UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

Generierung von Feedback-Regeln für Modelle aus annotierten Musterlösungen

Vorgelegt der Fakultät für Informatik 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. 12, 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.. 17

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. 19

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, 53, 55–57, 59, 61–63, 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. [33]. 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. 20

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.. 17

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 Abschluss der Prüfungsphasen in Sekundar- und Hochschuleinrichtungen stellt sich regelmäßig die Herausforderung einer umfangreichen Anzahl an Korrekturaufgaben, die von Lehrkräften und Dozenten bewältigt werden müssen. Diese Aufgabe kann mitunter mühsam und zeitaufwendig sein, insbesondere im Falle einer hohen Anzahl an Prüfungen [22]. 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 [20].

Heute ist die automatisierte und computerbasierte Bewertung von akademischen Arbeiten weit verbreitet im Hochschulbereich. Diverse Plattformen wie Dynexite [9], LPLUS [30], Questionmark [36], Turnitin [43], Leapsome [28] 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. In der Modellierung existiert keine eindeutige Lösungsstrategie, was die Aufgabe komplexer gestaltet. Die Resultate jedes Studierenden müssen wiederholt mit der Musterlösung verglichen werden, um Vollständigkeit sicherzustellen.
2. Richtige Modelle können von der erwarteten Lösung abweichen und Studierende benachteiligen, was Originalität hemmen könnte.
3. Die Beurteilung kann subjektiv sein. Unterschiedliche Auslegungen könnten zu ungleichen Bewertungen führen, besonders wenn mehrere Personen bewerten.
4. Das Evaluationsmodell ist zeitaufwendig, besonders bei vielen Studierenden. Der Abgleich der Studierendenarbeiten mit der Musterlösung erfordert Zeit und kann zu Verzögerungen 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” [33] 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 Feedbacks 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 ein Feedback geben. 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 sieben 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 Kon-

zept 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 acht 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 die verschiedene Ansätze für die automatisierte Bewertung von UML-Modellen 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 [10]. 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 [39]. 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 [39]. 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 [39].

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 [17]. 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 [17].

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 [2].
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 [17] [2].
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 [17] [2].
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 [2].
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 [2].
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 [2].
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 [17] [2].

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

2.1.2 Mögliche Aufgabentypen

In der Erstellung von Übungen 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** [39]. Diese Klassifikationen sind jeweils durch spezifische Charakteristika gekennzeichnet und bieten Vorzüge, die auf präzise Beurteilungsziele ausgerichtet sind [26].

Geschlossene Fragen (Closed questions)

Die Kategorie der Geschlossenen Fragen 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[17]. 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. 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 [17]. 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 [5] [6].
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 [24].
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 [17].

4. Lückentexte (Fill-in-the-Blanks): Bei Lückentexten müssen die Lernenden fehlende Wörter oder Phrasen in einem Satz oder Text ergänzen. In solchen Übungen wird sehr häufig eine Wortliste bereitgestellt, die von den Teilnehmenden zur Lösung der Übungen verwendet werden soll. Diese Art von Aufgabe kann verwendet werden, um das Verständnis von Kontext und Details zu überprüfen [17].

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 [39]. 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 [39]. 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 [17]. 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 [17].
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 [17].
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 [17].
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 [17].

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 [44].

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

2.2 Konzeptionelle Modelle

Ein konzeptionelles Modell ist 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 [1].

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 [1]. 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 [1]. 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 [32]. 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 [44].

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 [15].
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 [37].
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 [29].
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 [46].
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 [35].
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 [7].

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 [7]. 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 [7]. 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 [7].

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 [44]. 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 [14].
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 [31] [18].
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 [31].
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 [44] [11].
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 [44] [41].
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 [44] [19].
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 [44] [8] [23].
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 [44] .
   (a) **Werkzeuge zur Modelltransformation und Codegenerierung:** Diese Werkzeuge ermöglichen die automatische Generierung von Code aus UML-Diagrammen. Sie helfen dabei, den manuellen 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 [42].

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 [27]. 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 [45], die Beurteilung von Programmieraufgaben [16] sowie die Modellierung und Bewertung von Modellen in diversen Fachdisziplinen [38]. 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 [13]. 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 Automatisierte Bewertung von UML-Modellen: Verschiedene Ansätze

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” [8] stammt von den Autoren Younes Boubekeur, Gunter Mussbacher und Shane McIntosh. In ihrem Artikel stellen sie einen 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 Studienarbeiten zu identifizieren, sondern auch ungefähre Noten vorherzusagen.

