph Codes mit, sowie die Form bzw. das Format in welchem diese Informationen dem Modell mitgeteilt werden. Die vierte Nachricht stammt von der Partei *User* und teilt dem Modell die spezifischen Informationen eines Graph Codes mit. Diese Informationen sind eine Aufzählung der Merkmale im Wörterbuch eines Graph Codes, generiert durch die Methode *listTerms*, sowie die Beziehungen zwischen den Merkmalen in Form eines geeigneten Textformats, generiert durch die Methode *getFormattedTerms*. Diese beiden Methoden werden durch die Klasse *GraphCode* bereitgestellt und sind somit dem Bereich *Model* zuzuweisen. Die Methode *listTerms* fusioniert die Merkmale des Wörterbuchs Mithilfe der Methode *join* der String-Klasse (*String.join(", ", dictionary)*). Die Methode *getFormattedTerms* ist hingegen wesentlich komplexer und kann in Listing 4.27 eingesehen werden. Die letzte Nachricht stammt von der Partei *Assistant* und stellt einen unvollständigen Satz dar. Die Aufgabe des Assistenten ist es auf den Nutzer zu antworten. Durch die ersten Mitteilungen zum System weiß das Modell wie und auf welche Art es dem Nutzer zu antworten hat. Der Assistent

antwortet dabei auf die vom Nutzer eingegebenen Informationen in vom System vorgegebener Art und vervollständigt den unvollständigen Satz mit einer Erklärung in Abhängigkeit der vom Nutzer angegebenen Informationen.

```
1 public String getFormattedTerms() {
2     List<String> formats = new ArrayList<>();
3     for(int row = 0; row < matrix.length; row++) {
4         int[] rowArr = matrix[row];
5         // MultiMap sortiert nach:
6         Multimap<Integer, String> mm = MultimapBuilder
7             // 1. Schlüssel (0,1,...)
8             .treeKeys()
9             // 2. Reihenfolge des Hinzufügens
10            .arrayListValues()
11            .build();
12        for(int column = 0; column < rowArr.length; column++) {
13            // Diagonale Einträge ignorieren
14            if(column != row) {
15                int entry = rowArr[column];
16                if(entry > 0) {
17                    // Wert des Eintrags ist Schlüssel für Merkmal(e)
18                    mm.put(rowArr[column], dictionary.get(column));
19                }
20            }
21        }
22        // Liste an Formaten in jetziger Reihe
23        List<String> formatsInRow = new ArrayList<>();
24        int finalRow = row;
25        // Map durchlaufen
26        mm.asMap().forEach((k, v) -> {
27            // Formate generieren und hinzufügen
28            if(k == 1) {
29                formatsInRow.add(String.format("<%s> - <%s>",
30                    finalRow + 1,
31                    v.stream()
32                        .map(s -> String.valueOf(dictionary.indexOf(s) + 1))
33                        .collect(Collectors.joining(", ")))
34            );
35            } else if(k > 1) {
36                formatsInRow.add(String.format("<%s> <%s> <%s>",
37                    finalRow + 1,
38                    k,
39                    v.stream()
40                        .map(s -> String.valueOf(dictionary.indexOf(s) + 1))
41                        .collect(Collectors.joining(", ")))
42            );
43        }
44    });
45    // Formate in Reihe fusionieren
46    if(!formatsInRow.isEmpty()) {
```

```

47         formats.add(String.join(", ", formatsInRow));
48     }
49 }
// Formate insgesamt fusionieren und zurückgeben
51 return !formats.isEmpty() ? String.join(", ", formats) : "";
52 }

```

Listing 4.27: Quellcode für Methode *getFormattedTerms*.

Um die Beziehungen zwischen Merkmalen in einem Graph Code geeignet darzustellen, werden mit dem Algorithmus *getFormattedTerms* Textformate generiert. Hierfür durchläuft der Algorithmus einzeln die Reihen der zweidimensionalen, quadratischen Matrix und sammelt für jeden einzigartigen Wert eine Reihe an Merkmalen. Hierbei werden allerdings diagonale Einträge und Einträge mit dem Wert null ausgelassen. Dies bedeutet, dass jeder einzigartige Wert in einer Reihe mit mehreren Merkmalen assoziiert sein kann. Zu diesem Zweck wird die Klasse *MultiMap* der Abhängigkeit *Guava* verwendet. Diese hält für einen jeweiligen Schlüssel, welcher ein einzigartiger Wert sein wird, eine Kollektion an Elementen, in diesem Fall Merkmale. Die Schlüssel werden numerisch sortiert (*treeKeys*) und die in die Kollektion hinzugefügten Merkmale werden nach Reihenfolge ihres Hinzufügens sortiert (*arrayListValues*). Abschließend werden die Einträge in der *MultiMap* durchlaufen und es werden Textformate generiert, indem die Werte in einem Format (*String.format()*) zusammengefügt werden. Anstatt allerdings die Merkmale zu fusionieren, werden nur die Indizes der Merkmale im Wörterbuch ermittelt und zusammengefügt und dann dem Format beigefügt. Ergebnis ist eine Menge an Formaten für eine Reihe in der Matrix, die dann am Ende fusioniert werden. Schlussendlich werden alle fusionierten Formate in einem einzigen Text zusammengefügt und als Ergebnis ausgegeben. Die Implementierung der Transformation der in der Matrix enthaltenen Informationen entspricht somit dem in der Modellierung in Abschnitt 3.3.1 entwickelten Konzept zur Transformation der Matrix.

Folgendes Beispiel zeigt die transformierten Informationen für einen echten Graph Code: Die Aufzählung des Vokabular lautet: „root-image, water, sky, cloud, building, skyscraper, dusk, body of water, lake, atmospheric phenomenon, horizon“. Die in Textformat überführten Einträge der Matrix des Graph Codes lauten: < 1 > – < 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 >. Eingefügt in Dialog 4.1 und als Anfrage an den Endpunkt Text gesendet, ergibt sich folgender Text als Antwort bzw. Erklärung:

The root-image captures a breathtaking view at dusk. The horizon is beautifully painted with hues of the setting sun, creating an atmospheric phenomenon that is a sight to behold. Dominating the scene are a skyscraper and other buildings, their silhouettes stark against the sky. Below, a body of water, perhaps a lake, mirrors the sky. The water, calm and serene, adds a sense of tranquility to the scene. Floating above are clouds, their edges tinged with the colors of dusk, adding depth to the sky.

Weitere Beispiele für Graph Codes, transformierte Informationen und daraus generierten Erklärungen können in Abschnitt 4.4 „Zusammenfassung“ eingesehen werden.

4.3.2 Integration der Endpunkte

In diesem Abschnitt wird die Implementierung der Integration der Endpunkte, sowie die Interaktion zur Erstellung einer Anfrage an diese Endpunkte behandelt. Der Ablauf der Erklärung der Integration und Interaktion mit den Endpunkten der Schnittstelle entspricht dabei dem Ablauf der Modellierung und Implementierung, da für die Anfrage an den Endpunkt Bild zuerst eine Anfrage an den Endpunkt Text notwendig ist.

Im Folgenden zeigt Listing 4.28 die Methode *actionPerformed*, die Teil der Klasse *TextPanel* ist und mit Anwendungsfall UC-1.8 assoziiert ist. Diese Methode realisiert die Interaktion und damit auch die Integration der notwendigen Komponenten zum Erstellen einer Anfrage an den Endpunkt Text.

```
1 public void actionPerformed(ActionEvent e) {
2     // Schnittstelle für OpenAI erw. mit Timeout (60s)
3     OpenAiService service = new OpenAiService(key);
4     ExecutorService executor = Executors.newSingleThreadExecutor();
5     // Task
6     Thread t = new Thread(() -> {
7         ChatCompletionRequest chatComplReq = ChatCompletionRequest.
8             builder()
9             .messages(messages)
10            .model("<model>") // gpt-3.5-turbo-16k oder gpt-4
11            // Anfrage parametrisieren
12            .temperature((Double) temperatureSpinner.getValue())
13            .topP((Double) topPSpinner.getValue())
14            .n((Integer) nSpinner.getValue())
15            .maxTokens((Integer) maxTokensSpinner.getValue())
16            .presencePenalty((Double) presencePenaltySpinner.getValue())
17            .frequencyPenalty((Double) frequencyPenaltySpinner.getValue())
18            // Anfrage finalisieren
19            .build();
20            // Fehlerbehandlung
21            try {
22                // Anfrage an Endpunkt Text
23                ChatCompletionResult chatComplRes = service.
24                    createChatCompletion(chatComplReq);
25                // Ergebnis in Textfeld anzeigen
26                textResponseArea.setText(chatComplRes.getChoices().get(0)
27                    .getMessage().getContent());
28            } catch (Exception ex) {
29                // Fehler in Konsole einfügen
30            }
31        }
32    }
33}
```

```

29     }
30 );
31 // Task ausführen und danach beenden
32 executor.execute(t);
33 executor.shutdown();
34 }

```

Listing 4.28: Anfrage an den Endpunkt Text.

In Zeile 3 wird eine Instanz der Schnittstelle *OpenAiService* initialisiert und mit der Variable *service* eingeführt. Für die Initialisierung ist ein gültiger API-Schlüssel notwendig. Des Weiteren wird ein Zeitüberschreitung von 60 Sekunden eingestellt. In Zeile 4 wird dann ein ExecutorService zum Ausführen eines Tasks bzw. einer Aufgabe initialisiert und mit der Variable *executor* eingeführt. Die Aufgabe wird das Erstellen einer Anfrage an den Endpunkt Text sein und wird hierfür durch einen Thread dargestellt. Die erste Aktion in diesem Thread ist das Erstellen der Komponente *ChatCompletionRequest*, welche eine Anfrage zur Chatvervollständigung darstellt und mit der Variable *chatComplReq* eingeführt wird. Das Erstellen dieser Komponente erfolgt über die Komponente *ChatCompletionRequestBuilder*, mit welcher eine Anfrage zur Chatvervollständigung Stück für Stück aufgebaut, parametrisiert und finalisiert werden kann. Die Variable *chatComplReq* enthält alle wichtigen Informationen zur Anfrage und kann im Weiteren über *service* an den Endpunkt Text gesendet werden. Das Ergebnis dieser Anfrage ist die Komponente *ChatCompletionResult*, welche die Antwort des Endpunktes an die Anfrage darstellt und mit der Variable *chatComplRes* eingeführt wird. *chatComplRes* wirkt somit als Container für die Antwortinformationen der Anfrage. Diese Informationen, im Folgenden einfach nur die textuelle Erklärung, kann dann aus diesem Container entnommen und im Weiteren in einem Textfeld angezeigt werden. Schlussendlich wird durch *executor* die Aufgabe bzw. der Thread ausgeführt und nach Ausführung beendet.

```

1 public void actionPerformed(ActionEvent e) {
2     // Schnittstelle für OpenAI erw. mit Timeout (60s)
3     OpenAiService service = new OpenAiService(key);
4     ExecutorService executor = Executors.newSingleThreadExecutor();
5     // Task
6     Thread t = new Thread(() -> {
7         ChatCompletionRequest chatComplReq = ChatCompletionRequest.
8             builder()
9             .messages(messages)
10            .model("<model>") // gpt-3.5-turbo-16k oder gpt-4
11            // Anfrage parametrisieren
12            .maxTokens(75)
13            // Anfrage finalisieren
14            .build();
15            // Fehlerbehandlung

```

```

16    try {
17        // Anfrage an Endpunkt Text
18        ChatCompletionResult chatComplRes = service.
19            createChatCompletion(chatComplReq);
20        String chatResp = chatComplRes.getChoices().get(0)
21            .getMessage().getContent();
22        CreateImageRequest imgReq = CreateImageRequest.builder()
23            // Textergebnis als Prompt verwenden
24            .prompt(chatResp)
25            // Anfrage parametrisieren
26            .n((Integer) nSpinner.getValue())
27            .size(String.valueOf(sizeComboBox.getSelectedItem()))
28            .responseFormat("url")
29            // Anfrage finalisieren
30            .build();
31        // Anfrage an Endpunkt Bild
32        ImageResult imageResult = service.createImage(imgReq);
33        // Url abgreifen
34        URL url = new URL(imageResult.getData().get(0).getUrl());
35        // Ergebnis als Icon anzeigen
36        picLabel.setIcon(new ImageIcon(url));
37    } catch (Exception ex) {
38        // Fehler in Konsole einfügen
39    });
40    // Task ausführen und danach beenden
41    executor.execute(t);
42    executor.shutdown();
43 }

```

Listing 4.29: Anfrage an den Endpunkt Bild.

Listing 4.29 zeigt das Erstellen einer Anfrage an den Endpunkt Bild. Für das Erstellen einer Anfrage an diesen Endpunkt ist zuerst eine Anfrage an den Endpunkt Text notwendig, um eine textuelle Erklärung, die als Bildbeschreibung für die visuelle Erklärung bzw. das zu erzeugende Bild fungieren soll, zu erzeugen. Die Implementierung dieser Anfrage verhält sich analog zu der in Listing 4.28. Allerdings bedarf es einer passenden Parametrisierung dieser Anfrage, da der Endpunkt Bild nur eine maximale Anzahl von 1000 Zeichen akzeptiert. Durch diese Parametrisierung kann sichergestellt werden, dass die Antwort der ersten Anfrage an den Endpunkt Text eine maximale Länge von 75 Token beträgt. Im nächsten Schritt erfolgt das Erstellen einer Anfrage *CreateImageRequest* durch die Komponente *CreateImageRequestBuilder* und wird mit der Variable *imgReq* eingeführt. Die Antwort der vorherigen Anfrage ist dabei Teil der Parametrisierung dieser Anfrage. Im nächsten Schritt wird die Anfrage über *service* an den Endpunkt Bild gesendet. Das Ergebnis dieser Anfrage ist die Komponente *ImageResult*, welche als Container für die Antwortinformationen wirkt. Eine dieser Information ist eine URL, die zum erzeugten Bild führt und in

einem JLabel als Icon angezeigt werden kann. Schlussendlich wird durch *executor* die Aufgabe bzw. der Thread ausgeführt und nach Ausführung beendet.

In Bezug auf die Listings 4.28 und 4.29 kann festgehalten werden, dass die in der Implementierung der Interaktion und Integration der Endpunkte Text und Bild verwendeten Komponenten den im Mechanismus (siehe Abb. 3.20) identifizierten Komponenten entsprechen. Ebenso entspricht der Ablauf der Aktionen in der umgesetzten Interaktion dem im Sequenzdiagramm (siehe Abb. 3.21) beschriebenem Ablauf.

4.3.3 Diskussion

Im ersten Forschungsziel wurden bereits beim Beschreiben der Implementierung die Funktionen *setUpPrompt* und *actionPerformed* der Klasse *TextPanel* bzw. *ImagePanel* angedeutet, aber aus Gründen der Zuordnung nicht näher beschrieben. In diesem FZ wurde die Implementierung dieser Methoden, wie auch im Rahmen der Modellierung in den Abschnitte 3.3.1 und 3.3.2 vorgesehen, genauer beschrieben.

In Abschnitt 4.3.1 wurde dabei die Implementierung der Transformation von Informationen in Graph Codes genauer beschrieben. Dies umfasst eine Zusammenstellung von Textnachrichten in einem Dialog, die die Prompt bzw. die Eingabe in den Endpunkt Text darstellen, sowie die Methoden *listTerms* und *getFormattedTerms* zum Transformieren der in Graph Codes enthaltenen Informationen. Die Implementierung dieser Methoden wurde dabei mit kompaktem Quellcode dokumentiert und beschrieben, sowie durch Beispiele verdeutlicht. In Abschnitt 4.3.1 wurde somit der Bereich *Model* behandelt.

Des Weiteren wurde in Abschnitt 4.3.2 die Implementierung der Interaktion und damit auch die Integration der Endpunkte Text und Bild der Schnittstelle von OpenAI genauer beschrieben. Dies umfasst das Initialisieren der Schnittstelle, das Erstellen und Parametrisieren der notwendigen Komponenten einer Anfrage, sowie das Senden und Verarbeiten der Anfrage an die Schnittstelle. Hierfür wurden die Methoden mit kompaktem Quellcode dokumentiert und beschrieben. In Abschnitt 4.3.2 wurde somit der verbleibende Teil des Bereichs *Controller* behandelt.

4.4 Zusammenfassung

In diesem Kapitel wurde die Implementierung behandelt und es wurde in Abschnitt 4.2 die prototypische Implementierung einer Benutzungsschnittstelle mit geeigneten Interaktionsmöglichkeiten zur Verarbeitung von Graph Codes und zur Erzeugung von textuellen und visuellen Erklärungen vorgestellt und beschrieben, sowie in Abschnitt 4.3 die prototypische Proof-of-Concept Implementierung von Funktionen zur Transformation von Graph Codes, sowie zur Interaktion und Integration der Endpunkte der Schnittstelle von OpenAI zur Erzeugung von textuellen und visuellen Erklärungen vorgestellt und beschrieben. Im Folgenden werden die wesentlichen Ergebnisse und Erkenntnisse der Implementierung nach den Forschungszielen der Implementierungsphase, wie nach der Methodik nach Nunamaker vorgesehen, aufgeschlüsselt und in Abschnitt 4.4.1 kurz zusammengefasst. Des Weiteren wird in Abschnitt 4.4.2 das weitere Vorgehen festgehalten und schlussendlich in Tab. 4.2 eine Übersicht des aktuellen Arbeitsstands in einer Tabelle dargestellt.

4.4.1 Gewonnene Erkenntnisse

In diesem Abschnitt werden die Erkenntnisse aus den Forschungszielen „FZ 1.3/I Erklärbarkeit von MMIR mittels generativer KI“ und „FZ 2.3/I Integration generativer KI in das GMAF“ zusammengefasst.

Im ersten Forschungsziel „FZ 1.3/I Erklärbarkeit von MMIR mittels generativer KI“ wurde eine prototypische Proof-of-Concept Implementierung der in Kapitel 3 im Rahmen der Modellierung entwickelten Konzepte zur Erklärbarkeit von MMIR mittels generativer KI beschrieben. Dies umfasst eine Beschreibung des Aufbaus der Benutzeroberfläche und der in ihr verbauten Komponenten. Des Weiteren wurde die Interaktion zwischen Benutzern des GMAF und diesen Komponenten, sowie die Interaktion zwischen diesen Komponenten thematisiert.

Im zweiten Forschungsziel „FZ 2.3/I Integration generativer KI in das GMAF“ wurde eine prototypische Proof-of-Concept Implementierung der in Kapitel 3 im Rahmen der Modellierung entwickelten Konzepte zur Integration generativer KI in das GMAF beschrieben. Dies umfasst eine Beschreibung der Funktionen zur Transformation der in Graph Codes gespeicherten Informationen, sowie eine Beschreibung der Interaktion mit und Integration von Endpunkten der Schnittstelle.

