UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

Generierung von Feedback-Regeln für Modelle aus annotierten Musterlösungen

Vorgelegt der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen

von

Michael Vivian Mboni Saha
michael.mboni-saha@stud.uni-due.de
https://michael-mboni.dev/
Matrikelnummer: 3154361

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Betreuer:	Dr. Michael Striewe
Erstgutachter:	Prof. Dr. Michael Goedicke
Zweitgutachter:	Prof. Dr. Volker Gruhn
Studiengang:	M.Sc. Software and Network Engineering
Studiensemester:	Wintersemester 2023/2024
Datum:	01.02.2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

E-Assessment-Systemen auch als elektronische Bewertungssysteme bezeichnet, sind computergestützte Technologien und Plattformen, die entwickelt wurden, um den Prozess der Bewertung und Beurteilung von akademischen Leistungen, Prüfungen und Aufgaben zu unterstützen oder zu automatisieren. 11, 14

formative Bewertung ist ein kontinuierlicher Prozess, der während des Lernens stattfindet. Ihr Hauptziel ist es, den Fortschritt der Schüler zu verfolgen, ihre Bedürfnisse zu identifizieren und ihnen bei der Verbesserung ihrer Leistung zu helfen.. 16

Geschlossene Fragen Diese Fragen zeichnen sich durch ihre Fähigkeit aus, eine begrenzte Auswahl vorab definierter Antwort möglichkeiten anzubieten, aus denen die Lernenden die adäquate Antwort auswählen müssen. 18

GReQL Converter Dies ist das im Rahmen dieser Masterarbeit entwickelte Werkzeug, das die Umwandlung von PlantText-Code in GReQL-Code ermöglicht, der für die Auswertung von UML-Diagrammen auf der JACK-Plattform verwendet werden kann.. 3, 54, 56–63, 65, 67, 69

JACK ist ein e-Assessment System, das von der Universität Duisburg-Essen entwickelt wurde. Es handelt sich um eine webbasierte Plattform, die es Lehrkräften ermöglicht, Online-Bewertungen wie Quiz, Prüfungen und Umfragen zu erstellen und durchzuführen. Jack bietet eine Vielzahl von Funktionen, die den Lehrkräften bei der Verwaltung ihrer Prüfungen helfen, wie z. B. Benotung, Berichterstattung und Verfolgung des Lernfortschritts. [51]. 13, 14

konzeptuellen Modellen sind abstrakte Darstellungen oder konzeptionelle Strukturen, die darauf abzielen, Ideen, Beziehungen, Konzepte oder Entitäten eines bestimmten Bereichs zu beschreiben, ohne in konkrete Details oder Implementierungsdetails einzugehen.. 13

Offene Fragen Im Kontrast zu geschlossenen Fragen präsentieren sie keine vordefinierten Antworten und gestatten den Lernenden, ihre Gedanken selbstständig zu artikulieren. 19

summative Bewertung ist eine Art von Bewertung, die am Ende eines definierten Lernzeitraums durchgeführt wird, wie z.B. am Ende eines Kurses, eines Moduls oder eines Schuljahres.. 16

Abkürzungsverzeichnis

OPAL Open Academic Learning Platform

UML Unified Modeling Language

UDE Universität Duisburg Essen

GReQL Graph Repository Query Language

XMI XML Metadata Interchange

UXF UML eXchange Format

NLP Natural Language Processing

BPMN Business Process Model and Notation

DFD Data Flow Diagrams

ER-Modelle Entity-Relationship-Modelle

ERD Entity-Relationship-Diagramm

Abstract - Englisch

This master's thesis focuses on the development of the GReQL Converter to simplify the process of generating feedback rules for UML models in the context of E-Assessment systems. The tool is designed to provide (semi-)automated support to teachers in creating GReQL code required for evaluating student submissions related to UML class diagram tasks on the JACK platform. The GReQL Converter is designed to address the challenges that teachers face when creating GReQL code, such as the need for expertise in GReQL syntax and the time-consuming nature of the process. The tool aims to streamline the process of generating feedback rules for UML models, thereby improving the efficiency and effectiveness of E-Assessment systems. The results of this work demonstrate the feasibility and effectiveness of the GReQL Converter in simplifying the process of generating feedback rules for UML models.

Abstract - Deutsch

Diese Masterarbeit konzentriert sich auf die Entwicklung des GReQL Converters, um den Prozess der Generierung von Feedback-Regeln für UML-Modelle im Kontext von E-Assessment-Systemen zu vereinfachen. Das Tool soll Lehrern (halb-)automatisierte Unterstützung bei der Erstellung von GReQL-Code bieten, der für die Bewertung von Studenteneinreichungen bezüglich UML-Klassendiagramm-Aufgabe auf der JACK-Plattform erforderlich ist. Der GReQL Converter ist darauf ausgelegt, die Herausforderungen zu bewältigen, mit denen Lehrer bei der Erstellung von GReQL-Code konfrontiert sind, wie z.B. die Notwendigkeit von Expertise in GReQL-Syntax und der zeitaufwändige Charakter des Prozesses. Das Tool soll den Prozess der Generierung von Feedback-Regeln für UML-Modelle vereinfachen und damit die Effizienz und Effektivität von E-Assessment-Systemen verbessern. Die Ergebnisse dieser Arbeit zeigen die Machbarkeit und Wirksamkeit des GReQL Converter bei der Vereinfachung des Prozesses der Generierung von Feedback-Regeln für UML-Modelle.

1. Einleitung

Das erste Kapitel dieser Arbeit befasst sich mit der grundlegenden Motivation und definiert dann das allgemeine Problem, das gelöst werden soll. Es folgt eine Erläuterung der Zielsetzung und zum Schluss wird der Aufbau der Arbeit im Detail beschrieben.

1.1 Motivation

Nach den Prüfungsphasen in Bildungseinrichtungen auf Sekundarstufen und Universitätsniveau manifestiert sich regelmäßig die Herausforderung, der zahlreichen Korrekturaufgaben, welche Lehrer und Dozenten zu bewältigen haben. Diese Aufgabe kann mitunter mühsam und zeitaufwendig sein, insbesondere im Falle einer hohen Anzahl an Prüfungen [35]. Die Evaluation von Prüfungsleistungen von 20 Prüflingen mag noch als vertretbar erscheinen, doch wie verhält es sich in Lehrveranstaltungen, in denen sich 400 oder gar mehr Studierende beteiligen? Bereits in den 1960er Jahren wurden Bestrebungen unternommen, solche Probleme durch die Entwicklung von E-Assessment-Systemen zu lösen [33].

Heute ist die automatisierte und computerbasierte Bewertung von akademischen Arbeiten weit verbreitet im Hochschulbereich. Diverse Plattformen wie Blackboard [13], Open Academic Learning Platform (OPAL) [24], und Moodle [48] wurden konzipiert, um den Prüfungsprozess zu rationalisieren und vollständig zu automatisieren. Diese Plattformen finden insbesondere in der Beurteilung geschlossener Fragen wie Multiple-Choice-Aufgaben Anwendung, bei denen die Antworten klar und die Bewertung einfach automatisiert werden kann. Die Frage, die sich jedoch stellt, ist: Wie gestaltet sich die Evaluation offener Fragenstellungen in Übungen, wie beispielsweise die Erstellung konzeptioneller Modelle, bei denen Lösungsansätze variieren können, jedoch dennoch korrekt sind?

In der allgemeinen Informatikbildung ist es üblich, Modelle zur Beschreibung der Architektur eines Systems oder abstrakterer Konzepte heranzu-

ziehen. Diese Modelle können in Form von Diagrammen wie Unified Modeling Language (UML)-Klassendiagrammen, Sequenzdiagrammen oder Zustandsdiagrammen präsentiert werden. Bei einer Bewertungsaufgabe, bei der ein Studierender ein Modell auf Grundlage eines gegebenen Textes erstellen soll, sieht sich der Evaluierende oft mit der Herausforderung konfrontiert, das Ergebnis des Studierenden mit der erwarteten Lösung abzugleichen. Diese Form der Beurteilung kann jedoch diverse Schwierigkeiten aufwerfen:

1. Erstens, in der Modellierung existiert keine eindeutige Lösungsstrategie, was die Aufgabe komplexer gestaltet. Die Resultate jedes Studierenden müssen wiederholt mit der vorgeschlagenen Lösung verglichen werden, um sicherzustellen, dass die Arbeit alle wesentlichen Komponenten der Lösung berücksichtigt.
2. Das zweite Hindernis in diesem Evaluationsmodell ist die inhärente Subjektivität. Einige Studierende könnten vollständig funktionierende Modelle entwickeln, die jedoch von der erwarteten Lösung abweichen, und könnten dadurch benachteiligt werden. Dies kann zu einer gewissen Starre im Unterricht führen und verhindern, dass Studierende originelle Ansätze verfolgen.
3. Eine weitere bedeutende Herausforderung dieses Evaluationsmodells ist die potenzielle Subjektivität der Evaluierenden. Jeder Korrektor könnte eine individuelle Auslegung dessen haben, was als korrekte Antwort betrachtet wird, was zu Unstimmigkeiten in der Benotung und einer ungleichen Verteilung von Noten zwischen den Arbeiten der Studierenden führen kann. Dies könnte besonders problematisch sein, wenn mehrere Evaluierende dieselben Arbeiten bewerten, da dies zu signifikanten Unterschieden in den vergebenen Noten führen kann.
4. Zuletzt kann dieses Evaluationsmodell auch beträchtlich zeitaufwendig sein, insbesondere in Lehrveranstaltungen mit einer hohen Anzahl an Studierenden. Ein ausführlicher Abgleich der Studierendenarbeiten mit der Referenzlösung erfordert Zeit und könnte zu Verzögerungen bei der Mitteilung der Ergebnisse an die Studierenden und einer erhöhten Arbeitsbelastung für die Evaluierenden führen.

In diesem Zusammenhang könnte die Implementierung einer automatisierten Bewertung auf Basis klar definierter und spezifischer Kriterien, die in jeder Lösung berücksichtigt sein müssen, eine optimale Lösung darstellen. Eine automatisierte Evaluierung durch eine computerbasierte Anwendung könnte sämtliche dieser Probleme lösen, indem sie eine rasche, objektive

und vorurteilsfreie Bewertung ermöglicht. Die automatische Evaluierung von konzeptuellen Modellen ist kein neues Forschungsthema. Umfangreiche Untersuchungen in dieser Richtung wurden bereits durchgeführt, und verschiedene Ansätze wurden entwickelt, um dieses Ziel zu realisieren.

1.2 Zielsetzung und Abgrenzung

Im Rahmen dieser Masterarbeit wird das Ziel verfolgt, ein Verfahren zur (halb-)automatischen Erstellung von Bewertungsregeln auf Basis annotierter Musterlösungen zu entwickeln und in Form einer Softwareanwendung zu prototypisieren. Dieses Verfahren soll dazu beitragen, den Prozess der Regeldefinition zu vereinfachen und zu optimieren.

In der Fakultät für Informatik an der Universität Duisburg Essen (UDE) wird ein E-Assessment-System namens “JACK” [51] verwendet, um Studierende automatisch bei bestimmten Prüfungen und Übungen zu bewerten. JACK ist in der Lage, verschiedene Arten von Aufgaben, sowohl geschlossene als auch offene Fragen, zu bewerten. Unter den verschiedenen Aufgabentypen fallen auch Aufgaben zur Erstellung von UML-Diagrammen. JACK wurde entwickelt, um die Lösungen der Studierenden zu bewerten und Noten zu vergeben. Bei der Bewertung von UML-Diagrammen verwendet JACK regelbasierte Ansätze. Die Lehrenden müssen die Regeln definieren, die das Diagramm des Studierenden validieren und ihm eine Note zuweisen. Dieser Prozess der Regeldefinition ist jedoch zeitaufwändig und birgt das Risiko, dass kleine, aber bedeutende Unaufmerksamkeitsfehler und Flüchtigkeitsfehler auftreten.

Das angestrebte Verfahren soll daher Lehrkräften dabei unterstützen, solche Bewertungsregeln effizienter und fehlerminimiert zu erstellen, indem es auf vorhandene annotierte Musterlösungen zurückgreift. Durch die Entwicklung einer Softwareanwendung, die diesen Prozess (halb-)automatisch durchführt, wird das Ziel verfolgt, eine zeit- und ressourcensparende Lösung bereitzustellen. Damit kann die Qualität der automatischen Bewertung von UML-Diagrammen in JACK verbessert und der Aufwand für die Lehrkräfte reduziert werden.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in sechs Hauptkapitel, die jeweils einen spezifischen Aspekt des Forschungsgebiets beleuchten.

Im zweiten Kapitel werden die grundlegenden Konzepte und Informationen eingeführt, die für das Verständnis der Arbeit erforderlich sind. Dies umfasst eine Erklärung der E-Assessment-Systemen, konzeptioneller Modelle sowie der automatisierten Bewertung. Darüber hinaus werden verschiedene Ansätze zur automatisierten Bewertung im Detail betrachtet. Ein Überblick über den aktuellen Stand der Forschung auf diesem Gebiet wird ebenfalls gegeben, gefolgt von einer Einführung in das E-Assessment-System JACK.

Das dritte Kapitel widmet sich der detaillierten Analyse des zugrunde liegenden Problems. Hier wird der Prozess der Erstellung und Einreichung von Übungen sowie deren Bewertung durch die JACK-Plattform erläutert. Die spezifischen Herausforderungen und Probleme, die in dieser Arbeit adressiert werden, werden identifiziert und diskutiert.

Anschließend folgt das Kapitel vier, welches hauptsächlich das Konzept zur Bewältigung der zuvor im vorhergehenden Kapitel definierten Problematik behandelt. In diesem Kapitel wird eine theoretische Herangehensweise zur Problemlösung vorgestellt, und es wird ein methodischer Ansatz zur Problembewältigung skizziert. Die praktische Umsetzbarkeit dieser Lösungsstrategie wird im darauf folgenden Abschnitt ausführlich erörtert.

Das Kapitel fünf beschäftigt sich mit der praktischen Umsetzung der entwickelten Lösung. Hier werden die verwendeten Technologien und der Entwicklungsprozess vorgestellt. Die Gründe für die Auswahl bestimmter Technologien, Herangehensweisen und Implementierungsmethoden werden erläutert. Die Gesamtarchitektur des entwickelten Prototyps wird präsentiert und begründet.

Das sechste Kapitel widmet sich der Bewertung der entwickelten Lösung. Es werden Kriterien definiert, anhand derer der Prototyp bewertet wird. Der Evaluationsprozess wird erläutert, einschließlich der durchgeführten User Studies. Im Kapitel sieben werden die Ergebnisse der Arbeit kritisch diskutiert und interpretiert. Hierbei werden die Implikationen der Ergebnisse auf das Gesamtforschungsgebiet erörtert, eventuelle Limitationen der Arbeit aufgezeigt und mögliche Ansätze für zukünftige Forschung identifiziert.

Abschließend bietet Kapitel sechs eine Zusammenfassung der wichtigsten Erkenntnisse und Ergebnisse dieser Masterarbeit. Es werden auch Perspektiven für die Weiterentwicklung der vorgestellten Lösung sowie die Bereitstellung der vollständigen Dokumentation des entwickelten Tools dargelegt.

2. Hintergrund

Im vorliegenden Kapitel wird eine grundlegende Einführung in die zentralen Konzepte, Methoden und Ansätze gegeben, die für das Verständnis und die weiterführende Behandlung dieser Masterarbeit von Bedeutung sind. Zunächst erfolgt eine Erläuterung und Definition des Begriffs “E-Assessment-Systeme”, um eine klare Basis für die nachfolgende Diskussion zu schaffen. Im Anschluss wird die Thematik der “Konzeptionellen Modelle” ausführlich behandelt, wobei der Fokus auf den Modellen liegt, die in der Informatik allgemein angewandt werden. Darüber hinaus erfolgt eine umfassende Einführung in das Konzept der “Automatisierten Bewertung”, wobei insbesondere die verschiedenen Methoden und Ansätze, die in diesem Bereich von Relevanz sind, beleuchtet werden.

Des Weiteren wird ein Überblick über den aktuellen Stand der Technik (“State of the Art”) hinsichtlich der im Bereich der automatisierten Bewertung verwendeten Techniken und Tools gegeben. Schließlich erfolgt eine generelle Vorstellung des Tools “JACK”, welches im Rahmen dieser Arbeit eine herausragende Rolle spielt und in späteren Abschnitten ausführlicher behandelt wird.

Dieses Kapitel dient somit als Grundlage und Orientierungshilfe, um den Leser in die Thematik einzuführen und die notwendigen Begrifflichkeiten und Zusammenhänge zu vermitteln, die im weiteren Verlauf der Masterarbeit von zentraler Bedeutung sein werden.

2.1 E-Assessment-Systeme

Nach Eilers et al. sind E-Assessment-Systeme computergestützte Bildungstechnologien, die entwickelt wurden, um den Prozess der Bewertung von Lernleistungen in Bildungseinrichtungen zu automatisieren, zu verbessern und zu erweitern [18]. Diese Systeme ermöglichen die Erfassung, Bewertung und Analyse von Schüler- oder Studentenleistungen in einem digitalen Umfeld. E-Assessment-Systeme verwenden verschiedene Arten von

Aufgaben und Prüfungen, darunter Multiple-Choice-Fragen, Essays, Simulationen, interaktive Aufgaben und mehr, um das Wissen, die Fähigkeiten und die Kompetenzen der Lernenden zu bewerten.

2.1.1 Definition und Vorteile

Die Nutzung von E-Assessment-Systemen bietet eine Reihe von Vorteilen, die sich aus ihrer Fähigkeit zur Automatisierung ergeben. Diese Systeme verwenden vordefinierte Algorithmen und Kriterien, um die Leistung der Lernenden objektiv zu bewerten, was eine schnellere und effizientere Verarbeitung der Ergebnisse ermöglicht. Dies ermöglicht es Lehrern, mehr Zeit für pädagogische Aktivitäten aufzuwenden, anstatt sich mit manuellen Prüfungen und Aufgabenbewertungen zu befassen. Die Art der Verwendung eines E-Assessment-Systems hängt von den angestrebten Zielen ab [64]. Es handelt sich nicht nur um ein Werkzeug zur Bewertung von Schülern am Ende eines Semesters (summative), sondern sie können auch am Lernprozess der Schüler teilnehmen (formative).

Die summative Bewertung ist eine Art von Bewertung, die am Ende eines definierten Lernzeitraums durchgeführt wird, wie z.B. am Ende eines Kurses, eines Moduls oder eines Schuljahres. Ihr Hauptziel besteht darin, die Gesamtleistung der Schüler zu messen und festzustellen, inwieweit sie die zuvor festgelegten Lernziele erreicht haben [64]. Diese Art der Bewertung wird in der Regel nach Abschluss des Lernens durchgeführt, oft in Form einer Abschlussprüfung oder eines Abschlussprojekts. Die summative Bewertung wird hauptsächlich verwendet, um eine Note zu vergeben oder Entscheidungen wie die Promotion oder den Erwerb eines Abschlusses zu treffen. Ihr Fokus liegt weniger auf detailliertem Feedback an die Schüler als vielmehr auf der Bewertung ihres Kompetenzniveaus [64].

Im Gegensatz zur summative Bewertung handelt es sich bei der formativen Bewertung um einen kontinuierlichen Prozess, der während des Lernens stattfindet. Das Hauptziel ist es, den Fortschritt der Schüler zu verfolgen, ihre Bedürfnisse zu identifizieren und ihnen bei der Verbesserung ihrer Leistung zu helfen [29]. Formative Bewertungen werden regelmäßig während des Lernprozesses durchgeführt, oft in Form von Quiz, Übungen oder Klassendiskussionen. Sie bieten den Schülern konstruktives Feedback, das sie dazu ermutigt, ihr eigenes Lernen zu verstehen, ihre Schwächen zu identifizieren und sich entsprechend zu verbessern. Darüber hinaus leitet die formative Bewertung die Lehrer an und zeigt ihnen notwendige Anpassungen in ihrer Unterrichtsgestaltung auf, um den spezifischen Bedürfnissen der Schüler gerecht zu werden. Dies fördert effektiveres und

zielgerichtetes Lernen [29].

Angesichts dieser Unterschiede kann man je nach Art der Übung verschiedene Vorteile der Verwendung von E-Assessment-Systemen nutzen:

1. **Effizienz und Zeitersparnis:** Online-Bewertungssysteme ermöglichen eine automatische Bewertung von Prüfungen und Aufgaben, was den Zeitaufwand für Lehrer erheblich reduziert. Lehrer können so mehr Zeit für die Entwicklung von Lehrinhalten und die Unterstützung der Lernenden aufwenden [5].
2. **Skalierbarkeit:** Diese Systeme sind äußerst skalierbar und können gleichzeitig eine große Anzahl von Prüfungen und Aufgaben verwalten. Dies ist besonders nützlich für Bildungseinrichtungen, die viele Schüler oder Studenten betreuen [29] [5].
3. **Schnelles Feedback:** Online-Bewertungssysteme bieten den Lernenden sofortiges Feedback. Dies trägt zur Beschleunigung des Lernprozesses bei und ermöglicht es den Studenten, ihre Leistung sofort zu überprüfen und zu verbessern [29] [5].
4. **Individualisierung:** Einige E-Assessment-Systeme ermöglichen es, Prüfungen und Aufgaben an die individuellen Bedürfnisse und Ziele der Lernenden anzupassen. Dies fördert personalisiertes Lernen und ermöglicht es den Studenten, in ihrem eigenen Tempo zu arbeiten [5].
5. **Umfassende Datenanalyse:** Diese Systeme sammeln umfangreiche Daten zur Leistung der Lernenden. Durch die Analyse dieser Daten können Bildungseinrichtungen Trends identifizieren, Schwächen und Stärken erkennen und das Lehrplanangebot entsprechend anpassen [5].
6. **Reduzierung von Betrug und Plagiat:** Online-Bewertungssysteme verfügen über Sicherheitsmechanismen, die Betrug und Plagiat bei Prüfungen und Aufgaben minimieren. Dies trägt zur Integrität des Bewertungsprozesses bei [5].
7. **Flexibilität und Zugänglichkeit:** Online-Bewertungssysteme ermöglichen die Durchführung von Prüfungen und Aufgaben in verschiedenen Umgebungen, einschließlich Online- und Hybrid-Lernumgebungen. Dies bietet den Studenten Flexibilität und Zugänglichkeit zu Bildungsbewertungen [29] [5].

Zusammenfassend kann festgehalten werden, dass E-Assessment-Systeme als eine effiziente und vielseitige Methode zur Bewertung von Lernenden betrachtet werden können, die sowohl formative als auch summative Bewertungen ermöglicht. Zahlreiche Vorteile werden durch sie ge-

boten. Mit diesem Verständnis werden im nächsten Kapitel die Mögliche Aufgabentypen in E-Assessment-Systemen nähere Einblicke gewonnen.

2.1.2 Mögliche Aufgabentypen

In der Erstellung von Prüfungen zur Bewertung von Schülern und Studierenden lassen sich im Allgemeinen zwei primäre Aufgabentypen identifizieren, die konsequent zu zwei verschiedenen Arten von Übungsszenarien führen: **Geschlossene Fragen** und **Offene Fragen** [64]. Diese Klassifikationen sind jeweils durch spezifische Charakteristika gekennzeichnet und bieten Vorzüge, die auf präzise Beurteilungsziele ausgerichtet sind [39].

Geschlossene Fragen (Closed questions)

Die Kategorie der Geschlossenen Fragen, auch als “closed questions” bezeichnet, repräsentiert eine geläufige Form von Aufgaben in E-Assessment-Systemen. Diese Fragen zeichnen sich durch ihre Fähigkeit aus, eine begrenzte Auswahl vorab definierter Antwortmöglichkeiten anzubieten, aus denen die Lernenden die adäquate Antwort auswählen müssen [29]. In der Regel umfasst diese Kategorie Multiple-Choice-Fragen, Wahr-Falsch-Fragen und ähnliche Formate. Die Vorzüge dieses Aufgabentyps sind vielschichtig. Primär sind sie als effektive Instrumente zur Beurteilung des Verständnisses spezifischer Konzepte und der Retention von Wissen zu betrachten. Darüber hinaus ermöglichen sie eine prompte und automatisierte Evaluierung, was unzweifelhaft im Interesse der Lehrenden und Prüfenden liegt. Allerdings sollte in Betracht gezogen werden, dass geschlossene Fragen in ihrer Fähigkeit, kritisches Denken, Kreativität und eigenständige Problemlösungsfähigkeiten zu bewerten, eingeschränkt sein können. Infolgedessen eignen sie sich vorwiegend für Beurteilungsziele, die auf die Prüfung grundlegender Kenntnisse und konzeptioneller Fertigkeiten abzielen [29]. Einige Beispiele für eine typische Übung mit geschlossener Frage:

1. Multiple-Choice-Fragen: Bei Multiple-Choice-Fragen wird den Lernenden eine Frage gestellt, und sie müssen aus einer Liste von vorgegebenen Antwortmöglichkeiten die richtige auswählen. Diese Art von Frage eignet sich gut, um das Verständnis von Fakten, Konzepten und Definitionen zu überprüfen [10] [11].
2. Wahr/Falsch-Fragen: Wahr/Falsch-Fragen erfordern, dass die Lernenden entscheiden, ob eine gegebene Aussage wahr oder falsch ist. Diese Fragen sind besonders nützlich, um das Verständnis von Sachverhalten zu überprüfen [37].

3. Zuordnungsaufgaben (Matching Questions): Bei Zuordnungsaufgaben müssen die Lernenden Elemente aus zwei verschiedenen Listen miteinander in Beziehung setzen. Dies kann verwendet werden, um das Verständnis von Zusammenhängen und Beziehungen zwischen Konzepten zu prüfen [29].
4. Lückentexte (Fill-in-the-Blanks): Bei Lückentexten müssen die Lernenden fehlende Wörter oder Phrasen in einem Satz oder Text ergänzen. Diese Art von Aufgabe kann verwendet werden, um das Verständnis von Kontext und Details zu überprüfen [29].

Der Vorteil von geschlossenen Fragen in E-Assessment-Systemen liegt in ihrer klaren Struktur und ihrer objektiven Bewertbarkeit. Sie ermöglichen eine schnelle Auswertung und sind besonders geeignet, um das Wissen über Fakten und Grundlagen zu überprüfen. Darüber hinaus können sie automatisch bewertet werden, was die Effizienz bei der Beurteilung von Lernenden in großen Gruppen erhöht.