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 [21]. Darüber hinaus setzen die Autoren maschinelle Lernverfahren ein, um ungefähre Noten vorherzusagen, basierend auf den von dem heuristischen Algorithmus generierten Punktzahlen (siehe Abbildung 2.1).

Um die Effektivität ihres vorgeschlagenen Ansatzes zu validieren, führten die Autoren eine empirische Studie mit 50 Studierenden durch, die an einem Softwaretechnikkurs teilnahmen. Die Studierenden wurden gebeten, UML-Diagramme für ein gegebenes Problem einzureichen, und die Autoren wandten ihren 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 [8]. Diese

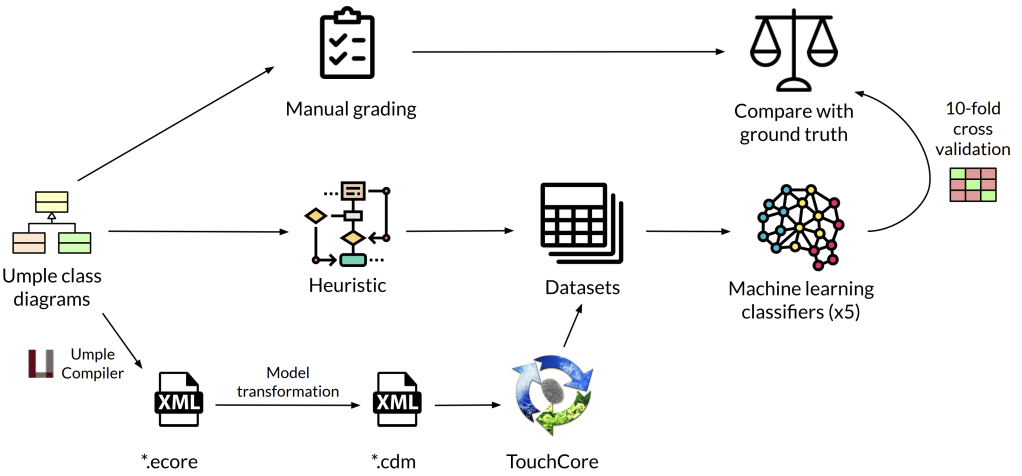


Abbildung 2.1: Ansatz der Studie [8]

Herangehensweise weist jedoch einige zu berücksichtigende Schwächen auf:

1. Es muss ein Zwischenmodell generiert werden, von dem aus die Diagramme der Studierenden bewertet werden, was zusätzliche Arbeit für die Lehrkräfte bedeutet [8].
2. Die Herangehensweise funktioniert für allgemeine Diagrammfälle, aber ihre Effektivität nimmt ab, wenn die Modellierung komplexer wird [8].
3. Die Autoren sprechen davon, dass die Herangehensweise mit einem begrenzten Datensatz getestet wurde und eine größere Menge an annotierten Daten benötigt, um sie zu verbessern, was für das Entwicklungsteam zeitintensiv sein kann [8].
4. Das Trainieren und Evaluieren einer künstlichen Intelligenz ist eine mühsame Arbeit, die Jahre dauern kann [8].

Die in dieser Masterarbeit behandelte Herangehensweise, wie sie im Konzeptkapitel 4.2 dargelegt wird, ermöglicht es hingegen, von einer bereits vorhandenen Musterlösung des Lehrers auszugehen, Regeln zu generieren und Feedback zu erhalten, um einen Student schnell und ohne Zwischenkomplexität zu bewerten. Diese Musterlösung sollte im Voraus annotiert werden, um zusätzliche Details hinzuzufügen, die zur Erstellung dieser Regeln erforderlich sind. Dieser Ansatz ist schneller und erleichtert die Arbeit der Lehrkräfte.