Die folgenden Abbildg. 4.12 bis 4.16 verdeutlichen die Erkenntnisse bzw. Ergebnisse der prototypischen Proof-of-Concept Implementierung anhand Screenshots. Abbldg. 4.12 und 4.13 zeigen dabei den Ausgangspunkt des Programms zur Erklärung von Graph Codes. Abb. 4.12 zeigt hierbei in der rechten Arbeitsfläche *ExplanationPanel* die Schnittstelle zur Erzeugung von visuellen Erklärungen bzw. Bildern für Graph Codes und Abb. 4.13 entsprechend die Schnittstelle zur Erzeugung von

textuellen Erklärungen. In keiner der beiden Abbildungen wurde zuvor ein Graph Code ausgewählt, wie es auch anhand des Platzhalters „No GraphCode selected!“ im rechten Teil der linken Arbeitsfläche bzw. dem *GraphCodeTable* erkennbar ist.

Abbildung. 4.15 und 4.16 zeigen daraufhin jeweils die Auswahl eines Graph Codes aus der Liste und wie diese Auswahl die anderen Komponenten der Schnittstelle beeinflusst, sowie eine für den entsprechend gewählten Erklärungstypen generierte Erklärung. Die beeinflussten Komponenten sind dabei *GraphCodeTable*, in welcher der ausgewählte Graph Code in Tabellenform dargestellt wird und *ExplanationPanel*, in welcher die aus den Informationen des Graph Codes erzeugte Prompt in einem Textfeld „Prompt“ dargestellt wird. Die aus dieser Prompt generierte Erklärung ist entsprechend im unteren Teil von *ExplanationPanel* in entweder einem Textfeld für die textuelle Erklärung, oder in einem Bild für die visuelle Erklärung dargestellt.

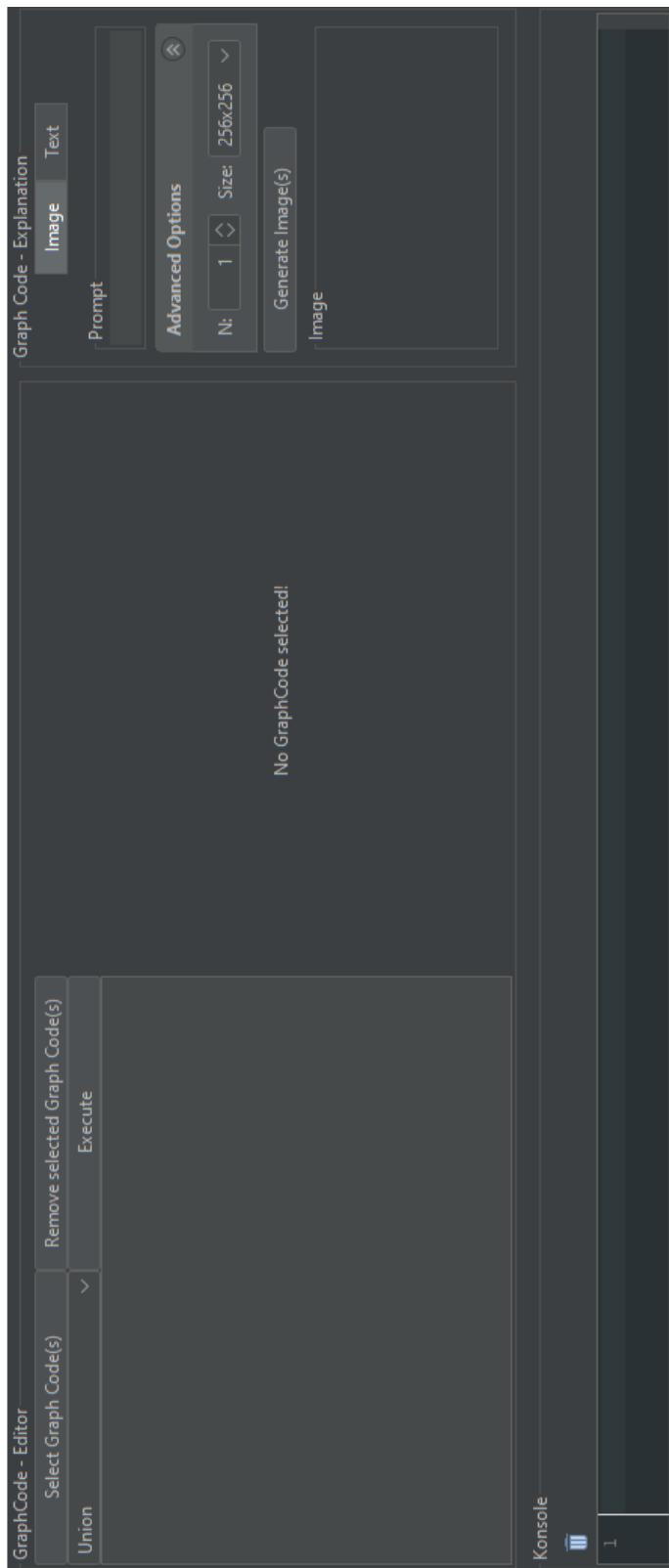


Abb. 4.12: Benutzungsschnittstelle zum Generieren einer visuellen Erklärung.

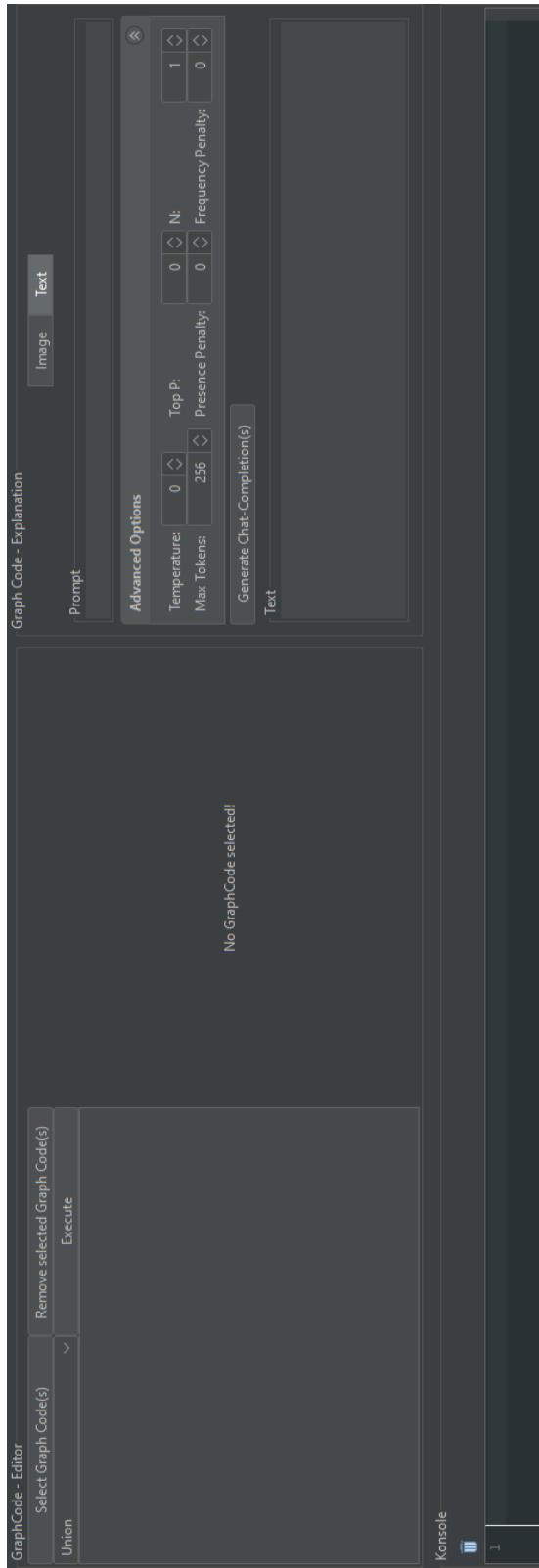


Abb. 4.13: Benutzungsschnittstelle zum Generieren einer textuellen Erklärung.

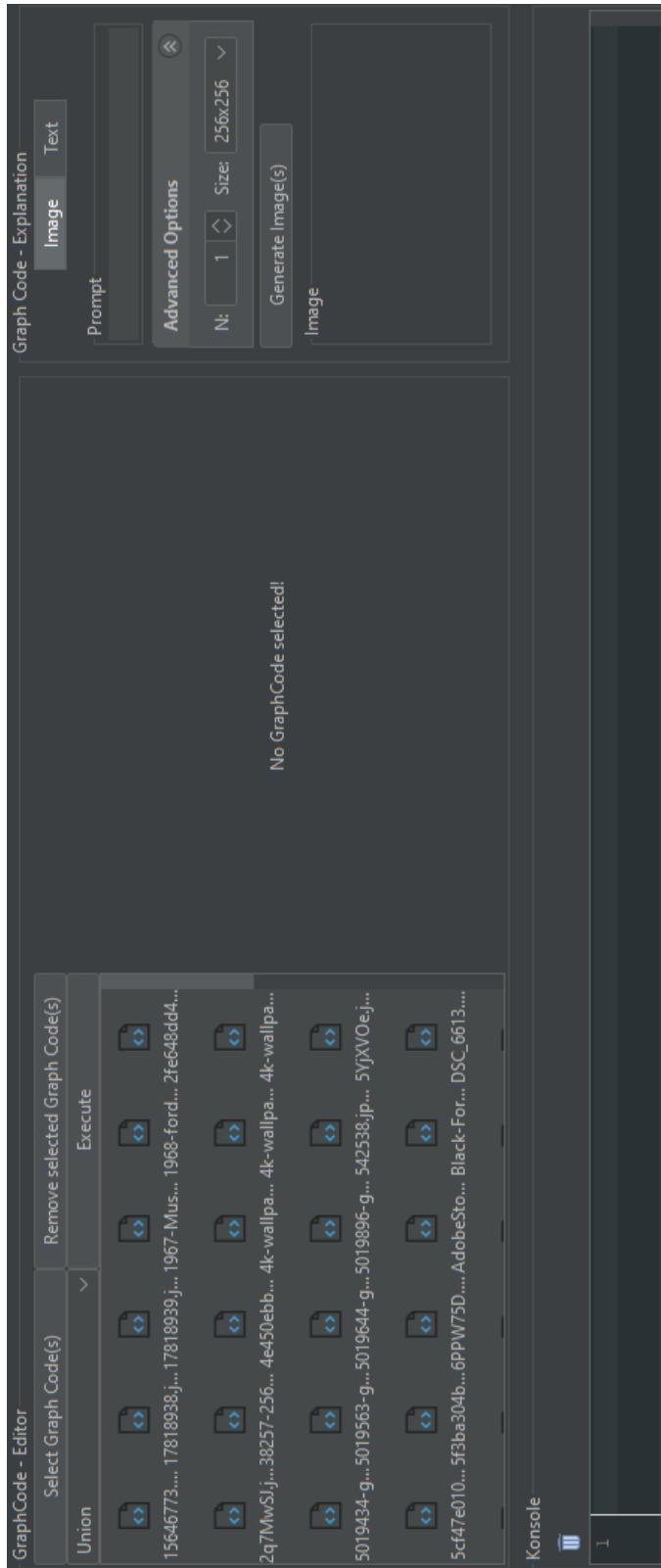


Abb. 4.14: Liste mit einer Reihe an importierten Graph Codes.

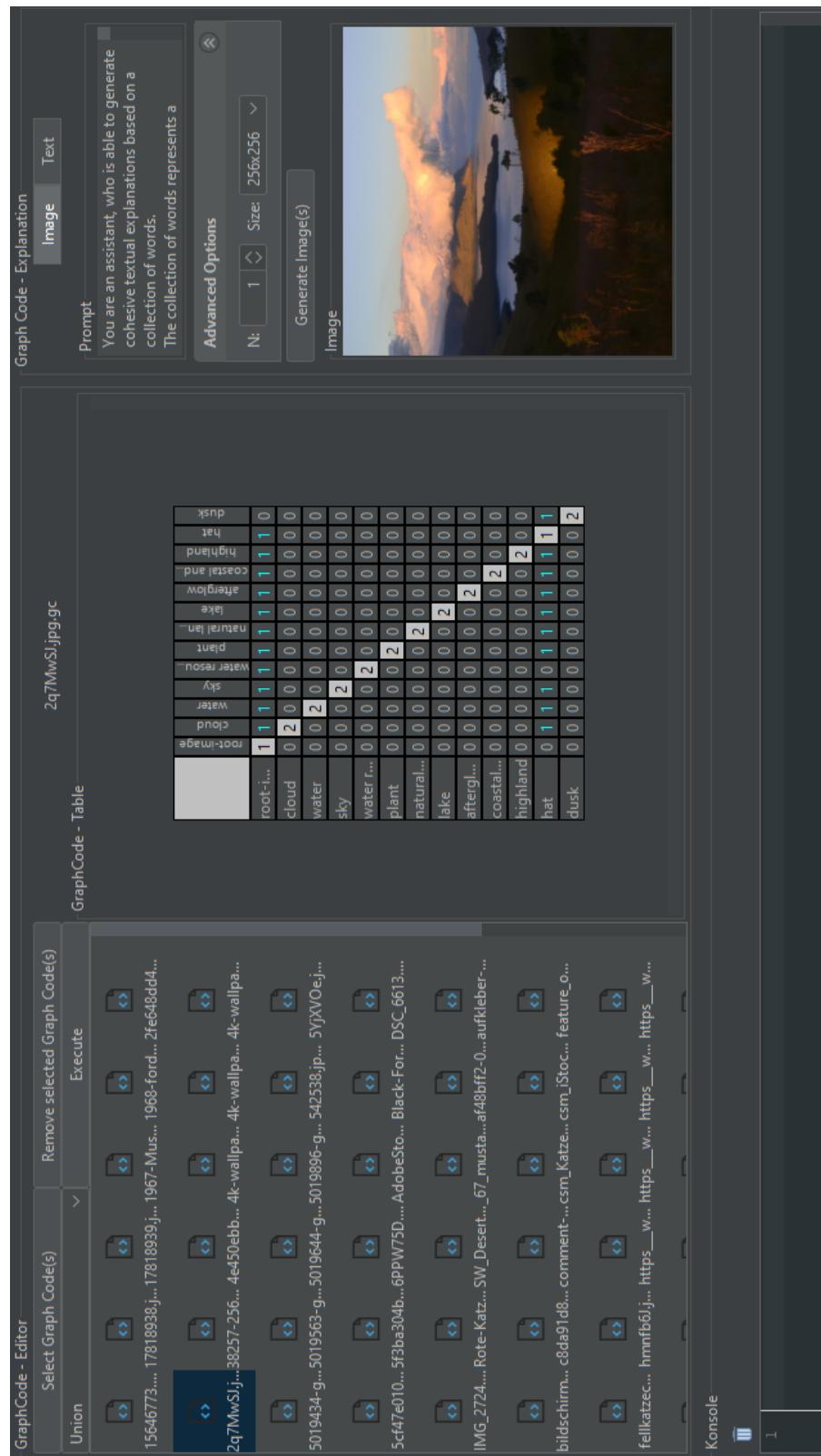


Abb. 4.15: Auswahl eines Graph Codes und eine generierte, visuellen Erklärung bzw. ein Bild.

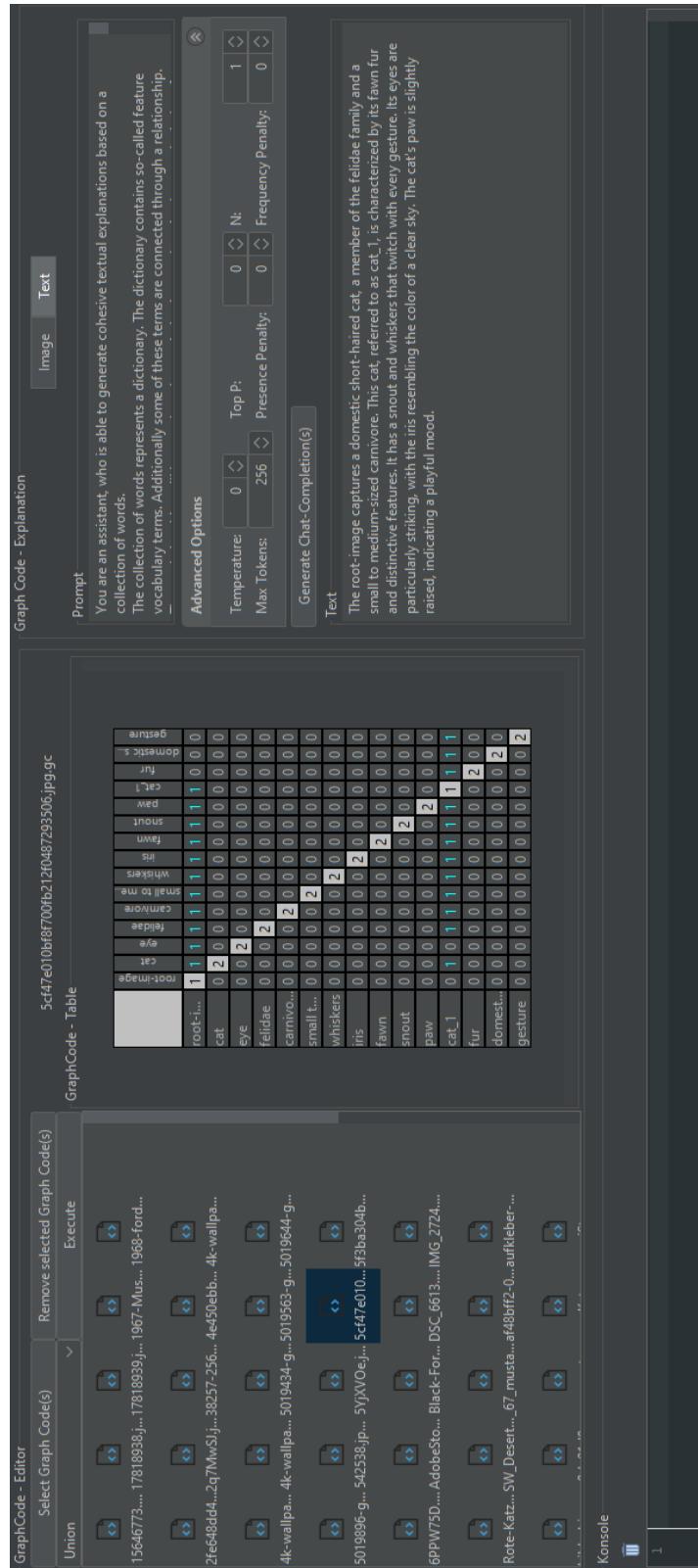


Abb. 4.16: Auswahl eines Graph Codes und eine generierte, textuelle Erklärung.

4.4.2 Weiteres Vorgehen

Im nachfolgenden Kapitel wird die Evaluierung der in diesem Kapitel beschriebenen, prototypischen Proof-of-Concept Implementierung vorgenommen. Genauer wird in Kapitel 5 in FZ 1.4/E eine Evaluierung, der im Rahmen der prototypischen Proof-of-Concept Implementierung umgesetzten Benutzungsschnittstelle und der Interaktion mit dieser, vorgenommen. In FZ 2.4/E wird dann eine Evaluierung, der im Rahmen der prototypischen Proof-of-Concept Implementierung umgesetzten Funktionen zur Transformation der in Graph Codes enthaltenen Informationen, vorgenommen.