Offene Fragen (Open-Ended questions)

Dem gegenüber bieten Offene Fragen, auch als “open-ended questions” bezeichnet, einen anpassungsfähigeren und nuancierteren Ansatz zur Evaluation. Im Kontrast zu geschlossenen Fragen präsentieren sie keine vordefinierten Antworten und gestatten den Lernenden, ihre Gedanken selbstständig zu artikulieren [64]. Offene Fragen können in verschiedenen Formen gestellt werden, wie schriftliche Antworten, Lösungen von Problemen, argumentative Erläuterungen oder ähnliche Formate. Einer der Hauptvorteile dieser Fragestellungen liegt darin, dass sie die Beurteilung von kritischem Denken, Kreativität, Synthese- sowie schriftlichen oder mündlichen Ausdrucksfertigkeiten ermöglichen [64]. Sie bieten zudem detailliertere Einblicke in das Verständnis und die Fertigkeiten der Lernenden. Es ist jedoch bedeutend zu betonen, dass die Bewertung der Antworten auf diese Fragen häufig komplexer und subjektiver ist, eine menschliche Bewertung erfordert und länger dauern kann als die automatisierte Auswertung [29]. Des Weiteren könnten offene Fragen aufgrund ihres ressourcenintensiven Charakters unter Umständen weniger geeignet sein für umfangreiche Beurteilungen. Einige Beispiele für eine typische Übung mit offener Frage:

1. Essay-Fragen: Essay-Fragen sind offene Fragen, bei denen die Lernenden in ausführlichen schriftlichen Antworten ihr Wissen, ihre Analysefähigkeiten und ihre Argumentationsfähigkeiten darlegen müssen. Diese Art von Frage eignet sich gut, um komplexe Konzepte zu vertiefen und kritisches Denken zu fördern [29].

2. Fallstudien und Szenario-basierte Fragen: Bei diesen Fragen werden den Lernenden reale oder fiktive Szenarien oder Fallstudien vorgelegt, die sie analysieren und Lösungen oder Empfehlungen entwickeln müssen. Dies fördert die Anwendung von Wissen auf komplexe Probleme [29].
3. Reflexionsfragen: Reflexionsfragen ermutigen die Lernenden dazu, über ihr eigenes Lernen, ihre Erfahrungen und ihre Entwicklung nachzudenken. Diese Art von Frage ist besonders nützlich, um metakognitive Fähigkeiten zu fördern und das Bewusstsein für den Lernprozess zu schärfen [29].
4. Problemstellungen und Aufgaben mit freier Lösung: Bei dieser Art von Fragen werden den Lernenden komplexe Probleme oder Aufgaben gestellt, für die es keine festen Lösungen gibt. Die Lernenden müssen ihre eigenen Lösungen entwickeln und ihre Entscheidungen begründen [29].

Offene Fragen bieten den Lernenden die Möglichkeit, ihr Verständnis und ihre Fähigkeiten auf eine tiefere Weise zu zeigen, die über reine Faktenkenntnisse hinausgeht. Sie fördern kritisches Denken, Problemlösungsfähigkeiten und die Fähigkeit zur Kommunikation komplexer Ideen. Allerdings erfordert die Bewertung von offenen Fragen in der Regel mehr Zeit und Aufwand von Lehrkräften oder Experten, da die Antworten vielfältig und subjektiver Natur sein können.

Übungen, bei denen aus einem Text ein UML-Diagramm erstellt werden soll, gehören in der Regel zu den offenen Fragen in E-Assessment-Systemen. Dies liegt daran, dass sie von den Lernenden verlangen, nicht nur Faktenwissen anzuwenden, sondern auch kreativ denken und die Informationen aus dem Text analysieren müssen, um ein geeignetes UML-Diagramm zu erstellen. Diese Übungen erfordern von den Lernenden, dass sie ein tieferes Verständnis für das gegebene Thema entwickeln und die Informationen aus dem Text in einen visuellen Kontext übertragen können [70].

Zusammenfassend sind geschlossene Fragen und offene Fragen zwei unterschiedliche Ansätze zur Bewertung in E-Assessment-Systemen. Geschlossene Fragen eignen sich gut zur Bewertung von Grundkenntnissen und zur automatischen Bewertung, während offene Fragen eine erhöhte Flexibilität bieten, um komplexe Fähigkeiten zu bewerten, obwohl sie möglicherweise eine intensivere Bewertung erfordern. Die Auswahl zwischen diesen beiden Arten von Aufgaben hängt von den spezifischen Bewertungszielen, der Art der zu bewertenden Fähigkeiten sowie den verfügbaren Ressourcen für die Bewertung und Auswertung ab. In Kom-

bination tragen diese beiden Fragekategorien wesentlich zu einer umfassenden und ausgewogenen Bewertung der Lernenden im Kontext des E-Assessment bei.

2.2 Konzeptionelle Modelle

Ausgehend von der Definition ist ein konzeptionelles Modell eine abstrakte Darstellung oder eine konzeptionelle Struktur, die darauf abzielt, Ideen, Beziehungen, Konzepte oder Entitäten eines bestimmten Bereichs zu beschreiben, ohne in konkrete Details oder Implementierungsdetails einzugehen. Solche Modelle werden häufig in verschiedenen Bereichen wie Informatik, Wissenschaft, Ingenieurwissenschaften, Management, Philosophie usw. verwendet, um das Verständnis eines komplexen Themas zu klären, die Kommunikation und Diskussion zu erleichtern und als Grundlage für die Gestaltung oder Analyse konkreterer Systeme zu dienen [2].

Konzeptionelle Modelle können verschiedene Formen annehmen, darunter Diagramme, Schemata, grafische Darstellungen, textuelle Beschreibungen sowie mathematische Repräsentationen. Diese dienen häufig als initialer Schritt innerhalb des Modellierungs- oder Problemlösungsprozesses, um die grundlegenden Konzepte und Zusammenhänge zu erfassen, bevor man sich in die detaillierten Facetten vertieft [2]. In der Fachdisziplin der Informatik, zum Beispiel, wird ein konzeptionelles Modell genutzt, um Schlüsselparameter und Verknüpfungen innerhalb einer Datenbank zu definieren, ohne Einzelheiten zur Datenspeicherung oder -abfrage preiszugeben [2]. Es kann auch verwendet werden, um die logische Architektur eines Systems zu beschreiben, wobei die Beziehung zwischen Objekten, Klassen und verschiedenen Entitäten hervorgehoben wird.

Im Allgemeinen nutzen Modellierungssprachen in den betreffenden Fachgebieten Notationen, die graphische Symbole einschließen und in zweidimensionalen visuellen Darstellungen resultieren [49]. Solche Darstellungen sind gebräuchlicherweise als Diagramme bekannt, und dies findet oft seinen Ausdruck in den Namen spezifischer Modelltypen wie “Entity-Relationship-Diagramm” oder “UML-Klassendiagramm”. Falls grafische Symbole innerhalb eines Modells eingesetzt werden, erfolgt ihre Annotation üblicherweise durch textliche Kennzeichnungen, um ihre Relevanz im Kontext des modellierten Objekts zu präzisieren. Die Modellierungssprachen, die im Rahmen der initialen Forschung herausragen (wie Entity-Relationship-Diagramm (ERD), UML, EPC,Business Process Model and Notation (BPMN) und Petri-Netze), repräsentieren beispielhafte Instanzen solcher graphischen Modellierungssprachen [70].

Im Fachgebiet der Informatik gibt es verschiedene Arten von konzeptionellen Modellen, die je nach ihrem Anwendungsbereich und Ziel unterschiedliche Formen und Eigenschaften aufweisen. Hier sind einige häufig vorkommende Arten von konzeptionellen Modellen in der Informatik:

1. **Entity-Relationship-Modelle (ER-Modelle):** Diese Modelle werden verwendet, um die Struktur von Datenbanken zu beschreiben, indem sie Entitäten (Objekte oder Konzepte) und deren Beziehungen zueinander darstellen. ER-Modelle verwenden typischerweise Diagramme, um Entitäten, Attribute und Beziehungen grafisch darzustellen [26].
2. **UML-Diagramme (Unified Modeling Language):** UML ist eine weit verbreitete Modellierungssprache in der Softwareentwicklung. Sie umfasst verschiedene Diagrammtypen, die zur Modellierung von Softwarearchitekturen, Prozessen und Verhaltensweisen verwendet werden [57].
3. **Data Flow Diagrams (DFD):** DFDs werden verwendet, um den Datenfluss und die Datenverarbeitung in Informationssystemen darzustellen. Sie zeigen, wie Daten zwischen Prozessen, Datenlagern und externen Entitäten fließen [42].
4. **BPMN-Diagramme (Business Process Model and Notation):** BPMN ist eine Modellierungssprache, die sich auf die Darstellung von Geschäftsprozessen konzentriert. Mit BPMN-Diagrammen können Abläufe, Aktivitäten und Entscheidungen innerhalb eines Unternehmensmodells visualisiert werden [73].
5. **Petri-Netze:** Petri-Netze sind mathematische Modelle, die zur Modellierung und Analyse von parallelen und verteilten Systemen verwendet werden. Sie sind besonders nützlich bei der Modellierung von Prozessen in der Softwareentwicklung und der Kommunikation zwischen Komponenten [55].
6. **Systemarchitekturmodelle:** Diese Modelle bieten eine Übersicht über die Architektur eines Software- oder Informationssystems und zeigen die Hauptkomponenten und ihre Interaktionen.

Diese Arten von konzeptionellen Modellen dienen dazu, komplexe Systeme, Prozesse und Datenstrukturen in der Informatik zu erfassen, zu analysieren und zu kommunizieren. Die Wahl des geeigneten Modells hängt von den spezifischen Anforderungen und Zielen eines Projekts ab.

Das Thema dieser Masterarbeit hat als Anwendungsfall Aufgabe mit offenen Fragen, die sich mit der Modellierung mit UML-Diagrammen befassen und bewertet werden sollen. Die Unified Modeling Language (UML) ist eine standardisierte und visuelle Modellierungssprache, die in der Softwareentwicklung und Systemmodellierung weit verbreitet ist. UML dient dazu,

komplexe Systeme, insbesondere Softwareanwendungen, zu beschreiben, zu analysieren, zu entwerfen und zu dokumentieren. Sie wurde erstmals in den 1990er Jahren von Grady Booch, James Rumbaugh und Ivar Jacobson entwickelt und hat sich seitdem zu einem Industriestandard für die Modellierung von Software- und Systemarchitekturen entwickelt [14].

UML bietet eine breite Palette von Diagrammtypen, darunter Klassendiagramme, Aktivitätsdiagramme, Sequenzdiagramme, Zustandsdiagramme und viele mehr. Jeder Diagrammtyp konzentriert sich auf bestimmte Aspekte eines Systems und ermöglicht es den Entwicklern und Ingenieuren, die verschiedenen Elemente und deren Beziehungen in einer klaren und leicht verständlichen visuellen Darstellung festzuhalten [14]. Dies fördert eine bessere Kommunikation und Zusammenarbeit zwischen den Mitgliedern eines Entwicklungsteams sowie zwischen den verschiedenen Interessengruppen eines Projekts.

Die Verwendung von UML in der Softwareentwicklung bietet eine Reihe von Vorteilen, darunter die Möglichkeit, Systeme zu abstrahieren, zu modularisieren und zu dokumentieren, was die Entwicklung, Wartung und Erweiterung von Software erleichtert [14]. Darüber hinaus unterstützt UML die frühzeitige Fehlererkennung und das systematische Design von Softwarelösungen, was zu einer höheren Qualität und Zuverlässigkeit von Anwendungen führt. Aufgrund seiner weitverbreiteten Akzeptanz und seiner Fähigkeit, komplexe Ideen in leicht verständlichen Diagrammen darzustellen, spielt UML eine entscheidende Rolle in der modernen Softwareentwicklung und Systemmodellierung und bildet die Grundlage für die Erstellung und den Austausch von Modellen und Entwurfsmustern in der Industrie [14].

2.3 Automatisierte Bewertung

2.3.1 Definition und Abgrenzung

Die automatisierte, computergestützte Bewertung von akademischen Einreichungen, wird in der Hochschulbildung häufig verwendet. Informatikbasierte automatisierte Bewertungssysteme existieren bereits seit den 1960er Jahren [70], während automatisierte Bewertung von Diagrammen seit etwa zwei Jahrzehnten bekannt ist. Diese Systeme werden eingesetzt, um sowohl offene Aufgaben zu bewältigen, als auch geschlossene Fragen. Bei einer Bewertung, in der ein Student ein Modell aus einem gegebenen Text erstellen soll, ist der Korrektor oft dazu verpflichtet, einen Vergleich zwischen dem vom Studenten vorgelegten Ergebnis und der erwarteten

Lösung durchzuführen. Allerdings kann dieses Korrekturmodell mehrere Probleme aufwerfen:

1. Zunächst einmal, da es im Bereich der Modellierung keine eindeutige Lösung gibt, wird diese Aufgabe weniger offensichtlich, da die Ergebnisse jedes Studenten mehrmals mit der vorgeschlagenen Lösung verglichen werden müssen, um sicherzustellen, dass die Arbeit des Studenten alle wesentlichen Komponenten der Lösung berücksichtigt [23].
2. Das zweite Hindernis, das aus diesem Korrekturmodell resultiert, liegt in der inhärenten Subjektivität dieser Methode. Einige Studenten könnten Modelle entwickeln, die vollkommen funktionsfähig sind, aber von der erwarteten Lösung abweichen, und für diese Abweichung benachteiligt werden. Darüber hinaus kann dies zu einer gewissen Starrheit in der Lehre führen und die Studenten daran hindern, originelle Ansätze zu erkunden [46] [31].
3. Eine weitere wichtige Herausforderung dieses Korrekturmodells besteht in der potenziellen Subjektivität der Korrektoren. Jeder Korrektor kann seine eigene Interpretation dessen haben, was eine richtige Antwort ausmacht, was zu Inkonsistenzen bei den Bewertungen und einer ungleichmäßigen Verteilung der Noten zwischen den Studentenarbeiten führen kann. Dies kann besonders problematisch werden, wenn mehrere Korrektoren dieselben Arbeiten bewerten, was zu erheblichen Unterschieden in den vergebenen Noten führen kann [46].
4. Schließlich kann dieses Korrekturmodell auch zeitaufwändig sein, insbesondere in Kursen mit einer großen Anzahl von Studenten. Der detaillierte Vergleich zwischen den Studentenarbeiten und der Referenzlösung erfordert Zeit und kann zu Verzögerungen bei der Mitteilung der Ergebnisse an die Studenten führen und die Arbeitsbelastung der Korrektoren erhöhen.

In diesem Zusammenhang könnte die Einführung einer automatisierten Korrektur auf der Grundlage klar definierter spezifischer Kriterien, die in jeder Lösung zwingend enthalten sein müssen, eine optimale Lösung darstellen. Eine automatisierte Korrektur durch ein Computerprogramm könnte jedes dieser Probleme lösen, indem sie eine schnelle, faire und vorurteilsfreie Bewertung ermöglicht.

2.3.2 Automatisierte Bewertungsmethoden für UML

Die Bewertung von UML-Diagrammen ist in verschiedenen Bildungskontexten von entscheidender Bedeutung, sei es in Informatiklehrgängen an

Universitäten oder in der beruflichen Weiterbildung. Dabei kann die manuelle Bewertung von UML-Diagrammen zeitaufwändig und subjektiv sein, insbesondere wenn es sich um eine große Anzahl von Diagrammen handelt. Um diese Herausforderungen zu bewältigen und eine effiziente und objektive Bewertung zu gewährleisten, werden verschiedene Methoden und Ansätze zur automatisierten Bewertung entwickelt und angewendet:

1. **Methoden zur Modellvergleich:** Diese Methoden beinhalten den Vergleich des bewerteten Modells mit einem oder mehreren Lösungsmodellen. Dieser Vergleich kann mithilfe von Ähnlichkeitsmaßen, wie Ähnlichkeitsmaßen oder Graphenabgleich, durchgeführt werden. Dabei werden Ähnlichkeiten und Unterschiede zwischen dem bewerteten Modell und den Lösungsmodellen ermittelt [70] [19].
2. **Regelbasierte Ansätze:** Regelbasierte Ansätze verwenden vordefinierte Regeln oder Kriterien zur Bewertung des bewerteten Modells. Diese Regeln können verschiedene Formen annehmen, darunter Graphabfragen, Eigenschaften, Metriken, Mängel oder Suchmuster innerhalb des bewerteten Modells. Die Bewertung erfolgt anhand der Einhaltung oder Nichteinhaltung dieser Regeln [70] [66].
3. **Constraints-basierte Ansätze:** Diese Ansätze sind eng mit regelbasierten Ansätzen verwandt und werden häufig in intelligenten Tutorensystemen eingesetzt. Sie beinhalten die Anwendung von Einschränkungen oder Regeln zur Bewertung des bewerteten Modells. Diese Einschränkungen können sich auf die Struktur, die Semantik oder andere Aspekte des Modells beziehen [70] [32].
4. **Methoden des maschinellen Lernens:** Einige aktuelle Forschungs-Artikel präsentieren Ansätze, die auf Methoden des maschinellen Lernens setzen, um die automatisierte Bewertung durchzuführen. Hierbei werden maschinelle Lernmodelle trainiert, um Modelle auf Grundlage vorheriger manueller Bewertungen zu bewerten. Dies beinhaltet das Training von Modellen zur Identifizierung von Mustern, Anomalien und potenziellen Problemen in UML-Diagrammen [70] [15] [36].
5. **Andere Techniken:** Neben den genannten Ansätzen gibt es verschiedene andere Techniken, wie die Simulation des bewerteten Modells, Teststrategien, die Gruppierung von Modellen und die Ausrichtung des bewerteten Modells mit einer annotierten textuellen Beschreibung [70] .
   (a) **Werkzeuge zur Modelltransformation und Codegenerierung:** Diese Werkzeuge ermöglichen die automatische Generierung von Code aus UML-Diagrammen. Sie helfen dabei, den manuel-

len Aufwand zur Übersetzung eines UML-Designs in ausführbaren Code zu reduzieren und die Korrektheit des Diagramms zu überprüfen. Dies geschieht durch die automatisierte Erzeugung von ausführbarem Code aus den Elementen und Strukturen des UML-Diagramms und ermöglicht die anschließende Überprüfung der Korrektheit des generierten Codes [67].

Diese verschiedenen Ansätze und Techniken tragen dazu bei, die automatisierte Bewertung von UML-Diagrammen in der Bildung und anderen Anwendungsbereichen effektiver und vielfältiger zu gestalten. Sie ermöglichen eine präzise und umfassende Bewertung von Modellen und tragen zur Verbesserung der Qualität von UML-Diagrammen bei.

2.3.3 Herausforderung der automatisierten Bewertung

Das vorrangige Ziel der automatisierten Bewertung besteht in der Verleihung von Bewertungen, was als eine Klassifizierungsaufgabe innerhalb des Bewertungskontexts betrachtet werden kann [40]. In Fällen von offenen Aufgaben erfolgt die Bewertung anhand der relativen Position der eingereichten Lösung innerhalb des umfassenden Lösungsraums. Dieser Lösungsraum beinhaltet eine Bandbreite von Antwortmöglichkeiten, die von vollständigen Lösungen über partielle Lösungen bis hin zu ungültigen oder nicht akzeptablen Lösungen reicht. Dieser methodische Ansatz zur automatisierten Bewertung, der auf der Unterscheidung und Beurteilung der Qualität von Antworten basiert, findet in verschiedenen Bereichen breite Anwendung. Beispiele hierfür sind die automatisierte Bewertung von Essays [71], die Beurteilung von Programmieraufgaben [27] sowie die Modellierung und Bewertung von Modellen in diversen Fachdisziplinen [62]. Dieser Ansatz ermöglicht eine effiziente, objektive und skalierbare Bewertung von eingereichten Arbeiten, und er trägt wesentlich dazu bei, den Bewertungsprozess zu rationalisieren und zu standardisieren.

Im Kontext der Erstellung eines Diagramms (UML, Sequenzdiagramm, Entity-Relationship-Diagramm oder andere) aus einem Text, der ein System beschreibt, wird diese Aufgabe in der Regel als offen angesehen. Die Studierenden müssen die im Text beschriebenen Konzepte, Entitäten und Beziehungen abstrakt interpretieren und darstellen, was bedeutet, dass es normalerweise keine eindeutige Antwort oder vordefinierte Lösung gibt. Verschiedene Studierende können leicht unterschiedliche Diagramme erstellen, um die gleiche textuelle Beschreibung darzustellen.

Da diese Aufgabe offen und subjektiv ist, müssen die Prüfer die Kreativität und das Verständnis jedes Studierenden bewerten. Ihr Ziel ist es zu

bestimmen, ob das erstellte Diagramm die im Ausgangstext beschriebenen Konzepte und Beziehungen angemessen erfasst [21]. Diese Komplexität macht den Ansatz auf Grundlage von Regeln für die Bewertung dieser Aufgabe geeignet, da er es ermöglicht, vordefinierte Kriterien anzuwenden und spezifisches Feedback entsprechend dieser Kriterien bereitzustellen.

2.4 State of the Art

In diesem Kapitel wird die Diskussion auf bestimmte, mehr oder weniger aktuelle Technologien ausgedehnt, die im Kontext des E-Assessments verschiedene Methoden aus dem vorherigen Kapitel verwenden.

2.4.1 Machine learning zur Bewertung von UML

Der Artikel mit dem Titel “Automatic Assessment of Students’ Software Models Using a Simple Heuristic and Machine Learning” [15] stammt von den Autoren Younes Boubekeur, Gunter Mussbacher und Shane McIntosh. In ihrem Artikel stellen sie einen innovativen Ansatz zur Bewertung von Studienarbeiten in Modellierungskursen vor. Ihr Hauptziel besteht darin, den zeitaufwändigen und subjektiven Bewertungsprozess von UML-Diagrammen zu bewältigen, der in der Softwaretechnikausbildung weit verbreitet ist. Die vorgeschlagene Methodik kombiniert einen einfachen heuristischen Algorithmus mit fortgeschrittenen maschinellen Lernverfahren, um nicht nur hochwertige Studienarbeiten zu identifizieren, sondern auch ungefähre Noten vorherzusagen.

Die Autoren beginnen ihre Arbeit damit, einen Überblick über bestehende Ansätze zur Bewertung von UML-Diagrammen zu geben. Diese Ansätze umfassen manuelle Bewertung, automatische Bewertung mithilfe regelbasierter Systeme und automatische Bewertung unter Verwendung von maschinellem Lernen. Anschließend stellen die Autoren ihren innovativen Ansatz vor, der einen einfachen heuristischen Algorithmus verwendet, um die Einreichungen der Studierenden mit einer Idealvorlage zu vergleichen [15]. Dieser heuristische Algorithmus basiert auf dem Prinzip, die Unterschiede zwischen der Einreichung des Studierenden und der Idealvorlage zu quantifizieren und anschließend eine Punktzahl aufgrund dieser Unterschiede zuzuweisen [34]. Darüber hinaus setzen die Autoren maschinelle Lernverfahren ein, um ungefähre Noten vorherzusagen, basierend auf den von dem heuristischen Algorithmus generierten Punktzahlen (siehe 2.1).

Um die Effektivität ihres vorgeschlagenen Ansatzes zu validieren, führten die Autoren eine empirische Studie mit 50 Studierenden durch, die an ei-

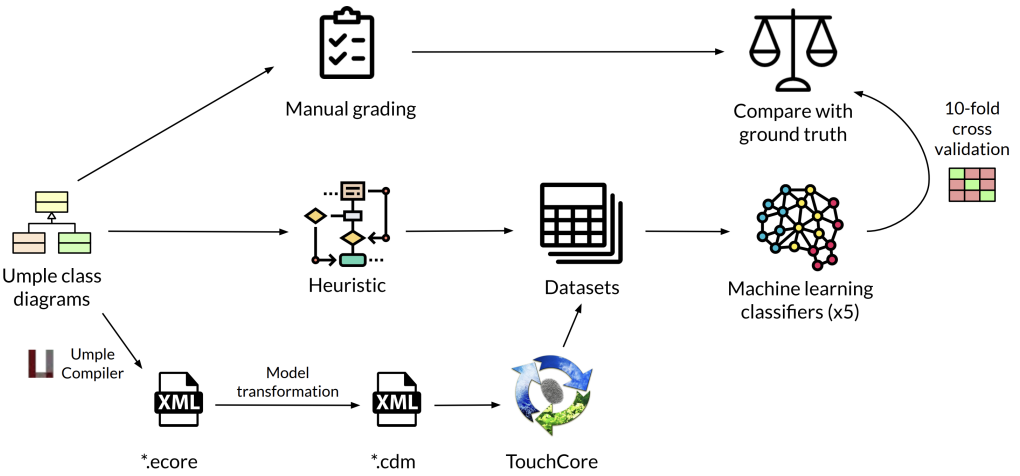


Abbildung 2.1: Ansatz der Studie [15].

nem Softwaretechnikkurs teilnahmen. Die Studierenden wurden gebeten, UML-Diagramme für ein gegebenes Problem einzureichen, und die Autoren wandten ihren innovativen Ansatz zur Bewertung dieser Einreichungen an. Um die Zuverlässigkeit der Bewertung sicherzustellen, wurden auch zwei menschliche Prüfer beauftragt, dieselben Einreichungen unabhängig voneinander zu bewerten.

Die Ergebnisse der Studie zeigen, dass der vorgeschlagene Ansatz mit der menschlichen Bewertung hinsichtlich der Identifizierung hochwertiger Einreichungen vergleichbar ist und erstaunlich präzise ungefähre Noten vorhersagen kann. Darüber hinaus verglichen die Autoren ihren Ansatz mit einer komplexen regelbasierten Technik und stellten fest, dass ihre Methode in Bezug auf Effizienz und Wartbarkeit überlegen ist [15].

Zusammenfassend bietet der innovative Ansatz eine vielversprechende Lösung zur Bewertung von UML-Diagrammen im Bereich der Softwaretechnikausbildung. Durch die Kombination eines einfachen heuristischen Algorithmus mit fortgeschrittenen maschinellen Lernverfahren identifiziert ihr Ansatz nicht nur hochwertige Einreichungen, sondern bietet auch eine effiziente Möglichkeit zur Vorhersage ungefährer Noten. Die Autoren betonen das potenzielle Zeitersparnis und die Reduzierung der Subjektivität im Bewertungsprozess und schlagen vor, dass ihr Ansatz auf andere Bereiche erweitert werden kann, sofern es eine klare Definition einer Idealvorlage gibt und eine Methode existiert, um die Einreichungen der Studierenden mit dieser Idealvorlage zu vergleichen.

2.4.2 Bewertung auf Basis von Ähnlichkeitsmaßen

Die Autoren des Artikels “A Different Approach on Automated Use Case Diagram Semantic Assessment” [19] sind Daniel Siahaan, Siti Rochimah, Reza Fauzan und Evi Triandini. In diesem Artikel stellen die Autoren eine innovative Methode zur automatischen semantischen Bewertung von Anwendungsfall-Diagrammen im Rahmen von UML-Diagrammen vor. Das Hauptziel dieser Methode besteht darin, die Herausforderungen anzugehen, vor denen Pädagogen bei der Bewertung von Anwendungsfall-Diagrammen stehen, sowohl in zwischenmenschlicher als auch intrapersonaler Hinsicht.

Die Autoren verwenden einen semantischen Bewertungsansatz, der in zwei Hauptdimensionen unterteilt ist: Eigenschaft (property) und Beziehung (relationship). Alle in dieser Bewertung verwendeten Informationen stammen aus Beschriftungen, die aus einem XML Metadata Interchange (XMI)-Dokument [1] übersetzt wurden. Zur Messung der Ähnlichkeit zwischen den Beschriftungen verwenden die Autoren die Kosinus-Ähnlichkeit und nutzen WuPalmer zur Unterstützung von WordNet-Suchen [19] [38].