2.4.2 Bewertung auf Basis von Ähnlichkeitsmaßen

Die Autoren des Artikels “A Different Approach on Automated Use Case Diagram Semantic Assessment” [11] sind Daniel Siahaan, Siti Rochimah, Reza Fauzan und Evi Triandini. Sie untersuchten einen semantischen Bewertungsansatz für Anwendungsfalldiagramme. Ihr Ansatz konzentriert sich auf Eigenschaften und Beziehungen. Durch die Verwendung von Cosinus-Ähnlichkeit und WuPalmer für WordNet-Suchen bewerteten sie 39 Diagramme [11, 25], gesammelt aus drei Projekten.

Vor der Bewertung etablierten sie einen Goldstandard als Referenz basierend auf Expertenbewertungen von Studentenantworten. Die vorgeschlagene Methode wurde mit dem Goldstandard verglichen, und die Ergebnisse zeigten eine hohe Übereinstimmung. Interessanterweise war die Methode sogar präziser als die durchschnittliche Übereinstimmung zwischen Experten [11].

Die Autoren stellten fest, dass Lehrkräfte bei der Bewertung von Anwendungsfalldiagrammen tendenziell mehr Wert auf Eigenschaften als auf Beziehungen legen. Dies könnte den Bewertungsprozess verbessern und zu einer objektiveren Beurteilung führen. Die vorgestellte semantische Bewertung konzentriert sich auf die Bedeutung von Informationen in Anwendungsfalldiagrammen und nicht nur auf ihre Form oder Struktur. Die Methode wurde jedoch noch nicht auf Klassendiagramme übertragen, da diese eine andere Struktur haben. Diese Herangehensweise weist einige zu berücksichtigende Schwächen auf:

1. Diese Herangehensweise erfordert jedoch eine spezifischere Terminologie und Semantik seitens der Lehrkraft, was zusätzliche Anstrengungen von seiner Seite bedeutet.
2. Diese Herangehensweise bedeutet, dass die Lehrkraft so präzise wie möglich sein muss, wenn er die Übung erstellt. Dadurch gibt er jedoch Anweisungen, die zu klar sind, was die Modellierungsaufgabe für den Studenten sehr offensichtlich macht.
3. Diese Herangehensweise lässt der Vorstellungskraft des Studenten bezüglich der Modellierung keinen freien Raum, da die Anweisungen so präzise sind, dass es nur eine mögliche Lösung geben würde. Dies ist im Allgemeinen nicht der Fall bei Modellierungsaufgaben.

Die Verwendung dieser Methode wäre daher nicht für das Hauptziel dieser Arbeit geeignet, das darin besteht, Lehrkräften die Beurteilung von UML-Klassendiagrammen zu erleichtern.

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 [3]. In ihrem Artikel widmen sich die Autoren der Aufgabe der Bewertung von UML-Klassendiagrammen, die von Studierenden erstellt wurden. Sie schlagen ein halbautomatisches System vor, das diese Aufgabe durch den Einsatz eines Vergleichs von 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 [4].
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 [12].
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 [34] (Siehe Abbildung 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 wurden ausgewählt, um das System offline zu konfigurieren und zu bewerten, wobei iterative Verbesserungen an den Ähnlichkeitskriterien und der Systemfunktionalität auf Grundlage der Übungsergebnisse vorgenommen wurden. 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 [3]. 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