Die nachfolgende Tabelle stellt eine Erweiterung der Tab. 3.3 dar und gibt eine Übersicht über den aktuellen Arbeitsstand nach Abschluss dieses Kapitels.

Tabelle 4.2: Tabelle zur Übersicht des aktuellen Arbeitsstands.

Forschungsziele		
FZ / OH	Beschreibung	Referenz
Erklärbarkeit von MMIR mittels generativer KI		
FZ 1.1/O	Recherche zur Erklärbarkeit von MMIR mittels generativer KI Grundlegende Technologien: <ul style="list-style-type: none"> • GMAF • MMFG • Graph Code 	Abschnitt 2.1.1 Abschnitt 2.1.2 Abschnitt 2.1.3
OH 1.1	Erste offene Herausforderung Systeme generativer KI und ein Überblick über aktuelle Systeme Diskussion und Auswahl von Systemen	OH 1.1 Abschnitt 2.1.4 Abschnitt 2.1.5
OH 1.2	Zweite offene Herausforderung	OH 2.1
FZ 1.2/TB	Modellierung der Erklärbarkeit von MMIR mittels generativer KI Erklärbarkeit durch generative KI → Behandlung der ersten offenen Herausforderung OH 1.1 Anwendungsfälle: <ul style="list-style-type: none"> • Textuelle Beschreibungen 	Abschnitt 3.2.1 Abschnitt 3.2.2
	Wireframes	Abschnitt 3.2.3
	Mechanismen	Abschnitt 3.2.4
	Sequenzdiagramme	Abschnitt 3.2.5

Forschungsziele		
FZ / OH	Beschreibung	Referenz
	→ Behandlung der zweiten offenen Herausforderung OH 1.2	
FZ 1.3/I	Implementierung der Erklärbarkeit von MMIR mittels generativer KI Benutzungsschnittstelle: <ul style="list-style-type: none">• Elemente der Benutzeroberfläche• Interaktion mit der Benutzeroberfläche	Abschnitt 4.2.1
FZ 1.4/E	Evaluierung der Erklärbarkeit von MMIR mittels generativer KI	
Integration generativer KI in das GMAF		
FZ 2.1/O	Recherche zur Integration generativer KI in das GMAF Aufzeigen der Integrationsmöglichkeiten von: <ul style="list-style-type: none">• Graph Codes Erste offene Herausforderung <ul style="list-style-type: none">• Systemen generativer KI Zweite offene Herausforderung	Abschnitt 2.2.1 OH 2.1 Abschnitt 2.2.2 OH 2.2
FZ 2.2/TB	Modellierung der Integration generativer KI in das GMAF Transformation von Graph Codes <ul style="list-style-type: none">• Transformation des Vokabulars• Transformation der Matrix• Anwendung von Graph Code Metriken → Behandlung der ersten offenen Herausforderung OH 2.1 Einbindung generativer KI in das GMAF → Behandlung der zweiten offenen Herausforderung OH 2.2	Abschnitt 3.3.1 Abschnitt 3.3.2
FZ 2.3/I	Implementierung der Integration generativer KI in das GMAF Transformation von Graph Codes Integration der Endpunkte	Abschnitt 4.3.1 Abschnitt 4.3.2

Forschungsziele		
FZ / OH	Beschreibung	Referenz
FZ 2.4/E	Evaluierung der Integration generativer KI in das GMAF	

5 Evaluierung

In diesem Kapitel werden die Ergebnisse der Forschungsziele der vorangegangenen Phasen

- Beobachtungsphase, Kapitel 2 „Stand der Wissenschaft und Technik“
- Theoriebildungsphase, Kapitel 3 „Modellierung“
- Implementierungsphase, Kapitel 4 „Implementierung“

im Vergleich zu den in Abschnitt 1.3 identifizierten, initialen Forschungsfragen in einer detaillierten Evaluierung überprüft. Ziel der Evaluierung ist es, Erkenntnisse zur Vollständigkeit, sowie Korrektheit der prototypischen Software für die Prototypen der Benutzungsschnittstelle(n) und Funktionen der Anwendung zu erlangen. Dieses Ziel ergibt sich aus der in diesem Kapitel gewählten Methodik der Evaluierung, sowie aus den Forschungszielen vom Typ Experiment und folgt dem Ansatz nach Nunamaker [25]. Zudem werden konkrete Ziele der Evaluierung, Auswahl von Methoden der Evaluierung, um diese Ziele zu erreichen, sowie die Vorbereitung, Durchführung und Auswertung von Experimenten nach den gewählten Methoden der Evaluierung, beschrieben. Mit diesem Ansatz werden alle Problembereiche im Vergleich zu den initialen Forschungszielen zufriedenstellend und ausführlich abgedeckt.

Die Gliederung des Kapitels lautet wie folgt: Abschnitt 5.1 konkretisiert die Ziele und zu beantwortenden Fragestellungen der Evaluierung. Des Weiteren wird für die Evaluierung eine geeignete Evaluierungsmethodik gewählt und erläutert. In Abschnitt 5.2 werden die in der Evaluierung durchzuführenden Experimente formuliert, sowie den entsprechenden Forschungszielen zugeordnet. In Abschnitte 5.3 und 5.4 werden jeweils die Vorbereitung, Durchführung und die Ergebnisse der jeweiligen Experimente beschrieben. In Abschnitt 5.5 werden in Abhängigkeit der in den Experimenten potentiell entdeckten Mängel, Fehlerbehandlungen beschrieben. In Abschnitt 5.6 werden dann die Ergebnisse der Evaluierung zusammengefasst und diskutiert.

5.1 Ziele der Evaluierung und geeignete Evaluierungsmethodik

Wesentliches Ziel der Evaluierung ist die Sicherstellung der Funktionalität der entwickelten, prototypischen Proof-of-Concept Implementierung. In diesem Abschnitt werden die konkreten Ziele der Evaluierung formuliert, auf deren Basis eine geeignete Evaluierungsmethodik ausgewählt wird. Konkrete Ziele lassen sich aus den Forschungszielen vom Typ Experiment ableiten. Folgende konkrete Ziele können

dabei formuliert werden: Bewertung der Vollständigkeit und Korrektheit der prototypischen Proof-of-Concept Implementierung in Bezug auf die in Kapitel 3 „Modellierung“ entwickelten Konzepte. Diese Ziele stellen die Grundlage für die Auswahl einer Evaluierungsmethodik dar. Durch Gruppierung und Zuweisung dieser Ziele zu einem Experiment können mehrere Ziele in einer Durchführung eines Experiments abgedeckt werden. Aus diesen festgelegten, konkreten Zielen ergibt sich folgendes Ziel der Evaluierung, für dessen Erreichen Experimente entworfen werden, die nach ihrer Durchführung Erkenntnisse bezüglich der Vollständigkeit und Korrektheit der prototypischen Proof-of-Concept Implementierung der Software zulassen.

In Abschnitt 2.3 wurden Methodiken vorgestellt, mit welchen eine systematische Evaluierung einer Anwendung möglich ist. Mittels dieser Methodiken können Aussagen und Bewertungen zur entwickelten Software vorgenommen werden. Es gilt aus der Sammlung dieser Methodiken eine geeignete Methodik zu wählen, die mit den in diesem Projekt begrenzten Ressourcen und Möglichkeiten zur Verfügung stehen und mit welcher möglichst viele formulierten Ziele erreicht werden können. Ein weiterer wichtiger Aspekt, den es bei der Auswahl eines geeigneten Evaluierungsverfahrens bzw. -methodik zu beachten gilt, ist, dass es sich bei der zu evaluierenden Software nicht um ein fertiges, einsetzbares Softwareprodukt handelt, sondern nur um eine prototypische Proof-of-Concept Implementierung. Da im Rahmen der Implementierung ein praktisch, benutzbarer Softwareprototyp entwickelt wurde, bietet es sich an, genau diesen Prototyp zur Evaluierung zu verwenden, sodass sich die Erkenntnisse und Ergebnisse der Evaluierung auf den aktuellsten Stand der Software und somit auch dieser Arbeit beziehen.

Für die Durchführung der Evaluierung kann auf einen Experten des Lehrgebiets Multimedia und Internetanwendungen zurückgegriffen werden. Aus den verfügbaren Evaluierungsmethodiken wird im Folgenden der Cognititve Walkthrough gewählt. Der Cognititve Walkthrough generiert keine Erkenntnisse bezüglich der Effektivität einer Anwendung, bietet sich jedoch trotzdem zur Evaluierung an, da dieser einfach, kostengünstig und mit limitierten Ressourcen durchführbar ist. Im Verfahren des Cognititve Walkthrough werden systematisch atomare Aktionen durchlaufen. Diese atomaren Aktionen sind Aufgaben, die ein Benutzer ausführen muss, um sein angestrebtes Ziel zu erreichen, und lassen sich aus den entsprechenden Interaktionen eines Benutzers mit dem System, festgehalten in Abschnitt 3.2.4 „Anwendungsfällen“, ableiten. Mit diesem Vorgehen wird sich versprochen, gezielt Erkenntnisse zu den Zielen „Bewertung der Vollständigkeit und Korrektheit“ zu erlangen. Andere umfangreichere Evaluierungsmethodiken sind im Rahmen dieser Arbeit nicht anwendbar aufgrund mangelnder zeitlicher, sowie personeller Ressourcen.

Im weiteren Verlauf dieses Kapitels wird ein Cognititve Walkthrough geplant, in seiner Durchführung beschrieben und schlussendlich anhand der gewonnenen Erkenntnisse ausgewertet. Ziel der Durchführung eines Cognititve Walkthrough ist es, neue

Erkenntnisse zur Funktionalität der entwickelten Anwendung durch Bewertung der Vollständigkeit und Korrektheit zu erlangen. Hierfür wird im nächsten Abschnitt eine Wahl und Zuweisung der in der Evaluierung durchzuführenden Experimente vorgenommen.

5.2 Wahl und Zuweisung durchzuführender Experimente

In Vorbereitung des Cognitive Walkthroughs werden aus den in Abschnitt 3.2.4 beschriebenen Anwendungsfällen zwei Experimente abgeleitet, deren Durchführung evaluiert werden soll und die durch Funktionen der Anwendung unterstützt werden. Diese Experimente lauten wie folgt:

1. **Exp. 1:** Interaktion mit Graph Codes.

Im Rahmen dieses Experiments interagiert der Experte unter Nutzung der Benutzungsschnittstelle der Anwendung mit Graph Codes. Interaktionen umfassen das Importieren, Entfernen und Verarbeiten von Graph Codes. Durch dieses Experiment werden die Anwendungsfälle UC-1.1 bis 1.6 abgedeckt.

2. **Exp. 2:** Erstellen einer Erklärung.

Im Rahmen dieses Experiments erstellt der Experte unter Nutzung der Benutzungsschnittstelle anhand eines vorher ausgewählten Graph Codes eine textuelle oder visuelle Erklärung. Wichtige Voraussetzung für dieses Experiments ist somit das erste Experiment **Exp. 1**. Durch dieses Experiment werden die Anwendungsfälle UC-1.6 bis 1.8 abgedeckt.

Experiment **Exp. 1** wird durch das Forschungsziel „FZ 1.4/E Erklärbarkeit von MMIR mittels generativer KI“ in Abschnitt 5.3 durchgeführt und beschrieben. Analog wird Experiment **Exp. 2** durch das Forschungsziel „FZ 2.4/E Integration generativer KI in das GMAF“ in Abschnitt 5.4 durchgeführt und beschrieben.

5.3 FZ 1.4/E Erklärbarkeit von MMIR mittels generativer KI

In diesem Abschnitt bzw. Forschungsziel wird das Experiment **Exp. 1** behandelt. In Abschnitt 5.3.1 wird die Vorbereitung des Cognitive Walkthroughs und in Abschnitt 5.3.2 die Durchführung des Cognitive Walkthroughs beschrieben. Schlussendlich werden in Abschnitt 5.3.3 dann die Ergebnisse der Durchführung, wie z.B. entdeckte Mängel festgehalten und Fehlerbehandlungen beschrieben.

5.3.1 Vorbereitung

Die Vorbereitung des Cognitive Walkthroughs beinhaltet den Aufbau einer adäquaten Kollektion an Graph Code Dateien. Für diese Graph Code Dateien ist wiederum eine geeignete Kollektion an Multimediainhalten notwendig. Das GMAF bietet hierfür bereits eine Kollektion an Multimediainhalten bzw. Graph Code Dateien in einem dafür dedizierten Ordner „/graphcodes“ an.

Ein wichtiger Teil für die Vorbereitung eines Cognitive Walkthroughs ist die Beschreibung der Aktionssequenz, die den Ablauf atomarer Aktionen auflistet, die ein Benutzer zur Lösung einer Aufgabe abarbeiten muss. Die Aktionssequenz für das Experiment **Exp. 1**.

1. Starten der Anwendung, Modul Explainer auswählen und starten.
2. Importieren von Graph Code(s) durch Mausklick auf den Knopf „Import Graph Code(s)“.
3. Auswahl von Graph Code(s) aus der Liste in der Benutzungsschnittstelle „GraphCode - Editor“.

Ab hier kann zwischen zwei weiteren Interaktionen in einer Fallunterscheidung differenziert werden.

4. Fall 1: Entfernen der ausgewählten Graph Code(s) durch Mausklick auf den Knopf „Remove selected Graph Code(s)“.
5. Fall 2: Ausführen einer Operation auf den ausgewählten Graph Code(s).
 - 5.1. Auswahl einer Operation in einer ein- und ausklappbaren Liste.
 - 5.2. Ausführen der ausgewählten Operation auf den ausgewählten Graph Codes durch Mausklick auf den Knopf „Execute“.
6. Ggf. Wiederholung der Schritte 2 bis 5 oder 3 bis 5.

Diese Aktionssequenz ist die Grundlage für die folgende Durchführung des Experiments **Exp. 1**. Hierbei wird jede atomare Aktion aus der Aktionssequenz durchgeführt und durch einen Expertennutzer analysiert.

Eine weitere, wichtige Aufgabe im Rahmen der Vorbereitung dieses Experiments ist die Beschreibung der Gruppe zu erwartender Anwender bzw. Benutzer. Aufgrund der Tatsache, dass die Anwendung „Explainer“ ein Modul bzw. Teil des GMAF ist, können Rückschlüsse auf mögliche Benutzergruppen getroffen werden. Daher wird angenommen, dass ein typischer Benutzer der Anwendung mindestens durchschnittlich erfahren mit der Bedienung von Computersystemen im Allgemeinen ist und darüber hinaus ebenfalls Kenntnisse über Funktionen und der Bedienung (kommerzieller) Programme bzw. Systeme zur Text- und Bildgenerierung besitzt. Zwecks der Validierung der Funktionalität des Moduls wird weiterhin angenommen, dass ein typischer Benutzer des Moduls nur über wenig Erfahrung im Umgang mit diesem Modul besitzt. Weitere, speziellere Kenntnisse werden nicht erwartet, oder vorausgesetzt.

Zum Abschluss der Vorbereitung werden eine Reihe an potentiellen Zielen festgehalten, die ein Benutzer des GMAF beim Start der Aufgabe bzw. einer Variation dieser erfüllen will. Eine Variation der ersten Aufgabe könnte hierbei das Bestimmen der Gemeinsamkeiten der Merkmale aus einer Kollektion an Multimediainhalten sein.

Bestimme die Gemeinsamkeiten der Merkmale aus einer Kollektion an Multimediainhalten.

- ↪ Starte die Benutzeroberfläche des Moduls Explainer.
- ↪ Wähle und importiere eine Reihe an Graph Codes aus der Kollektion aus.
- ↪ Wähle eine Operation, in diesem Fall Gemeinsamkeiten, aus und führe diese aus.

Damit ist die Vorbereitung des Cognitive Walkthroughs zur ersten Aufgabe abgeschlossen und in den folgenden Abschnitten werden die Durchführung, sowie die aus dieser Durchführung erschlossenen Ergebnisse vorgestellt, beschrieben und in einer Diskussion eingeordnet.

5.3.2 Durchführung

Der in dieser Durchführung beschriebene Cognitive Walkthrough wurde in einer Live-Video-Sitzung am 16.10.2023 durchgeführt. An dieser Sitzung nahmen Jens Nathan Andreß, der Autor dieser Arbeit und Prof. Dr.-Ing. Stefan Wagenpfeil, der Betreuer dieser Bachelorarbeit, teil. Anstelle eines formalen Cognitive Walkthroughs als Evaluierungsmethodik wird in der Durchführung eine informelle Variante des Cognitive Walkthroughs angewandt. Diese informelle Variante der Evaluierungsmethodik zeichnet sich dadurch aus, dass der Experte die in der Vorbereitung beschriebene Aktionssequenz schrittweise bearbeitet und das Verhalten eines Benutzers simuliert bzw. nachempfindet. Anders als bei einem formalen Cognitive Walkthrough mit ei-

nem vorher angefertigten Fragenkatalog für jede in der Aktionssequenz definierte Aktion, werden in dieser Durchführung die Aussagen des Expertennutzers zu den jeweiligen Aktionen informell schriftlich festgehalten. Diese Aussagen können insbesondere potentiell auftretende Probleme umfassen, mit denen sich ein nachempfundener Benutzer konfrontiert sehen könnte. In Bezug auf die bereits ohnehin schon eingeschränkten zeitlichen und personellen Ressourcen muss daher mit qualitativen Abzügen bei den durch diesen informellen Durchlauf gesammelten Erkenntnissen gerechnet werden. Nichtsdestotrotz können mit dieser informellen Variante und den begrenzten Ressourcen in dieser Arbeit, die in Abschnitt 5.1 definierten Ziele zufriedenstellend erreicht werden. Prof. Dr.-Ing. Stefan Wagenpfeil nahm während der Sitzung die Rolle des Experten an und imitierte das Verhalten eines Benutzers aus der in der Vorbereitung angenommenen Benutzergruppe und machte Anmerkungen zu den jeweils durchgeföhrten Aktionen aus der Aktionssequenz, die in der folgenden Beschreibung der Durchführung hervorgehoben werden. Zudem werden ggfs. ausgewählte Schritte der Aktionssequenz mit Bildschirmaufnahmen relevanter Ausschnitte der jeweiligen Benutzungsschnittstelle dokumentiert.

Die Durchführung des ersten Experiments **Exp. 1** beginnt mit der ersten Aktion in der Aktionssequenz, dem Starten der GMAF-Anwendung und dem Auswählen des Moduls „Explainer“. Der Experte merkt an, dass das Fenster, in welchem sich das Modul öffnet, aufgrund der in den Komponenten dargestellten, zu antizipierenden Inhalten zu klein ist. Eine mögliche Anpassung der Benutzungsschnittstelle wird im nachfolgenden Abschnitt 5.3.3 besprochen. Nach dem Starten des Moduls sieht sich der Benutzer mit der in Abb. 4.12 dargestellten Benutzungsschnittstelle konfrontiert.