Für ihre empirische Studie sammelten die Autoren einen Datensatz aus drei unterschiedlichen Projekten: Outlay, QuickBill und dem Restaurant Management System (RMS). Diese Projekte umfassen die Finanzaufzeichnung, den Point of Sale und das Restaurant-Bestellsystem. Der Quellcode für diese Projekte wurde von GitHub bezogen und folgt einem objektorientierten Ansatz. Den Studierenden wurden Demonstrationen zur Funktionsweise jedes Projekts gezeigt, und anschließend wurden sie aufgefordert, Anwendungsfall-Diagramme auf der Grundlage ihres Verständnisses der Projekte und der zugehörigen Terminologie zu erstellen. Insgesamt sammelten die Autoren 36 Anwendungsfall-Diagramme von den Studierenden, wobei jedes Projekt ein Antwortschlüssel-Diagramm hatte. Somit umfasst der Datensatz insgesamt 39 Anwendungsfall-Diagramme [19].

Bevor sie ihre vorgeschlagene Methode bewerteten, etablierten die Autoren einen Goldstandard als Referenz für die Bewertung. Dieser Goldstandard repräsentiert das durchschnittliche Ergebnis der Expertenbewertungen von Studentenantworten auf der Grundlage des Antwortschlüssels. Insgesamt beteiligten sich 21 Experten an dieser Bewertung, die von zwölf verschiedenen Universitäten stammten. Die Mindestanforderung für eine Expertin oder einen Experten war ein Master-Abschluss in Informatik.

Zur Bewertung ihrer vorgeschlagenen Methode nutzten die Autoren Gwets AC1, ein Maß, das den Grad der Übereinstimmung zwischen zwei Experten anzeigt. In diesem Fall repräsentiert der erste Experte den Durchschnitt der Bewertungen der Experten, während der zweite Experte die Bewertung durch die vorgeschlagene Methode darstellt. Die Autoren

übertrugen die Ähnlichkeitswerte der beiden Experten auf eine Skala von 1 bis 5, wobei eine Wertung von eins ein Bewertungsergebnis von weniger als zwanzig, zwei ein Bewertungsergebnis von zwanzig bis vierzig, drei ein Ergebnis von vierzig bis sechzig, vier ein Ergebnis von sechzig bis achtzig und fünf ein Ergebnis von mehr als achtzig darstellt. Anschließend berechneten die Autoren die Übereinstimmung zwischen den beiden Experten unter Verwendung von Gwets AC1 [19].

Die Ergebnisse ihrer Bewertung zeigten, dass die vorgeschlagene Methode eine erhebliche Übereinstimmung mit der Bewertung durch den Experten aufwies. Interessanterweise stellten die Autoren fest, dass ihre vorgeschlagene Methode eine höhere Übereinstimmung mit dem Experten aufwies als die durchschnittliche Übereinstimmung zwischen den Experten. Darüber hinaus beobachteten die Autoren, dass Pädagogen tendenziell mehr Wert auf Informationen über Eigenschaften legen als auf Informationen über Beziehungen bei der Bewertung von Anwendungsfall-Diagrammen. Diese Erkenntnis könnte dazu beitragen, den Bewertungsprozess zu verbessern und eine konsistentere und objektivere Bewertung von Studentenantworten zu gewährleisten [19].

Zusammenfassend stellen die Autoren eine innovative Methode zur automatischen semantischen Bewertung von Anwendungsfall-Diagrammen vor. Ihr Ansatz umfasst die semantische Bewertung in den Dimensionen (property) und Beziehung (relationship), wobei ein Datensatz aus realen Projekten gesammelt und ein Goldstandard als Referenz etabliert wurde. Die Evaluation ihrer Methode unter Verwendung von Gwets AC1 zeigt eine erhebliche Übereinstimmung mit Expertenbewertungen. Darüber hinaus liefert ihre Erkenntnis darüber, dass Pädagogen bei der Bewertung von Anwendungsfall-Diagrammen tendenziell Eigenschaftsinformationen stärker gewichten als Beziehungsinformationen, wertvolle Einblicke zur Verbesserung des Bewertungsprozesses und zur Förderung einer konsistenteren und objektiveren Bewertung von Studentenantworten.

2.4.3 Graph matching zur Bewertung von UML

Der Artikel mit dem Titel “New method for summative evaluation of UML class diagrams based on graph similarities” wurde von Outair Anas, Mohammed Ouadou und Abdelhadi Lotfi verfasst [7]. In ihrem Artikel widmen sich die Autoren der anspruchsvollen Aufgabe der Bewertung von UML-Klassendiagrammen, die von Studierenden erstellt wurden. Diese Aufgabe ist oft komplex für Lehrende, da UML-Klassendiagramme mehrere gültige Darstellungen haben können. Die Autoren schlagen ein halbautomatisches System vor, das dieses Problem durch den Einsatz eines Vergleichs von

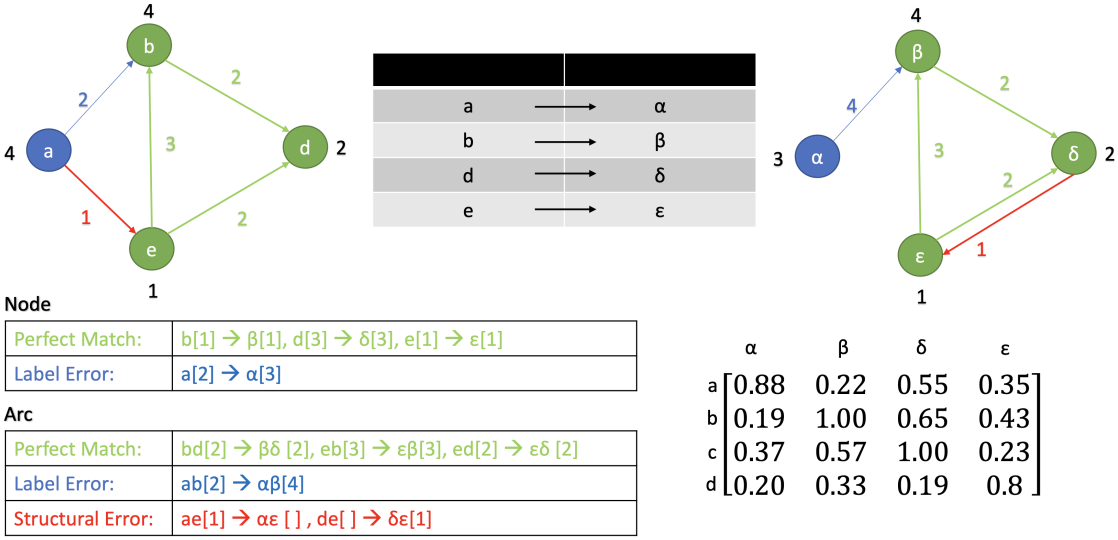


Abbildung 2.2: Beispiel für einen Graphenabgleich [7].

syntaktischen, strukturellen und semantischen Ähnlichkeiten angeht, um Fehler von Studierenden zu identifizieren und wertvolles Feedback zu ihrem Lernprozess zu geben. Ihre Methode besteht aus drei Schritten:

1. **UML-Diagramme in Graphen umwandeln:** Die Autoren beginnen ihren Ansatz, indem sie UML-Klassendiagramme in Graphdarstellungen umwandeln und das Metamodell durch die Einführung neuer Elemente zur Erleichterung der Bewertung verbessern. Sie stellen Klassen als Knoten dar, Attribute als mit Klassen verbundene Knoten und Assoziationen als beschriftete Kanten [9].
2. **Definition von Ähnlichkeitsmaßen:** Die Autoren definieren eine Reihe von Ähnlichkeitsmaßen, die auf die transformierten UML-Graphen anwendbar sind. Sie untersuchen verschiedene Techniken zur Graphabstimmung und Metriken zur Knotenähnlichkeit, um den Vergleich von Graphen und die Fehlererkennung zu erleichtern [20].
3. **Abgleich und Vergleich von Graphen:** Unter Verwendung der zuvor definierten Ähnlichkeitsmaße führen die Autoren einen Abgleich und Vergleich der von Studierenden erstellten UML-Diagramme und der vom Lehrer bereitgestellten Referenzdiagramme durch [52] (Siehe 2.2).

Die Leistung des Systems wird anhand eines Datensatzes bewertet, der aus 100 von Studierenden erstellten Klassendiagrammen besteht, wobei

für jedes Diagramm ein entsprechendes Referenzdiagramm zur Bewertung bereitgestellt wird. Drei Übungen werden ausgewählt, um das System offline zu konfigurieren und zu bewerten, wobei iterative Verbesserungen an den Ähnlichkeitskriterien und der Systemfunktionalität auf Grundlage der Übungsergebnisse vorgenommen werden. Die Ergebnisse zeigen, dass das System eine Genauigkeitsrate von 70 % bei der Erkennung von Fehlern von Studierenden erreicht und minimalen Eingriff erfordert, um Übereinstimmungen für 80 % der verarbeiteten Diagramme zu korrigieren [7]. Die Autoren betonen, dass der in das System integrierte formative Bewertungsansatz gut geeignet ist, um UML-Klassendiagramme zu bewerten, da er den Studierenden Feedback zur Verbesserung ihres Verständnisses des Lehrstoffes liefert. Darüber hinaus hat das System das Potenzial, die Belastung der Lehrenden zu verringern, indem es den Bewertungsprozess automatisiert und ihnen ermöglicht, individuelleres Feedback an einzelne Studierende zu geben.

Zusammenfassend bietet die in diesem Artikel vorgestellte Methode eine vielversprechende Möglichkeit zur summativen Bewertung von UML-Klassendiagrammen unter Berücksichtigung der Möglichkeit mehrerer gültiger Darstellungen. Durch den umfassenden Vergleich von syntaktischen, strukturellen und semantischen Ähnlichkeiten kann das System Fehler in von Studierenden erstellten Diagrammen erkennen und zur Verbesserung ihres Lernfortschritts beitragen. Die Autoren erkennen an, dass weitere Forschung erforderlich ist, um das System zu verfeinern und seine Genauigkeit zu verbessern, aber die Ergebnisse dieser Studie legen nahe, dass es das Potenzial hat, eine wertvolle Unterstützung sowohl für Lehrende als auch für Studierende bei der Bewertung von UML-Klassendiagrammen zu sein.

2.4.4 Automatisierte Bewertung mit GReQL

In ihrem Artikel “Automated checks on UML diagrams” [66] stellen Michael Striewe und Michael Goedicke eine innovative Technik zur Evaluierung von UML-Klassendiagrammen auf der Grundlage von Graphabfragen vor. Sie betrachten UML-Klassendiagramme als Graphen und formalisieren ihren Ansatz mithilfe einer Graphabfragesprache namens GReQL, die von der JGraLab-Bibliothek bereitgestellt wird.

Graph Repository Query Language (GReQL), ähnlich wie SQL, eignet sich gut für die Implementierung regelbasierter Prüfungen von graphenbasierten Daten [65]. Sie ermöglicht die Abfrage von Elementen bestimmter Typen, die Untersuchung ihrer Verbindungen und die Überprüfung ihrer Attribute. Im Kontext von UML-Klassendiagrammen können diese Abfragen

das Vorhandensein von Diagrammelementen wie Klassen, Schnittstellen, Eigenschaften, Operationen, Parametern, Assoziationen und Generalisierungen ermitteln und ihre Beziehungen bewerten [66].

```
1  <rule type="presence" points="5">
2    <query>
3    from x : V{Class},
4    y : V{Property}, z : V{PrimitiveType}
5           with
6              isDefined(x.name) and x.name="A" and
7              x --> y and
8              isDefined(y.name) and y.name="b" and
9              y --> z and
10             isDefined(z.name) and z.name="String"
11          report 1 end
12   </query>
13   <feedback>Ein "A" soll ein Attribut für die
14   Eigenschaft "b" bereitstellen.</feedback>
15 </rule>
```

Quelltext 2.1: Codebeispiel in GReQL

Um ein UML-Diagramm mithilfe dieses Ansatzes zu evaluieren, parsen die Autoren das Diagramm zunächst in eine graphenbasierte Darstellung seiner abstrakten Syntax, in der Regel durch einen XML-Parser. Anschließend verwenden sie GReQL-Abfragen, um die Anwesenheit bestimmter Diagrammelemente und ihrer Verbindungen zu überprüfen. Die Autoren stellen eine Reihe von Regeln vor, die als GReQL-Abfragen implementiert sind und an die Anforderungen bestimmter Kurse oder Aufgaben angepasst werden können.

In ihrem Bewertungssystem werden einzelnen Regeln individuelle Punktzahlen zugewiesen, wobei unterschiedliche Gewichtungen zur Unterscheidung zwischen Korrektheits- und Qualitätsaspekten verwendet werden. Zum Beispiel könnte eine Regel, die das Vorhandensein einer Klasse mit einem bestimmten Namen sicherstellt, als wichtiger angesehen werden als eine, die das Vorhandensein eines Kommentars zu einer Klasse überprüft. Die Autoren schlagen auch eine Methode zur Berechnung von Gesamtnoten vor, indem die Punktzahlen jeder Regel aggregiert werden [66].

Um die praktische Anwendung ihres Ansatzes zu zeigen, bieten die Autoren ein Beispiel aus einem UML-Modellierungskurs an, in dem Lösungen von Studierenden automatisch mit ihrer Methodik bewertet wurden. Das Ergebnis dieser Beispielbewertung zeigt die Effektivität ihres Ansatzes bei

der Erkennung von Fehlern und der Bereitstellung rechtzeitigen Feedbacks an die Studierenden, wodurch die Arbeitsbelastung der Dozenten reduziert wird [66].

Über UML hinaus sehen die Autoren die Vielseitigkeit ihres Ansatzes bei der Bewertung anderer Sprachen, die in eine graphenbasierte Darstellung ihrer abstrakten Syntax transformiert werden können, was praktisch alle Programmiersprachen einschließt. Sie sehen seine Anwendung in intelligenten Tutoringsystemen und automatisiertem Tutoring, da es eine sofortige Rückmeldung an Studierende ermöglicht und den Bewertungsprozess für Lehrkräfte vereinfacht.

Mögliche Einschränkungen dieses Ansatzes sind jedoch die Abhängigkeit von der Annahme einer korrekten Umwandlung von UML-Diagrammen in graphenbasierte Darstellungen, die zu Ungenauigkeiten führen kann, wenn der Umwandlungsprozess fehlerhaft ist. Die Autoren schlagen jedoch vor, den Umwandlungsprozess durch eine Reihe von Testfällen zu validieren, um die Zuverlässigkeit sicherzustellen [66].

Zusätzlich konzentriert sich der Ansatz hauptsächlich darauf, das Vorhandensein von Diagrammelementen und ihren Verbindungen zu überprüfen, bewertet jedoch nicht intrinsisch die Korrektheit des Inhalts innerhalb dieser Elemente. Zum Beispiel kann er überprüfen, ob eine Klasse einen bestimmten Namen hat, aber er überprüft nicht die Richtigkeit der zugehörigen Attribute und Methoden. Die Autoren weisen jedoch darauf hin, dass ihr Ansatz erweitert werden kann, um komplexere Regeln für die Bewertung des Inhalts von Diagrammelementen zu umfassen [65].

Zusammenfassend bietet der Ansatz von Striewe und Goedicke eine vielversprechende Alternative zur Bewertung von UML-Klassendiagrammen. Indem sie diese Diagramme als Graphen behandeln und GReQL für Abfragen nutzen, bieten sie eine flexible und anpassbare Möglichkeit zur Bewertung. Ihre Methodik hat das Potenzial für breitere Anwendungen in verschiedenen Sprachen und Bildungsbereichen und verspricht, das Feedback an Studierende zu verbessern, während sie die Arbeit für Lehrende vereinfacht.

2.5 JACK

Das E-Assessment-System namens “JACK3” (dritte Version von JACK) [51], entwickelt von der Universität Duisburg-Essen, stellt eine webbasierte Plattform dar, die Pädagogen eine effektive Lösung für die Erstellung und Durchführung von Online-Übungen bietet. Diese vielseitige Plattform ermöglicht Lehrkräften die Erstellung einer breiten Palette von Übungen,

die nahtlos über eine benutzerfreundliche Oberfläche bereitgestellt werden können. Neben der Erstellung von Bewertungen verfügt JACK3 über eine Auswahl an Funktionen, die den Bewertungsprozess optimieren sollen. Diese beinhalten die automatisierte Bewertung, umfassende Berichterstellungswerkzeuge und die Echtzeitverfolgung des Lernfortschritts der Schüler.

Die Bedeutung von JACK3 reicht über die reine Bequemlichkeit hinaus, da es Lehrern ermöglicht, wertvolle Zeit und Ressourcen zu sparen, während gleichzeitig die Effizienz und Effektivität der Bewertungen gesteigert werden. Des Weiteren bereichert JACK3 das Lernerlebnis, indem es den Schülern ansprechende und personalisierte Bewertungen bietet, begleitet von sofortigem Feedback. Die Plattform erleichtert die Datensammlung zur Überwachung des Lernfortschritts der Schüler und zur Identifizierung von Verbesserungsmöglichkeiten. Sie unterstützt verschiedene Lehr- und Lernmethoden, darunter das umgekehrte Klassenzimmer, Blended Learning und Fernunterricht [51].

Darüber hinaus bietet JACK3 eine Vielzahl weiterer Vorteile, wie die Steigerung der Schülerbeteiligung und -motivation, die Möglichkeit zur schnellen Bewertung und Rückmeldung von Aufgaben sowie die Prävention von Betrug und Plagiat. Zusätzlich bietet es erweiterte Möglichkeiten zur Verfolgung der Schulentwicklung. JACK3 bietet eine breite Palette von Übungstypen an, die den unterschiedlichen Lernzielen der Pädagogen und den spezifischen Bedürfnissen ihrer Schüler gerecht werden. Zu den Beispielen gehören Multiple-Choice-Quizfragen, Wahr/Falsch-Quizfragen, Zuordnungsübungen, Lückentextübungen, Kurzantwortfragen Aufsatzfragen und vieles mehr. Des Weiteren ermöglicht JACK3 die Erstellung komplexerer Übungen wie Drag-and-Drop-Aufgaben, Hotspot-Übungen, Umfragen zur Datensammlung und Webquests, die eine tiefgehende Exploration von Themen ermöglichen [51].

Es ist erwähnenswert, dass JACK3 seine Fähigkeiten zur Unterstützung der Bewertung von Übungen zur Unified Modeling Language (UML) erweitert hat, was seine Anwendbarkeit in der Softwaretechnik-Ausbildung verbessert. Lehrer können das System nutzen, um UML-Diagramme wie Klassendiagramme, Sequenzdiagramme und Aktivitätsdiagramme automatisch zu bewerten.

Im Bereich der UML-Übungen bietet JACK3 Lehrern die Flexibilität, Übungen zu gestalten, die Klassendiagramme, Sequenzdiagramme und Aktivitätsdiagramme umfassen. Diese Übungen ermöglichen es den Schülern, die Struktur und das Verhalten von Softwaresystemen zu visualisieren, die komplexen Beziehungen zwischen Systemkomponenten zu verstehen, ihre Ideen effektiv an Kommilitonen zu kommunizieren und ihre Fähigkeiten im

Bereich Problemlösung und kritisches Denken zu schärfen [51].

Das Hauptaugenmerk dieser wissenschaftlichen Arbeit liegt auf der Analyse und Evaluierung des Instrumentariums JACK, insbesondere seiner Kapazität zur kritischen Beurteilung von UML-Diagrammen. Im darauf folgenden Abschnitt wird die spezifische Problematik in umfassender Weise erörtert und eine breite Palette möglicher Lösungsansätze erörtert.

3. Problemanalyse

In diesem Kapitel erfolgt eine gründliche Analyse der zugrunde liegenden Problematik. Die Identifikation und eingehende Diskussion der spezifischen Herausforderungen und Schwierigkeiten, die im Rahmen dieser Arbeit behandelt werden, stehen dabei im Fokus.

3.1 Beschreibung des Bewertungsprozesses

Wie bereits in den vorherigen Kapiteln erwähnt, verwendet die Fakultät für Informatik der Universität Duisburg-Essen (UDE) ein elektronisches Bewertungssystem namens “JACK” [51], um bestimmte Prüfungen und Übungen automatisch zu bewerten. Unter den verschiedenen Arten von Übungen konzentriert sich diese Masterarbeit auf Übungen vom Typ UML.

Eine Übung vom Typ UML-Klassendiagramm besteht darin, die statische Struktur eines Software-Systems mithilfe der UML-Modellierungssprache zu modellieren [57]. Den Studierenden wird in der Regel eine Beschreibung des Systems, der wichtigsten Entitäten und ihrer Beziehungen zur Verfügung gestellt, und sie müssen dann ein Klassendiagramm erstellen, das diese Elemente darstellt. In diesem Diagramm sind die Klassen die Hauptobjekte, mit ihren Attributen (Variablen) und Methoden (Funktionen), und die Beziehungen zwischen den Klassen werden durch Assoziationen, Aggregationen oder Kompositionen dargestellt. Die Studierenden müssen besonders auf die Genauigkeit der Namen, Multiplizitäten und Kardinalitäten achten, um die Struktur des Systems korrekt widerzuspiegeln [57]. Das Hauptziel dieser Übung ist es, das Verständnis der objektorientierten Modellierungskonzepte zu vertiefen und eine solide Grundlage für die Softwareentwicklung zu schaffen. In Bezug auf das JACK-System kann der Bewertungsprozess in mehrere verschiedene Phasen unterteilt werden:

Phase 1: Erstellung der Übung
In dieser ersten Phase erstellt der Lehrer die Übung, indem er eine

schriftliche Beschreibung eines zu modellierenden Systems bereitstellt. Besonderes Augenmerk wird auf die Genauigkeit der Bezeichnungen der Entitäten und die Klarheit und Explizitheit der Beziehungen zwischen ihnen gelegt. Der Lehrer hat dann einen Bereich in der JACK-Anwendung, in den er diesen Text einfügen kann, der von den Studierenden eingesehen wird.

Phase 2: Erstellung einer Musterlösung oder Anmerkung (optional)

In dieser Phase kann der Lehrer entscheiden, ein UML-Diagramm als Musterlösung für die Übung zu erstellen. Dies erleichtert das Verständnis der Übung, ermöglicht die Überprüfung der Kohärenz und Durchführbarkeit des Systems. Alternativ kann der Lehrer die Übung lediglich annotieren, um die Schlüsselbegriffe und -elemente hervorzuheben, die in der späteren Phase nützlich sein werden.

Phase 3: Entwicklung des GReQL-Codes

In dieser Phase verwendet der Lehrer seine Anmerkungen und die Musterlösung, um den GReQL-Code zu entwickeln, der von JACK interpretiert wird, um die von den Studierenden eingereichten Lösungen zu bewerten. JACK verwendet den GReQL-Code sowie verschiedene in dieser Sprache definierte Regeln, um die UML-Diagramme zu bewerten. Wenn ein Student seine Lösung einreicht, wird eine grafische Darstellung dieser Lösung erstellt, und der GReQL-Code führt Abfragen auf dieser Darstellung aus, um eine Note für die Lösung des Studenten zu vergeben [66].

Phase 4: Einreichung der Lösung durch den Studenten

Nachdem der Student an der Lösung der Übung gearbeitet hat, lädt er ein XML-Dokument im XMI Format auf JACK hoch. Zur Generierung dieses kompatiblen XMI-Dokuments verwenden Studierende Tools wie BOUML [53] oder Software Ideas Modeler [58], mit denen sie XMI-Code aus einer zuvor erstellten grafischen UML Darstellung ableiten können. Dieses Dokument repräsentiert die von ihm entwickelte Lösung.

Phase 5: Bewertung des Diagramms

Das von den Studierenden eingereichte Diagramm wird anhand des von den Lehrern verfassten GReQL-Codes bewertet. Anschließend wird dem Studenten eine Note zugeteilt.

Diese Phasen veranschaulichen die grundlegende Funktionsweise der JACK-Plattform in Bezug auf die automatisierte Bewertung von Übungen zur Erstellung von UML-Klassendiagrammen.

3.2 Untersuchung des Problems

Im Verlauf der dritten Phase, wie im vorherigen Kapitel beschrieben, sehen sich Lehrende mit der Notwendigkeit konfrontiert, GReQL-Code zu

verfassen, der vom JACK-System zur Bewertung der Einreichungen verschiedener Studierender verwendet wird. Diese Phase stellt jedoch bereits auf verschiedenen Ebenen eine Herausforderung für die Lehrenden dar, aus folgenden Gründen:

Erforderliche Expertise für die Erstellung von GReQL-Code: Das Verfassen von GReQL-Code erfordert eine gewisse Expertise. Auf den ersten Blick mag die Syntax von GReQL nicht schwer zu verstehen sein. Dennoch kann es anspruchsvoll sein, die Feinheiten der Code-Erstellung zu beherrschen. Dies erfordert von den Lehrenden mehrere Stunden, intensives Üben und umfangreiche Tests, um einen Code zu erstellen, der präzise bewertet, insbesondere in Fällen von komplexeren Beziehungen zwischen Entitäten.

Hohe Möglichkeit von Fehlern im GReQL-Code: Selbst bei Beherrschung der Feinheiten von GReQL sind Lehrende nicht vor möglichen Fehlern gefeit. Diese Fehler können zwar geringfügig erscheinen, haben jedoch das Potenzial, das Bewertungssystem erheblich zu beeinträchtigen.

Zeitaufwand für die Erstellung von GReQL-Code: Die Bewertung jedes Diagramms erfordert erheblichen Zeitaufwand, um einen GReQL-Bewertungscode zu erstellen, insbesondere für komplexere Beziehungen zwischen verschiedenen Entitäten. Die Zeit, um die erforderliche Expertise zu erlangen, kann stark variieren und erfordert zahlreiche Stunden intensiver Übung.

Wartung und Anpassungsfähigkeit des GReQL-Codes: Nachdem der GReQL-Code erstellt und für die Bewertung von UML-Diagrammen implementiert wurde, ist kontinuierliche Wartung erforderlich. Mit steigenden Anforderungen an die Bewertung muss der GReQL-Code regelmäßig aktualisiert und angepasst werden. Dies stellt für Lehrende eine fortwährende Herausforderung dar und erfordert Zeit und Aufwand, um sicherzustellen, dass das Bewertungssystem präzise und relevant bleibt.

Bedarf an Ressourcen und technischer Unterstützung: Lehrende benötigen möglicherweise Zugang zu technischen Ressourcen und angemessener Unterstützung, um die Kunst der effizienten GReQL-Code-Erstellung zu beherrschen. Dies kann Schulungen, Orientierungsdokumente, Diskussionsforen oder andere Formen technischer Unterstützung einschließen. Die Beschaffung dieser Ressourcen kann Zeit in Anspruch nehmen und eine institutionelle Koordination erfordern.

Dies sind die verschiedenen Herausforderungen, die bei der Erstellung von GReQL-Code für die Bewertung von UML-Diagrammen auf der JACK-Plattform auftreten können.