In Schritt 2 der Aktionssequenz klickt der Benutzer durch einen Mausklick auf den Knopf „Import Graph Code(s)“. Daraufhin wird dem Benutzer ein systemweiter Auswahldialog zum Auswählen von Dateien, genauer Graph Code Dateien angezeigt. Dieser Dialog verweist bereits auf den in der Vorbereitung angemerkt, für die Kollektion an Graph Codes dedizierten Ordner im GMAF. Aus diesem Auswahldialog wählt der Benutzer ein oder mehrere Graph Code Datei(en) aus. Diese ausgewählten Graph Code Dateien werden dann in der Benutzungsschnittstelle „GraphCode - Editor“ einer Liste hinzugefügt und angezeigt. Der Experte stellt fest, dass dies der Erwartungshaltung eines Benutzers aus der in der Vorbereitung definierten Benutzergruppe entspricht. Durch diese Reihe an Aktionen wird der Anwendungsfall UC-1.1 abgedeckt.

In Schritt 3 der Aktionssequenz wählt der Benutzer aus der Liste in der Benutzungsschnittstelle „GraphCode - Editor“ ein oder mehrere Graph Code(s) aus. Die Auswahl eines Graph Codes in der Liste wird dem Benutzer standardmäßig durch das visuelle Hervorheben des ausgewählten Elements bestätigt (siehe Abb. 4.7). Darüber hinaus werden durch die Auswahl eines Graph Codes aus der Liste andere Benutzungsschnittstellen beeinflusst, die dem Benutzer die Wirkung der Auswahl

verdeutlichen. Der Experte stellt fest, dass dies der Erwartungshaltung eines Benutzers aus der in der Vorbereitung definierten Benutzergruppe entspricht. Durch diese Aktion wird der Anwendungsfall UC-1.3 abgedeckt.

In Schritt 4 der Aktionssequenz klickt der Benutzer durch einen Mausklick auf den Knopf „Remove selected Graph Code(s)“. Daraufhin werden die bereits in der Liste ausgewählten Graph Codes aus dieser entfernt. Der Experte stellt fest, dass dies der Erwartungshaltung eines Benutzers aus der in der Vorbereitung definierten Benutzergruppe entspricht. Durch diese Aktion wird der Anwendungsfall UC-1.2 abgedeckt.

In Schritt 5 der Aktionssequenz führt der Benutzer eine Operation auf den in der Liste ausgewählten Graph Codes aus. Hierfür trifft der Benutzer in einer Unteraktion eine Auswahl bezüglich der auszuführenden Operation auf den ausgewählten Graph Codes. Der Experte stellt fest, dass die Benutzungsschnittstelle zur Auswahl der Operation nur die Namen der Operationen auflistet, es aber nicht erkenntlich ist, dass es sich dabei um Operationen auf Graph Codes handelt (siehe Abb. 4.4). Es ist daher möglich, dass dem Benutzer nicht ersichtlich ist, dass es sich dabei um eine Operation auf Graph Codes handelt. Eine mögliche Anpassung der Benutzungsschnittstelle wird im nachfolgenden Abschnitt 5.3.3 besprochen. In einer weiteren Unteraktion klickt der Benutzer durch einen Mausklick auf den Knopf „Execute“. Daraufhin wird die ausgewählte Operation auf den in der Liste ausgewählten Graph Codes ausgeführt. Der Experte hält fest, dass die Ergebnisse der ausgeführten Operationen augenscheinlich korrekt sind und dies der Erwartungshaltung eines Benutzers aus der in der Vorbereitung definierten Benutzergruppe entspricht. Durch diese Aktionen werden die Anwendungsfälle UC-1.4 und UC-1.5 abgedeckt.

Im nächsten Abschnitt werden die aus dieser Durchführung gewonnenen Erkenntnisse aufgezählt, diskutiert und mögliche Anpassungen der Benutzungsschnittstelle besprochen.

5.3.3 Ergebnisse und Diskussion

Das Ergebnis des im vorigen Abschnitts durchgeföhrten Experiments unter der Anwendung des Cognitive Walkthroughs als Evaluierungsmethodik ist eine Liste an Mängeln, die in der prototypischen Software entdeckt werden konnten. In diesem Abschnitt werden die entdeckten Mängel aufgezählt, beschrieben und es wird jeweils ein Lösungsansatz vermerkt, mit welchem der entsprechende Mangel abgestellt werden kann.

Mangel 1: Das Fenster, in welchem die Benutzungsschnittstelle dargestellt wird, ist zu klein.

Der Mangel kann behoben werden, indem das Fenster für die Benutzungsschnittstelle

beim Starten des Moduls maximiert dargestellt wird.

Mangel 2: Dem Benutzer ist nicht ersichtlich, dass es sich bei den in der Auswahlliste angebotenen Elementen um gleichnamige Operationen auf Graph Codes handelt.

Der Mangel kann behoben werden, indem der Auswahlliste an der ersten Position ein Element „Graph Code Operation“ hinzugefügt wird.

Nach Durchführung des Experiments und dem Beschreiben der identifizierten Mängel kann festgehalten werden, dass die getesteten Funktionen der Anwendung vollständig und korrekt umgesetzt wurden. Als solches wurden beide formulierten Ziele, die Ziele der Vollständigkeit und der Korrektheit zufriedenstellend erfüllt bzw. abgedeckt.

5.4 FZ 2.4/E Integration generativer KI in das GMAF

In diesem Abschnitt bzw. Forschungsziel wird das Experiment **Exp. 2** behandelt. In Abschnitt 5.4.1 wird die Vorbereitung des Cognitive Walkthroughs und in Abschnitt 5.4.2 die Durchführung des Cognitive Walkthroughs beschrieben. Schlussendlich werden in Abschnitt 5.4.3 dann die Ergebnisse der Durchführung, wie z.B. entdeckte Mängel festgehalten und Fehlerbehandlungen beschrieben.

5.4.1 Vorbereitung

Die Vorbereitung des Cognitive Walkthroughs beinhaltet den Aufbau einer adäquaten Kollektion an Graph Code Dateien. Für diese Graph Code Dateien ist wiederum eine geeignete Kollektion an Multimediainhalten notwendig. Das GMAF bietet hierfür bereits eine Kollektion an Multimediainhalten bzw. Graph Code Dateien in einem dafür dedizierten Ordner „/graphcodes“ an.

Ein wichtiger Teil für die Vorbereitung eines Cognitive Walkthroughs ist die Beschreibung der Aktionssequenz, die den Ablauf atomarer Aktionen auflistet, die ein Benutzer zur Lösung einer Aufgabe abarbeiten muss. Die Aktionssequenz für das Experiment **Exp. 2**.

1. Starten der Anwendung, Modul Explainer auswählen.
2. Importieren von Graph Code(s) durch Mausklick auf den Knopf „Import Graph Code(s)“.
3. Auswahl eines Graph Codes aus der Liste in der Benutzungsschnittstelle „GraphCode - Editor“.
4. Erklärungstyp (Image / Text) in der Benutzungsschnittstelle „GraphCode - Explanation“ auswählen.
5. Anfrage mittels erweiterten Optionen parametrisieren.
6. Generieren einer Erklärung durch Mausklick auf den Knopf „Generate ...“.

Diese Aktionssequenz ist die Grundlage für die folgende Durchführung des Experiments **Exp. 2**. Hierbei wird jede atomare Aktion aus der Aktionssequenz durchgeführt und analysiert.

Eine weitere, wichtige Aufgabe im Rahmen der Vorbereitung dieses Experiments ist die Beschreibung der Gruppe zu erwartender Anwender bzw. Benutzer. Aufgrund der Tatsache, dass die Anwendung „Explainer“ ein Modul bzw. Teil des GMAF ist, können Rückschlüsse auf mögliche Benutzergruppen getroffen werden. Daher wird angenommen, dass ein typischer Benutzer der Anwendung mindestens durchschnittlich erfahren mit der Bedienung von Computersystemen im Allgemeinen ist

und darüber hinaus ebenfalls Kenntnisse über Funktionen und der Bedienung (kommerzieller) Programme bzw. Systeme zur Text- und Bildgenerierung besitzt. Zwecks der Validierung der Funktionalität des Moduls wird weiterhin angenommen, dass ein typischer Benutzer des Moduls nur über wenig Erfahrung im Umgang mit diesem Modul besitzt. Weitere, speziellere Kenntnisse werden nicht erwartet, oder vorausgesetzt.

Zum Abschluss der Vorbereitung werden eine Reihe an potentiellen Zielen festgehalten, die ein Benutzer des GMAF beim Start der Aufgabe erfüllen will.

Generiere eine Erklärung zu einem ausgewählten Graph Code.

- ↪ Starte die Benutzeroberfläche des Moduls Explainer.
- ↪ Wähle und importiere eine Reihe an Graph Codes aus der Kollektion aus.
- ↪ Wähle einen Graph Code, zu welchem eine Erklärung generiert werden soll, aus.
- ↪ Wähle einen Erklärungstyp (Image / Text) aus.
- ↪ Starte die Generierung einer Erklärung.

Damit ist die Vorbereitung des Cognitive Walkthroughs zur ersten Aufgabe abgeschlossen und in den folgenden Abschnitten werden die Durchführung, sowie die aus dieser Durchführung erschlossenen Ergebnisse vorgestellt, beschrieben und in einer Diskussion eingeordnet.

5.4.2 Durchführung

Der in dieser Durchführung beschriebene Cognitive Walkthrough wurde in einer Live-Video-Sitzung am 16.10.2023 durchgeführt. An dieser Sitzung nahmen Jens Nathan Andreß, der Autor dieser Arbeit und Prof. Dr.-Ing. Stefan Wagenpfeil, der Betreuer dieser Bachelorarbeit, teil. Anstelle eines formalen Cognitive Walkthroughs als Evaluierungsmethodik wird in der Durchführung eine informelle Variante des Cognitive Walkthroughs angewandt. Diese informelle Variante der Evaluierungsmethodik zeichnet sich dadurch aus, dass der Experte die in der Vorbereitung beschriebene Aktionssequenz schrittweise bearbeitet und das Verhalten eines Benutzers simuliert bzw. nachempfindet. Anders als bei einem formalen Cognitive Walkthrough mit einem vorher angefertigten Fragenkatalog für jede in der Aktionssequenz definierte Aktion, werden in dieser Durchführung die Aussagen des Expertennutzers zu den jeweiligen Aktionen informell schriftlich festgehalten. Diese Aussagen können insbesondere potentiell auftretende Probleme umfassen, mit denen sich ein nachempfundener Benutzer konfrontiert sehen könnte. In Bezug auf die bereits ohnehin schon eingeschränkten zeitlichen und personellen Ressourcen muss daher mit qualitativen Abzügen bei den durch diesen informellen Durchlauf gesammelten Erkenntnissen

gerechnet werden. Nichtsdestotrotz können mit dieser informellen Variante und den begrenzten Ressourcen in dieser Arbeit, die in Abschnitt 5.1 definierten Ziele zufriedenstellend erreicht werden. Prof. Dr.-Ing. Stefan Wagenpfeil nahm während der Sitzung die Rolle des Experten an und imitierte das Verhalten eines Benutzers aus der in der Vorbereitung angenommenen Benutzergruppe und machte Anmerkungen zu den jeweils durchgeführten Aktionen aus der Aktionssequenz, die in der folgenden Beschreibung der Durchführung hervorgehoben werden. Zudem werden ggfs. ausgewählte Schritte der Aktionssequenz mit Bildschirmaufnahmen relevanter Ausschnitte der jeweiligen Benutzungsschnittstelle dokumentiert.

Die ersten Schritte 1 bis 3 der Aktionssequenz verhalten sich analog zu denen in **Exp. 1**. Als solches werden die in diesen Schritten gewonnenen Erkenntnisse im nachfolgenden Abschnitt 5.4.3 nicht berücksichtigt.

In Schritt 4 der Aktionssequenz wählt der Benutzer in der Benutzungsschnittstelle „GraphCode - Explanation“ einen Erklärungstypen, „Image“ oder „Text“ aus. Daraufhin wird dem Benutzer, in Abhängigkeit des jeweiligen Erklärungstyps, jeweils eine spezifische Benutzungsschnittstelle zum Erstellen einer entsprechenden Erklärung für einen ausgewählten Graph Code angezeigt (siehe Abbildg. 4.12 und 4.13). Der Experte hat zwei Anmerkungen in Bezug auf die Benutzungsschnittstelle, speziell auf das Textfeld „Generated Prompt“, in welchem die generierte Prompt dargestellt wird: 1. Das Textfeld sollte nicht editierbar sein. Dies kann den Anschein erwecken, dass der Inhalt der Prompt in der Benutzungsschnittstelle modifiziert und angepasst werden kann, wobei diese höchstens parametrisiert, nicht aber der Inhalt angepasst werden kann. Der Experte merkt zudem an, dass 2. das Textfeld unterhalb der Antwort und nicht oberhalb der Antwort platziert sein sollte, da dies zu Verwirrung beim Benutzer führen könnte (siehe Abb. 4.9). Durch diese Aktion wird der Anwendungsfall UC-1.6 abgedeckt.

In Schritt 5 der Aktionssequenz wird die Anfrage, deren Grundlage die generierte Prompt ist, durch den Benutzer mit den in der Benutzungsschnittstelle „Advanced Options“ angebotenen Optionen parametrisiert (siehe Abb. 4.9). Der Experte merkt an, dass in beiden Benutzungsschnittstellen *ImagePanel* und *TextPanel* die erweiterte Option zur Auswahl der Anzahl zu generierenden Erklärungen sinnfrei ist, da immer nur eine Erklärung angezeigt wird. Weiterhin vermerkt der Experte, dass grundsätzlich erweiterte Optionen im Rahmen der erwarteten Benutzergruppe eine untergeordnete Rolle spielen und die Ergebnisse der Parametrisierung nicht ohne weiteres und sicher nicht mit den in dieser Arbeit vorhandenen Ressourcen validiert werden können. Schlussendlich hält der Experte fest, dass die Parametrisierung einer Anfrage augenscheinlich korrekt ist und dies der Erwartungshaltung eines Benutzers entspricht. Durch diese Aktion wird der Anwendungsfall UC-1.7 abgedeckt.

In Schritt 6 der Aktionssequenz klickt der Benutzer durch einen Mausklick auf den

Knopf „Generate ...“. Daraufhin wird durch das Senden der Anfrage an den für den ausgewählten Erklärungstypen entsprechenden Endpunkt eine Erklärung zu einem ausgewählten Graph Code erzeugt. Der Experte merkt an, dass es keine visuelle Rückmeldung zum Prozess der Generierung einer Erklärung gibt und der Benutzer somit im Unklaren ist, ob der Knopf überhaupt funktioniert. Eine mögliche Anpassung der Benutzungsschnittstelle wird im nachfolgenden Abschnitt 5.4.3 besprochen. Durch diese Aktion wird der Anwendungsfall UC-1.8 abgedeckt.

Im nächsten Abschnitt werden die aus dieser Durchführung gewonnenen Erkenntnisse aufgezählt, diskutiert und mögliche Anpassungen der Benutzungsschnittstelle besprochen.

5.4.3 Ergebnisse und Diskussion

Das Ergebnis des im vorigen Abschnitts durchgeföhrten Experiments unter der Anwendung des Cognitive Walkthroughs als Evaluierungsmethodik ist eine Liste an Mängeln, die in der prototypischen Software entdeckt werden konnten. In diesem Abschnitt werden die entdeckten Mängel aufgezählt, beschrieben und es wird jeweils ein Lösungsansatz vermerkt, mit welchem der entsprechende Mangel abgestellt werden kann.

Mangel 1: Das Textfeld für die generierte Prompt ist editierbar. Die Möglichkeit den im Textfeld „Generated Prompt“ dargestellten Text anzupassen, kann bei einem Benutzer den Anschein erwecken, dass der Inhalt der Prompt und somit auch die Anfrage angepasst werden kann. Dies ist allerdings nicht der Fall und die Logik der Anfrage hat keinen Bezug zum Textfeld.

Dieser Mangel kann behoben werden, indem die Editierbarkeit des Textfeldes „Generated Prompt“ deaktiviert wird.

Mangel 2: Das Textfeld für die generierte Prompt ist oberhalb der Fläche für die erzeugten Erklärungen. Das Textfeld „Generated Prompt“ hat nur informativen Charakter und sollte unterhalb der Antwort- bzw. Erklärungsfläche platziert sein. Ist das Textfeld oberhalb der Erklärungsfläche platziert, erweckt das ebenfalls den Anschein, dass dieses Textfeld eine Eingabefläche sein könnte.

Dieser Mangel kann mit einer Anpassung der Benutzeroberfläche behoben werden, genauer durch das Verschieben des Textfeldes unterhalb der Erklärungsfläche.

Mangel 3: Die in beiden Benutzungsschnittstellen verfügbare, erweiterte Option „N“, die die Anzahl der zu generierenden Erklärungen in einer Anfrage an einen Endpunkt darstellt, hat keine Wirkung, da in den Benutzungsschnittstellen jeweils zu einem Zeitpunkt immer nur eine Erklärung dargestellt werden kann.

Dieser Mangel kann behoben werden, indem in den Benutzungsschnittstellen *Image-*

Panel oder *TextPanel* jeweils in der Fläche für die generierte Erklärung, in Abhängigkeit der gewählten Anzahl, mehrere Tabs hinzugefügt werden, die jeweils eine generierte Erklärung darstellen.

Mangel 4: Keine visuelle Rückmeldung zum Status der Verarbeitung einer Anfrage. Dies führt dazu, dass der Benutzer sich nicht im Klaren darüber ist, ob eine Anfrage erfolgreich stattgefunden hat bzw. ob der Knopf „Generate ...“ überhaupt eine Wirkung hat.

Dieser Mangel kann behoben werden, indem die Darstellung des Mauszeigers für den Zeitraum, in welchem die Anfrage erstellt und versendet, aber noch keine Antwort empfangen wurde, verändert wird. Eine geeignete Darstellung für den Mauszeiger wäre eine Sanduhr, die dem Benutzer verdeutlicht, dass im Hintergrund des Programms ein Prozess begonnen hat bzw. im Gange ist.

Darüber hinaus schlägt der Experte eine Modifizierung bzw. Erweiterung der Benutzungsschnittstelle vor. Diese Erweiterung sieht das Gegenüberstellen des Graph Codes, der Original-Datei und der generierten Erklärung vor. Durch diese Erweiterung soll eine verbesserte Übersicht bzw. eine Verdeutlichung des direkten Bezugs einer Erklärung zur Original-Datei, z.B. bei einem Bild und einer visuellen Erklärung, entstehen. Eine solche Erweiterung könnte einfach durch das Hinzufügen einer weiteren Benutzungsschnittstelle für die Original-Datei in den rechten Bereich der linken Arbeitsfläche erfolgen.