3.3 Ableitung und Abgrenzung der Anforderungen

Das Ziel dieser Masterarbeit besteht nicht darin, Lehrkräfte von der Notwendigkeit zu befreien, GReQL-Code zu schreiben oder zu bearbeiten. Vielmehr geht es darum, den Prozess der Regeldefinition erheblich zu erleichtern. Um diesen Prozess zu vereinfachen, zielt diese Masterarbeit darauf ab, ein Verfahren zu entwickeln und in Form einer Softwareanwendung zu prototypisieren, mit dem (halb-)automatisch Bewertungsregeln aus annotierten Musterlösungen erstellt werden können. Dieses System soll in der Lage sein, folgende Ziele zu erreichen:

1. **Verminderung der Einstiegshürde:** Es strebt danach, den Prozess der Erstellung von GReQL-Regeln zu erleichtern. Selbst jemand, der keine Vorkenntnisse in GReQL hat, sollte unser Tool verwenden können, um Regeln zu erstellen, die bereits eine Bewertung von einfachen Diagrammen und Beziehungen ermöglichen.
2. **Verbesserung der Präzision und Zuverlässigkeit der GReQL-Regeln:** Durch die Nutzung von annotierten Musterlösungen zielt die Anwendung darauf ab, die Präzision und Zuverlässigkeit der generierten Regeln zu verbessern. Dies würde das Risiko von Fehlern und Inkonsistenzen in den Bewertungsregeln verringern und somit eine konsistentere Bewertung der studentischen Arbeiten gewährleisten.
3. **Optimierung von Zeit und Aufwand der Lehrkräfte:** Durch Automatisierung eines Teils des Regelbildungsprozesses zielt das System darauf ab, Zeit und Aufwand der Lehrkräfte zu sparen. Dies würde es ihnen ermöglichen, sich mehr auf das Lehren und Betreuen der Studierenden zu konzentrieren, anstatt auf mühsame administrative Aufgaben.
4. **Förderung der Skalierbarkeit und Wartbarkeit der Regeln:** Durch die Implementierung eines Mechanismus zur Pflege und Aktualisierung der generierten Regeln würde die Anwendung dazu beitragen sicherzustellen, dass die Regeln relevant und anpassungsfähig bleiben und sich den sich ändernden Anforderungen von Lehre und Bewertung anpassen.
5. **Unterstützung einer breiten Palette von Bewertungsszenarien:** Die Anwendung sollte flexibel genug sein, um eine Vielzahl von Bewertungsszenarien zu unterstützen, einschließlich solcher mit komplexen Diagrammen und Beziehungen. Dadurch würde sie eine vielseitige Lösung für Lehrkräfte bieten.

Das Ziel dieser Initiative ist es, den Prozess der Bewertung von UML-Diagrammen effizienter, zugänglicher und präziser zu gestalten, während die administrativen Arbeitslasten der Lehrkräfte reduziert werden. Dadurch soll die Qualität von Lehre und Bewertung im Bereich der Softwaremodellierung verbessert werden.

4. Entwicklung des Konzepts

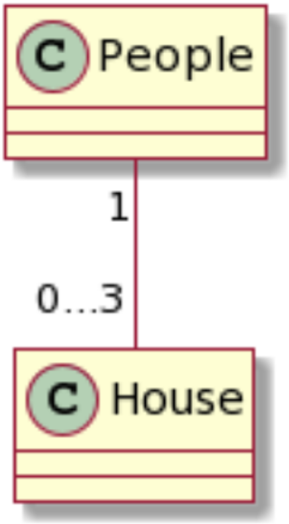
In diesem Kapitel wird der Übergang von der Problemanalyse zur kreativen Konzeptentwicklung im Forschungsprozess hervorgehoben. Es markiert die zentrale Phase, in der theoretisches Wissen und praktische Erkenntnisse zusammenfließen, um innovative Lösungsansätze zu formulieren. Wir integrieren theoretische Grundlagen und Modelle aus vorangegangenen Kapiteln und erläutern die angewandten Methoden zur Entwicklung tragfähiger Lösungen. Dieses Kapitel dient als Brücke zwischen Analyse und Umsetzung, betont die kreative Synthese von Ideen und Lösungsansätzen und bietet einen detaillierten Einblick in den Prozess der Konzeptentwicklung.

4.1 Verschiedene Herangehensweisen

Vor der Ausarbeitung des endgültigen Implementierungskonzepts wurden mehrere methodische Ansätze einer eingehenden Untersuchung unterzogen. Das vorrangige Ziel dieser methodischen Herangehensweisen bestand in der Extraktion einer Reihe von Regelwerken, die später in GReQL-Ausdrücke transformiert werden sollten. Diese Regelwerke sollten aus den Kommentaren oder Anmerkungen extrahiert werden, welche von Lehrkräften in Bezug auf eine UML-Übung hinterlassen wurden. Die Wahl des Notenformats ist frei, was viele Möglichkeiten bietet.

4.1.1 YAML-basierten Annotationen

Der erste untersuchte methodische Ansatz fokussierte auf die Entwicklung eines benutzerfreundlichen Annotationsystems, das eine umfassende Modellierung der Interaktionen innerhalb eines UML-Diagramms ermöglichen sollte. Infolgedessen wurden editierbare Regelobjekte abgeleitet, welche aus diesem Annotationsystem generiert wurden. Sobald der Nutzer sämtliche Regeln, die aus dem Annotationsystem resultierten, verifizierte, konnte er GReQL-Code generieren, welcher anschließend in das JACK-System ein-




```yaml
associations:
        - name: OneToMany
          class1: People
          class2: House
          multiplicity1: 1
          multiplicity2: 0..3
```

Abbildung 4.1: Beispiel einer Annotation von UML mit YAML

gefügt wurde. Die Konzeption dieses schlichten Notationssystems diente dem Zweck der Abbildung von Beziehungen innerhalb eines UML Klassendiagramms. Zu diesem Zweck wurde das YAML-Format aus mehreren Gründen präferiert:

1. Erstens, das Schreiben im YAML-Format erweist sich als unkompliziert.
2. Zweitens, es liefert Daten in einem strukturierten Format, welches leicht manipuliert werden kann.

Die Notation in YAML mag den Eindruck vermitteln, bereits eine Form der Regeldefinition darzustellen, weil der Lehrende bereits alle Interaktionen zwischen den verschiedenen Objekten im Diagramm definiert. Um jedoch die Klarheit bezüglich dieser Frage zu gewährleisten, könnte es sinnvoll sein, ein weniger formelles Notationssystem zu etablieren. Beispielweise:

- Die Klasse A erbt von der Klasse B.
- Die Klasse A besitzt drei Attribute:
  - Attribut a vom Typ x mit öffentlicher Sichtbarkeit.
  - Attribut b vom Typ y mit öffentlicher Sichtbarkeit.
  - Attribut c vom Typ z mit privater Sichtbarkeit.

oder ein noch weniger wortreiches System:

- A $\Rightarrow$ B
- A:
  - +a:x (+ für öffentlich)
  - +b:y
  - -c:z (- für privat)

Dieses System ähnelt eher herkömmlichen Kommentaren, jedoch könnte es herausfordernder sein, den darin enthaltenen Text zu analysieren und daraus anwendbare Regelvorschläge zu extrahieren. Trotzdem birgt dieser methodische Ansatz mehrere Limitationen:

1. Es existieren bereits seit geraumer Zeit vielfältige Notationssysteme und Datenformate für UML, wie beispielsweise XMI [61], UML eXchange Format (UXF) [68], Textbasierte [72] oder JSON-basierte Datenformate [12], um nur einige zu nennen.
2. Die Entwicklung eines vollständig neuen und umfassenden Notationssystems innerhalb eines begrenzten Zeitraums kann äußerst zeitintensiv sein, insbesondere wenn es die umfassende Darstellung sämtlicher Varianten eines UML-Klassendiagramms anstrebt.
3. Dieses System verlagert die Aufgabe der Ableitung von Beziehungen zwischen den verschiedenen Objekten lediglich in ein anderes Format, ohne dabei die Arbeitsbelastung für die Lehrkraft zu mindern.

In Anbetracht dieser diversen Überlegungen wurde von der Verfolgung dieses methodischen Ansatzes Abstand genommen, und es erfolgte die Exploration alternativer Vorgehensweisen.

4.1.2 Verwendung von Natural Language Processing Tools

Die zweite konzeptionelle Idee besteht darin, ein Natural Language Processing (NLP)-Tool zu verwenden, um die Kommentare oder Notizen des Lehrers zu analysieren. Anschließend kann das NLP-Tool verwendet werden, um Empfehlungen auf der Grundlage einer vorher festgelegten Reihe von Parametern für den Lehrer abzugeben. Diese Parameter könnten eine Reihe von UML-Regeln in einem spezifischen Format sein, so dass die KI in der Lage ist, die richtige Lösung auf der Grundlage des Kommentars zu finden. Der Vorschlag könnte dann in GReQL-Code übersetzt werden. Diese Herangehensweise hat jedoch auch einige Herausforderungen:

1. Die von dieser Methode generierten Lösungen neigen dazu, ungenau zu sein, und es sind möglicherweise zahlreiche Anpassungen erforderlich, um das gewünschte Ergebnis zu erzielen [16]. Dies könnte

möglicherweise die Arbeitsbelastung der Lernenden erhöhen, anstatt sie zu erleichtern.
2. Es wäre notwendig, die KI darauf zu trainieren, bestimmte Muster zu erkennen und sie mit vordefinierten Regeln in Verbindung zu bringen, um die Zuverlässigkeit zu erhöhen.

In ihrem aktuellen Stand erfordert diese Herangehensweise erhebliche Anstrengungen, um diese Herausforderungen zu bewältigen und eine effektive Implementierung zu erreichen. Aufgrund der potenziellen Schwierigkeiten bei der Implementierung und der möglicherweise nicht zufriedenstellenden Ergebnisse wurde dieser Ansatz auch aufgegeben.

4.2 Konzept

Bei der Entwicklung des Konzepts wurde ein radikal andersartiger Ansatz als der in dem vorherigen Abschnitt dargelegte verfolgt. Die Herangehensweise, Regeln aus Kommentaren abzuleiten, erweist sich als ein komplexer Prozess und potenziell schwierig umzusetzen. Die Gewährleistung der Zuverlässigkeit der erzielten Ergebnisse stellt ebenfalls eine bedeutende Herausforderung dar. In diesem Zusammenhang wurde eine intuitivere Herangehensweise in Betracht gezogen, nämlich die direkte Ableitung von Regeln aus der Mustervorlage.

Bevor dieses Konzept im Detail beschrieben wird, ist es unerlässlich, das Endziel in Erinnerung zu rufen, nämlich die Unterstützung von Lehrenden bei der Bewertung von UML-Übungsaufgaben (insbesondere Klassendiagrammen). Dies soll durch erhebliche Vereinfachung des Schreibens von GReQL-Code auf der JACK-Plattform erreicht werden, indem Lehrern mit Hilfe eines im Rahmen dieser Masterarbeit zu entwickelnden Tools (halb-)automatische Unterstützung geboten wird. Das Konzept zur Erfüllung dieser Aufgabe kann in drei Wichtige Schritte unterteilt werden:

Schritt 1: Erstellung einer Musterlösung mit Hilfe eines Modellierungstools

Wie im Abschnitt 3.1 erwähnt wurde, umfasst Phase 2 die optionalen Schritte zur Erstellung einer Musterlösung. In diesem Konzept ist diese Phase unerlässlich und gewinnt ihre volle Bedeutung. Nachdem ein UML-Klassendiagrammübungsaufgabe entwickelt wurde, erstellen die meisten Lehrer eine Musterlösung, oft in Form eines Klassendiagramms. Diese Lösung wird dann mit den verschiedenen Einreichungen der Studenten

verglichen, und Punkte werden gemäß den vom Lehrer festgelegten Kriterien an die einzelnen Studenten vergeben. Da in den meisten Fällen eine Musterlösung in Form eines Klassendiagramms für die Aufgabe vorhanden ist, wäre es sinnvoll, diese zur Ableitung relevanter Regeln zu verwenden. Dies stellt die erste Phase des Ansatzes dar, bei der eine Musterlösung mithilfe einer Modellierungsanwendung erstellt wird, die anschließend die Möglichkeit bietet, das Diagramm in einem leicht programmierbaren Format zu exportieren.

Schritt 2: Verarbeitung der exportierten Datei

Nachdem der Lehrer die Musterlösung erfolgreich modelliert hat, ergibt sich die Notwendigkeit, diese Vorlage in ein geeignetes Format zu exportieren, welches eine programmgesteuerte Bearbeitung ermöglicht. Dieser Transformationsprozess kann mittels diverser Datenformate wie JSON, XML oder sogar YAML realisiert werden. Sobald das exportierte Dokument verfügbar ist, eröffnet sich im Kontext eines Klassendiagramms die Möglichkeit, zunächst sämtliche in der Musterlösung enthaltenen Klassen sowie sämtliche Interdependenzen zwischen den verschiedenen Elementen zu extrahieren. Infolge dieses Phasenschritts besteht die Option zur Ableitung von “Regel”-Objekten, welche auf der Frontend-Oberfläche sichtbar sind und vom Lehrer interaktiv bearbeitet werden können. Die Bereitstellung dieser Funktionalität befähigt den Lehrer dazu, wertvolles Feedback hinzuzufügen, die zugehörigen Punktzahlen zu definieren und sämtliche erforderlichen Informationen für den nächsten Schritt und die anschließende Evaluierung umfassend zu vervollkommnen.

Schritt 3: Generierung des GReQL-Codes

Der abschließende Schritt dieses Vorgehens umfasst die Generierung von GReQL-Code aus den Regeln, die in der vorherigen Phase extrahiert und/oder hinzugefügt wurden. Nachdem die Konfiguration abgeschlossen ist, muss der GReQL-Code unter Verwendung verschiedener XMI-Vorlagen generiert werden. Der resultierende Code kann vom Lehrer auf der JACK-Plattform verwendet werden.

Durch dieses Konzept (siehe 4.2) und seine verschiedenen Schritte ist es möglich, von einer Musterlösung zur Generierung des erforderlichen GReQL-Codes für die Bewertung von Diagrammen durch JACK zu gelangen. Dieser Ansatz hat das Potenzial, den Bewertungsprozess von Diagrammen über die JACK-Plattform erheblich zu vereinfachen, was das zentrale Ziel dieser Masterarbeit ist. Die tatsächliche Umsetzung dieses

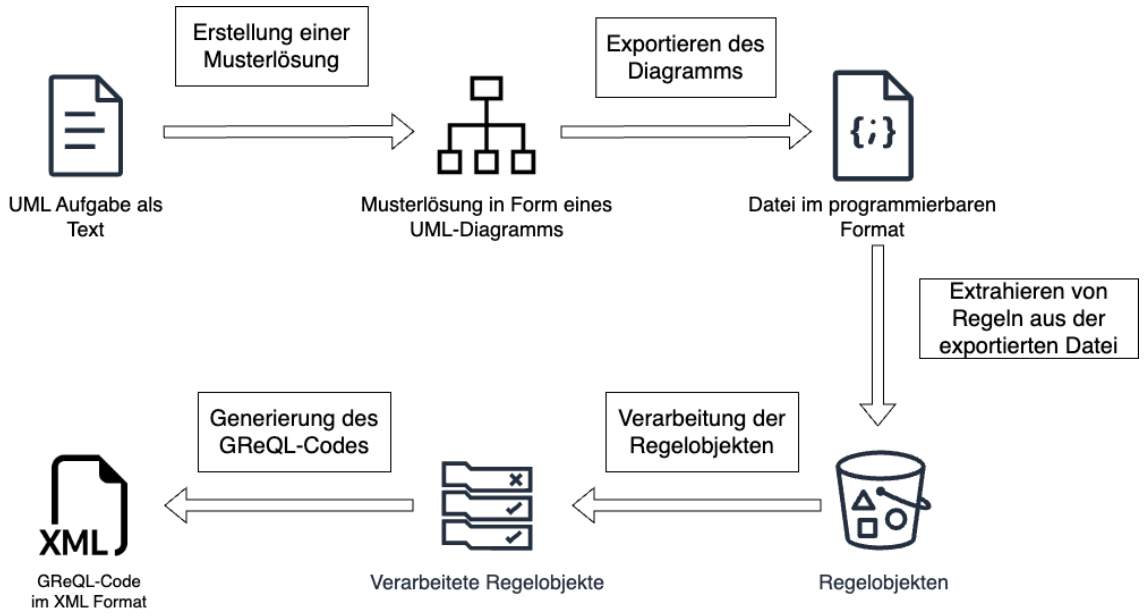


Abbildung 4.2: Repräsentatives schema des konzepts.

Konzepts hängt jedoch eng vom gewählten Workflow und der Art und Weise ab, wie das Konzept umgesetzt wird. Dieser Ausblick leitet zum nächsten Kapitel über, das die Implementierung behandelt.

5. Implementierung

Das vorliegende Kapitel widmet sich der umfassenden Dokumentation des Implementierungsprozesses des zuvor beschriebenen Konzepts. Es bietet eine detaillierte Aufarbeitung der technischen Umsetzung und des Entwicklungsprozesses, der im Rahmen dieser Masterarbeit durchgeführt wurde. Beginnend mit einer umfassenden Beschreibung der verwendeten Technologien und Werkzeuge sowie einer ausführlichen Begründung für die Wahl dieser spezifischen Technologien, wird dieses Kapitel einen tiefen Einblick in die präzise Umsetzung des Konzepts gewähren. Die Implementierung ist ein entscheidender Schritt zur Realisierung des in den vorherigen Kapiteln skizzierten Ansatzes zur Bewertung von UML-Diagrammen. Durch die Dokumentation dieses Schrittes wird das Verständnis für die technischen Aspekte des Projekts vertieft und ermöglicht eine transparente Darstellung des Entwicklungsprozesses.

5.1 Überblick über angewendete Werkzeuge und Technologien

Dieses Kapitel konzentriert sich auf die Vorstellung der Werkzeuge, die für die Umsetzung des im vorherigen Kapitel vorgestellten Konzepts verwendet werden. Es wird jedes Werkzeug vorgestellt und die Auswahl begründet.

5.1.1 PlantText

Der erste Schritt des im vorherigen Kapitel vorgestellten Konzepts besteht in der Modellierung eines UML-Diagramms. Diese Modellierungsphase verlangt die Anwendung einer Software-Anwendung, welche in der Lage ist, das entworfene Diagramm in ein Format zu überführen, welches für die anschließende Extraktion der enthaltenen Regelsätze dienlich ist. Eine facettenreiche Auswahl an digitalen Werkzeugen steht zur Verfügung, um

diese spezifische Aufgabe zu bewältigen, wobei PlanText exemplarisch zu nennen ist.

PlantText [59] ist ein webbasiertes Instrument zur Diagrammmodellierung, das insbesondere in den Domänen der UML und anderer Modellierungssprachen einen renommierten Status innehat. Dieses Instrument wurde entwickelt, um die bequeme Erstellung, Bearbeitung und gemeinsame Nutzung von Diagrammen in einer kollaborativen Umgebung zu ermöglichen. Seine Auszeichnungen resultieren aus der Benutzerfreundlichkeit, der Flexibilität und der Leistungsfähigkeit, wodurch es zu einer favorisierten Wahl für Softwareentwickler, Systemarchitekten und Projektmanager avanciert [59].

PlantText basiert auf einer schlichten, aber wirkungsvollen Konzeption, nämlich der Generierung von Diagrammen durch Verwendung von textuellen Notationen. Benutzer können Diagramme unter Einsatz von natürlicher Sprache und vordefinierten Schlüsselwörtern kreieren, wodurch der gesamte Prozess simplifiziert wird. Eine prototypische Darstellung einer Klasse in einem UML-Klassendiagramm kann etwa wie folgt aussehen:

```
1 class A {
2    + attribute1: Typ1
3    - attribute2: Typ2
4    # operation1(): void
5 }
```

Quelltext 5.1: PlantText code Example

In diesem Illustrationsfall repräsentieren einfache Textnotationen die Klasse “A”, ihre Attribute und Methoden. Die Verwendung von “+” für öffentliche Attribute, “-” für private Attribute und “” für Methoden gestaltet sich intuitionsgetreu und erleichtert die Entwicklung von UML-Diagrammen erheblich. PlantText beinhaltet eine leistungsstarke Rendering-Engine, die diese textlichen Notationen automatisiert in visuell ansprechende Diagramme konvertiert. Benutzer sind in der Lage, die Diagramme in Echtzeit zu visualisieren und zu editieren, ohne sich mit den komplexen Details der grafischen Gestaltung auseinandersetzen zu müssen. Diese textorientierte Herangehensweise führt zu einer höchst effizienten und adaptierbaren Gestaltung und Veränderung von Diagrammen. Die Vorzüge der Anwendung von PlantText manifestieren sich unter anderem in:

1. **Benutzerfreundlichkeit:** PlantText ist für Einsteiger und erfahrene Modellierer gleichermaßen zugänglich. Die Verwendung von textuellen Notationen vereinfacht den Einstieg und reduziert die Lernkurve,

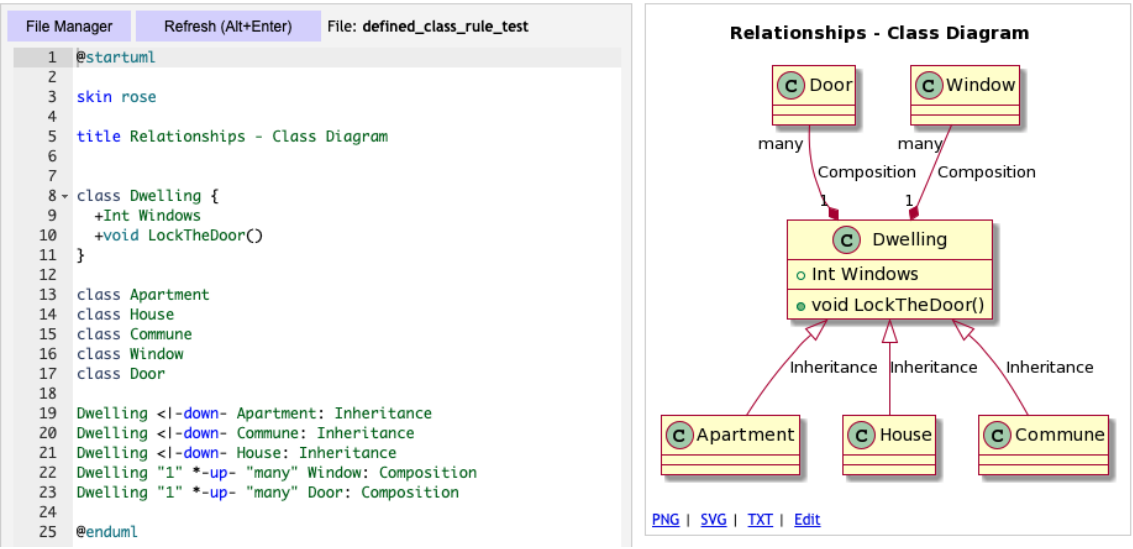


Abbildung 5.1: Grafische Benutzeroberfläche von plantText

da sie natürlicher und verständlicher sind als grafische Schnittstellen [45].

2. **Kollaboration und gemeinsame Nutzung:** PlantText bietet eine eingebettete Kollaborationsplattform, auf der mehrere Benutzer simultan an Diagrammen arbeiten können. Dies fördert die Teamarbeit und erlaubt die Echtzeit-Erstellung und Überarbeitung von Modellen [44].
3. **Plattformunabhängigkeit:** Da PlantText webbasiert ist, ist es plattformneutral. Benutzer können von jedem Gerät mit Internetzugang auf ihre Modelle zugreifen und sie editieren, ohne Softwareinstallationen durchführen zu müssen [59].
4. **Erweiterbarkeit:** PlantText unterstützt nicht ausschließlich UML, sondern auch diverse andere Modellierungssprachen und Diagrammtypen. Infolgedessen entwickelt sich PlantText zu einem vielseitigen Werkzeug für eine Vielzahl von Anwendungsszenarien [59].

Vor der Wahl von PlantText als Instrument für die Entwurfsphase der Anwendung wurden mehrere Modellierungswerkzeuge einer eingehenden Prüfung unterzogen. Diese Werkzeuge schlossen namhafte Anwendungen wie Enterprise Architect [63], Astah UML [8], MagicDraw [69], Visual Paradigm [54], Umbrello [17] und Draw.io [43] ein. Eine gemeinsame Eigenschaft dieser Werkzeuge besteht darin, dass sie sich für die rasche Erstellung von Diagrammen eignen, was auf ihre intuitive Benutzeroberfläche, leichte Verständlichkeit und Nutzerfreundlichkeit zurückzuführen ist. Bedauerlicherweise wiesen sie jedoch einen bedeutenden Mangel auf,

der ihre Anwendbarkeit in Bezug auf unsere spezifischen Anforderungen einschränkte. Keines dieser Programme bot die Möglichkeit, die erstellten Diagramme in eine leicht interpretierbare textuelle Form zu überführen.

Die meisten dieser Tools gestatten zwar das Exportieren der erstellten Diagramme im XML-Format, doch dieses Format präsentiert lediglich eine räumliche Repräsentation der verschiedenen Objekte in einer Ebene, ohne eine semantische Tiefenstruktur. Ein weiteres Hindernis bestand darin, dass die Informationen bezüglich der Verbindungen zwischen den diversen UML-Objekten nur mit großem Aufwand und erheblichen Schwierigkeiten aus dem generierten XML extrahiert werden konnten. Dies wäre in der Praxis äußerst zeitaufwendig und würde die Entwicklung eines eigenen Parsers erfordern, um die relevanten Informationen zu extrahieren und in eine verwertbare Form zu überführen.

Die Nutzung von PlantText hingegen bietet einen klaren Vorteil in dieser Hinsicht. Dies resultiert aus der bereits implementierten PlantUML-Engine, die über einen eingebauten Parser verfügt. Diese Funktionalität ermöglicht es, den erstellten Code direkt in einem Format zu erhalten, das für die Weiterverarbeitung und Interpretation äußerst zugänglich ist. Diese grundlegende Unterscheidung führt dazu, dass PlantText in unserem Anwendungsfall als überlegen angesehen wird. Durch die Fähigkeit zur Bereitstellung des Modells in einer textuellen Form ermöglicht es eine tiefere und bedeutungsvollere Analyse der erstellten Diagramme. Dies fördert die Genauigkeit bei der Modellierung und stellt sicher, dass die erstellten Diagramme nicht nur als visuelle Darstellungen betrachtet werden, sondern auch als Quellen semantischer Informationen dienen können.

5.1.2 PlantUML Parser + Nodejs

Im vorangegangenen Unterabschnitt wurde die Anwendung PlantText als Instrument zur Erstellung von Diagrammen vorgestellt. Es ist jedoch essenziell zu betonen, dass PlantText lediglich die grafische Benutzeroberfläche darstellt, da die zugrundeliegende Engine, auf der PlantText basiert, “PlantUML” [60] ist. PlantUML ist ein Open-Source-Tool, das von Arnaud Roques entwickelt wurde und erstmals im Jahr 2009 veröffentlicht wurde [60]. Als Java-Anwendung ermöglicht PlantUML, ähnlich wie PlantText, die Modellierung von Diagrammen, wobei die Verwendung durch die Entwicklung von PlantText vereinfacht wurde. Alle im vorherigen Abschnitt hervorgehobenen Vorteile der Nutzung von PlantText sind in Wirklichkeit Funktionen von PlantUML.