Nach Durchführung des Experiments und dem Beschreiben der identifizierten Mängel kann festgehalten werden, dass die getesteten Funktionen der Anwendung vollständig und korrekt umgesetzt wurden. Als solches wurden beide formulierten Ziele, die Ziele der Vollständigkeit und der Korrektheit zufriedenstellend erfüllt bzw. abgedeckt.

5.5 Anpassung der prototypischen Proof-of-Concept Implementierung anhand der Evaluierungsergebnisse

In den vorhergehenden Abschnitte 5.3 und 5.4 wurden die in Abschnitt 5.1 formulierten Experimente behandelt. Dies umfasste die Vorbereitung und Durchführung der Experimente, sowie das Sammeln und Diskutieren der in der Durchführung gesammelten Ergebnisse. Ergebnis der Abschnitte 5.3.3 und 5.4.3 sind wiederum mögliche Anpassungen der Benutzungsschnittstelle, um die in den Durchführungen identifizierten Mängel zu beheben. In diesem Abschnitt werden diese möglichen Anpassungen näher beschrieben und ggfs. mit Bildschirmaufnahmen dokumentiert.

In der Durchführung des ersten Experiments konnten zwei Mängel identifiziert werden. Diese Mängel betrafen die Größe des Fensters der Benutzungsschnittstelle und die Intuitivität der Auswahl von Operationen auf Graph Codes aus einer Auswahlliste. Diese beiden Mängel erfordern keine aufwendigen Anpassungen und können durch das Hinzufügen oder Anpassen von ein bis zwei Zeilen Quellcode umgesetzt werden. Im Fall der Fenstergröße erfolgt die Anpassung in der Methode *initFrame* in der Klasse *ExplainerFrame*, in welcher die Zeile *setExtendedState(getExtendedState() / MAXIMIZED_BOTH)*; als Quellcode hinzugefügt werden muss. Im Fall der Auswahlliste erfolgt die Anpassung durch das Hinzufügen des Elements „GraphCode Operation“ an erster Position in der Auswahlliste. Darüber hinaus wird im Steuer-element *GraphCodeCalculationController* eine Kontrolle eingefügt, die die Auswahl dieses Elements abdeckt bzw. unterbindet.

In der Durchführung des zweiten Experiments konnten vier Mängel identifiziert werden. Diese Mängel betrafen die Editierbarkeit der Textfelder, die die generierten Prompts darstellen, die Positionierung der Textfelder relativ zur Antwortfläche, den fehlenden Nutzen der Auswahl der Anzahl zu generierenden Erklärungen und die Visualisierung bzw. Verdeutlichung der im Hintergrund ablaufenden Arbeitsprozesse zur Generierung von Erklärungen. Das Beheben des ersten Mangels erfordert keine aufwendige Anpassung des Quellcodes. Durch das Hinzufügen der Zeile *promptArea.setEditable(false);* im Quellcode, genauer dem Quellcode der Klassen *ImagePanel* und *TextPanel*, kann die Editierbarkeit der Textfelder deaktiviert werden. Das Beheben des zweiten Mangels erfordert eine Anpassung des Layouts, sowie der Positionierung der Textfelder im Layout. In den Benutzungsschnittstellen bzw. Klassen *ImagePanel* und *TextPanel* rutschen alle anderen Elemente eine Zelle nach oben, während die Textfelder der letzten Zelle zugewiesen werden. Entsprechend werden in der Spezifikation des Layouts auch die Beschränkungen der Zellen umgeordnet. Die aus diesen Umordnung resultierenden Benutzungsschnittstellen können in Abb. 5.1 eingesehen werden.

Das Beheben des dritten Mangels erfordert eine umfassendere Anpassung der Benutzeroberfläche, sowie eine Anpassung der Verarbeitung der generierten Ergebnisse.

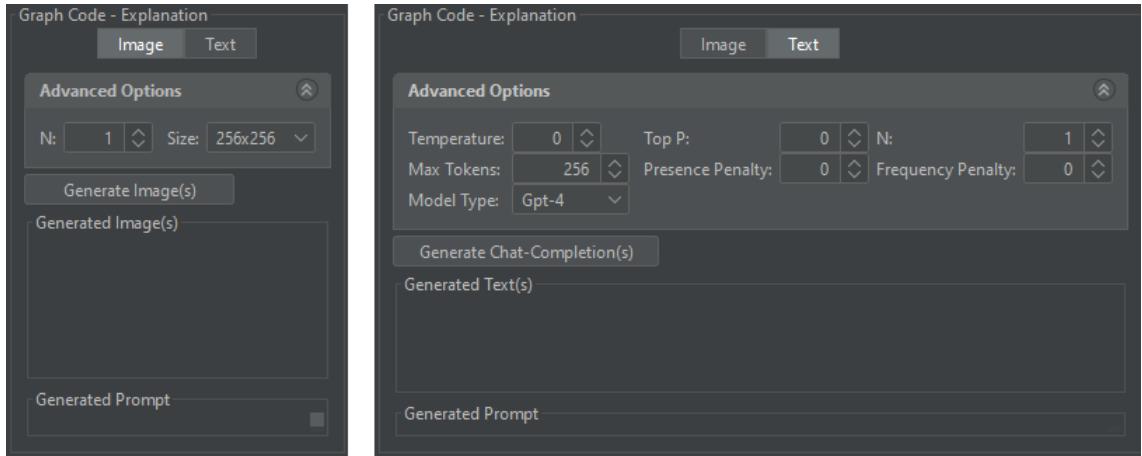


Abb. 5.1: Umordnung der Layouts von *ImagePanel* (links) und *TextPanel* (rechts).

Anstatt eines Labels zur Darstellung des generierten Bildes, bzw. eines Textfeldes zur Darstellung des generierten Textes, müssen nun mehrere Darstellungen in einer Benutzungsschnittstelle eingebunden werden. Hierfür wird auf ein Registerkartenpanel *JTabbedPane* anstelle des Labels *JLabel* bzw. des Textfeldes *JTextArea* zurückgegriffen. Dieses Registerkartenpanel bietet das Darstellen mehrere Registerkarten mit eigenständigem Inhalt, sowie dem Hin- und Herschalten zwischen diesen Registerkarten. Die Komponenten zur Darstellung der Inhalte bleiben unverändert ein *JLabel* bzw. eine *JTextArea*, die nun allerdings als Registerkarten dem Registerkartenpanel dynamisch hinzugefügt werden. Das dynamische Hinzufügen erfolgt durch eine Anpassung der Verarbeitung, die in Bezug auf die Ergebnisse vorsieht, dass alle Datenelemente bzw. Inhalte einer Antwort durchlaufen werden. Abb. 5.2 zeigt die Darstellung mehrerer Registerkarten mit generierten Bildern in einem Registerkartenpanel in der Benutzungsschnittstelle *ImagePanel*.

Das Beheben des letzten Mangels erfolgt durch das Modifizieren der Darstellung des Mauszeigers. Eine geeignete Darstellung für einen Mauszeiger, der einen laufenden Hintergrundprozess darstellen soll, ist eine Sanduhr. Die Modifikation dieser Darstellung kann in den entsprechenden Benutzungsschnittstellen *ImagePanel* oder *TextPanel* durch die Methode *setCursor(...)* erfolgen. In dem Moment, in dem ein Benutzer den Knopf „Generate ...“ betätigt, wird im Hintergrund ein entsprechender Arbeitsprozess zum Generieren einer Erklärung begonnen und der Mauszeiger wird zu einer Sanduhr angepasst. Sobald der Arbeitsprozess abgeschlossen wurde, erfolgreich oder nicht, wird der Mauszeiger auf den Standardwert zurückgesetzt.

Unabhängig von diesen Mängeln schlägt der Experte eine optionale Erweiterung der Benutzungsschnittstelle vor. Um den Benutzern der Anwendung eine besseres Einordnen bzw. ein besseres in Bezug setzen der generierten Erklärungen zu den

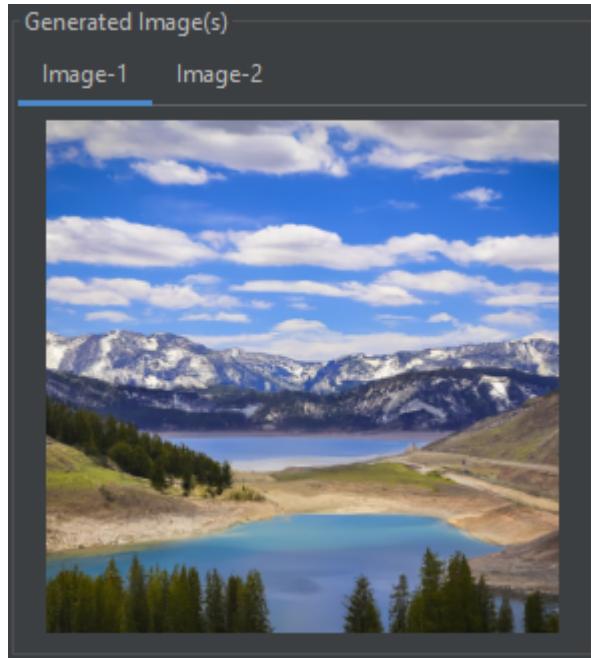


Abb. 5.2: Registerkartenpanel in der Benutzungsschnittstelle *ImagePanel* mit zwei Registerkarten.

ursprünglichen Dateien zu ermöglichen, schlägt der Experte das Umsetzen einer Gegenüberstellung des entsprechenden Graph Codes, der Original-Datei und der generierten Erklärung vor. Diese Gegenüberstellung wird erreicht durch das Hinzufügen einer Registerkartenpanels im rechten Teil des Grundbereichs *EditorGraphCode* mit entsprechenden Registerkarten. Diese Registerkarten sind zum einen die tabellarische Darstellung des Graph Codes, *GraphCodeTable* und zum anderen eine neue Benutzungsschnittstelle *OriginalAssetPanel*, die anhand des Dateinamens einer Graph Code Datei in der Kollektion nach einer entsprechenden Datei sucht und diese geeignet darstellt. Hierfür kann zum Großteil auf bereits bestehenden Quellcode der Komponente *AssetDetailPanel* zurückgegriffen werden. Abb. 5.3 zeigt die neue Benutzungsschnittstelle *OriginalAssetPanel* zum Darstellen der originalen Datei. Eine komplette Gegenüberstellung einer generierten visuellen Erklärung zu ihrer entsprechenden originalen Datei kann in Abb. 5.4 eingesehen werden.

Im Folgenden werden noch weitere Anpassungen an der prototypischen Proof-of-Concept Implementierung beschrieben, die unabhängig von den in der Evaluierung durchgeföhrten Experimenten erfolgten.

Im Umgang mit generativer KI sind Tokens von großer Bedeutung. Es wurde daher in der Benutzungsschnittstelle *TextPanel* eine Informationsoberfläche hinzugefügt, die es einem Benutzer ermöglicht, die Tokenanzahl eines generierten Textes einzusehen.

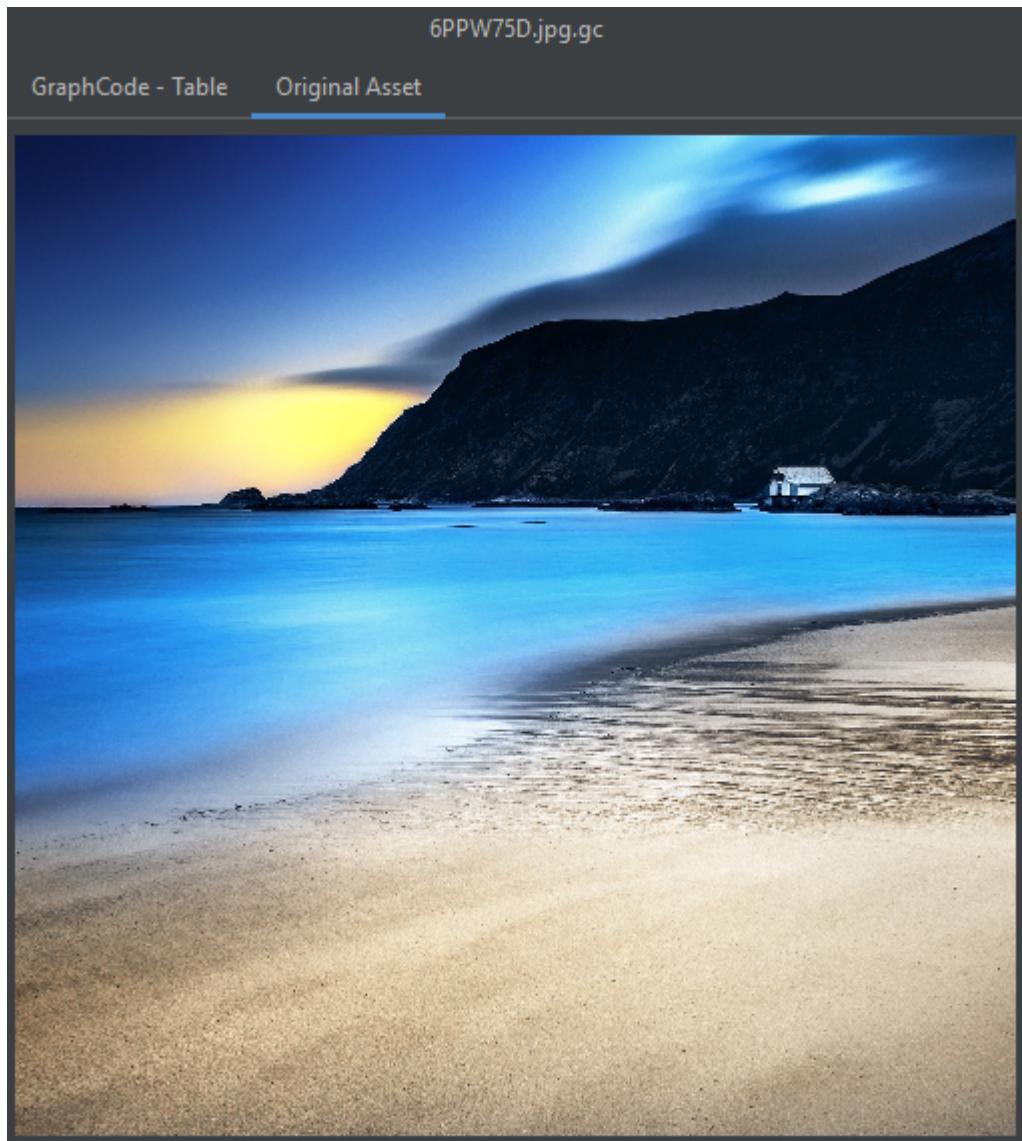


Abb. 5.3: Ausgewählte Registerkarte mit der Darstellung der mit einem Graph Code ursprünglich assoziierten Datei.

Darüber hinaus können durch diese Informationsoberfläche die Tokens in jenem Text farbig hervorgehoben werden. Diese Anpassung kann Abb. 5.5 eingesehen werden.

Eine weitere Anpassung, die eine sinnvolle Erweiterung der Funktionen der Anwendung darstellt, ist das automatische Sichern der in der Benutzungsschnittstelle *ImagePanel* generierten Bilder in einem dafür vorgesehenen Ordner „explanations“.

Infolge des angekündigten Updates [135] von OpenAI am 6. November 2023 (OpenAI DevDay [136]) konnten weitere Anpassungen und Erweiterungen an der entwickelten

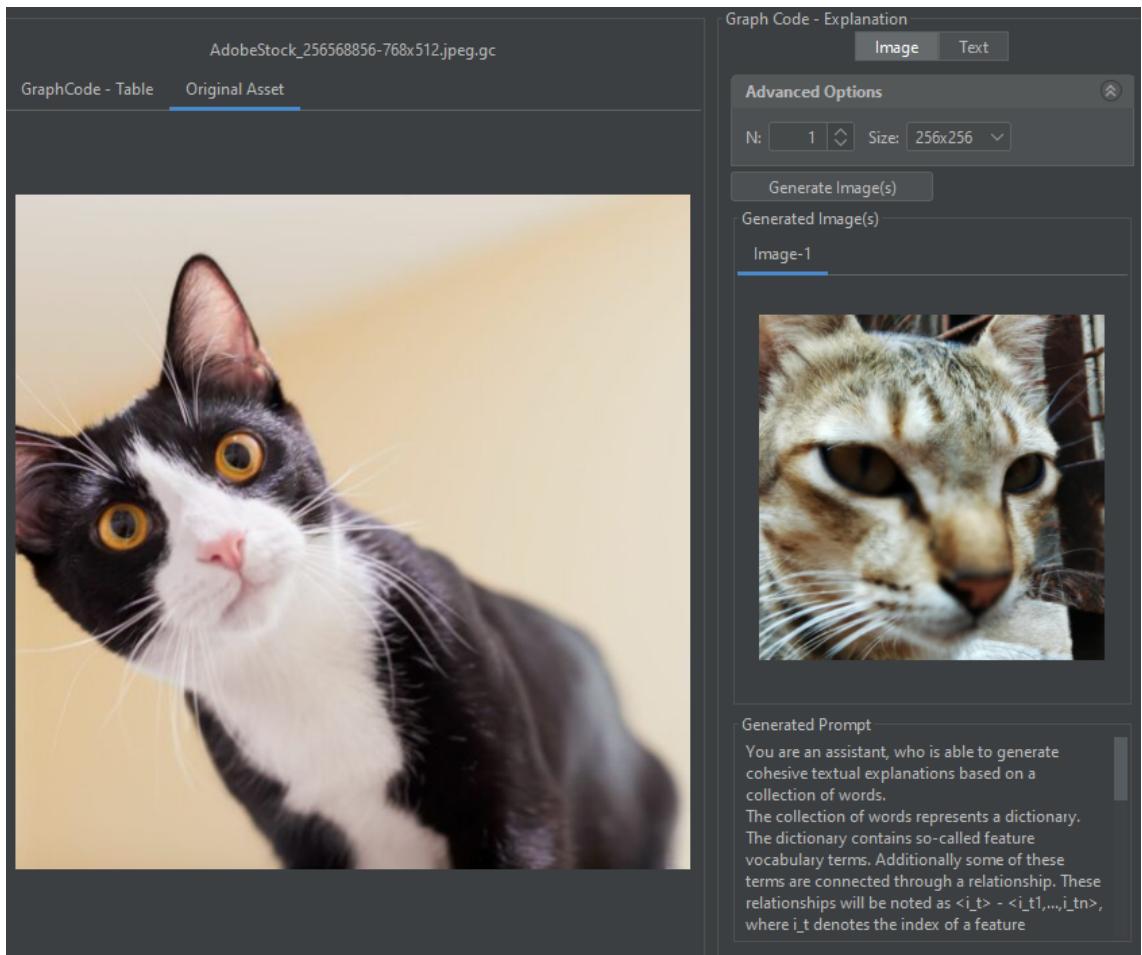


Abb. 5.4: Gegenüberstellung eines Graph Codes, der originalen Datei und der erzeugten visuellen Erklärung.

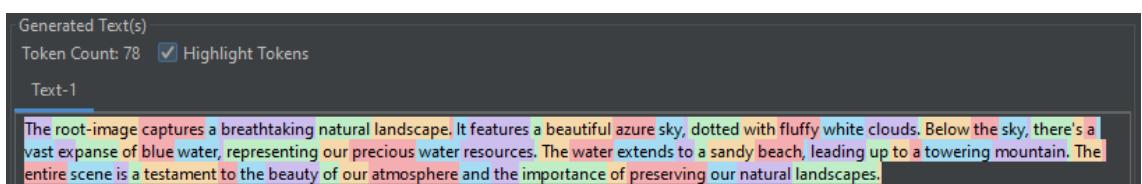


Abb. 5.5: Informationsoberfläche für Tokens in einem generierten Text.

Anwendung vorgenommen werden. Dieses Update umfasst eine Reihe an Verbesserungen, Anpassungen und Erweiterungen. Zu den Erweiterungen gehören die neue Dall-E 3 Schnittstelle, sowie eine Text-To-Speech (TTS, z.Dt. Text zu Sprache) Schnittstelle, mit welcher eine Audiodatei bzw. ein clip für einen beliebigen Text generiert werden kann. Diese Neuerungen wurden ausgewählt und in die Anwendung

integriert. Die Integration dieser Erweiterungen äußert sich in einer Anpassung der Benutzungsschnittstelle *ImagePanel* des Erklärungstyps Bild, welches hauptsächlich neue Auswahlmöglichkeiten in den erweiterten Optionen umfasst, sowie durch eine neue Benutzungsschnittstelle *AudioPanel*, einsehbar in Abb. 5.6, für einen neuen Erklärungstyp Audio, mit welchem auditive Erklärungen in Form von (kurzen) Audiodateien bzw. clips zu Graph Codes generiert werden können. Aufgrund der Größe der durch Dall-E 3 generierten Bilder (1024x1024, 1792x1024, 1024x1792) können diese nicht mehr in der Benutzungsschnittstelle angezeigt werden und es wird stattdessen ein Knopf zum Öffnen des Bildes in einem externen Fenster angeboten.

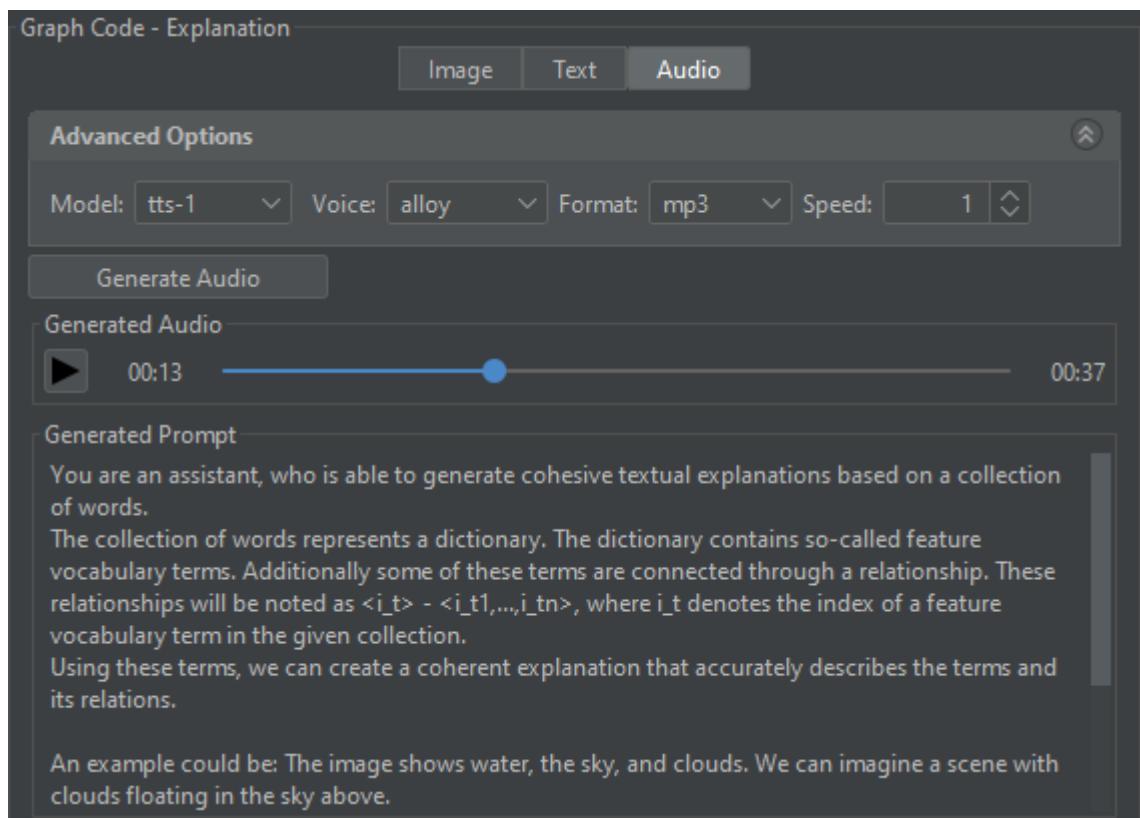


Abb. 5.6: Benutzungsschnittstelle für den Erklärungstyp Audio.

5.6 Zusammenfassung

Inhalt dieses Kapitels war die Evaluierung der im Rahmen der Implementierung entwickelten prototypischen Proof-of-Concept Implementierung. In Abschnitt 5.1 wurden zuerst die Ziele der Evaluierung formuliert. In Vorbereitung der Evaluierung wurden dann in Abschnitt 5.2 Aufgaben bzw. Experimente formuliert, die aus den Anwendungsfällen in Kapitel 3 abgeleitet wurden. Diese Experimente wurden dann, der Methodik nach Nunamaker entsprechend, den Forschungszielen zugewiesen. In den Forschungszielen FZ 1.4/E und FZ 2.4/E wurde dann jeweils die Vorbereitung, die Durchführung und die Einordnung der Ergebnisse des jeweiligen Experiments behandelt. Die in diesen Forschungszielen gesammelten Erkenntnisse umfassen in erster Linie eine Reihe an Mängeln, die in der Durchführung der jeweiligen Experimente identifiziert werden konnten. Als Reaktion auf diese Mängel wurden dann in Abschnitt 5.5 Anpassungen an der Benutzungsschnittstelle und Funktionen der Anwendung beschrieben.

5.6.1 Gewonnene Erkenntnisse

In diesem Abschnitt werden die Erkenntnisse aus den Forschungszielen „FZ 1.4/E Erklärbarkeit von MMIR mittels generativer KI“ und „FZ 2.4/E Integration generativer KI in das GMAF“ zusammengefasst.

In Abschnitt 5.3 „FZ 1.4/E Erklärbarkeit von MMIR mittels generativer KI“ konnte durch die Durchführung des Experiments **Exp. 1** festgestellt werden, dass die Anwendung eine Reihe an nicht fatalen Mängeln aufweist. Nicht fatal soll ausdrücken, dass diese Mängel die Funktionen und Abläufe in dem Programm nicht negativ beeinträchtigen, sodass Benutzer der Anwendung ihre formulierten Ziele erreichen können. Allerdings beeinträchtigen diese Mängel durchaus die Intuivität bzw. den Umgang mit dem Programm. Zum Abschluss des Experiments wurden diese Mängel gesammelt, beschrieben und jeweils potentielle Lösungsvorschläge formuliert. Auf die Ziele der Evaluierung rückblickend konnte festgehalten werden, dass beide Ziele, Vollständigkeit und Korrektheit der Implementierung, zufriedenstellend erfüllt wurden.

In Abschnitt 5.4 „FZ 2.4/E Integration generativer KI in das GMAF“ konnte durch die Durchführung des Experiments **Exp. 2** festgestellt werden, dass die Anwendung eine Reihe an nicht fatalen Mängeln aufweist. Auch diese Mängel betreffen nicht die Funktionen und Abläufe des Programms, sondern beeinträchtigen nur die Intuivität bzw. den Umgang mit dem Programm. Zum Abschluss des Experiments wurden diese Mängel gesammelt, beschrieben und jeweils potentielle Lösungsvorschläge formuliert. Auf die Ziele der Evaluierung rückblickend konnte festgehalten werden, dass beide Ziele, Vollständigkeit und Korrektheit der Implementierung, zufriedenstellend erfüllt wurden.

Infolge der in den Experimenten identifizierten Mängel und den darauffolgend beschriebenen Lösungsvorschlägen wurden in Abschnitt 5.5 Anpassungen an der prototypischen Proof-of-Concept Implementierung beschrieben, um diese Mängel zu beseitigen. Diese Anpassungen variieren stark in ihrem Umfang, betreffen aber in erster Linie nur den Aufbau oder die Interaktion mit der Benutzungsschnittstelle. Da keine fatalen Mängel festgestellt werden konnten, mussten keine Anpassungen an Datenmodell bezogene Algorithmen vorgenommen werden.

Darüber hinaus wurden, unabhängig von den in den Experimenten identifizierten Mängeln, in Abschnitt 5.5 weitere Anpassungen und Erweiterungen von Funktionen der Anwendung beschrieben, die vom Experten vorgeschlagen oder vom Autor als sinnvoll erachtet wurden.

Aufgrund mangelnder Ressourcen konnte im Rahmen der Evaluierung keine umfassendere Prüfung der prototypischen Anwendung auf Fehler und Defekte durchgeführt werden. Daher wurde mit den verfügbaren Ressourcen eine andere Evaluierungsmethodik durchgeführt, nach welchem für die Evaluierung festgehalten werden kann, dass die in dieser Arbeit entwickelte prototypische Proof-of-Concept Implementierung eine funktionierende Anwendung darstellt und welche das übergeordnete Ziel, Erklärungen von Graph Codes mittels generativer KI zu erzeugen, zufriedenstellend erfüllt.

Die nachfolgende Tabelle stellt eine Erweiterung der Tab. 4.2 dar und gibt eine Übersicht über den aktuellen Arbeitsstand nach Abschluss dieses Kapitels.

Tabelle 5.1: Tabelle zur Übersicht des aktuellen Arbeitsstands.

Forschungsziele		
FZ / OH	Beschreibung	Referenz
Erklärbarkeit von MMIR mittels generativer KI		
FZ 1.1/O	Recherche zur Erklärbarkeit von MMIR mittels generativer KI Grundlegende Technologien: <ul style="list-style-type: none"> • GMAF • MMFG • Graph Code 	Abschnitt 2.1.1 Abschnitt 2.1.2 Abschnitt 2.1.3
OH 1.1	Erste offene Herausforderung Systeme generativer KI und ein Überblick über aktuelle Systeme Diskussion und Auswahl von Systemen	OH 1.1 Abschnitt 2.1.4 Abschnitt 2.1.5

Forschungsziele		
FZ / OH	Beschreibung	Referenz
OH 1.2	Zweite offene Herausforderung	OH 2.1
FZ 1.2/TB	<p>Modellierung der Erklärbarkeit von MMIR mittels generativer KI</p> <p>Erklärbarkeit durch generative KI → Behandlung der ersten offenen Herausforderung OH 1.1</p> <p>Anwendungsfälle:</p> <ul style="list-style-type: none"> • Textuelle Beschreibungen <p>Wireframes</p> <p>Mechanismen</p> <p>Sequenzdiagramme</p> <p>→ Behandlung der zweiten offenen Herausforderung OH 1.2</p>	Abschnitt 3.2.1 Abschnitt 3.2.2 Abschnitt 3.2.3 Abschnitt 3.2.4 Abschnitt 3.2.5
FZ 1.3/I	<p>Implementierung der Erklärbarkeit von MMIR mittels generativer KI</p> <p>Benutzungsschnittstelle:</p> <ul style="list-style-type: none"> • Elemente der Benutzeroberfläche • Interaktion mit der Benutzeroberfläche 	Abschnitt 4.2.1
FZ 1.4/E	<p>Evaluierung der Erklärbarkeit von MMIR mittels generativer KI</p> <p>Experiment Exp. 1:</p> <ul style="list-style-type: none"> • Vorbereitung des Experiments • Durchführung des Experiments • Erkenntnisse des Experiments 	Abschnitt 5.3.1 Abschnitt 5.3.2 Abschnitt 5.3.3
Integration generativer KI in das GMAF		
FZ 2.1/O	<p>Recherche zur Integration generativer KI in das GMAF</p> <p>Aufzeigen der Integrationsmöglichkeiten von:</p> <ul style="list-style-type: none"> • Graph Codes <p>Erste offene Herausforderung</p> <ul style="list-style-type: none"> • Systemen generativer KI <p>Zweite offene Herausforderung</p>	Abschnitt 2.2.1 OH 2.1 Abschnitt 2.2.2 OH 2.2

Forschungsziele		
FZ / OH	Beschreibung	Referenz
FZ 2.2/TB	<p>Modellierung der Integration generativer KI in das GMAF</p> <p>Transformation von Graph Codes</p> <ul style="list-style-type: none"> • Transformation des Vokabulars • Transformation der Matrix • Anwendung von Graph Code Metriken <p>→ Behandlung der ersten offenen Herausforderung OH 2.1</p> <p>Einbindung generativer KI in das GMAF</p> <p>→ Behandlung der zweiten offenen Herausforderung OH 2.2</p>	Abschnitt 3.3.1 Abschnitt 3.3.2
FZ 2.3/I	<p>Implementierung der Integration generativer KI in das GMAF</p> <p>Transformation von Graph Codes</p> <p>Integration der Endpunkte</p>	Abschnitt 4.3.1 Abschnitt 4.3.2
FZ 2.4/E	<p>Evaluierung der Integration generativer KI in das GMAF</p> <p>Experiment Exp. 2:</p> <ul style="list-style-type: none"> • Vorbereitung des Experiments • Durchführung des Experiments • Erkenntnisse des Experiments 	Abschnitt 5.4.1 Abschnitt 5.4.2 Abschnitt 5.4.3

Im nächsten Kapitel 6 „Zusammenfassung“ wird die gesamte Arbeit zusammengefasst, auf die Forschungsfragen aus dem ersten Kapitel zurückgesprungen und weitere Informationen in Bezug auf diese Arbeit beschrieben.

6 Zusammenfassung und Diskussion

In den letzten Kapiteln 2 bis 5 wurden, aufeinander aufbauend und der Methodik nach Nunamaker folgend, in entsprechenden Forschungszielen, Lösungen für die in Kapitel 1 „Einleitung“ identifizierten Probleme erarbeitet. In Kapitel 2 wurden in einer ausführlichen und gründlichen Recherche bereits bestehende Konzepte und Softwarelösungen, denen für die Lösung der identifizierten Probleme Relevanz beigemessen wird, beschrieben und untersucht. In Kapitel 3 wurden dann Konzepte, Modelle und Vorgehensweisen zur weiteren Verwendung und Einbettung der in Kapitel 2 recherchierten Konzepte und Softwarelösungen modelliert. In Kapitel 4 wurde dann die in Kapitel 3 entwickelten Konzepte, Modelle und Vorgehensweisen in einer prototypischen Proof-of-Concept Implementierung umgesetzt und beschrieben. In Kapitel 5 wurden dann abschließend die Funktionen der in Kapitel 4 entwickelten prototypischen Proof-of-Concept Implementierung in einer Reihe an Experimenten getestet und validiert. Mit dem Abschluss des fünften Kapitels ist die in den letzten Kapiteln angewandte Methodik nach Nunamaker nun formal abgeschlossen.

Das Ziel dieses Kapitels ist es nun, die gesamte Arbeit abzurunden, d.h. die Ergebnisse der gesamten Arbeit kompakt darzustellen und diese mit den in der Einleitung angesprochenen Problembereichen und ihren entsprechend identifizierten Problemen in Beziehung zueinander zu setzen, sowie die in dieser Arbeit erlangten Erkenntnisse in den übergeordneten Themenbereich des MMIR einzuordnen.

Im Verlauf dieses Kapitels wird in Abschnitt 6.1 die Einordnung der Arbeitsergebnisse vorgenommen. In Abschnitt 6.2 werden verbleibende Herausforderungen angemerkt und Aspekte der Verarbeitung dieser Herausforderungen in zukünftige Arbeit, auch als *Future Work* bezeichnet, eingeordnet. Schlussendlich wird in Abschnitt 6.3 die Bearbeitung dieser Bachelorarbeit aus der Perspektive des Autors beschrieben.

6.1 Einordnung der Arbeitsergebnisse

In diesem Abschnitt werden die in dieser Arbeit gewonnenen Erkenntnisse in kompakter Darstellung in Beziehung zu den in der Einleitung angesprochenen Problembereichen und ihren entsprechend identifizierten Problemen gesetzt. Darüber hinaus wird diskutiert, ob das in Abschnitt 1.2.3 definierte Ziel für diese Arbeit erfolgreich abgedeckt werden konnte. Schlussendlich werden die in dieser Arbeit gewonnenen Erkenntnisse in den übergeordneten Themenbereich des MMIR eingeordnet und es wird in den nächsten Abschnitt 6.2 übergeleitet.

Der erste Problembereich, angesprochen in Abschnitt 1.2.1, beschreibt Probleme in Beziehung auf die Erklärbarkeit im GMAF. Die in diesem Problembereich beschrie-

benen Probleme werden im Folgenden noch einmal kurz beschrieben, bevor diese dann in Beziehung zu den gewonnenen Erkenntnisse gesetzt werden und erklärt wird, wie diese Probleme durch die Arbeit angesprochen und behandelt wurden.

1. Auch wenn das GMAF bereits Verfahren zur Erklärbarkeit von Prozessen des MMIR bietet, so basieren diese auf statischen und rein mathematischen Ansätzen.

Im Rahmen dieser Arbeit konnte die Erkenntnis erlangt werden, dass durch den Einsatz generativer KI die statischen und rein mathematischen Ansätze zur Erklärbarkeit von Prozessen des MMIR durch dynamischere Verfahren ersetzt werden können. Die entwickelte Anwendung bietet eine Lösung für dieses Problems, indem durch den Einsatz der von OpenAI entwickelten Systeme generativer KI Benutzern die Möglichkeit geboten wird Erklärungen für Prozesse des MMIR, genauer Graph Codes, auf dynamische Art und Weise zu generieren.

2. Die einzige Form von Erklärungen, die das GMAF bietet, sind (eingeschränkte) textuelle Erklärungen, obwohl auch andere Formen von Erklärungen, wie z.B. in Form von kurzen prägnanten Bildern, infrage kämen.

Im Rahmen dieser Arbeit konnte die Erkenntnis erlangt werden, dass durch eine geeignete Benutzungsschnittstelle auch andere Formen der Erklärung möglich sind. Die entwickelte Anwendung bietet in ihrer Benutzungsschnittstelle eine Lösung für dieses Problem in Form einer Arbeitsfläche, in welcher zwischen den Formen der Erklärung bzw. den Erklärungstypen hin- und hergeschaltet werden kann. Diese Formen umfassen die textuelle Form, also Erklärungen im Textformat und die visuelle Form, also Erklärungen in Form eines Bildes. Auf diese Weise sind Benutzer in der Lage originelle und qualitativ hochwertige Texte und Bilder als Erklärungen zu erzeugen.