Eine besonders bemerkenswerte Funktion, die in diesem Abschnitt präsentiert und im Folgenden verwendet wird, ist jedoch der PlantUML-

Parser [56]. PlantUML ist im Wesentlichen eine Backend-Anwendung, die auf einem Server ausgeführt wird. Wenn ein Benutzer ein Modell erstellen und Code in PlantText eingeben möchte, wird dieser Code an einen Server übertragen, der ihn analysiert, interpretiert und ein Diagramm mithilfe von Graphviz [25] generiert. Mit Hilfe des PlantUML-Parsers besteht die Möglichkeit, das erzeugte Diagramm in ein Format zu exportieren, das für programmatische Anwendungen geeignet ist. Es ist genau diese Funktion, die im weiteren Verlauf dieser Abhandlung ausführlich behandelt wird.

Der PlantUML Parser [56] ist ein Open-Source-Tool, mit dem der PlantUML-Code in ein JSON-Format geparst werden kann. Dieses Format kann zur Modellierung verschiedener Regelobjekte (wie im Konzept beschrieben 4.2) verwendet werden. Da der Parser ausschließlich in einer Serverumgebung funktioniert und somit in verschiedenen Java-, Node.js-Umgebungen verfügbar ist, kann er ausschließlich in solchen Umgebungen eingesetzt werden.

Node.js [22] wird verwendet, um diese Serverumgebung zu erstellen und eine gewisse Kohärenz zwischen dem Frontend und dem Backend sicherzustellen, indem die Verwendung mehrerer Programmiersprachen in einem Projekt vermieden wird. Dies trägt zur Effizienz und Integration des Gesamtprojekts bei.

5.1.3 Vue.js

Vue.js, oft auch einfach als Vue bezeichnet, ist ein leistungsfähiges Open-Source-JavaScript-Framework, konzipiert und entwickelt von Evan You [75]. Der Ursprung von Vue.js entsprang der Vision, eine zeitgemäße, wandelbare und leicht handhabbare Lösung zur Gestaltung von Benutzeroberflächen in Webanwendungen zu schaffen. In seiner Erstveröffentlichung im Jahr 2014 initiiert, hat Vue.js eine bemerkenswerte Entfaltung erfahren, die es zu einem herausragenden Akteur unter den JavaScript-Frameworks [41] in der Sphäre der Webentwicklung gemacht hat.

In den Zielen und Vorzügen von Vue.js kristallisiert sich eine Antwort auf die Herausforderungen bei der Generierung interaktiver Webanwendungen und die Optimierung des Entwicklungsprozesses in ein ergötzliches Narrativ. Wesentliche Prämissen und Gewinnpunkte von Vue.js offenbaren sich in folgender Weise:

1. **Nahtlose Integration:** Vue.js kann ohne Mühe in laufende Projekte integriert werden, unabhängig davon, ob es als das Hauptframework oder als eine ergänzende Komponente in Kombination mit anderen Technologien fungiert. Hierdurch ergibt sich ein gestaffelter

Übergangsprozess und eine vorherrschende Flexibilität in der Architektur von Anwendungen.

2. **Komponentenbasierte Architektur:** Vue begünstigt die Einsetzung wiederverwendbarer Komponenten, die nicht nur die Strukturierung und Organisation des Quellcodes erleichtern, sondern auch eine präzise Abgrenzung von Aufgaben und eine verbesserte Wartbarkeit ermöglichen.
3. **Reaktive Datenbindung:** Vue bietet eine reaktive Datenbindung, die die automatische Synchronisierung von Daten und Benutzeroberfläche gestattet. Hierbei erfolgt die Anpassung von Daten an die Benutzeroberfläche und umgekehrt ohne das Hinzufügen von Zusatzcode.
4. **Deklarative Rendering:** Die Einbindung deklarativer Syntax in Vue.js vereinfacht die Implementierung von Benutzeroberflächenelementen erheblich. Entwickler können die gewünschte Darstellung der Benutzeroberfläche beschreiben, während Vue für die entsprechende Logikumsetzung sorgt.
5. **Gemeinschaftsunterstützung:** Vue.js profitiert von einer florierenden und stetig wachsenden Entwicklergemeinschaft, die eine Vielzahl von Ressourcen, Bibliotheken und Erweiterungen zur Verfügung stellt. Dies vergrößert die Möglichkeiten zur Erweiterung und Anpassung von Vue-Projekten erheblich.

Die Entscheidung zur Verwendung von Vue.js in einem Projekt kann vielschichtige Vorteile entfalten. Primär ermöglicht die komponentenbasierte Architektur eine effiziente Code-Entwicklung, indem wiederverwendbare Komponenten zur Strukturierung komplexer Benutzeroberflächen genutzt werden. Dies resultiert in einer verbesserten Wartbarkeit des Quellcodes und einer beschleunigten Entwicklungszeit [74].

Die reaktive Datenbindung von Vue.js bewirkt eine harmonische Synchronisation von Daten und Benutzeroberfläche, was die Schöpfung interaktiver und ansprechender Webanwendungen begünstigt. Die deklarative Syntax von Vue reduziert gleichzeitig den Boilerplate-Code und erleichtert die Nachvollziehbarkeit des Quellcodes. Vue.js ist zudem für seine aktive Entwicklergemeinschaft und die Verfügbarkeit einer Vielzahl von Erweiterungen und Plugins bekannt. Dies erlaubt den Zugriff auf bewährte Lösungen und bewährte Praktiken, was die Effizienz und Qualität eines Projekts immens steigern kann [74].

Schließlich bietet Vue.js eine attraktive Option für jene, die ein flexibles und gut dokumentiertes Framework suchen, das sich reibungslos in bestehende Projekte einfügt. Vue kann schrittweise übernommen und je nach

Projektanforderungen sowohl als Hauptframework als auch für spezifische Aufgaben verwendet werden [74].

Zusammengefasst gewährleistet Vue.js eine stabile Grundlage für die Gestaltung zeitgemäßer, interaktiver Webanwendungen und trägt entscheidend dazu bei, die Effizienz und Qualität von Projekten zu erhöhen. Vor diesem Hintergrund erweist sich die Verwendung von Vue.js in diesem Projekt als empfohlen, um die Vorzüge dieses robusten Frameworks voll auszuschöpfen. Für die Umsetzung des Konzepts wurden verschiedene andere Werkzeuge und Bibliotheken verwendet, jedoch wurden in diesem Abschnitt nur die wichtigsten vorgestellt.

5.2 Darlegung des Workflow-Prozesses

In diesem Abschnitt wird der umfassende Prozess zur Umsetzung des im vorherigen Kapitels skizzierten Konzepts 4.2 detailliert erörtert. Ein zentrales Element dieser Umsetzung ist das derzeit in der Entwicklungsphase befindliche Werkzeug, welches unter dem Namen **"GReQL Converter"** firmiert. Dieses Instrument dient dazu, das zuvor skizzierte Konzept in die praktische Anwendung zu überführen. Der GReQL Converter ist eine Webanwendung, die in der Lage ist, GReQL Code aus PlantText zu generieren, um damit die Evaluation von UML-Klassendiagrammen zu ermöglichen. Die technischen Einzelheiten der Implementierung dieses Tools werden im ausführlichen Abschnitt 5.3 eingehend beleuchtet. Der Workflow-Prozess gliedert sich in vier Phasen:

Tabelle 5.1: Workflowphasen

Darstellung der Workflowphasen
Phase 1: Modellierung einer Musterlösung mit PlantText
Phase 2: Verarbeitung der auf dem GReQL Converter erzeugten Regeln
Phase 3: Generierung des GReQL-Codes
Phase 4: Übertragung des GReQL-Codes auf JACK

Phase 1: Modellierung einer Musterlösung mit PlantText

Nachdem der Lehrer eine Übung in Textform erstellt hat, um die Studierenden zu bewerten, muss er die Musterlösung mithilfe von PlantText modellieren. Dabei müssen bestimmte Kriterien beachtet werden, die in der Dokumentation zur Verwendung des GReQL Converters ausführlich beschrieben sind. Diese Kriterien umfassen beispielsweise:

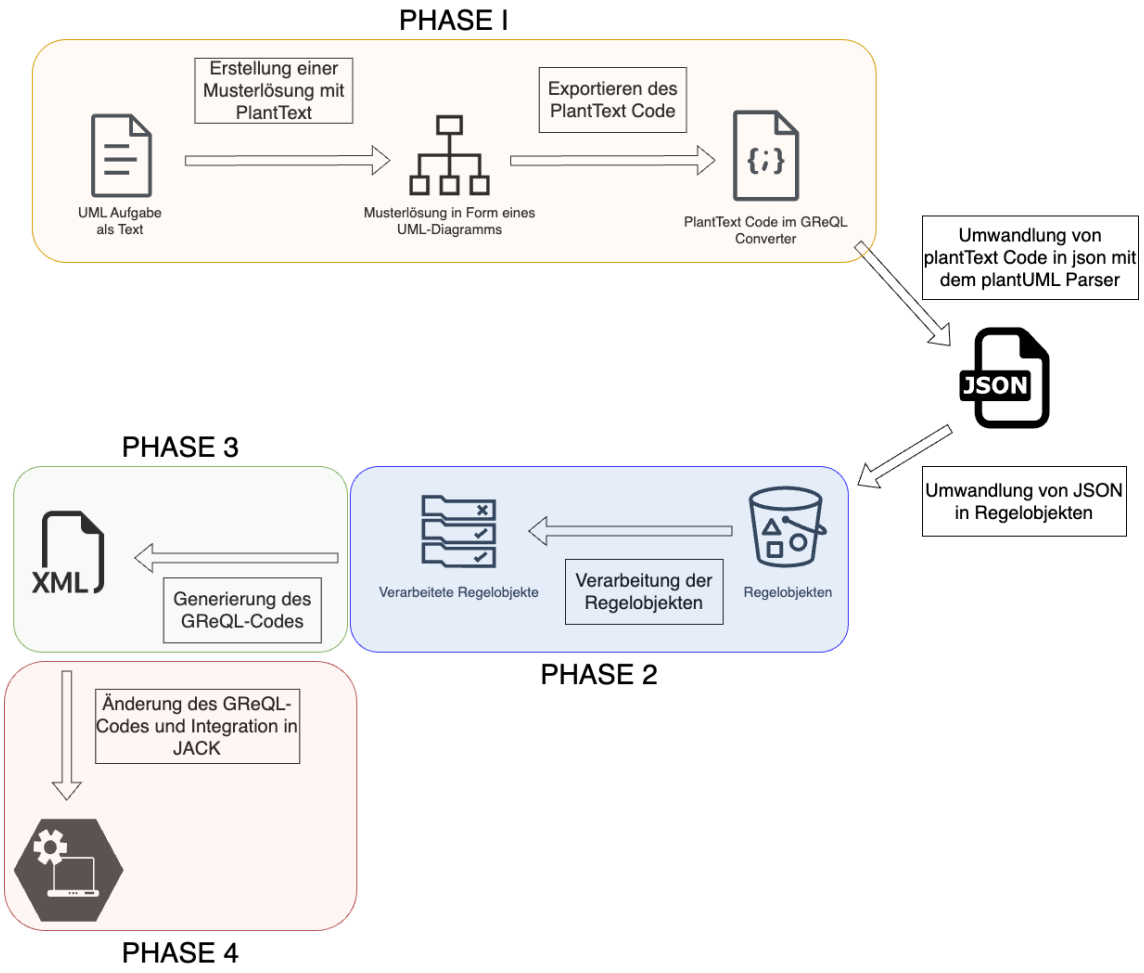


Abbildung 5.2: Repräsentatives Bild des Workflows des GReQL-Converters.

- Enums immer mit $<< enum >>$ zu kennzeichnen.
- Interfaces immer mit $<< interface >>$ zu annotieren.
- Eine spezielle Syntax, die für bestimmte Assoziationen einzuhalten ist.

Nachdem der Code für die Musterlösung generiert wurde, kann der Lehrer zur Phase 2 übergehen.

Phase 2: Verarbeitung der auf dem GReQL Converter erzeugten Regeln

In der Phase der Regelbearbeitung, die auf die Codegenerierung folgt, hat die Lehrkraft die Aufgabe, den generierten Code in den GReQL Converter zu übertragen. In dieser Phase entstehen eine Vielzahl von anpassbaren Regeln. Die Anpassung erfolgt gemäß den spezifischen Bewertungskriterien, die die Lehrkraft verfolgt. Es ist hierbei möglich, Änderungen an den Regeln vorzunehmen, je nachdem, welche Aspekte der Übung bewertet werden sollen. Weiterhin besteht die Option, Regeln hinzuzufügen oder zu entfernen. In der Bearbeitung der Regeln ergibt sich auch die Möglichkeit, individuelles Feedback für jede einzelne Regel hinzuzufügen und die Punktzahl für jede Regel erneut zu justieren.

Phase 3: Generierung des GReQL-Codes

Sobald die Phase der Regelbearbeitung (Phase 2) abgeschlossen ist, kann die Lehrkraft den GReQL-Code auf der Plattform generieren. Dabei orientiert sich die Generierung an den zuvor definierten Regeln. Der generierte GReQL-Code kann, falls erforderlich, weiterhin angepasst werden.

Dieser Bearbeitungsschritt ist nur dann sinnvoll, wenn der Lehrer bereits über Kenntnisse in GReQL zur Generierung von Regeln für Klassendiagramme verfügt. Dies ermöglicht es ihm, einige Details des Codes zu verfeinern. Der Zweck des GReQL Converters ist es, den GReQL-Code zu generieren, der am besten auf die Erwartungen der Lehrkraft eingeht, so dass dieser Änderungsschritt einfach übersprungen wird. Es ist jedoch möglich, dass dieser Schritt zu Beginn noch notwendig ist.

Phase 4: Übertragung des GReQL-Codes auf JACK

Nach Beendigung der Phase 3 kann die Lehrkraft den generierten GReQL-Code auf die JACK-Plattform übertragen. Dies erfolgt, nachdem die Übung auf der Plattform erstellt wurde. Anschließend haben die Benutzer die

Möglichkeit, ihre Übungen einzureichen, woraufhin eine Bewertung mit entsprechendem Feedback erfolgt.

Die hier skizzierten Phasen repräsentieren einen essenziellen Schritt in der Umsetzung des im vorherigen Kapitel Ansatzes zur Bewertung von UML-Diagrammen. Die Dokumentation dieses Prozesses gewährt einen umfassenden Einblick in die technischen Aspekte des Projekts und erlaubt eine transparente Darstellung des Implementierungsprozesses.

5.3 Einrichtung und Entwicklung des “GReQL Converters”

Der GReQL Converter stellt eine Webanwendung dar, die es ermöglicht, GReQL-Regeln aus einem UML-Klassendiagramm zu extrahieren, das mithilfe von PlantText erstellt wurde. Diese extrahierten Regeln dienen anschließend der Bewertung von UML-Klassendiagrammen auf der JACK-Plattform. Die Realisierung dieser Anwendung erfolgte mit dem Vue.js-Framework im Frontend und Node.js im Backend, wie in fig- 5.3 veranschaulicht. In diesem Kapitel wird der Implementierungsprozess der Plattform sowie ihrer verschiedenen Komponenten ausführlich erläutert.

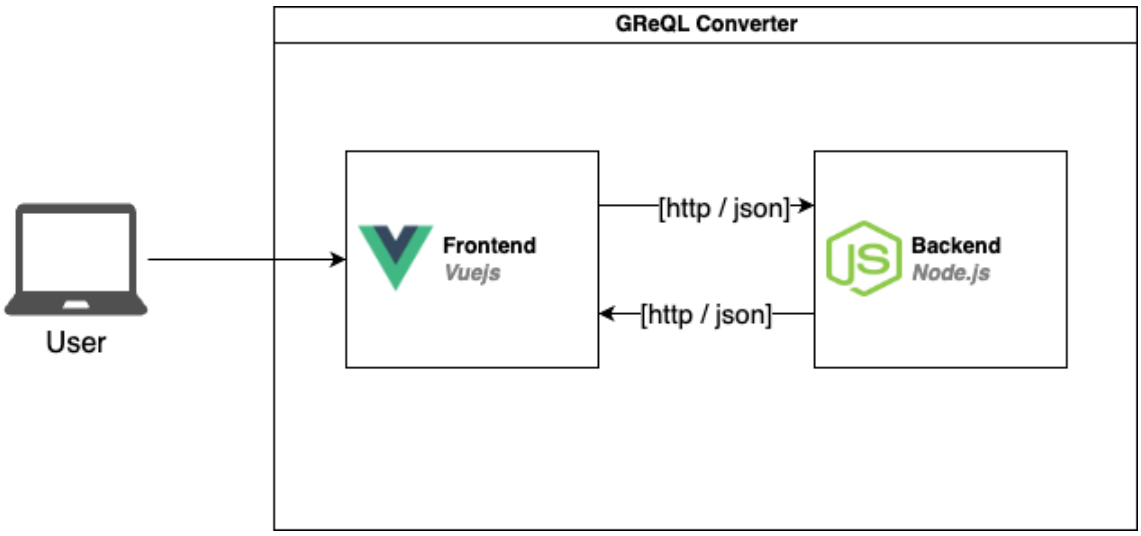


Abbildung 5.3: GReQL-Converter global infrastructure

5.3.1 Einrichtung des PlantUML-Parsers

Die Entwicklungsphase der Anwendung startete mit der Konfiguration des PlantUML-Parsers. Das ursprüngliche Ziel bestand darin, eine Anwendung zu entwickeln, die in der Lage ist, PlantText-Code als Eingabe zu akzeptieren und als Ausgabe ein JSON zu generieren, welches später zur Modellierung von Regelobjekten verwendet werden könnte. Ursprünglich war geplant, den Parser direkt im Frontend einzusetzen. Es stellte sich jedoch heraus, dass der Parser nur in einer Serverumgebung effektiv funktioniert. An dieser Stelle gab es zwei Optionen zur Auswahl: die Verwendung eines Tomcat-Servers mit Java oder eines Node.js-Servers mit JavaScript. Die zweite Option erwies sich als die geeignetere Wahl, da sie es ermöglichte, dieselbe Programmiersprache für das gesamte Projekt beizubehalten und die Gesamtarchitektur des Systems, einschließlich seiner zukünftigen Bereitstellung, zu vereinfachen. Node.js eignet sich besonders gut für kleinere Projekte wie dieses.

Im Großen und Ganzen wurde Node.js ausschließlich für die Bereitstellung des Parsers genutzt, was den Backend-Code erheblich vereinfachte A.1. Dies ermöglichte einen reibungslosen Übergang zum nächsten Schritt, nämlich der Erstellung des Grunddesigns der Anwendung.

5.3.2 Erstellung des Grunddesigns der Anwendung

Um die Ziele des GReQL Converters zu erreichen, ist es von entscheidender Bedeutung, den Benutzern eine elegante, intuitive und benutzerfreundliche grafische Benutzeroberfläche zur Verfügung zu stellen. Die möglichen Aktionen müssen auf den ersten Blick leicht erkennbar sein, um das Verständnis zu fördern und dem Benutzer eine schnelle und effiziente Arbeit zu ermöglichen [30]. Die Seite “Class Converter” stellt die Benutzeroberfläche dar, auf der der Benutzer seinen PlantText-Code in GReQL-Code umwandeln kann. Diese Seite ist in drei Hauptabschnitte unterteilt:

Teil 1: Code editor

Dieser Abschnitt der grafischen Benutzeroberfläche ist für die Eingabe des PlantUML-Codes durch den Benutzer vorgesehen. Nachdem der Benutzer die Musterlösung in PlantText modelliert hat, muss er den generierten Code kopieren und in den Editor einfügen, der durch die Nummer 1 (siehe 5.4) gekennzeichnet ist. Darüber hinaus hat der Benutzer die Möglichkeit, Beispielscodes auszuwählen, wie dies in PlantText der Fall ist und in der Abbildung unter Nummer 2 (siehe 5.4) dargestellt ist. Da der geschriebene

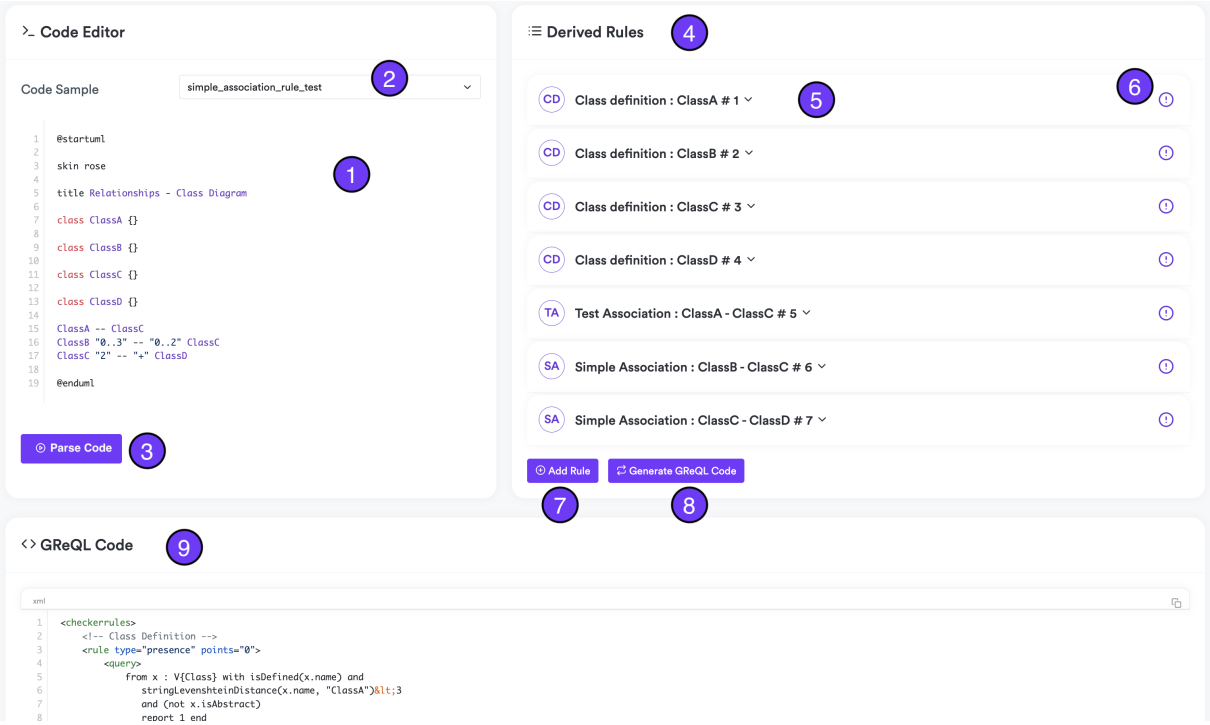


Abbildung 5.4: Dashboard im Überblick

Code jedoch der in der Dokumentation definierten Struktur entsprechen muss, sind diese Beispielscodes für den Benutzer nützlich, da sie funktionierende Codebeispiele liefern, von denen er sich inspirieren lassen und seinen eigenen Code schreiben kann, um die Plattform zu testen. Schließlich ermöglicht der Button “Parse Code” (Nummer 3 - siehe 5.4) das Extrahieren der Regelobjekte aus dem Code, was uns zum Teil 2 führt.

Teil 2: Derived Rules

Nachdem der Benutzer auf den Button “Parse Code” geklickt hat, wird eine Reihe von Regelobjekten (Nummer 4 - siehe 5.4) aus dem im Code-Editor vorhandenen Code generiert. Diese Regeln werden durch ein Symbol und einen Titel dargestellt, was es ermöglicht, sie schnell zu unterscheiden und zu identifizieren. Ganz rechts neben dem Regelobjekt befindet sich ein Ausrufezeichen, das tatsächlich ein Button ist (Nummer 6 - siehe 5.4). Wenn auf diesen Button geklickt wird, öffnet sich eine Dokumentation, die erklärt, was die Regel ist und wie sie verwendet wird. Wenn der Benutzer auf die Regel klickt (Nummer 5 - siehe 5.4), öffnet sich ein Menü, das es ihm ermöglicht, die in der Regel vorhandenen Informationen nach Belieben

zu ändern (siehe 5.5), wodurch der GReQL Converter zu einem halbautomatischen Werkzeug wird. Diese Informationen variieren je nach Art der Regel und den für die Generierung des GReQL-Codes erforderlichen Informationen.

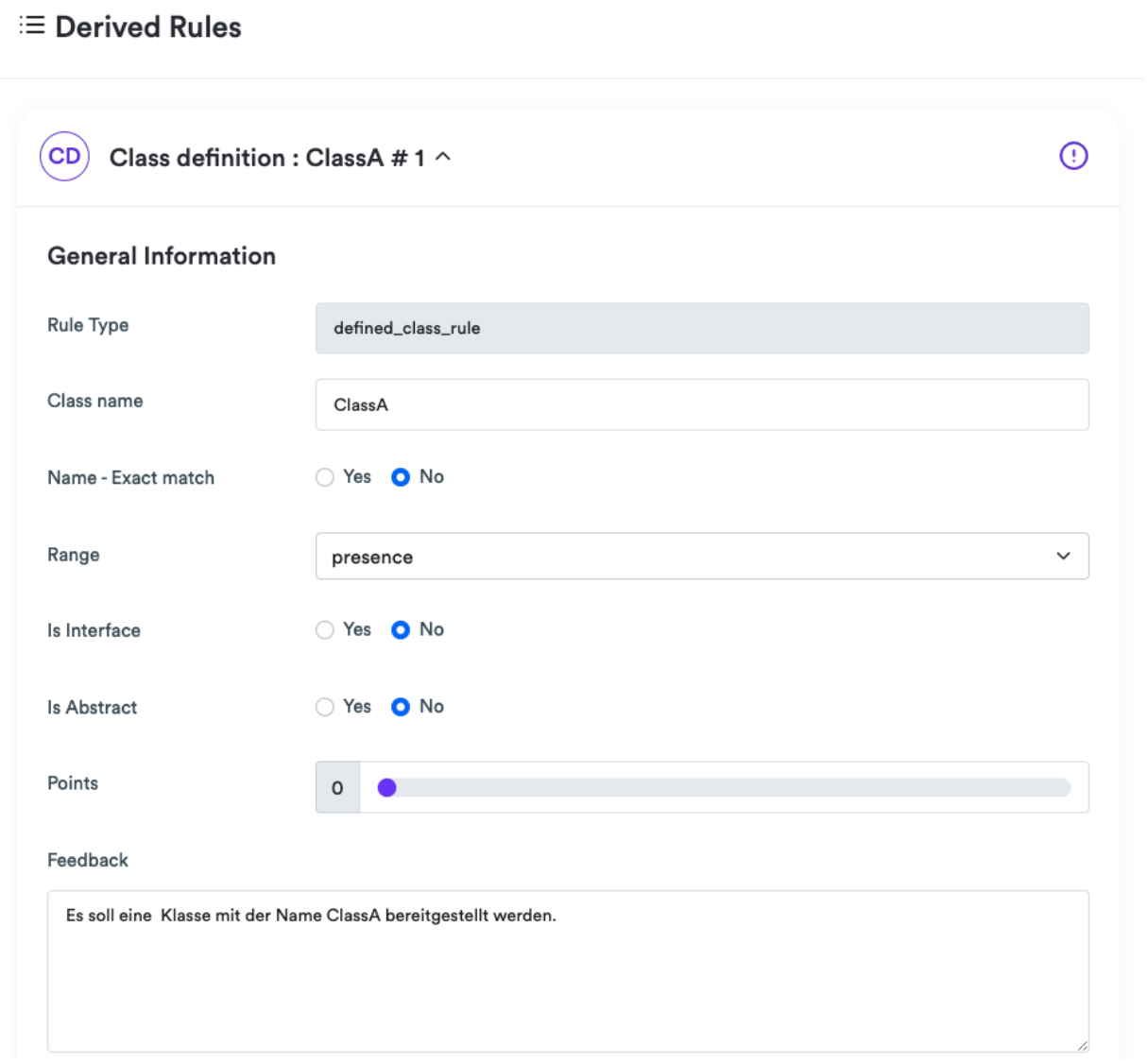


Abbildung 5.5: Beispiel eines Regelobjekts

Dann gibt es der Button “Add Rule” (Nummer 7 - siehe 5.4), die es dem Benutzer ermöglicht, zusätzliche Regeln hinzuzufügen. Nachdem die Konfiguration und die Änderungen abgeschlossen sind, kann der Benutzer den GReQL-Code generieren, indem er auf den Button “Generate GReQL

Code” klickt (Nummer 8 - siehe 5.4). Der GReQL-Code wird aus den zuvor vom Benutzer konfigurierten Regelobjekten generiert.