3. Das GMAF bietet keine Möglichkeit, mit welcher Benutzer unterschiedliche Anwendungsfälle, wie z.B. Gemeinsamkeiten oder Differenzen zwischen Graph Codes, in Bezug auf Erklärungen abdecken können.

Im Rahmen dieser Arbeit konnte die Erkenntnis erlangt werden, dass durch eine geeignete Benutzungsschnittstelle den Benutzern der Anwendung Berechnungen auf Graph Codes zugänglich gemacht werden können, um verschiedene Anwendungsfälle in Bezug auf Erklärungen abdecken zu können. Die entwickelte Anwendung bietet in ihrer Benutzungsschnittstelle eine Lösung für dieses Problem in Form einer listen artigen Arbeitsfläche, in welcher eine Reihe an Graph Codes ausgewählt und eine Reihe an Aktionen und Operationen auf diesen ausgeführt werden können.

4. Das GMAF bietet nur grobe Möglichkeiten zur Anpassung der generierten Er-

klärungen, obwohl Anforderungen an Erklärungen je nach Anwendungsbereich erheblich variieren können.

Im Rahmen dieser Arbeit konnte die Erkenntnis erlangt werden, dass durch eine geeignete Benutzungsschnittstelle umfassende Möglichkeiten zur Anpassung von generierten Erklärungen möglich sind. Die entwickelte Anwendung bietet in ihrer Benutzungsschnittstelle durch ihre Elemente „Erweiterte Optionen“ umfassende Möglichkeiten die Anfragen zur Generierung von Erklärungen an die Endpunkte der Systeme generativer KI von OpenAI so zu parametrisieren, dass diese eine Breite an Anforderungen für verschiedenste Anwendungsbereiche abdecken.

Der zweite Problembereich, angesprochen in Abschnitt 1.2.2, beschreibt Probleme in Beziehung auf die Integration generativer KI in das GMAF. Die in diesem Problembereich beschriebenen Probleme werden im Folgenden noch einmal kurz beschrieben, bevor diese dann in Beziehung zu den gewonnenen Erkenntnissen gesetzt werden und erklärt wird, wie diese Probleme durch die Arbeit angesprochen und behandelt wurden.

1. Das GMAF besitzt keine Anbindung von Systemen generativer KI.

Im Rahmen dieser Arbeit wurde die Erkenntnis erlangt, dass eine einfache und flexible Integration der von OpenAI angebotenen Systeme generativer KI und ihrer Endpunkte möglich ist. Die entwickelte Anwendung löst dieses Problem durch die Integration der von OpenAI entwickelten Systeme generativer KI und ihrer Endpunkte zu den GPT-Modellen und Dall-E 2.

2. Das GMAF besitzt keine Möglichkeit die in MMIR-Prozessen identifizierten Merkmale in gültige und geeignete Eingabedaten für Systeme generativer KI zu überführen.

Im Rahmen dieser Arbeit wurde die Erkenntnis erlangt, dass die integrierten Systeme generativer KI als Eingabe eine Prompt benötigen. Eine Prompt besteht dabei aus ein oder mehreren Textnachrichten. Es wurde zudem die Erkenntnis erlangt, dass die in Graph Codes gespeicherten Informationen nicht stumpf und unstrukturiert in die Eingabe überführt werden können, sondern diese in einer logischen Abfolge geordnet übertragen und dem System generativer KI möglichst detailliert erklärt werden müssen. Die entwickelte Anwendung bietet eine Lösung für dieses Problem, indem das Datenmodell für Graph Codes so erweitert wird, dass die in Graph Codes gespeicherten Informationen in Textformat abgerufen werden können.

Das Ziel für diese Arbeit lautete: *Die im GMAF implementierten Konzepte zur Erklärung von MMIR-Prozessen sollen durch **generative KI** abgelöst werden, um*

bessere Erklärungen, sowie andere Formen von Erklärungen zu ermöglichen.

Das Ziel für diese Arbeit konnte erfolgreich erfüllt werden, da die entwickelte Anwendung das GMAF durch die Integration generativer KI erweitert, bereichert und zugleich die statischen und rein mathematischen Verfahren im GMAF ablöst. Des Weiteren konnten durch die Integration generativer KI auch andere Formen von Erklärungen in das GMAF eingebunden werden.

Es kann festgehalten werden, dass die im Rahmen dieser Arbeit entwickelte Anwendung in dieser Form und ihrem Umfang bislang im Themenbereich des MMIR noch nicht entwickelt oder vorgestellt wurde und somit einen wichtigen Beitrag im Themenbereich MMIR leisten kann.

Im nachfolgenden Abschnitt werden abschließend Aspekte beschrieben, die im Rahmen dieser Arbeit nicht abgedeckt werden konnten, oder die sich im Rahmen dieser Arbeit ergeben haben und folglich als Herausforderung verbleiben und im Rahmen zukünftiger Arbeiten / Projekte behandelt werden könnten.

6.2 Verbleibende Herausforderungen und „Future Work“

Drei verbleibende Herausforderungen können formuliert werden, die im Rahmen zukünftiger Arbeiten behandelt werden könnten:

- Optimierung der Prompt, z.B. Testen und Evaluierung von Variationen an Instruktionen.
- Andere Formen von Erklärungstypen in die Benutzungsschnittstelle einbinden. Dies würde die Integration von anderen Systemen generativer KI voraussetzen und führt zum nächsten Punkt.
- GPT-Modelle unterstützten bislang nur eine Eingabemodalität: Text [137]. Dadurch ist die Breite an potentiellen Anwendungsfällen dieser Modelle begrenzt [137]. OpenAI hat in einem Update (siehe [135]) ein Derivat des GPT-4 Modells, GPT-4V(ision) bzw. GPT-4-Vision-Preview, vorgestellt, welches in der Lage sein soll auch Bilder als Eingabemodalität zu unterstützen [137, 135]. Auf diese Weise soll dieses neue Modell in der Lage sein echte Bilder zu verstehen, zu beschreiben und Bildunterschriften zu erzeugen [135]. Dies verspricht eine Vielzahl von neuen Anwendungsmöglichkeiten in Bezug auf das GMAF. Dieses Modell zu integrieren und seine Funktionen im Kontext des MMIR voll auszuspielen, kann Gegenstand zukünftiger Arbeiten sein.
- Ein weiterer interessanter Punkt im Rahmen zukünftiger Arbeiten und/oder der Forschung wäre das Integrieren und Testen anderer Systeme generativer

KI, welche nicht zwangsläufig von OpenAI entwickelt wurden oder angeboten werden. Wie bereits in Abschnitt 2.1.4.2 angemerkt wurde, gibt es viele Systeme generativer KI und die Landschaft dieser Systeme ändert sich rapide.

Die aufgezählten verbleibenden Herausforderungen zeigen nur einen Ausschnitt an möglichen Herausforderungen, demonstrieren aber auch klar und deutlich den Umfang der in dieser Arbeit behandelten Aspekte, sowie ihre Relevanz für weitere zukünftige Forschung und Entwicklung im Bereich des MMIR.

6.3 Persönliches Fazit

Bereits im Wintersemester WS 22/23 hatte ich große Freude an der Bearbeitung des Kurses „01519 Fachpraktikum Multimedia Information Retrieval“ im Lehrgebiet Multimedia und Internetanwendungen. In diesem Fachpraktikum lernte ich das erste Mal die Methodik nach Nunamaker kennen und anwenden, die auch in dieser Arbeit zur Anwendung kam und eine große Hilfe beim Problemverständnis und der Problemlösung darstellte. Die Methodik ermöglichte mir durch das direkte Vorgeben der Struktur eine optimale Vorgehensweise zur Problemlösung, mit welcher ich meine Ideen für die Arbeit frei, aber gezielt ausführen und entfalten konnte. Gekoppelt mit einer gewissen Programmieraffinität hatte ich ebenfalls große Freude und keine Probleme bei der Entwicklung der prototypischen Proof-of-Concept Implementierung.

Literatur

- [1] BENDEL, O. „Digitalisierung • Definition | Gabler Wirtschaftslexikon.“ Springer Nature. (2021), Verfügbar: <https://wirtschaftslexikon.gabler.de/definition/digitalisierung-54195/version-384620> (Zugriff am 30.12.2023).
- [2] VAILSHERY, L. S. „Number of IoT connected devices worldwide 2019-2021, with forecasts to 2030.“ (2022), Verfügbar: <https://www-statista-com.ub-proxy.fernuni-hagen.de/statistics/1183457/iot-connected-devices-worldwide/> (Zugriff am 30.12.2023).
- [3] RABE, L. „Anzahl der Internetnutzer weltweit bis 2022.“ ITU. (2022), Verfügbar: <https://de-statista-com.ub-proxy.fernuni-hagen.de/statistik/daten/studie/805920/umfrage/anzahl-der-internetnutzer-weltweit/> (Zugriff am 30.12.2023).
- [4] McKinsey & Company. „How COVID-19 has pushed companies over the technology tipping point—and transformed business forever.“ SEILER, D., Hrsg. (2020), Verfügbar: <https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushes-companies-over-the-technology-tipping-point-and-transformed-business-forever> (Zugriff am 30.12.2023).
- [5] RABE, L. „Wie wirkt sich das Coronavirus auf die Nutzung digitaler Medien aus?“ (2022), Verfügbar: <https://de.statista.com/themen/6289/auswirkungen-des-coronavirus-covid-19-auf-digitale-medien/#topicOverview> (Zugriff am 30.12.2023).
- [6] Instagram.com. „Instagram.“ (2023), Verfügbar: <https://www.instagram.com> (Zugriff am 30.12.2023).
- [7] Facebook.com. „Facebook.“ (2023), Verfügbar: <https://www.facebook.com> (Zugriff am 30.12.2023).
- [8] Snapchat.com. „Snapchat.“ (2023), Verfügbar: <https://www.snapchat.com> (Zugriff am 30.12.2023).
- [9] TAYLOR, P. „Mobile data usage worldwide, by device 2020-2027.“ (2023), Verfügbar: <https://www-statista-com.ub-proxy.fernuni-hagen.de/statistics/1370201/global-mobile-data-usage/> (Zugriff am 30.12.2023).
- [10] KWAŚNICKA, H. und JAIN, L. C., *Bridging the Semantic Gap in Image and Video Analysis*. Berlin, Germany: Springer Cham, 2018, ISBN: 978-3-319-73891-8. DOI: <https://doi.org/10.1007/978-3-319-73891-8>. (Zugriff am 30.12.2023).

- [11] ANYOHA, R. „The History of Artificial Intelligence.“ (2017), Verfügbar: <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/> (Zugriff am 30.12.2023).
- [12] HEINRICH, L. „Anwendungsgebiete von künstlicher Intelligenz.“ (2022), Verfügbar: <https://datasolut.com/anwendungsgebiete-von-kuenstlicher-intelligenz/> (Zugriff am 30.12.2023).
- [13] Edley. „Artificial Intelligence vs. Machine Learning vs. Deep Learning.“ (2023), Verfügbar: <https://edley.de/insights/unterschied-artificial-intelligence-machine-deep-learning/> (Zugriff am 30.12.2023).
- [14] BENDEL, O. „Generative KI • Definition | Gabler Wirtschaftslexikon.“ (2023), Verfügbar: <https://wirtschaftslexikon.gabler.de/definition/generative-ki-124952/version-388556> (Zugriff am 30.12.2023).
- [15] McKinsey & Company, „What is generative AI?“ 2023. Verfügbar: <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai> (Zugriff am 30.12.2023).
- [16] OpenAI. „Introducing ChatGPT.“ (2022), Verfügbar: <https://openai.com/blog/chatgpt> (Zugriff am 30.12.2023).
- [17] OpenAI. „DALL · E 2.“ (2022), Verfügbar: <https://openai.com/product/dall-e-2> (Zugriff am 30.12.2023).
- [18] WAGENPFEIL, S.: „Smart Multimedia Information Retrieval“. Fernuniversität in Hagen, Hagen, 2022. Fernuniversität in Hagen, Fakultät für Mathematik und Informatik, Dissertation, 2022. DOI: 10.18445/20220718-214959-0. Verfügbar: <https://doi.org/10.18445/20220718-214959-0> (Zugriff am 30.12.2023).
- [19] MAZZONI, D. „Using AI to give people who are blind the ‚full picture‘.“ (2019), Verfügbar: <https://blog.google/outreach-initiatives/accessibility/get-image-descriptions/> (Zugriff am 30.12.2023).
- [20] Lego Group. „Easter Chick.“ (2019), Verfügbar: <https://zusammengebaut.com/anleitung-jetzt-dieses-knuffige-lego-kueken-nachbauen-68584/> (Zugriff am 30.12.2023).
- [21] HEUWERS, I. „Schmetterling.“ (2019), Verfügbar: <https://www.gedichte-oase.de/gedicht/kleiner-schmetterling> (Zugriff am 30.12.2023).
- [22] WAGENPFEIL, S., MC KEVITT, P. und HEMMJE, M., „Smart Multimedia Information Retrieval,“ *Analytics*, Jg. 2, Nr. 1, S. 198–224, 2023, ISSN: 2813-2203. DOI: 10.3390/analytics2010011. Verfügbar: <https://www.mdpi.com/2813-2203/2/1/11> (Zugriff am 30.12.2023).

- [23] KFZ Serviceportal. „Sicherheitsrelevante Bauteile an Kraftfahrzeugen.“ (2023), Verfügbar: <https://kfz-serviceportal.de/ratgeber/sicherheitsrelevante-bauteile-am-kfz/> (Zugriff am 30.12.2023).
- [24] Kia Austria GmbH. „Kia-Stinger-Preisliste.pdf.“ (2018), Verfügbar: <https://www.kia.com/content/dam/kwcmss/kme/at/de/assets/contents/utility/Pricelist/Kia-Stinger-Preisliste.pdf> (Zugriff am 30.12.2023).
- [25] NUNAMAKER, J. F. Jr, CHEN, M. und PURDIN, T., „Systems Development in Information Systems Research.,“ *Journal of Management Information Systems*, Jg. 7, Nr. 3, S. 89–106, 1991, ISSN: 0742-1222. DOI: 10.1080/07421222.1990.11517898. (Zugriff am 30.12.2023).
- [26] NORMAN, D. A. und DRAPER, S. W., *User Centered System Design - New Perspectives on Human-Computer Interaction*. USA: L. Erlbaum Associates Inc., 1986, ISBN: 978-0-898-59872-8. (Zugriff am 30.12.2023).
- [27] FOWLER, M., *UML Distilled - A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, Boston, ISBN: 978-0-321-19368-1. (Zugriff am 30.12.2023).
- [28] WAGENPFEIL, S. „Github Repository of GMAF and MMFVG.“ (2023), Verfügbar: <https://github.com/stefanwagenpfeil/GMAF> (Zugriff am 30.12.2023).
- [29] WAGENPFEIL, S., ENGEL, F., MC KEVITT, P. und HEMMJE, M., „AI-Based Semantic Multimedia Indexing and Retrieval for Social Media on Smartphones,“ *Information*, Jg. 12, Nr. 1, 2021, ISSN: 2078-2489. DOI: 10.3390/info12010043. Verfügbar: <https://www.mdpi.com/2078-2489/12/1/43> (Zugriff am 30.12.2023).
- [30] WAGENPFEIL, S., MC KEVITT, P., CHEDDAD, A. und HEMMJE, M., „Explainable Multimedia Feature Fusion for Medical Applications,“ *Journal of Imaging*, Jg. 8, Nr. 4, 2022, ISSN: 2313-433X. DOI: 10.3390/jimaging8040104. Verfügbar: <https://www.mdpi.com/2313-433X/8/4/104> (Zugriff am 30.12.2023).
- [31] WAGENPFEIL, S., VU, B., MC KEVITT, P. und HEMMJE, M., „Fast and Effective Retrieval for Large Multimedia Collections,“ *Big Data and Cognitive Computing*, Jg. 5, Nr. 3, 2021, ISSN: 2504-2289. DOI: 10.3390/bdcc5030033. Verfügbar: <https://www.mdpi.com/2504-2289/5/3/33> (Zugriff am 30.12.2023).
- [32] WAGENPFEIL, S., MC KEVITT, P. und HEMMJE, M., „Towards Automated Semantic Explainability of Multimedia Feature Graphs,“ *Information*, Jg. 12, Nr. 12, 2021, ISSN: 2078-2489. DOI: 10.3390/info12120502. Verfügbar: <https://www.mdpi.com/2078-2489/12/12/502> (Zugriff am 30.12.2023).

- [33] W3C.org. „W3C Semantic Web Activity.“ (2001), Verfügbar: <https://www.w3.org/2001/sw/> (Zugriff am 30.12.2023).
- [34] HAUSSER, R. R., *Grundlagen der Computerlinguistik: Mensch-Maschine-Kommunikation in natürlicher Sprache*. Springer Berlin Heidelberg, 2013, ISBN: 978-3-64257-306-4. Verfügbar: https://books.google.de/books?id=FW_zBQAAQBAJ (Zugriff am 30.12.2023).
- [35] WAGENPFEIL, S., ENGEL, F., MC KEVITT, P. und HEMMJE, M., „Graph Codes-2D Projections of Multimedia Feature Graphs for Fast and Effective Retrieval,“ *International Journal of Computer and Information Engineering*, Jg. 15, Nr. 12, S. 636–648, 2021, ISSN: eISSN: 1307-6892. Verfügbar: <https://publications.waset.org/vol/180> (Zugriff am 30.12.2023).
- [36] LAWTON, G. „What is generative AI? Everything you need to know.“ (2023), Verfügbar: <https://www.techtarget.com/searchenterpriseai/definition/generative-AI> (Zugriff am 30.12.2023).
- [37] MARTINEAU, K. „What is synthetic data?“ (2023), Verfügbar: <https://research.ibm.com/blog/what-is-synthetic-data> (Zugriff am 30.12.2023).
- [38] RIEMANN, R. „Synthetic Data.“ (2023), Verfügbar: https://edps.europa.eu/press-publications/publications/techsonar/synthetic-data_en (Zugriff am 30.12.2023).
- [39] MOSTLY AI. „What is synthetic data?“ (2023), Verfügbar: <https://mostly.ai/synthetic-data/what-is-synthetic-data> (Zugriff am 30.12.2023).
- [40] FRUHLINGER, J. „Was ist Generative AI?“ (2023), Verfügbar: <https://www.computerwoche.de/a/was-ist-generative-ai,3614061> (Zugriff am 30.12.2023).
- [41] FOSTER, D., *Generatives Deep Learning*. O'Reilly, 2020, https://www.assets.dpunkt.de/leseproben/13535/2_GenerativeModellierung.pdf, ISBN: 978-3-96009-128-8. Verfügbar: <https://dpunkt.de/produkt/generatives-deep-learning/> (Zugriff am 30.12.2023).
- [42] Turing. „Generative Models vs Discriminative Models: Which One to Choose?“ (2023), Verfügbar: <https://www.turing.com/kb/generative-models-vs-discriminative-models-for-deep-learning> (Zugriff am 30.12.2023).
- [43] RONKOWITZ, K. „Eliza, a chatbot therapist.“ (2017), Verfügbar: <https://web.njit.edu/~ronkowitz/eliza.html> (Zugriff am 30.12.2023).