Teil 3: GReQL Editor

Genau wie der erste Abschnitt ist dieser Bereich auch ein Code-Editor, jedoch für XML, da der GReQL-Code im XMI-Format vorliegt (das auf XML basiert). Dieser Editor (Nummer 9 - siehe 5.4) ermöglicht es erfahrenen Benutzern, die bereits Erfahrung mit GReQL haben, detaillierte und fortgeschrittene Änderungen am generierten Code vorzunehmen. Auf diese Weise können sie die Abfragen verbessern und spezifizieren, wenn sie sie zu allgemein finden, oder sie einfach anpassen, je nachdem, was sie im UML-Diagramm bewerten möchten.

Diese drei verschiedenen Teile bilden das grundlegende Design des GReQL Converters. Es gibt auch andere Seiten wie die Dokumentation oder die Startseite, die nicht erwähnt wurden, da sie in der weiteren Entwicklung keine wesentliche Rolle spielen. Dieses Design bietet jedoch die Möglichkeit zur Erweiterung, um die Hinzufügung weiterer Seiten und sogar weiterer Parser zu erleichtern. Der nächste Abschnitt konzentriert sich auf den Prozess der Extraktion von Regelobjekten aus dem zuvor geparsten Code.

5.3.3 Regel-Extraktionsprozess

Um den Extraktionsprozess, der in diesem Kapitel beschrieben wird, einzuführen, wird mit einem Beispiel aus der Modellierung in PlantText angefangen. Die Schritte bis zur Erstellung der Regeln werden verfolgt. Hierzu wird der folgende PlantText-Code verwendet, der ein UML-Diagramm modelliert (siehe 5.6).

Das Diagramm enthält vier Klassen: ClassA, ClassB, ClassC und ClassD. ClassA hat drei Elemente: ein privates Attribut namens “Windows” vom Typ Integer (Int), ein öffentliches Attribut namens “Time” vom Typ Date und eine öffentliche Methode namens “Lock” ohne Parameter und mit dem Rückgabetyp “void”. ClassB und ClassC sind leere Klassen ohne definierte Attribute oder Methoden. ClassD ist ebenfalls eine leere Klasse. Im Diagramm sind verschiedene Beziehungen zwischen diesen Klassen dargestellt:

1. Es gibt eine Assoziationsbeziehung zwischen ClassA und ClassB mit einer Multiplizität von “1..2” auf der Seite von ClassA und einer Multiplizität von “*” auf der Seite von ClassB, was bedeutet, dass eine Instanz von ClassA mit 1 bis 2 Instanzen von ClassB verknüpft ist.

```
1  @startuml
2  class ClassA {
3    - Windows : Int
4    + Time : Date
5    +void Lock()
6  }
7  class ClassB {}
8  class ClassC {}
9  class ClassD {}
10
11 ClassA "1..2" -- "*" ClassB
12 ClassA o-- "*" ClassC
13 ClassB <|-- ClassC
14 ClassC -- ClassD
15
16 @enduml
```

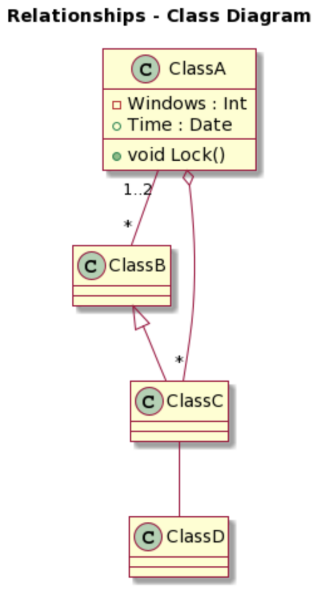


Abbildung 5.6: PlantText-Code mit der grafischen Darstellung

2. Es besteht eine Kompositionsbeziehung zwischen ClassA und ClassC, die durch ein Rautensymbol auf der Seite von ClassA gekennzeichnet ist, mit einer Multiplizität von “*” auf der Seite von ClassC. Diese Komposition zeigt an, dass ClassA eine Sammlung von Instanzen von ClassC besitzt und dass diese Instanzen von ClassC von den Instanzen von ClassA verwaltet werden.
3. Eine Vererbungsbeziehung (oder Generalisierung) ist zwischen ClassB und ClassC etabliert, gekennzeichnet durch einen Pfeil, der von ClassC auf ClassB zeigt, mit einer gestrichelten Linie. Dies bedeutet, dass ClassC eine Unterklasse von ClassB ist, was auf eine Vererbungsbeziehung hinweist.
4. Schließlich gibt es eine einfache Assoziationsbeziehung zwischen ClassC und ClassD, dargestellt durch eine durchgehende Linie, was darauf hinweist, dass es eine Verbindung zwischen den beiden Klassen gibt, obwohl die Details dieser Assoziation im Diagramm nicht spezifiziert sind.

Dies ist eine Darstellung des PlantText-Codes. Sobald dieser Code im GReQL Converter vorliegt und der Benutzer auf “Parse Code” klickt (Nummer 5 siehe 5.4), wird dieser Code über das HTTP-Protokoll an den Node.js-Server gesendet. Der Server ist dafür verantwortlich, den Code

zu analysieren, parsen und ein JSON-Objekt zurückzugeben, ähnlich dem unten dargestellten:

```
[
  {
    "elements": [
      {
        "name": "ClassA",
        "title": "ClassA",
        "isAbstract": false,
        "members": [
          {
            "name": "Windows",
            "isStatic": false,
            "accessor": "-",
            "type": "Int"
          },
          {
            "name": "Time",
            "isStatic": false,
            "accessor": "+",
            "type": "Date"
          },
          {
            "name": "Lock",
            "isStatic": false,
            "accessor": "+",
            "returnType": "void",
            "_arguments": ""
          }
        ],
        ...
      },
      ...
      {
        "left": "ClassA",
        "right": "ClassB",
        "leftType": "Unknown",
        "rightType": "Unknown",
        "leftArrowHead": "",
        "rightArrowHead": "",
        "leftArrowBody": "-",
        "rightArrowBody": "-",
        "leftCardinality": "1..2",
```

```
42          "rightCardinality": "*",
43          "label": "",
44          "hidden": false
45        },
46        ...
47        {
48          "left": "ClassC",
49          "right": "ClassD",
50          "leftType": "Unknown",
51          "rightType": "Unknown",
52          "leftArrowHead": "",
53          "rightArrowHead": "",
54          "leftArrowBody": "-",
55          "rightArrowBody": "-",
56          "leftCardinality": "",
57          "rightCardinality": "",
58          "label": "",
59          "hidden": false
60        }
61      ]
62    }
63  ]
```

Quelltext 5.2: Snippet des JSON-Codes, der nach dem Parsen des PlantText-Codes durch den Server erhalten wurde.

Nach Erhalt des JSON wird der Regelgenerierungsprozess gestartet. Basierend auf dem JSON kann für jedes Element eine zugehörige Regel identifiziert werden. Zum Beispiel, um eine Generalisierung (Vererbung) zwischen zwei Klassen zu erkennen, reicht es aus, ein Objekt aus dem JSON (hier als “elem” bezeichnet) zu betrachten und zu überprüfen, ob folgende Bedingung erfüllt ist:

```
1 elem.leftArrowHead.includes("<|") &&
2 elem.leftArrowBody.includes("-") &&
3 elem.rightArrowBody.includes("-")
```

Quelltext 5.3: Evaluationsbedingung für eine Generalisierung

Dies gilt auch für alle anderen Regeln, die anhand des generierten JSON identifiziert werden können. Einige UML-Feinheiten werden jedoch nicht vom Parser erkannt, was in einem späteren Abschnitt erläutert wird. Bei der Entwicklung des GReQL Converters wurden eine Reihe von Regeln definiert, darunter:

Class definition

Diese Regel hat zum Ziel, die Anwesenheit oder Abwesenheit einer Klasse in einem UML-Diagramm zu bestimmen. Darüber hinaus ermöglicht sie die Extraktion verschiedener Merkmale dieser Klasse. Mit Hilfe dieser Regel können Informationen wie die Abstraktion der Klasse, ihre Interface-Natur, die Methoden (einschließlich der Parameter und des Rückgabetyps), die Attribute und ihre Typen extrahiert werden (siehe A.2). Nach Extraktion dieser Informationen wird ein entsprechendes JSON-Objekt erstellt und einer Liste von Objekten hinzugefügt.

Enum definition

Das Ziel dieser Regel besteht darin, Enumerationen in einem UML-Diagramm zu identifizieren. Nachdem eine Enumeration ordnungsgemäß identifiziert wurde, geht es darum, alle verschiedenen Attribute zu identifizieren, die sie zusammensetzen, was ebenfalls in dieser Regel durchgeführt wird (siehe A.2). Nach dieser Identifizierung wird ein entsprechendes JSON-Objekt erstellt und der Liste von Objekten hinzugefügt.

Generalization

Diese Regel dient dazu, die verschiedenen Vererbungsbeziehungen zwischen Klassen und Schnittstellen zu definieren. Sie wird erstellt, wenn eine Klasse von einer anderen erbt oder eine Schnittstelle implementiert. Die Details dieser Beziehung werden erkannt, und ein JSON-Objekt wird generiert und zu einer Objektliste hinzugefügt (siehe A.2).

Simple Association

Diese Regel zielt darauf ab, die Assoziationsbeziehungen zwischen Klassen mit ihren jeweiligen Vielfachen zu definieren. Wenn eine Assoziation erkannt wird, werden die Klassennamen und die Vielfachen in einem JSON-Objekt gespeichert, das anschließend zu einer Objektliste hinzugefügt wird (siehe A.2).

Composition und Aggregation

Diese beiden Regeln dienen dazu, die verschiedenen Compositions- und Aggregationsbeziehungen im UML-Diagramm zu identifizieren und zu definieren. Das generierte Objekt enthält auch Multiplizitäten, die bei der

Generierung von GReQL-Code wichtig sind. Dieses Objekt wird dann zu einer Liste von Objekten hinzugefügt (siehe A.2).

Association Class

Diese Regel ermöglicht es, die Beziehung von Assoziationsklassen zu identifizieren und zu definieren. Obwohl sie in den meisten UML-Diagrammen während der Modellierung selten verwendet wird, spielt sie dennoch bei der Bewertung eine wichtige Rolle. Das generierte JSON-Objekt enthält im Wesentlichen Informationen zur zugehörigen Klasse und nicht direkt zur Beziehung (siehe A.2).

Nomination Consistency (optional)

Diese Regel dient dazu, festzustellen, ob das zu bewertende Diagramm eine Konsistenz bei der Benennung der verschiedenen Attribute aufweist. Zum Beispiel gilt es als schlechte Praxis, Attribute gleichzeitig mit Groß- und Kleinbuchstaben zu benennen [4]. Ein JSON-Objekt wird generiert und zu einer Liste von Objekten hinzugefügt. Diese Regel ist optional, was bedeutet, dass sie zu den Regeln gehört, die vom Lehrer selbst hinzugefügt werden müssen. Sie wird also nicht automatisch generiert (siehe A.2).

Count Methods (optional)

Diese Regel ermöglicht es, die Anzahl der im UML-Diagramm vorhandenen Methoden zu definieren. Der Lehrer muss als Parameter die genaue Anzahl der Methoden angeben, die im UML-Diagramm des Schülers vorhanden sein müssen. Ein JSON-Objekt wird generiert und zu einer Liste von Objekten hinzugefügt. Diese Regel ist optional, was bedeutet, dass sie zu den Regeln gehört, die vom Lehrer selbst hinzugefügt werden müssen. Sie wird also nicht automatisch generiert (siehe A.2).

Count Attributes (optional)

Diese Regel ermöglicht es, die Anzahl der im UML-Diagramm vorhandenen Attribute zu definieren. Der Lehrer muss als Parameter die genaue Anzahl der Attribute angeben, die im UML-Diagramm des Schülers vorhanden sein müssen. Diese Regel ähnelt der Regel Count Methods (siehe A.2).

Test Association

Diese Regel dient einfach dazu, festzustellen, ob es eine Beziehung zwischen zwei Klassen gibt. Ein JSON-Objekt wird generiert und zu einer Liste von Objekten hinzugefügt (siehe A.2).

Die zuvor erwähnte Objektliste enthält nun alle Regeln, die aus dem vom Node.js-Server bereitgestellten JSON extrahiert wurden. Jede dieser Regeln wird auf der Frontend-Seite angezeigt und ist anpassbar (siehe 5.5). Der Benutzer hat somit die Möglichkeit, jede Regel nach seinen Wünschen anzupassen, indem er beispielsweise Kommentare, Punkte (Points) sowie jeden mit der jeweiligen Regel verknüpften Attribut ändert. Auf diese Weise werden die Regeln aus dem JSON extrahiert und angepasst. Anschließend folgt der nächste Schritt, bei dem diese Regeln in GReQL-Code umgewandelt werden.

5.3.4 Umwandlung der Regel-Objekte in GReQL-Code

Nach dem Extrahieren und Anpassen der Regeln ist es nun möglich, GReQL-Code aus der Liste zu generieren, die die Objekte darstellt, welche diese Regeln repräsentieren. Tatsächlich hat jede Regel eine entsprechende Vorlage. Obwohl es verschiedene Möglichkeiten gibt, GReQL-Code zu schreiben, wurde im Verlauf der Entwicklung nach einer optimierten Variante gesucht, die das schnellste und effizienteste Ergebnis mit minimalem Code ermöglicht. Konkret geht es darum, den GReQL-Code so anzupassen, dass er perfekt zur zu konvertierenden Regel passt, und ihn dann als Vorlage zu verwenden. Zum Beispiel für die Regel, die das Vorhandensein einer Klasse definiert (siehe 5.4):

```
1  <rule type="${rule.existence}" points="${rule.points}">
2      <query>
3          from x : V{Class}
4             with
5             isDefined(x.name) and
6             stringLevenshteinDistance(x.name,
7             "${rule.rule_specific.class_name}")&lt;3
8             ${abstractCode}
9             report 1 end
10     </query>
```

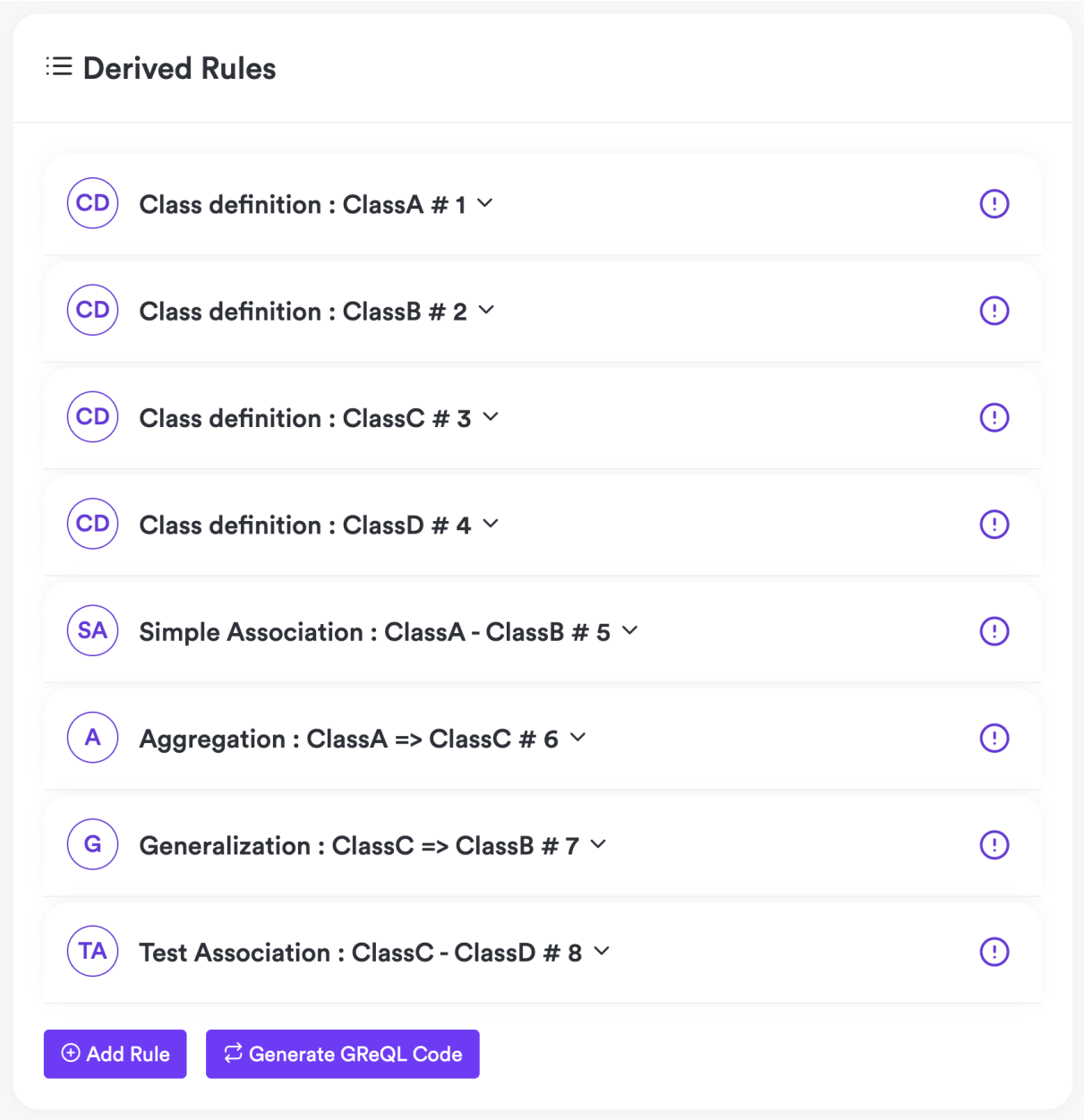


Abbildung 5.7: Regeln, die dem geparsten PlantText-Code entsprechen

```
11      <feedback>
12          ${rule.feedback}
13      </feedback>
14  </rule>
```

Quelltext 5.4: Class Definition Template $ersterTeil$

Nachdem die Vorlage erhalten wurde, werden die Parameter durch diejenigen aus dem Regelobjekt ersetzt. Auf diese Weise wird für jedes Objekt eine entsprechende Regel erstellt. Dieser Prozess kann je nach den verschiedenen Regeln und Objekten variieren, bleibt jedoch im Wesentlichen für die meisten Regeln gleich. Am Ende dieses Prozesses wird einen gültigen GReQL-Code erhalten, der auf der JACK-Plattform verwendet werden kann.

```
1   <checkerrules>
2
3       <!-- Class Definition -->
4       <rule type="presence" points="0">
5           <query>
6               from x : V{Class} with isDefined(x.name) and
7                  stringLevenshteinDistance(x.name,
8                  "ClassA")&lt;3
9                  and (not x.isAbstract)
10                 report 1 end
11          </query>
12          <feedback>
13              Es soll eine Klasse mit der Name
14              ClassA bereitgestellt werden.
15          </feedback>
16      </rule>
17
18      <!-- Class Definition -->
19      <rule type="presence" points="0">
20          <query>
21              from x : V{Class} with
22                 isDefined(x.name) and
23                 stringLevenshteinDistance(x.name,
24                 "ClassB")&lt;3
25                 and (not x.isAbstract)
26                 report 1 end
27          </query>
```

```
28          <feedback>
29              Es soll eine Klasse mit der Name
30              ClassB bereitgestellt werden.
31          </feedback>
32      </rule>
33
34      ...
35
36      <!-- Generalization rule -->
37      <rule type="presence" points="0">
38          <query>
39              from a,b : V{Class}
40                     with
41                        isDefined(a.name) and
42                        a.name="ClassC" and
43                        isDefined(b.name) and
44                        b.name="ClassB" and
45                        a --> V{Generalization} --> b
46                     report 1 end
47          </query>
48          <feedback>
49              Das Diagramm sollte eine Klasse
50              ClassC enthalten, die von einer
51              Oberklasse ClassB erbt.
52          </feedback>
53      </rule>
54      <!-- Test Association Rule -->
55      <rule type="presence" points="0">
56          <query>
57              from x,y : V{Class}
58                             with
59                                isDefined(x.name) and
60                                x.name="ClassC" and
61                                isDefined(y.name) and
62                                y.name="ClassD" and
63                                x --> V{Property}
64                                --> V{Association}
65                                &lt;-- V{Property}
66                                &lt;-- y
67                             report 1 end
68          </query>
69          <feedback prefix="Hinweis">
70              Im Diagramm gibt es keine directe Association
```

```
71              zwischen die Klasse "ClassD" und
72              die Klasse "ClassC". Das kann durch eine
73              bessere Modellierung vermieden werden.
74          </feedback>
75      </rule>
76  </checkerrules>
```

Quelltext 5.5: Ausschnitt aus dem GReQL-Code, der nach der Konvertierung erhalten wurde

5.4 Entwicklung eines Annotationssystems

In der in Abschnitt 5.2 beschriebenen ersten Phase (die das Schreiben des plantText-Codes in den GReQL Converter umfasst), folgt die zweite Phase, in der die generierten Regelobjekte modifiziert werden. Es ist jedoch wichtig zu beachten, dass dieser Prozess zur Modifikation der Regelobjekte als äußerst arbeitsintensiv angesehen werden kann. Um dies zu verdeutlichen, kann das Beispiel der Anpassung der Anzahl der Punkte (Points) für jede Regel in einem Diagramm mit mehr als zwanzig Regeln dienen. Diese Aufgabe erweist sich schnell als zeitaufwändig, da jede Regel individuell bearbeitet werden muss, um die erforderlichen Änderungen vorzunehmen. Ebenso stellt sich die gleiche Problematik ein, wenn es um die Eigenschaft “Exact match” (siehe 5.6) geht (die festlegt, ob der Name eines Attributs, einer Methode oder einer Klasse genau mit dem im Diagramm angegebenen Namen übereinstimmen muss oder ob eine gewisse Fehlermarge akzeptiert wird).

Um diesen Prozess zu erleichtern, wurde im GReQL Converter ein Annotationssystem implementiert. Dieses Vorhaben zielt zunächst darauf ab, einige Attribute des PlantUML-Parsers zu nutzen, die in Bezug auf UML-Diagramme vergleichsweise selten verwendet werden, insbesondere Generika, sowie das Labeling-System von PlantText. Hierfür wurde eine spezifische Syntax entwickelt, deren Einzelheiten in der Dokumentation des GReQL Converters erläutert sind. Das folgende Beispiel 5.8 zeigt einen Anwendungsfall dieser Syntax für die Annotation einer Klasse:

> **!class**: Die Verwendung der Direktive “!class” dient dazu, die Funktion der exakten Übereinstimmung (Exact match) anhand von Klassennamen zu aktivieren.
> **!attr(0,1,3)**: Die Direktive “!attr(0,1,3)” wird verwendet, um die exakte Übereinstimmung (Exact match) in den ersten, zweiten und vierten Attributen, nämlich “windows”, “x” und “time” zu aktivieren.

```
class A <!class, !attr(0,1,3), !method(1,2),
p=10, ad-p=3, md-p=5> {
  - windows : Int
  - x : Int
  - y : Int
  + time : Date
  + double lock(int age, bool status)
  + double unlock()
  + void block()
}
```

Abbildung 5.8: Beispiel für die Verwendung des Annotationssystems

!attr(*): Wenn beabsichtigt wird, die exakte Übereinstimmung (Exact match) in allen Attributen zu aktivieren, sollte die Direktive "!attr(*)" verwendet werden.
!method(1,2): Die Direktive "!method(1,2)" wird verwendet, um die exakte Übereinstimmung (Exact match) in den zweiten und dritten Methoden, nämlich "unlock" und "block" zu aktivieren.
!method(*): Sollte der Wunsch bestehen, die exakte Übereinstimmung (Exact match) in allen Methoden zu aktivieren, sollte die Direktive "!method(*)" verwendet werden.
p: Die Variable "p" dient dazu, das Ausmaß zu bestimmen, in dem die Punkte (Points) der Klassendefinitionsregel zuzuordnen sind.
ad-p: In ähnlicher Weise bestimmt "ad-p" die Punkte (Points), die den Attributregeln zugeordnet sind.
md-p: Gleichzeitig dient "md-p" als Indikator für die Punkte (Points), die den Methodenregeln zugeordnet sind.

Dieses Annotierungssystem beschleunigt und optimiert signifikant den Prozess der Generierung von GReQL-Code. Sobald die Beherrschung dieses Annotierungssystems erreicht ist, werden Lehrende nicht länger auf die Zwischenrepräsentation von Regelobjekten angewiesen sein, da sie in der Lage sein werden, GReQL-Code direkt mit der PlantText-Code und der

Annotation zu generieren.

5.5 Erreichte Ergebnisse

Der GReQL Converter ist ein Instrument zur Generierung von GReQL-Code aus zuvor bereitgestelltem PlantText-Code. Der vorherige Abschnitt hat im Detail den Prozess der Extraktion von Regeln aus PlantText-Code bis hin zur Generierung von GReQL-Code beschrieben. Es ist jedoch von entscheidender Bedeutung, die Frage zu beantworten, ob dieses Werkzeug effektiv funktioniert. Selbst wenn es funktioniert, ist seine Nützlichkeit von Interesse. Im folgenden Kapitel wird die Frage der Evaluation erörtert. Dabei werden die verschiedenen Prozesse ausführlich beschrieben, die zur Prüfung des Tools verwendet wurden. Darüber hinaus wird eine gründliche Untersuchung durchgeführt, um die Relevanz dieses Tools für verschiedene Lehrkräfte nachzuweisen.

6. Evaluation

Dieses Kapitel widmet sich einer umfassenden Analyse der Relevanz und Effektivität des GReQL Converters als innovatives Werkzeug. Spezifisch zielt diese Abschnitt darauf ab, die grundlegende Frage zu beantworten, ob dieses Instrument tatsächlich im akademischen und pädagogischen Kontext nützlich ist. Um dieses Ziel zu erreichen, ist es von größter Wichtigkeit, eine systematische Herangehensweise zu verfolgen, die die Anwendung verschiedener Bewertungsmethoden beinhaltet, um die Vorzüge und Effektivität des GReQL Converters nachzuweisen. Dieses Kapitel behandelt ausführlich die angewandten Ansätze zur Prüfung des GReQL Converters, die Datensammlungsmethoden und die durchgeführten Analysen zur Messung seiner Nützlichkeit. Letztendlich geht es darum, empirisch festzustellen, ob dieses Werkzeug konkrete Vorteile und einen signifikanten Mehrwert für Lehrende und Lernende bietet.

6.1 Erreichte Ziele

Die Untersuchung der Effizienz und Funktionalität des GReQL Converters als Instrument zur Extraktion relevanter GReQL-Regeln aus einer UML-Diagrammannotation, die mittels PlantText zur Evaluation von UML-Diagrammen erstellt wurde, stellt eine essenzielle Fragestellung dar, welche eine systematische und tiefgehende Herangehensweise erfordert. In dem Implementierungskapitel (siehe 5) wurden diverse Regeln vorgestellt, begleitet von einer detaillierten Erläuterung des spezifischen Extraktionsprozesses für jede dieser Regeldefinitionen. Gleichwohl erwies sich eine umfassende Testphase als unverzichtbar, um eine eingehende Evaluierung der Leistungsfähigkeit jeder einzelnen Regel zu ermöglichen.

Zur Durchführung dieser Evaluierungen wurden verschiedene Testverfahren appliziert, welche eine Vielzahl von Szenarien und Variationen abdeckten, mit dem Ziel, die Effizienz der durch den GReQL Converter generierten Regeldefinitionen im Detail zu beurteilen. Der Testprozess

kann in vier diskrete Schritte unterteilt werden:

1. Initiale Generierung eines Diagramms mithilfe von PlantText, wobei dieses Diagramm gezielt konzipiert wurde, um die spezifischen Merkmale der zu evaluierenden Regel zu inkorporieren. Zum Beispiel wurde ein Diagramm erstellt, welches eine Aggregation zwischen zwei Klassen darstellte, um eine Regel zur Aggregation zu prüfen.
2. Erstellung eines Evaluationsdiagramms (welches in diesem Kontext mittels BOUML modellisiert wurde), welches von den generierten Regeldefinitionen bewertet werden sollte.
3. Extraktion der verschiedenen Regeldefinitionen aus dem PlantText-Code.
4. Evaluierung mithilfe der GReQL-Engine von JACK, um festzustellen, ob die Regeldefinitionen die Aggregation im Diagramm effektiv erkannt haben.

Parallelen dazu wurde besonderes Augenmerk auf die Identifikation von falsch positiven (False Positiv) Ergebnissen gerichtet, welche den Eindruck einer korrekten Funktionsweise des Werkzeugs erwecken könnten, obwohl dem nicht so ist. Hierzu wurde eine sorgfältige Analyse des generierten Codes durchgeführt, und die Protokolle der GReQL-Engine wurden überprüft, um die präzise Identifikation der Aggregationsregel sicherzustellen. Es sei hervorgehoben, dass dieser gewissenhafte Überprüfungsprozess in systematischer Weise auf alle vorab definierten Regeldefinitionen der Implementierungsphase angewandt wurde.

Im Anschluss an die individuelle Prüfung jeder Regeldefinition erfolgte eine Evaluierung anhand zunehmend komplexerer Diagramme. Das Ziel bestand darin, mehrere Funktionen simultan zu bewerten und zu ermitteln, ob die Existenz mehrerer Verknüpfungen zwischen verschiedenen Elementen im Diagramm spezifische Regeldefinitionen nicht beeinträchtigte. Die Bedingungen, die zu potenziellen Konflikten führen könnten, wurden somit eingehend untersucht, und der GReQL Converter wurde bei jeder Fehlererkennung oder -identifikation angepasst, um seine Effizienz zu steigern.

In einem darauffolgenden Schritt wurden verschiedene gängige Übungen zu UML-Diagrammen selektiert, um die akademische Leistung der Studierenden an der Universität zu evaluieren. Unter diesen Übungen findet sich das Diagramm “Mendelssohn & Sohn Maschinenbau GmbH”, welches im Rahmen des Kurses “Einführung in die Unified Modeling Language” an der Universität Potsdam zum Einsatz kam (siehe 6.1). Dieses spezifische Diagramm weist signifikante Ähnlichkeiten zu den traditionellen Modellen auf, die in universitären Prüfungskontexten verwendet werden, wodurch es

sich als exemplarisches Testobjekt zur Ermittlung der Leistungsfähigkeit des GReQL Converters erweist.

Zu diesem Ziel wurde das Musterdiagramm, das als Repräsentation der Übungsvorlage diente, mittels des Werkzeugs BOUML modelliert. Anschließend erfolgte die Extraktion der XMI-Datei, welche als Bewertungsinstrument diente. Parallel dazu wurde die Anwendung von PlantText zur Erstellung der Musterdiagrammvorlage in Erwägung gezogen, wobei der GReQL Converter im Anschluss dazu verwendet wurde, um die Regeln gemäß dem vorab beschriebenen Verfahren formal zu erfassen. Subsequent zur Extraktion der Regeldefinitionen und der willkürlichen Punktevergabe für jede einzelne Regel, unterzog man diese Regeldefinitionen einer Evaluation im Rahmen des GReQL-Motors von JACK.

Hervorzuheben ist, dass für ein mittelkomplexes Diagramm wie das genannte insgesamt **82 GReQL-Regeln** generiert wurden. Nach abschließenden Tests ergab sich, dass der GReQL-Motor eine Erfolgsrate von 100% aufwies, was auf die korrekte Funktionalität der erzeugten Regeldefinitionen hinweist. Dieser Erfolg illustriert eindrucksvoll die Befähigung des GReQL Converters zur effektiven Evaluierung von UML-Diagrammen, insbesondere jener von erhöhter Komplexität.

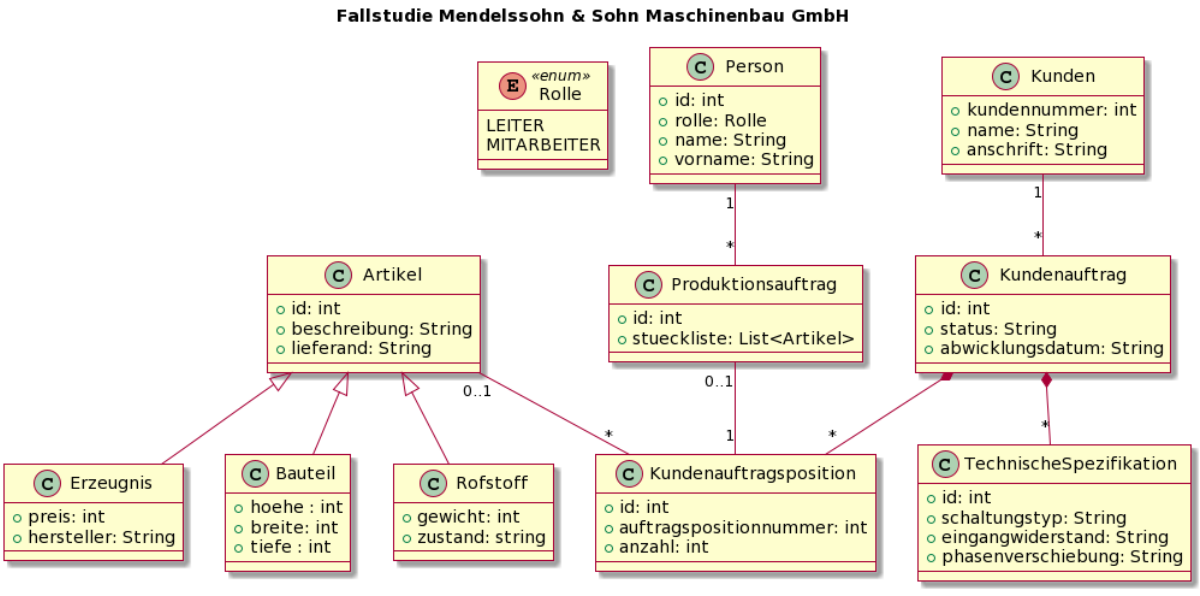


Abbildung 6.1: Fallstudie UML - Klassendiagramm

In Bezug auf die Leistungen, die dem GReQL Converter zugeschrieben werden können, lassen sich folgende Punkte feststellen:

1. Der GReQL Converter ermöglicht die umfassende Bewertung eines Diagramms. Im vorherigen Beispiel ermöglichte er die Generierung von 82 Regeln, eine Aufgabe, die besonders mühsam und fehleranfällig wäre, wenn sie manuell von einem Lehrer durchgeführt würde.
2. Der GReQL Converter führt Bewertungen äußerst präzise durch und verhindert somit potenzielle Fehler, die Lehrer bei manuellen Bewertungen begehen könnten.
3. Der GReQL Converter verhindert Fehler in Bezug auf die Syntax und Formulierung von Regeln, insbesondere solche, die bereits vom Tool definiert sind, was den Benutzer von Sorgen in Bezug auf diese Aspekte entlastet.
4. Darüber hinaus ist eine direkte Interaktion mit dem GReQL-Code nicht mehr erforderlich, da der Benutzer die Regeln problemlos ändern kann, indem er die generierten Objekte nach dem Schritt der syntaktischen Analyse verwendet, was den Bearbeitungsprozess vereinfacht.
5. Der GReQL Converter erleichtert erheblich das Verständnis des GReQL-Codes, indem er klaren, lesbaren und zugänglichen Code generiert, was es Dritten ermöglicht, seine Funktionsweise zu verstehen und bei Bedarf Änderungen vorzunehmen.
6. Darüber hinaus berücksichtigt das Design des Tools die Vielfalt grundlegender Varianten in UML-Diagrammen, was seine Vielseitigkeit und Anpassungsfähigkeit stärkt.

Wenn man diese Vorteile mit den in Teil drei aufgeführten Problemen vergleicht, die die Problemanalyse (siehe 3) behandelten, steht außer Frage, dass der GReQL Converter seine Rolle als Hilfsmittel bei der Erstellung von GReQL-Code für die Bewertung von Klassendiagrammen erfüllt und Lehrern erheblich Zeit spart. Im nächsten Abschnitt wird eine Studie durchgeführt, um zu ermitteln, ob Lehrende diese Ansicht teilen.

6.2 Umfrage zur Bewertung des GReQL Converters

todo - write something here

7. Diskussion

Dieses Kapitel wird in mehrere Abschnitte unterteilt sein. Zunächst wird eine Interpretation der Ergebnisse der Evaluation vorgenommen, um fundierte Schlussfolgerungen zu ziehen. Anschließend wird eine eingehende Diskussion über die verschiedenen Herausforderungen geführt, die im Laufe des Entwicklungsprozesses des Tools bewältigt wurden. Abschließend wird eine umfassende Untersuchung durchgeführt, um die verschiedenen Möglichkeiten zur Verbesserung des GReQL Converters zu erörtern und sein fortwährendes Potenzial zu bewerten. Diese thematische Struktur zielt darauf ab, eine ganzheitliche und gründliche Analyse der Leistung, der Herausforderungen und der Verbesserungsaussichten im Zusammenhang mit diesem Tool bereitzustellen.

7.1 Interpretation der Evaluationsergebnisse

todo - write something here

7.2 Herausforderungen während des Entwicklungsprozesses

Im Verlauf des Entwicklungsprozesses manifestierten sich verschiedene herausfordernde Sachverhalte, die eine gezielte Entwicklungsdynamik bedingten. Dies wiederum zwang die Notwendigkeit zur Implementierung spezifischer Beschränkungen oder die Abkehr von bestimmten funktionalen Aspekten.

PlantText Parser

Bezüglich des PlantText Parsers sind gewisse Limitationen zu konstatieren. Er ist nicht in der Lage, statische Attribute, statische Methoden und

statische Klassen zu erfassen. Indessen fand rasch eine Lösung in Form eines Kompromisses Anklang, indem dem Anwender ermöglicht wird, diese Modifikationen manuell im Rahmen des Regel-Editors vorzunehmen.

GReQL Engine Optimizer

Der GReQL Engine Optimizer verfügt über einen Algorithmus zur Optimierung von Abfragen, um deren Ausführung zu erleichtern und mögliche Probleme wie die Verwendung undefinierter Variablen, welche eine Abfrage fehlerhaft machen könnten, zu umgehen. Nichtsdestoweniger kann dieser Optimierer zuweilen Unklarheit in der Abfrageausführung stiften. Es besteht die Möglichkeit, dass eine Abfrage verfasst wird, die auf den ersten Blick in vollkommen korrektem Einklang erscheint, jedoch bei der Ausführung vom Optimierer in einer Art und Weise modifiziert wird, welche die Abfrage invalide werden lässt (Wie es bei einigen Regeln im WIKI der Fall ist [50]). Daraus resultiert, dass die GReQL Engine Fehlermeldungen retourniert. Zur Bewältigung dieser Thematik waren eigens maßgeschneiderte Abfragen erforderlich, welche verschiedene Prüfungen vor der Ausführung durchführen. In dieser Hinsicht erweist sich die Verwendung des GReQL Converters als vorteilhaft, indem er ausschließlich valide Abfragen zur Optimierung generiert und dem Nutzer die Frustration erspart, scheinbar korrekte, aber nicht funktionierende Abfragen manuell zu konzipieren.

Beschränkung auf BOUML

Der Kern des GReQL Converters liegt in der Erstellung und Definition von Vorlagen, die für jede Regel festgelegt wurden. Zur Herstellung dieser Vorlagen war es erforderlich, zunächst ein Diagramm, welches die jeweilige Regel in Anspruch nimmt, mittels der Software BOUML zu modellieren. Anschließend erfolgte die grafische Darstellung mithilfe des GReQL Engine, um abschließend die Regel aus der grafischen Darstellung abzuleiten. Diese Vorgehensweise impliziert, dass die Mehrzahl der in Gebrauch genommenen Regeln ihren Ursprung in einer bildlichen Repräsentation eines Diagramms haben, welches mithilfe von BOUML erstellt wurde. Dies stellt ein substantielles Problem dar, da die XMI-Repräsentationen der Diagramme abhängig vom verwendeten Tool variieren. Als Beispiel generiert der Enterprise Architect offensichtlich eine XMI-Datei, die sich von derjenigen generiert durch BOUML zu unterscheiden scheint. Dies hätte zur Konsequenz, dass die Mehrzahl der durch den GReQL Converter generierten Regeln ungültig würde, sofern das zu beurteilende Diagramm mittels eines alternativen Tools geschaffen wurde. Das bedeutet, dass die Auswahl des

Tools, das für die Generierung der Lösungen zur Beurteilung eingesetzt wird, von entscheidender Relevanz ist, was wiederum den GReQL Converter auf eine spezifische Werkzeugauswahl oder auf die Nutzung von BOUML für die Gestaltung der zu beurteilenden Diagramme beschränkt.

Primitive Datentypen

Im Zusammenhang mit den von BOUML generierten Diagrammen ist zu berücksichtigen, dass sie einem ausgewiesenen Standard entsprechen, nämlich dem UML-Standard 2.3 [28], der von BOUML in Gebrauch genommen wird. Dieser Standard erkennt jedoch lediglich vier primitive Datentypen, nämlich int, bool, string und UnlimitedNatural [28]. Diese Beschränkung führt dazu, dass Typen, die im Grundsatz als primitiv erachtet werden könnten, wie double, float, char und dergleichen, schlichtweg nicht berücksichtigt werden. Dieses Problem hat zur Folge, dass Typen in GReQL-Abfragen nicht überprüft werden können, sofern sie nicht den Kriterien des UML 2.3-Standards genügen. In praktischer Konsequenz mussten gewisse Funktionen aufgegeben werden, etwa die Überprüfung des Rückgabetyps einer Methode oder des Typs einer Variablen (sofern diese nicht gemäß UML 2.3 als primitiv gelten), da die Repräsentation, die durch die GReQL Engine erzeugt wird (basierend auf dem XMI von BOUML), diese Typen nicht erkennt und daher nicht darstellen kann. Infolgedessen können derlei Abfragen nicht ausgeführt werden.

Diese genannten Beschränkungen stellen zweifelsohne vielversprechende Ansatzpunkte für eine substantielle Verbesserung des GReQL Converters dar. Daher wird in dem folgenden Abschnitt eine Diskussion darüber eingeleitet, wie der GReQL Converter möglicherweise verbessert werden kann, um einige dieser inhärenten Einschränkungen zu überwinden.

7.3 Potenziale für Weiterentwicklungen

Der GReQL Converter, obwohl er vielversprechend ist, verwehrt sich der Illusion der Vollkommenheit. In diversen Domänen sind signifikante Verbesserungen realisierbar, um seine Effektivität bei der Bewältigung spezifischer Herausforderungen zu optimieren.

Hinzufügen neuer Regeln

Eine solche Möglichkeit zur Verbesserung manifestiert sich in der Erweiterung des Regelkatalogs. Obwohl die Entwicklung des GReQL Converters

bereits eine umfassende Berücksichtigung der Regeln, die der Modellierung von UML-Klassendiagrammen zugrunde liegen, einschloss, bleiben einige subtile Nuancen unvollständig berücksichtigt. Zum Beispiel wurden keine Regeln für Assoziationen mit spezifischer Richtung oder für nicht-ausgerichtete Beziehungen integriert. Während die Assoziation zwischen zwei Klassen betrachtet wird, sofern eine Beziehung zwischen ihnen besteht, erfolgt keine explizite Erfassung der Ausrichtung dieser Assoziation. Ebenso bleiben die mit Assoziationen verknüpften Rollennamen unberücksichtigt. Diese und andere Feinheiten könnten zukünftige Erweiterungen des GReQL Converters sein, um das Tool in Bezug auf die Generierung präziserer GReQL-Regeln zu bereichern.

Erweiterung bestehender Regeln

Mehrere Regeln könnten von Verbesserungen profitieren, um mit einer breiteren Palette von Entwurfsvarianten kompatibel zu sein. Hierfür ist es unerlässlich, den GReQL Converter auf einer signifikanten Anzahl von Diagrammen zu testen, um seine Grenzen zu ermitteln und sie umfassend anzugehen.

Erweiterung der Kompatibilität des GReQL Converter

Wie zuvor erwähnt, ist der von GReQL Converter generierte GReQL-Code derzeit zu 100% kompatibel mit Lösungsdiagrammen, die mit BOUML erstellt wurden. Der GReQL Converter wurde jedoch mit Blick auf die Erweiterbarkeit zu anderen Technologien entwickelt, die die Modellierung von UML-Diagrammen und die Generierung von XMI-Dateien ermöglichen. Die Erweiterung auf andere Diagramm-Modellierungstools sollte hauptsächlich das Hinzufügen von regelbasierten Vorlagen für diese Tools und die Möglichkeit zur Auswahl des bevorzugten Tools über die grafische Benutzeroberfläche des GReQL Converter einschließen. Glücklicherweise sollte diese Aufgabe nicht besonders komplex sein.

Bei der Entwicklung des GReQL Converter war eine der entscheidenden Überlegungen, ein Tool zu schaffen, das sich im Laufe der Zeit weiterentwickeln kann. Aus diesem Grund wurde der Code unter Einhaltung von Designprinzipien und bewährten Praktiken geschrieben, um zukünftigen Entwicklern, die an dem Tool arbeiten, die Orientierung zu erleichtern und ihre Entwicklererfahrung erheblich zu vereinfachen [47].

8. Zusammenfassung und Ausblick

Dieses Kapitel markiert den Abschluss dieser Masterarbeit. In erster Linie wird eine kurze Zusammenfassung der erreichten Ergebnisse in dieser Arbeit präsentiert. Anschließend wird die Diskussion über potenzielle Ansätze und Alternativen eröffnet, die zur Lösung der in diesem Kontext aufgeworfenen Problematik erkundet werden könnten.

8.1 Zusammenfassung

In dieser Masterarbeit wurde die Generierung von Feedback-Regeln für UML-Modelle im Kontext von E-Assessment-Systemen untersucht. Die Untersuchung umfasste die Analyse der Herausforderungen, die im Zusammenhang mit der manuellen Erstellung von GReQL-Code für die Bewertung von UML-Modellen auftreten können, sowie die Vorstellung des GReQL Converters als Lösungsansatz. Im dritten Kapitel erfolgte eine detaillierte Analyse der Problematiken, die im Rahmen der manuellen Erstellung von GReQL-Code auftreten können. Diese Probleme umfassen die notwendige Expertise für die Erstellung von GReQL-Code, die Komplexität bei der Formulierung von Regelwerken sowie den zeitlichen Aufwand, der für die manuelle Erstellung von Feedback-Regeln erforderlich ist. Die erwähnten Schwierigkeiten können zu einer zeitaufwändigen und fehleranfälligen manuellen Erstellung von Feedback-Regeln führen.

Der GReQL Converter präsentiert sich als Lösung für diese Problematiken. Durch den Einsatz dieses Konverters können Feedback-Regeln automatisch generiert werden, ohne dass umfassende Kenntnisse im Bereich GReQL-Code erforderlich sind. Zusätzlich dazu ist der Konverter in der Lage, komplexe Regeln zu formulieren und den zeitlichen Aufwand für die manuelle Erstellung von Feedback-Regeln zu reduzieren. Die Ergebnisse dieser Untersuchung legen nahe, dass der GReQL Converter als ein nützliches Instrument für die automatisierte Bewertung von UML-Modellen betrachtet werden kann. Er trägt dazu bei, den Prozess der Generierung

von Feedback-Regeln zu vereinfachen.

In der Gesamtschau der vorliegenden Masterarbeit wird aufgezeigt, dass die Nutzung des GReQL Converters eine wirksame Lösung für die Herausforderungen bei der manuellen Erstellung von Feedback-Regeln für UML-Modelle darstellt. Die hier erzielten Erkenntnisse haben das Potenzial, den Einsatz von E-Assessment-Systemen zu optimieren und den Lernprozess für Lehrende und Lernende zu verbessern.

8.2 Ausblick

Für die Entwicklung des GReQL Converters wurde ursprünglich PlantText aufgrund seiner Fähigkeit gewählt, mithilfe des PlantUML Parser Beziehungen und Entitäten aus einem UML-Diagramm zu extrahieren und sie in einem leicht verwertbaren JSON-Format zu exportieren. Es gibt jedoch mehrere andere, komplexere Strategien, die in Erwägung gezogen werden können. Aktuell erfordert die Verwendung des GReQL Converters die Anwendung von PlantText, während für die Erstellung des zu bewertenden Diagramms auf der JACK-Seite die Generierung einer XMI-Datei mit BOUML erforderlich ist. Dies führt zu einer gewissen Heterogenität im Arbeitsablauf. Die Erkundung der Möglichkeit, Regeln direkt aus einer XMI-Datei zu generieren, könnte ein interessanter Entwicklungsbereich für den GReQL Converter sein. Dies wäre jedoch aufgrund der Entwicklung eines angepassten Parsers, die bereits eine recht komplexe Aufgabe darstellt, anspruchsvoll. Dennoch würde dies zweifellos einen Mehrwert für den Entwicklungsprozess bieten und es ermöglichen, aus einer Lösung, die beispielsweise in BOUML erstellt wurde, nicht nur GReQL-Regeln zu generieren, sondern sie auch direkt zu überprüfen. Der Benutzer müsste keine zwei unterschiedlichen Technologien erlernen, um diese Aufgabe zu erfüllen.

Ein weiterer Entwicklungsbereich, der von Anfang an aufgegeben wurde, aber dennoch von Interesse sein könnte, ist die Integration eines KI-Algorithmus. Ein solcher Algorithmus könnte dazu befähigt sein, ein zuvor annotiertes UML-Übungsszenario zu lesen und daraus abzuleiten, welche Regeln generiert werden sollten. Eine ähnliche Aufgabe wurde bereits in einem anderen Kontext von Mohammed Amraouy et al.[6] realisiert, bei dem mithilfe von Studentenkommentaren auf einer E-Assessment-Plattform eine Analyse und Klassifizierung der Emotionen durchgeführt wurde, um das Engagement der Studenten in einem bestimmten Kurs zu bewerten. In einem Kontext, der dem dieser Masterarbeit ähnlicher ist, ist auch die Arbeit von Aggarwal et al.[3] relevant, die verschiedene Algorithmen des maschi-

nellen Lernens verwenden, um verschiedene Entitäten auf der Grundlage von Text in Kategorien zu klassifizieren.

Da das Tool nicht vollständig automatisiert werden kann, ist nach wie vor das Eingreifen eines Benutzers erforderlich, um die Richtigkeit und Konsistenz der generierten Regeln zu überprüfen.

Literatur

[1] O. M. 2.0. *XML Metadata Interchange (XMI)*. 2007.
URL: http://www.omg.org/spec/XMI/2.1.1/..

[2] W. Abramowicz.
Business Information Systems: 16th International Conference, BIS 2013, Poznań, Poland, June 19-21, 2013, Proceedings. Bd. 157.
Springer, 2013.

[3] C. C. Aggarwal und C. C. Aggarwal.
Machine learning for text: An introduction. Springer, 2018.

[4] M. Albert, V. Pelechano, J. Fons, M. Ruiz und O. Pastor.
„Implementing UML association, aggregation, and composition. A particular interpretation based on a multidimensional framework“.
In: *Advanced Information Systems Engineering: 15th International Conference, CAiSE 2003 Klagenfurt/Velden, Austria, June 16–20, 2003 Proceedings 15*. Springer. 2003, S. 143–158.

[5] N. Alruwais, G. Wills und M. Wald.
„Advantages and challenges of using e-assessment“.
In: *International Journal of Information and Education Technology* 8.1 (2018), S. 34–37.

[6] M. Amraouy, M. Bellafkih, A. Bennane und J. Talaghzi.
„Sentiment Analysis for Competence-Based e-Assessment Using Machine Learning and Lexicon Approach“. In: *The International Conference on Artificial Intelligence and Computer Vision*.
Springer. 2023, S. 327–336.

[7] O. Anas, T. Mariam und L. Abdelouahid. „New method for summative evaluation of UML class diagrams based on graph similarities“.
In: *International Journal of Electrical and Computer Engineering* 11.2 (2021), S. 1578–1590.

[8] Astah. *Astah UML*. 2023.
URL: https://astah.net/products/astah-uml/.

[9] L. Auxepaules, M. Alonso, M. Alonso, L. Auxepaules, D. Py und D. Py. „Diagram, a Learning Environment for Initiation to Object-Oriented Modelling with UML Class Diagrams“. In: (2015).

[10] J. Azevedo, E. P. Oliveira und P. D. Beites. „E-assessment and multiple-choice questions: a literature review“. In: *Handbook of Research on E-Assessment in Higher Education* (2019), S. 1–27.

[11] J. M. Azevedo. „E-Assessment in mathematics courses with multiple-choice questions tests“. In: *International Conference on Computer Supported Education*. Bd. 2. SCITEPRESS. 2015, S. 260–266.

[12] T. Benson, G. Grieve, T. Benson und G. Grieve. „UML, XML and JSON“. In: *Principles of Health Interoperability: FHIR, HL7 and SNOMED CT* (2021), S. 399–426.

[13] Blackboard. *Educational Technology Services — Blackboard — North America*. Accessed on September 6, 2023. 2023. URL: https://www.blackboard.com/.