- [44] DWHITE, M. „A Brief History of Generative AI.“ (2023), Verfügbar: [http://matthewdwhite.medium.com/a-brief-history-of-generative-ai-cb1837e67106](https://matthewdwhite.medium.com/a-brief-history-of-generative-ai-cb1837e67106) (Zugriff am 30.12.2023).
- [45] LUBER, S. und LITZEL, N. „Was ist Machine Learning?“ (2016), Verfügbar: <https://www.bigdata-insider.de/was-ist-machine-learning-a-592092/> (Zugriff am 30.12.2023).
- [46] „Machine Learning: Definition, Algorithmen, Methoden und Beispiele.“ (2022), Verfügbar: <https://datasolut.com/was-ist-machine-learning/> (Zugriff am 30.12.2023).
- [47] RAY, S., „A Quick Review of Machine Learning Algorithms,“ in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, S. 35–39. DOI: [10.1109/COMITCon.2019.8862451](https://doi.org/10.1109/COMITCon.2019.8862451).
- [48] IBM. „What is unsupervised learning?“ (2023), Verfügbar: <https://www.ibm.com/topics/unsupervised-learning> (Zugriff am 30.12.2023).
- [49] IBM. „Was ist Deep Learning?“ (2023), Verfügbar: <https://www.ibm.com/de-de/topics/deep-learning> (Zugriff am 30.12.2023).
- [50] IBM Data and AI Team. „AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the difference?“ (2023), Verfügbar: <https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/> (Zugriff am 30.12.2023).
- [51] WUTTKE, L. „Deep Learning: Definition, Beispiele & Frameworks.“ (2023), Verfügbar: <https://datasolut.com/was-ist-deep-learning/> (Zugriff am 30.12.2023).
- [52] GOODFELLOW, I. J. u. a., *Generative Adversarial Networks*, 2014. Verfügbar: <https://arxiv.org/abs/1406.2661>.
- [53] KINGMA, D. P. und WELLING, M., „An Introduction to Variational Autoencoders,“ *Foundations and Trends® in Machine Learning*, Jg. 12, Nr. 4, S. 307–392, 2019. DOI: [10.1561/2200000056](https://doi.org/10.1561/2200000056). Verfügbar: <https://doi.org/10.1561%2F2200000056> (Zugriff am 30.12.2023).
- [54] Nvidia. „Large Language Models Explained.“ (2023), Verfügbar: <https://www.nvidia.com/en-us/glossary/data-science/large-language-models/> (Zugriff am 30.12.2023).
- [55] VASWANI, A. u. a., *Attention Is All You Need*, 2017. Verfügbar: <https://arxiv.org/abs/1706.03762>.
- [56] NABI, J. „In-Context Learning Approaches in Large Language Models.“ (2023), Verfügbar: <https://towardsdatascience.com/in-context->

- learning - approaches - in - large - language - models - 9c0c53b116a1 (Zugriff am 30.12.2023).
- [57] Kobold AI. „Was ist Explainable AI (XAI)? Alles was man wissen muss.“ (2023), Verfügbar: <https://www.kobold.ai/explainable-ai/> (Zugriff am 30.12.2023).
 - [58] GAO, T., „Prompting: Better Ways of Using Language Models for NLP Tasks,“ *The Gradient*, 2021. Verfügbar: <https://thegradient.pub/prompting/> (Zugriff am 30.12.2023).
 - [59] Hopsworks. „Prompt Engineering.“ (2023), Verfügbar: <https://www.hopsworks.ai/dictionary/prompt-engineering> (Zugriff am 30.12.2023).
 - [60] SkimAI. „Understanding the Role of Prompts in ChatGPT and other Language Models.“ (2023), Verfügbar: <https://skimai.com/role-of-prompts-in-gpt-and-ai-language-models/> (Zugriff am 30.12.2023).
 - [61] DAIR.AI. „Basics of Prompting.“ (2023), Verfügbar: <https://www.promptingguide.ai/introduction/basics> (Zugriff am 30.12.2023).
 - [62] DAIR.AI. „Prompt Engineering Guide.“ (2023), Verfügbar: <https://www.promptingguide.ai/> (Zugriff am 30.12.2023).
 - [63] DAIR.AI. „Zero-Shot Prompting.“ (2023), Verfügbar: <https://www.promptingguide.ai/techniques/zeroshot> (Zugriff am 30.12.2023).
 - [64] DAIR.AI. „Few-Shot Prompting.“ (2023), Verfügbar: <https://www.promptingguide.ai/techniques/fewshot> (Zugriff am 30.12.2023).
 - [65] DAIR.AI. „Chain-of-Thought Prompting.“ (2023), Verfügbar: <https://www.promptingguide.ai/techniques/cot> (Zugriff am 30.12.2023).
 - [66] WEI, J. u. a., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2023. Verfügbar: <https://arxiv.org/abs/2201.11903>.
 - [67] KOJIMA, T., GU, S. S., REID, M., MATSUO, Y. und IWASAWA, Y., *Large Language Models are Zero-Shot Reasoners*, 2023. Verfügbar: <https://arxiv.org/abs/2205.11916>.
 - [68] McKinsey & Company. „The economic potential of generative AI: The next productivity frontier.“ (2023), Verfügbar: <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#key-insights> (Zugriff am 30.12.2023).
 - [69] THORMUNDSSON, B. „Artificial intelligence (AI) market size worldwide in 2021 with a forecast until 2030.“ (2023), Verfügbar: <https://www.statista.com/statistics/1365145/artificial-intelligence-market-size/> (Zugriff am 30.12.2023).

- [70] OpenAI. „OpenAI.“ (2023), Verfügbar: <https://openai.com/> (Zugriff am 30.12.2023).
- [71] StabilityAI. „Stable Diffusion — Stability AI.“ (2023), Verfügbar: <https://stability.ai/stablediffusion> (Zugriff am 30.12.2023).
- [72] StabilityAI. „Stability AI.“ (2023), Verfügbar: <https://stability.ai/> (Zugriff am 30.12.2023).
- [73] Midjourney. „Midjourney Documentation and User Guide.“ (2023), Verfügbar: <https://docs.midjourney.com/> (Zugriff am 30.12.2023).
- [74] OpenAI. „Models - OpenAI API.“ (2023), Verfügbar: <https://platform.openai.com/docs/models> (Zugriff am 30.12.2023).
- [75] Google. „PaLM 2 Technical Repor.“ (2023), Verfügbar: <https://ai.google/static/documents/palm2techreport.pdf> (Zugriff am 30.12.2023).
- [76] GHAHRAMANI, Z. „Introducing PaLM 2.“ (2023), Verfügbar: <https://blog.google/technology/ai/google-palm-2-ai-large-language-model/> (Zugriff am 30.12.2023).
- [77] Google. „Bard.“ (2023), Verfügbar: <https://bard.google.com/> (Zugriff am 30.12.2023).
- [78] ElevenLabs Inc. „ElevenLabs - Generative AI Text to Speech & Voice Cloning.“ (2023), Verfügbar: <https://elevenlabs.io/> (Zugriff am 30.12.2023).
- [79] Resemble.AI. „AI Voice Generator with Text-to-Speech and Speech-to-Speech.“ (2023), Verfügbar: <https://www.resemble.ai/> (Zugriff am 30.12.2023).
- [80] Suno.AI. „Text-Prompted Generative Audio Model.“ (2023), Verfügbar: <https://github.com/suno-ai/bark> (Zugriff am 30.12.2023).
- [81] Runway AI, Inc. „Runway - Advancing creativity with artificial intelligence.“ (2023), Verfügbar: <https://runwayml.com/> (Zugriff am 30.12.2023).
- [82] Synthesia. „Synthesia | #1 AI Video Generator.“ (2023), Verfügbar: <https://www.synthesia.io/> (Zugriff am 30.12.2023).
- [83] Rephrase.ai. „Rephrase.ai: Convert Text into Engaging AI Videos in Minutes.“ (2023), Verfügbar: <https://www.rephrase.ai/> (Zugriff am 30.12.2023).
- [84] Sequoia. „Generative AI: A Creative New World.“ (2023), Verfügbar: <https://www.sequoiacap.com/article/generative-ai-a-creative-new-world/> (Zugriff am 30.12.2023).

- [85] KLINKHAMMER, D., *Künstliche Intelligenz in der Radikalisierungsforschung: Synopse von Grundlagenwissen, Weiterbildungsangeboten und Limitationen* (MOTRA-Spotlight). Wiesbaden: MOTRA-Verbund Monitoringsystem und Transferplattform Radikalisierung, 2023, Bd. 06/23, S. 13. DOI: <https://doi.org/10.57671/motra-2023006>. (Zugriff am 30.12.2023).
- [86] KUNDU, R. „Large Language Models (LLMs): Challenges, Predictions, Tutorial.“ (2023), Verfügbar: <https://www.v7labs.com/blog/large-language-models-llms#limitations-of-large-language-models> (Zugriff am 30.12.2023).
- [87] BENDEL, O. „Responsible AI • Definition | Gabler Wirtschaftslexikon.“ (2021), Verfügbar: <https://wirtschaftslexikon.gabler.de/definition/responsible-ai-123232/version-384809> (Zugriff am 30.12.2023).
- [88] Maintenance_Mental. „Famous Cities as a Venice : r/midjourney.“ (2023), Verfügbar: https://www.reddit.com/r/midjourney/comments/13dzenp/famous_cities_as_a_venice/ (Zugriff am 30.12.2023).
- [89] Midjourney. „Midjourney Quick Start Guide.“ (2023), Verfügbar: <https://docs.midjourney.com/docs> (Zugriff am 30.12.2023).
- [90] Midjourney. „Midjourney Subscription Plans.“ (2023), Verfügbar: <https://docs.midjourney.com/docs/plans> (Zugriff am 30.12.2023).
- [91] Midjourney. „Midjourney Terms of Service.“ (2023), Verfügbar: <https://docs.midjourney.com/docs/en/community-guidelines> (Zugriff am 30.12.2023).
- [92] StabilityAI. „Credits + Billing.“ (2023), Verfügbar: <https://platform.stability.ai/docs/getting-started/credits-and-billing> (Zugriff am 30.12.2023).
- [93] W3C. „REST.“ (2011), Verfügbar: <https://www.w3.org/2001/sw/wiki/REST> (Zugriff am 30.12.2023).
- [94] OpenAI. „Pricing.“ (2023), Verfügbar: <https://openai.com/pricing> (Zugriff am 30.12.2023).
- [95] OpenAI. „API Reference - OpenAI API.“ (2023), Verfügbar: <https://platform.openai.com/docs/api-reference> (Zugriff am 30.12.2023).
- [96] KANNING, T. „OpenAI-Java.“ (2023), Verfügbar: <https://github.com/TheoKanning/openai-java> (Zugriff am 30.12.2023).
- [97] JANG, J. „How DALL·E Credits Work.“ (2023), Verfügbar: <https://help.openai.com/en/articles/6399305-how-dall-e-credits-work> (Zugriff am 30.12.2023).

- [98] Google. „Häufig gestellte Fragen zu Bard.“ (2023), Verfügbar: <https://bard.google.com/faq> (Zugriff am 30.12.2023).
- [99] HUFFMAN, S. und WOODWARD, J. „PaLM API & MakerSuite: an approachable way to start prototyping and building generative AI applications.“ (2023), Verfügbar: <https://developers.googleblog.com/2023/03/announcing-palm-api-and-makersuite.html> (Zugriff am 30.12.2023).
- [100] Google. „Meet the PaLM API.“ (2023), Verfügbar: <https://developers.generativeai.google/products/palm> (Zugriff am 30.12.2023).
- [101] OpenAI. „Introducing ChatGPT.“ (2022), Verfügbar: <https://openai.com/blog/chatgpt> (Zugriff am 30.12.2023).
- [102] Google. „Vertex AI - Preise.“ (2023), Verfügbar: <https://cloud.google.com/vertex-ai/pricing?hl=de#europe> (Zugriff am 30.12.2023).
- [103] Google. „Public preview FAQ.“ (2023), Verfügbar: https://developers.generativeai.google/guide/preview_faq (Zugriff am 30.12.2023).
- [104] BROWN, T. B. u. a., *Language Models are Few-Shot Learners*, 2020. Verfügbar: <https://arxiv.org/abs/2005.14165>.
- [105] OpenAI. „GPT-4.“ (2023), Verfügbar: <https://openai.com/research/gpt-4> (Zugriff am 30.12.2023).
- [106] BASTIAN, M. „GPT-4 hat mehr als eine Billion Parameter - Bericht Mid-journey prompted by GPT-4.“ (2023), Verfügbar: <https://the-decoder.de/gpt-4-hat-eine-billion-parameter/> (Zugriff am 30.12.2023).
- [107] OpenAI. „GPT-4 API general availability and deprecation of older models in the Completions API.“ (2023), Verfügbar: <https://openai.com/blog/gpt-4-api-general-availability> (Zugriff am 30.12.2023).
- [108] IOANNIDOU, X. „Generative AI & Large Language Models for Life Sciences.“ (2023), Verfügbar: https://ml-architects.ch/blog_posts/llm_for_life_sciences.html (Zugriff am 30.12.2023).
- [109] SabrePC. „What Causes Large Language Models to Hallucinate.“ (2023), Verfügbar: <https://www.sabrepc.com/blog/Deep-Learning-and-AI/what-causes-large-language-models-to-hallucinate> (Zugriff am 30.12.2023).
- [110] Google. „Gson.“ (2023), Verfügbar: <https://github.com/google/gson> (Zugriff am 30.12.2023).
- [111] OpenAI. „Libraries - OpenAI API.“ (2023), Verfügbar: <https://platform.openai.com/docs/libraries/community-libraries> (Zugriff am 30.12.2023).

- [112] OpenAI. „OpenAI and Microsoft extend partnership.“ (2023), Verfügbar: <https://openai.com/blog/openai-and-microsoft-extend-partnership> (Zugriff am 30.12.2023).
- [113] OpenAI. „Introduction.“ (2023), Verfügbar: <https://platform.openai.com/docs/introduction> (Zugriff am 30.12.2023).
- [114] Postman. „2023 State of the API Report | API Technologies.“ (2023), Verfügbar: <https://www.postman.com/state-of-api/api-technologies/#api-technologies> (Zugriff am 30.12.2023).
- [115] OpenAI. „OpenAPI.yaml.“ (2023), Verfügbar: <https://github.com/openai/openai/blob/master/openapi.yaml> (Zugriff am 30.12.2023).
- [116] Swagger. „API Code & Client Generator | Swagger Codegen.“ (2023), Verfügbar: <https://swagger.io/tools/swagger-codegen/> (Zugriff am 30.12.2023).
- [117] J., J. „ChatGPT API Transition Guide.“ (2023), Verfügbar: <https://help.openai.com/en/articles/7042661-chatgpt-api-transition-guide> (Zugriff am 30.12.2023).
- [118] OpenAI. „Tokenizer.“ (2023), Verfügbar: <https://platform.openai.com/tokenizer> (Zugriff am 30.12.2023).
- [119] OpenAI. „Model.py.“ (2023), Verfügbar: <https://github.com/openai/tiktoken/blob/main/tiktoken/model.py> (Zugriff am 30.12.2023).
- [120] OpenAI. „tiktoken.“ (2023), Verfügbar: <https://github.com/openai/tiktoken> (Zugriff am 30.12.2023).
- [121] Knuddels GmbH & Co. KG. „JTokkit.“ (2023), Verfügbar: <https://jtokkit.knuddels.de/> (Zugriff am 30.12.2023).
- [122] IEEE. „IEEE - The world’s largest technical professional organization dedicated to advancing technology for the benefit of humanity.“ (2023), Verfügbar: <https://www.ieee.org/> (Zugriff am 30.12.2023).
- [123] IEEE. „IEEE Standard for Software Reviews and Audits,“ *IEEE Std 1028-2008*, S. 1–53, 2008. DOI: 10.1109/IEEESTD.2008.4601584. (Zugriff am 30.12.2023).
- [124] Quality Services & Wissen GmbH. „RUP - Rational Unified Process | QUALITY.DE.“ (2023), Verfügbar: <https://www.quality.de/lexikon/rup-rational-unified-process/> (Zugriff am 30.12.2023).
- [125] Informatik Forum Simon. „Rational Unified Process (RUP).“ (2023), Verfügbar: <http://www.infforum.de/themen/anwendungsentwicklung/se-rup.htm> (Zugriff am 30.12.2023).

- [126] Utecht • Softwareentwicklung & Consulting. „Utecht Softwareentwicklung & Consulting - RUP.“ (2023), Verfügbar: <https://www.utecht.net/vorgehensmodelle/rup.php> (Zugriff am 30.12.2023).
- [127] Object Management Group. „UML.“ (2015), Verfügbar: <http://www.omg.org/spec/UML/> (Zugriff am 30.12.2023).
- [128] Visual Paradigm. „What is Unified Modeling Language (UML)?“ (2023), Verfügbar: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> (Zugriff am 30.12.2023).
- [129] KREISCHE, D., „Geschäftsprozessmodellierung mit der Unified Modeling Language (UML),“ Diss., Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2004.
- [130] BELL, D. (2003), Verfügbar: <https://developer.ibm.com/articles/introduction-to-uml/> (Zugriff am 30.12.2023).
- [131] Object Management Group. „OMG | Object Management Group.“ (2023), Verfügbar: <https://www.omg.org/> (Zugriff am 30.12.2023).
- [132] SIX, H. und WINTER, M., „Kurseinheit 3: Funktions- und Verhaltensmodellierung,“ in Fernuniversität in Hagen, 2020, S. 132–133, Lehrtext des Kurses: Software Engineering, Kurseinheit 3. Verfügbar: <https://vu.fernuni-hagen.de/1vuweb/1vuauth/file/FeU/Informatik/2020WS/01793/material/ke/ke1/01793-03-S1-002628910.pdf>.
- [133] Oracle. „Java | Oracle.“ (2023), Verfügbar: <https://www.java.com/de/> (Zugriff am 30.12.2023).
- [134] Apache Maven Project. „Maven - Welcome to Apache Maven.“ (2023), Verfügbar: <https://maven.apache.org/> (Zugriff am 30.12.2023).
- [135] OpenAI. „New models and developer products announced at DevDay.“ (2023), Verfügbar: <https://openai.com/blog/new-models-and-developer-products-announced-at-devday> (Zugriff am 30.12.2023).
- [136] OpenAI. „OpenAI DevDay.“ (2023), Verfügbar: <https://devday.openai.com/> (Zugriff am 30.12.2023).
- [137] OpenAI. „Vision - OpenAI API.“ (2023), Verfügbar: <https://platform.openai.com/docs/guides/vision> (Zugriff am 30.12.2023).