[14] G. Booch. „The History of Software Engineering“. In: *IEEE Software* 35.5 (2018), S. 108–114. DOI: 10.1109/MS.2018.3571234.

[15] Y. Boubekeur, G. Mussbacher und S. McIntosh. „Automatic assessment of students' software models using a simple heuristic and machine learning“. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2020, S. 1–10.

[16] K. Chowdhary und K. Chowdhary. „Natural language processing“. In: *Fundamentals of artificial intelligence* (2020), S. 603–649.

[17] K. e.V. *Umbrello UML*. 2023. URL: https://uml.sourceforge.io/.

[18] B. Eilers, S. Gruttmann und H. Kuchen. „Konzeption eines integrierbaren Systems zur computergestützten Lernfortschrittskontrolle“. In: *E-learning-management* (2008), S. 213–232.

[19] R. Fauzan, D. Siahaan, S. Rochimah und E. Triandini. „A different approach on automated use case diagram semantic assessment“. In: *International Journal of Intelligent Engineering and Systems* (2021).

[20] R. Fauzan, D. Siahaan, S. Rochimah und E. Triandini.
„Class diagram similarity measurement: a different approach“.
In: *2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE)*.
IEEE. 2018, S. 215–219.

[21] M. Fellmann et al. „Evaluation automatisierter Ansätze für die Bewertung von Modellierungsaufgaben“.
In: *DeLFI 2016–Die 14. E-Learning Fachtagung Informatik* (2016).

[22] O. Foundation. *Node.js*. 2023. URL: https://nodejs.org/en.

[23] J. G. Geer. „What do open-ended questions measure?“
In: *Public Opinion Quarterly* 52.3 (1988), S. 365–367.

[24] B. GmbH. *OPAL LERNPLATTFORM*.
Accessed on September 6, 2023. 2023.
URL: https://www.bps-system.de/opal-lernplattform/.

[25] Graphviz. *GraphViz*. 2023. URL: https://graphviz.org/.

[26] H. Gregersen und C. S. Jensen.
„Temporal entity-relationship models-a survey“. In: *IEEE Transactions on knowledge and data engineering* 11.3 (1999), S. 464–497.

[27] S. Gross, B. Mokbel, B. Hammer und N. Pinkwart.
„Feedback provision strategies in intelligent tutoring systems based on clustered solution spaces“. In: (2012).

[28] O. M. Group.
Unified Modeling Language (UML). Infrastructure, Version 2.3. 2010.
URL: https://www.omg.org/spec/UML/2.3/Infrastructure/PDF.

[29] S. Gruttmann. „Formatives E-Assessment in der Hochschullehre: computerunterstützte Lernfortschrittskontrollen im Informatikstudium“.
Diss. Zugl.: Münster (Westfalen), Univ., Diss., 2010, 2009.

[30] R. C. C. Guntupalli. *User interface design: methods and qualities of a good user interface design*. 2008.

[31] C. L. Hancock.
„Implementing the assessment standards for school mathematics: Enhancing mathematics learning with open-ended questions“.
In: *The Mathematics Teacher* 88.6 (1995), S. 496–499.

[32] J. Holland, N. Baghaei, M. Mathews und A. Mitrovic.
„The effects of domain and collaboration feedback on learning in a collaborative intelligent tutoring system“.
In: *International Conference on Artificial Intelligence in Education*. Springer. 2011, S. 469–471.

[33] J. Hollingsworth. „Automatic Graders for Programming Classes“.
In: *Commun. ACM* 3.10 (Okt. 1960), S. 528–529. ISSN: 0001-0782.
DOI: 10.1145/367415.367422.
URL: https://doi.org/10.1145/367415.367422.

[34] C. R. Huyck und S. L. Lytinen.
„Efficient heuristic natural language parsing“. In: *Proceedings of the eleventh national conference on Artificial intelligence*. 1993, S. 386–391.

[35] R. P. Jayawardena, G. D. Thiwanthi, P. S. Suriyaarachchi, K. I. Withana und C. Jayawardena. „Automated Exam Paper Marking System for Structured Questions and Block Diagrams“.
In: *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. 2018, S. 1–5.
DOI: 10.1109/ICIAFS.2018.8913351.

[36] R. P. Jayawardena, G. D. Thiwanthi, P. S. Suriyaarachchi, K. I. Withana und C. Jayawardena. „Automated Exam Paper Marking System for Structured Questions and Block Diagrams“.
In: *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. 2018, S. 1–5.
DOI: 10.1109/ICIAFS.2018.8913351.

[37] T. Khdour.
„A semantic assessment framework for e-learning systems“.
In: *International Journal of Knowledge and Learning* 13.2 (2020), S. 110–122.

[38] M. A.-R. Al-Khiaty und M. Ahmed. „Matching UML class diagrams using a Hybridized Greedy-Genetic algorithm“.
In: *2017 12th International scientific and technical conference on computer sciences and information technologies (CSIT)*. Bd. 1. IEEE. 2017, S. 161–166.

[39] S. Kocdar, A. Karadeniz, R. Peytcheva-Forsyth und V. Stoeva.
„Cheating and plagiarism in e-assessment: Students' perspectives“.
In: *Open Praxis* 10.3 (2018), S. 221–235.

[40] S. Laumer, A. von Stetten und A. Eckhardt. „E-assessment“. In: *Business & Information Systems Engineering* 1 (2009), S. 263–265.

[41] M. Levlin. „DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte“. In: (2020).

[42] Q. Li und Y.-L. Chen. „Data flow diagram“. In: *Modeling and Analysis of Enterprise and Information Systems*. Springer, 2009, S. 85–97.

[43] J. Ltd. *Draw.io*. 2023. URL: https://www.drawio.com/doc/.

[44] R. Madanayake, G. Dias und N. D. Kodikara. „Transforming Simplified Requirement in to a UML Use Case“. In: *International Journal of Computer Science and Software Engineering* 6.3 (2017), S. 61.

[45] M. Mazanec und O. Macek. „On General-purpose Textual Modeling Languages.“ In: *Dateso*. Bd. 12. Citeseer. 2012, S. 1–12.

[46] A. L. McCann. „Factors affecting the adoption of an e-assessment system“. In: *Assessment & Evaluation in Higher Education* 35.7 (2010), S. 799–818.

[47] S. McConnell. *Software estimation: demystifying the black art*. Microsoft press, 2006.

[48] Moodle. *Moodle LMS - Moodle-Lernplattform*. Accessed on September 6, 2023. 2023. URL: https://moodle.de/.

[49] D. Moody. „The “physics” of notations: toward a scientific basis for constructing visual notations in software engineering“. In: *IEEE Transactions on software engineering* 35.6 (2009), S. 756–779.

[50] B. Mschypula M. Striewe. *GReQL Regeln - WIKI*. Accessed on October 19, 2023. 2023. URL: https://wiki.uni-due.de/jack/index.php?title=GReQL_Regeln#Beispiele_f.C3.BCr_statische_Pr.C3.BCfregeln_f.C3.BCr_UML-Modelle_im_XMI2-Format.

[51] B. Mschypula M. Striewe. *JACK - WIKI*. Accessed on September 6, 2023. 2023. URL: https://wiki.uni-due.de/jack/index.php?title=Hauptseite.

[52] A. Outair, M. Tanana und A. Lyhyaoui.
„TOWARDS AN AUTOMATIC EVALUATION OF UML CLASS DIAGRAMS BY MEASURING GRAPH SIMILARITY.“ In: *Journal of Theoretical & Applied Information Technology* 95.4 (2017).

[53] B. Pagès. *BOUML*. 2021. URL: https://www.bouml.fr/index.html.

[54] V. Paradigm. *Visual Paradigm*. 2023.
URL: https://www.visual-paradigm.com/.

[55] C. A. Petri und W. Reisig. „Petri net“.
In: *Scholarpedia* 3.4 (2008), S. 6477.

[56] *PlantUML Parser*. 2023.
URL: https://www.npmjs.com/package/plantuml-parser.

[57] G. Reggio, M. Leotta, F. Ricca und D. Clerissi.
„What are the used UML diagrams? A Preliminary Survey.“
In: *EESSMod@ MoDELS*. 2013, S. 3–12.

[58] I. D. Rodina. *Software Ideas Modeller*. 2023.
URL: https://www.softwareideas.net/.

[59] A. Roques. *PlantText*. 2023.
URL: https://blog.planttext.com/about/.

[60] A. Roques. *PlantUML*. 2023. URL: https://plantuml.com/.

[61] D. Skogan.
„UML as a schema language for XML based data interchange“.
In: *Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99)*. Citeseer. 1999.

[62] R. Sousa und J. P. Leal.
„A structural approach to assess graph-based exercises“.
In: *Languages, Applications and Technologies: 4th International Symposium, SLATE 2015, Madrid, Spain, June 18-19, 2015, Revised Selected Papers 4*. Springer. 2015, S. 182–193.

[63] Sparxsystems. *Enterprise Architect*. 2023.
URL: https://www.sparxsystems.de/.

[64] U. Stödberg. „A research review of e-assessment“. In: *Assessment & Evaluation in Higher Education* 37.5 (2012), S. 591–604.
DOI: 10.1080/02602938.2011.557496.
eprint: https://doi.org/10.1080/02602938.2011.557496.
URL: https://doi.org/10.1080/02602938.2011.557496.

[65] M. Striewe. „Automated Assessment of Software Artefacts-A Use Case in E-Assessment“. In: *University of Duisburg-Essen* (2014).

[66] M. Striewe und M. Goedicke. „Automated checks on UML diagrams“. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. 2011, S. 38–42.

[67] T. Sturm, J. von Voss und M. Boger. „Generating code from UML with velocity templates“. In: *International Conference on the Unified Modeling Language*. Springer. 2002, S. 150–161.

[68] J. Suzuki und Y. Yamamoto. „Making UML models interoperable with UXF“. In: *International Conference on the Unified Modeling Language*. Springer. 1998, S. 78–91.

[69] D. Systemes. *Magicdraw*. 2023. URL: https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/.

[70] M. Ullrich et al. „Automated assessment of conceptual models in education“. In: *Manuskript unter Begutachtung* (2021).

[71] S. Valenti, F. Neri und A. Cucchiarelli. „An overview of current research on automated essay grading“. In: *Journal of Information Technology Education: Research* 2.1 (2003), S. 319–330.

[72] H. Washizaki, M. Akimoto, A. Hasebe, A. Kubo und Y. Fukazawa. „TCD: A text-based UML class diagram notation and its model converters“. In: *Advances in Software Engineering: International Conference, ASEA 2010, Held as Part of the Future Generation Information Technology Conference, FGIT 2010, Jeju Island, Korea, December 13-15, 2010. Proceedings*. Springer. 2010, S. 296–302.

[73] S. A. White. „Introduction to BPMN“. In: *Ibm Cooperation* (2004).

[74] E. Wohlgethan. „Supportingweb development decisions by comparing three major javascript frameworks: Angular, react and vue. js“. Diss. Hochschule für Angewandte Wissenschaften Hamburg, 2018.

[75] E. You. *Vue.js*. 2023. URL: https://vuejs.org/.

A. Appendix

A.1 Einige Teile des Quellcode

A.1.1 Backend Quellcode

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const { parse } = require('plantuml-parser');
4  const cors = require('cors');
5  const app = express();
6  const port = 3000;
7  app.use(cors());
8  app.use(bodyParser.json());
9  app.post('/convert', (req, res) => {
10   const code = req.body.code;
11   try {
12     const parsedCode = parse(code);
13     if(parsedCode.length === 0)
14        throw new Error('Failed to parse PlantUML code')
15
16     res.json(parsedCode);
17   } catch (error) {
18     res.status(500)
19 .json({ error: 'Failed to parse PlantUML code' });
20   }
21 });
22 app.listen(port, () => {
23   console.log(`Server is running on port: ${port}`);
24 });
```

Quelltext A.1: Node/Express Backend Quelltext

A.1.2 Frontend Quellcode

Rules Definition JSON

```
export default {
    RULE_TYPE: {
        // CLASS & INTERFACE
        'defined_class': 'defined_class_rule',
        'defined_enum': 'defined_enum_rule',

        // GENERALIZATION & SPECIALIZATION
        'generalization': 'generalization_rule',

        // RELATIONSHIPS
        'simple_association': 'simple_association_rule',
        'composition': 'composition_rule',
        'aggregation': 'aggregation_rule',

        // ASSOCIATION CLASS
        'association_class': 'association_class_rule',

        // OPTIONAL
        'nomination_consistency':
'nomination_consistency_rule',
        'test_association': 'test_association_rule',
        'count_methods': 'count_methods_rule',
        'count_attributes': 'count_attributes_rule'

    },
    RULE_TYPE_JSON: {
        // CLASS & INTERFACE
        'defined_class_rule': {
            rule_type: 'defined_class_rule',
            rule_name: 'Class definition',
            feedback: '... no feedback yet',
            points: 0,
            existence: 'presence',
            rule_specific: {
                class_name: "Car",
                exact_match: false,
                abstract: false,
                interface: false,
                methods: [],
```

```
            attributes: [],
        }
    },
    // ENUM
    'defined_enum_rule': {
        rule_type: 'defined_enum_rule',
        rule_name: 'Enum definition',
        feedback: '... no feedback yet',
        points: 0,
        existence: 'presence',
        rule_specific: {
            enum_class_name: "Car",
            exact_match: false,
            attributes: [],
        }
    },
    // GENERALIZATION & SPECIALIZATION
    'generalization_rule': {
        rule_type: "generalization_rule",
        rule_name: "Generalization",
        existence: "presence",
        points: 0,
        feedback: '... no feedback',
        rule_specific: {
            class_child: "Child",
            class_parent: "Parent",
            exact_match: false,
            type: "inheritance" // implementation
        }
    },
    // RELATIONSHIPS
    'simple_association_rule': {
        rule_type: "simple_association_rule",
        rule_name: "Simple Association",
        existence: "presence",
        points: 0,
        feedback: "... no feedback",
        rule_specific: {
            class_A: "Class A",
            class_B: "Class B",
            exact_match: false,
            A_multiplicity: "1",
            B_multiplicity: "1"
```

```
83              }
84          },
85          'composition_rule': {
86              rule_type: "composition_rule",
87              rule_name: "Composition",
88              existence: "presence",
89              points: 0,
90              feedback: "... no feedback",
91              rule_specific: {
92                  class_composite: "Composite",
93                  class_element: "Element",
94                  exact_match: false,
95                  element_multiplicity: "*",
96              }
97          },
98          'aggregation_rule': {
99              rule_type: "aggregation_rule",
100             rule_name: "Aggregation",
101             existence: "presence",
102             points: 0,
103             feedback: "... no feedback",
104             rule_specific: {
105                 class_aggregate: "Aggregate",
106                 class_element: "Element",
107                 exact_match: false,
108                 element_multiplicity: "*",
109             }
110         },
111         // ASSOCIATION CLASS
112         'association_class_rule': {
113             rule_type: "association_class_rule",
114             rule_name: "Association Class",
115             existence: "presence",
116             points: 0,
117             feedback: "...",
118             rule_specific: {
119                 class_A: "Class A",
120                 class_B: "Class B",
121                 class_C: "Association Class",
122                 exact_match: false,
123             }
124         },
125         // OPTIONAL
```

```
126         'nomination_consistency_rule': {
127             rule_type: "nomination_consistency_rule",
128             rule_name: "Nomination Consistency",
129         },
130         'count_methods_rule': {
131             rule_type: "count_methods_rule",
132             rule_name: "Count Methods",
133             existence: 'absence',
134             points: 0,
135             rule_specific: {
136                 methods: 0,
137             }
138         },
139         'count_attributes_rule': {
140             rule_type: "count_attributes_rule",
141             rule_name: "Count Attributes",
142             existence: 'absence',
143             points: 0,
144             rule_specific: {
145                 attributes: 0,
146             }
147         },
148         'test_association_rule': {
149             rule_type: "test_association_rule",
150             rule_name: "Test Association",
151             existence: "absence",
152             points: 0,
153             rule_specific: {
154                 class_A: "Class A",
155                 class_B: "Class B",
156                 exact_match: false,
157             }
158         },
159     },
160     METHODS_TYPE: {
161         name: "public_method_name",
162         exact_match: false,
163         return_type: "void",
164         visibility: "public",
165         arguments: "",
166         points: 0,
167         feedback: '... no feedback',
168         is_static: false
```

```
169     },
170     ATTRIBUTE_TYPE: {
171         name: "attribute_name",
172         exact_match: false,
173         type: "string",
174         visibility: "public",
175         points: 0,
176         feedback: '... no feedback',
177         is_static: false
178     },
179     ENUM_ATTRIBUTE_TYPE: {
180         name: "ENUM_ATTR",
181         exact_match: false,
182         points: 0,
183         feedback: '... no feedback',
184     },
185     EXISTENCE_TYPE: {
186         'presence': 'presence',
187         'absence': 'absence'
188     },
189     GENERALIZATION_TYPE: {
190         'inheritance': 'inheritance',
191         'implementation': 'implementation'
192     }
193 }
194 }
```

Quelltext A.2: Rules Definition JSON

Rule Object Generierung

```
generateClassRule(elem) {
    const rule = JSON.parse(JSON.stringify(rulesDefinitions
        .RULE_TYPE_JSON.defined_class_rule));

    rule.rule_specific.class_name = elem.name;

    const annotation_rules =
        this.annotationConverter(elem.generics[0])

    rule.points = annotation_rules.p

    rule.rule_specific.exact_match =
        annotation_rules.classMatch

    let abstractText = ""
    if (elem.isAbstract) {
        rule.rule_specific.abstract = elem.isAbstract;
        abstractText = "abstracte "
    }

    if (elem.stereotypes.includes("interface"))
        rule.rule_specific.interface = true;


    let attr_index = 0
    let method_index = 0
    elem.members.forEach((member) => {
        // METHODS
        if (member.returnType) {
            const method = JSON.parse(
                JSON.stringify
                    (rulesDefinitions.METHODS_TYPE));
            method.name = member.name;
            method.return_type = member.returnType;
            method.arguments = member._arguments;

            method.visibility =
                this.getVisibility(member.accessor);
            method.feedback =
                `Die Klasse ${rule.rule_specific.class_name}
                 soll eine Methode namens ${method.name}
```

```
42                  bereitstellen.`;
43
44              rule.rule_specific.methods.push(method);
45              if(annotation_rules.method === '*')
46                  method.exact_match = true
47              else if (annotation_rules.method
48                      .includes(method_index))
49                  method.exact_match = true
50              method.points = annotation_rules.mdp
51              method_index++
52          }
53          // ATTRIBUTES
54          else if (member.type) {
55              const attribute = JSON.parse(
56                  JSON.stringify(rulesDefinitions
57                      .ATTRIBUTE_TYPE));
58              attribute.name = member.name;
59              attribute.type = member.type;
60
61              attribute.visibility =
62                  this.getVisibility(member.accessor);
63              attribute.feedback = `Die Klasse
64                  ${rule.rule_specific.class_name} soll ein
65                  ${attribute.visibility} Attribut für die
66                  Eigenschaft ${attribute.name} und type
67                  ${attribute.type} bereitstellen.`;
68
69              rule.rule_specific.attributes.push(attribute);
70              if(annotation_rules.attr === '*')
71                  attribute.exact_match = true
72              else if (annotation_rules.attr
73                  .includes(attr_index))
74                  attribute.exact_match = true
75              attribute.points = annotation_rules.adp
76              attr_index++
77          }
78      });
79
80      const entityName = rule.rule_specific.interface
81          === true ? "Interface" : "Klasse"
82      rule.feedback = `Es soll eine ${abstractText}
83          ${entityName} mit der Name
84          ${rule.rule_specific.class_name}
```

```
85          bereitgestellt werden.`;
86      return rule;
87  },
88
89  annotationConverter(input){
90      const result = {};
91
92      result.classMatch = /!class/.test(input);
93
94      const attrMatch = /!attr\((.*?)\)/.exec(input);
95      if (attrMatch) {
96          const attrValue = attrMatch[1].trim();
97          result.attr = attrValue === '*' ? '*' :
98              attrValue.split(',').map(Number);
99      } else
100         result.attr = [];
101
102     const methodMatch = /!method\((.*?)\)/.exec(input);
103     if (methodMatch) {
104         const methodValue = methodMatch[1].trim();
105         result.method = methodValue === '*' ? '*' :
106             methodValue.split(',').map(Number);
107     } else
108         result.method = [];
109
110     const pMatch = /p=(\d+)/.exec(input);
111     if (pMatch)
112         result.p = parseInt(pMatch[1]);
113     else
114         result.p = 0
115
116     const adpMatch = /ad-p=(\d+)/.exec(input);
117     if (adpMatch)
118         result.adp = parseInt(adpMatch[1]);
119     else
120         result.adp = 0
121
122     const mdpMatch = /md-p=(\d+)/.exec(input);
123     if (mdpMatch)
124         result.mdp = parseInt(mdpMatch[1]);
125     else
126         result.mdp = 0
127
```

```
128     return result;
129 },
```

Quelltext A.3: Class Definition Rule

GReQL Code Generierung

```
generateDefineClassRule: function (rule) {
    const isInterface = rule.rule_specific.interface
    let code = ""
    if (isInterface) {
        code += "<!-- Interface Definition -->"
        code += `<rule type="${rule.existence}"
        points="${rule.points}">
        <query>from x : V{Interface}
               with
                  isDefined(x.name) and
                  stringLevenshteinDistance(x.name,
                  "${rule.rule_specific.class_name}")&lt;3
              report 1 end
        </query>
        <feedback>${rule.feedback}</feedback>
     </rule>`
    } else {
        const isAbstract = rule.rule_specific.abstract
        let abstractCode
        if (isAbstract)
            abstractCode = `and x.isAbstract`
        else
            abstractCode = `and (not x.isAbstract)`
        code += "<!-- Class Definition -->"
        code += `<rule type="${rule.existence}"
        points="${rule.points}">
        <query>from x : V{Class}
               with
                  isDefined(x.name) and
                  stringLevenshteinDistance(x.name,
                  "${rule.rule_specific.class_name}")&lt;3
                  ${abstractCode}
               report 1 end
        </query>
        <feedback>${rule.feedback}</feedback>
    </rule>`
    }

    if(rule.rule_specific.attributes.length !== 0){
    rule.rule_specific.attributes.forEach(attribute => {
    code += this.generateAttributeRule(rule, attribute)
```

```
42      })}
43
44      if(rule.rule_specific.methods.length !== 0){
45      rule.rule_specific.methods.forEach(method => {
46      code += this.generateMethodRule(rule, method)
47      })}
48
49      return code
50  },
51
52  generateAttributeRule: function (rule, attribute) {
53      /***
54       1- Only 3 primitive type are working
55       Integer - Boolean - String
56       2- GReQL Engine cannot handle this case from BOUML XMI
57       */
58      let code = ""
59      code += "<!-- Attribute Rule -->"
60
61      const visibility = this.getVisibility(attribute)
62      const isStatic = this.isStatic(attribute)
63      const primitiveType = this.getType(attribute.type)
64
65      let vType = "from x: V{Class}, y : V{Property}"
66      let vTypeText = ""
67      if (primitiveType !== '!prim') {
68          vType = "from x : V{Class},
69          y : V{Property},
70          z : V{PrimitiveType}"
71          vTypeText = `and y --> z and
72          isDefined(z.name) and
73          z.name="${primitiveType}"`
74      }
75
76      code += `<rule type="presence"
77      points="${attribute.points}">
78      <query>${vType}
79              with
80                 isDefined(x.name) and
81                 stringLevenshteinDistance(x.name,
82                 "${rule.rule_specific.class_name}")&lt;3 and
83                 x --> y and isDefined(y.name) and
84                 stringLevenshteinDistance(y.name,
```

```
85              "${attribute.name}")&lt;3 and
86              ${visibility}
87              ${isStatic}
88              ${vTypeText}
89          report 1 end
90      </query>
91      <feedback>${attribute.feedback}</feedback>
92  </rule>`
93      return code
94  },
95  generateMethodRule: function (rule, method) {
96      let code = ""
97      const visibility = this.getVisibility(method)
98      const isStatic = this.isStatic(method)
99      const retType = this.getType(method.return_type)
100     /***
101      1- Only 3 primitive type are working
102      Integer - Boolean - String
103      */
104     let vType = "from x: V{Class}, y : V{Operation}"
105     let vTypeText = ""
106     if (retType !== '!prim') {
107         vType = "from x : V{Class},
108         y : V{Operation},
109         ret: V{Parameter},
110         retType: V{PrimitiveType}"
111         vTypeText = ` and y --> ret and
112         isDefined(ret.name) and
113         ret.name="return"  and
114         ret --> retType and
115         isDefined(retType.name) and
116         retType.name="${retType}"`
117     }
118
119     code += "<!-- Method Rule -->"
120     code += `<rule type="presence"
121     points="${method.points}">
122     <query>${vType}
123             with
124                 isDefined(x.name) and
125                 x.name="${rule.rule_specific.class_name}" and
126                 isDefined(y.name) and
127                 stringLevenshteinDistance(y.name,
```

```
128                 "${method.name}")&lt;3 and
129                 ${visibility}
130                 ${isStatic} and
131                 x --> y
132                 ${vTypeText}
133               report 1 end
134     </query>
135     <feedback>${method.feedback}</feedback>
136   </rule>`
137
138     this.extractVariableNames(method.arguments)
139         .forEach( arg => {
140         code += "<!-- Method Param -->"
141         code += `<rule type="presence" points="0">
142         <query>from x: V{Class}, y : V{Operation},
143         param: V{Parameter}
144         with
145         isDefined(x.name) and
146         stringLevenshteinDistance(x.name,
147         "${rule.rule_specific.class_name}")&lt;3 and
148         isDefined(y.name) and
149         stringLevenshteinDistance(y.name,
150         "${method.name}")&lt;3 and
151         x --> y and
152         y --> param and isDefined(param.name) and
153         param.name="${arg}"
154        report 1 end
155         </query>
156         <feedback>Die Methode ${method.name} muss
157         ein Attribut ${arg} haben.</feedback>
158       </rule>`
159     })
160     return code
161 },
162
163 extractVariableNames: function (inputString) {
164     inputString += ';'
165     const variablePattern = /\b(\w+)\s*,?\s*(?=[,;])/g;
166     const matches = inputString.match(variablePattern);
167     if (matches)
168         return  matches.map(match => match.trim());
169     return [];
170 }
```

```
171 getVisibility: function (accessor) {
172     switch (accessor.visibility) {
173         case 'public':
174             return 'y.visibility="public" and '
175         case 'private':
176             return 'y.visibility="private" and'
177         case 'protected':
178             return 'y.visibility="protected" and'
179         default:
180             return ""
181     }
182 },
183 isStatic: function (accessor) {
184     if (accessor.is_static)
185         return 'y.isStatic=true '
186     else
187         return 'y.isStatic=false '
188 },
189 getType: function (type) {
190     switch (type.toLowerCase()) {
191         case 'int':
192             return "Integer"
193         case 'string':
194             return "String"
195         case 'bool':
196         case 'boolean':
197             return "Boolean"
198         default:
199             return "!prim"
200     }
201 },
```

Quelltext A.4: Fall der Class Definition Rule

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Michael Vivian Mboni Saha, Campus Essen, den 01.02.2024