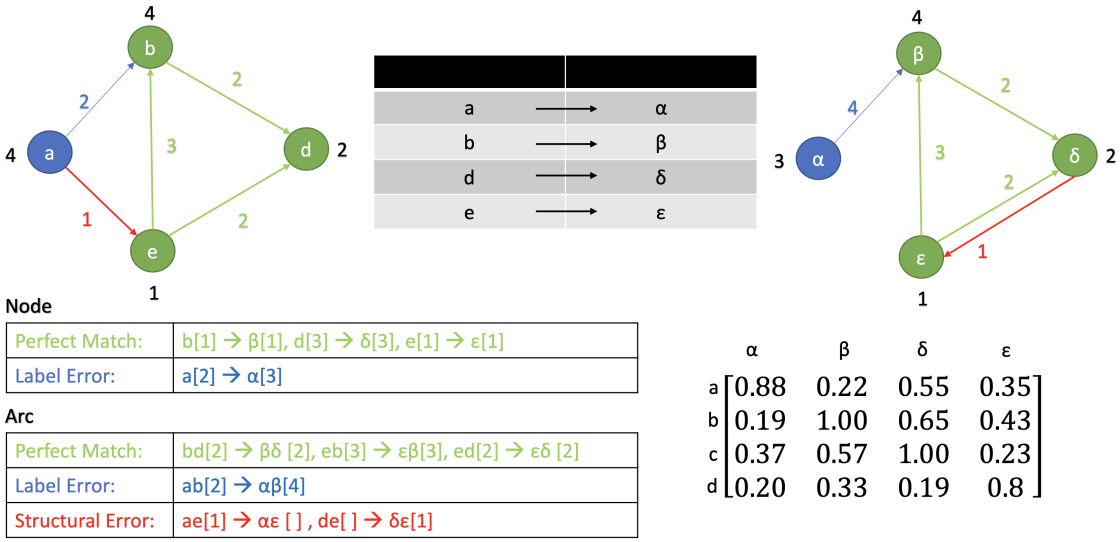


Abbildung 2.2: Beispiel für einen Graphenabgleich [3]

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. Dieser Ansatz weist einige zu berücksichtigende Schwächen auf:

1. Die Autoren erkennen an, dass weitere Forschung erforderlich ist, um das System zu verfeinern und seine Genauigkeit zu verbessern.
2. Dieser Ansatz ermöglicht keine Varianten oder Alternativen in den Lösungen, die die Studierenden präsentieren könnten. Sie konzentriert sich ausschließlich auf die Genauigkeit und lässt somit keinen Raum für mögliche Alternativen.

Die in der Sektion zum Konzept 4.2 vorgestellte Methode bietet eine direkte Lösung für dieses Problem. Durch die Annotation der Lehrerlösung ist es möglich, verschiedene Modellierungsvarianten des Studierenden zu berücksichtigen. Diese Methode ist zudem zuverlässiger, denn wenn das Modell und die Alternativen klar definiert sind, liegt die Präzision stets bei 100%.

2.4.4 Automatisierte Bewertung mit GReQL

In ihrem Artikel “Automated checks on UML diagrams” [41] stellen Michael Striewe und Michael Goedicke eine 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.

Graph Repository Query Language (GReQL), ähnlich wie SQL, eignet sich gut für die Implementierung regelbasierter Prüfungen von graphenbasierten Daten [40]. 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 [41].

```
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. Die Autoren schlagen auch eine Methode zur Berechnung von Gesamtnoten

vor, indem die Punktzahlen jeder Regel aggregiert werden [41].

Diese Herangehensweise wurde für die Entwicklung des Konzepts verwendet (siehe Abschnitt 4.2). Durch sie ist es möglich, ein Diagramm schnell zu bewerten und ein Feedback vom Lehrer zu erhalten. Im Vergleich zu anderen Methoden ist diese Herangehensweise zuverlässiger und genauer hinsichtlich der erzielten Ergebnisse. Allerdings weist sie auch mehrere Schwächen auf, die im Abschnitt zur Problemanalyse (siehe Kapitel 3) dargelegt werden. Eine Lösung zur Verbesserung des Bewertungsprozesses wird ebenfalls im Kapitel zum Konzept (siehe Abschnitt 4.2) präsentiert.

2.5 JACK

Das E-Assessment-System namens “JACK3” (dritte Version von JACK)[33], entwickelt von der Universität Duisburg-Essen, stellt eine webbasierte Plattform dar, die Lehrkräfte 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.

JACK3 bietet eine breite Palette von Übungstypen an, die den unterschiedlichen Lernzielen der Lehrkräfte 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 [33].

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 [33]. JACK3 verwendet die im Abschnitt 2.4.4 vorgestellte Methode, um diese verschiedenen Diagrammtypen zu bewerten.

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” [33], 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 [37]. 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 [37]. 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. Zusätzlich zum GReQL-Code wird vom Lehrenden auch ein Feedbacktext sowie die mit jeder Regel verbundenen Gewichtungen bereitgestellt. Alle diese verschiedenen Informationen werden in einem XML-Format verfasst, das auf JACK hochgeladen wird. 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 Bewertung für die Lösung des Studenten zu vergeben [41].

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 [**bouml**] oder Software Ideas Modeler [**sim**], 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 ein Feedback 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.

Wartung und Anpassungsfähigkeit des GReQL-Codes: Nachdem der GReQL-Code erstellt und für die Bewertung von UML-Diagrammen implementiert wurde, kann eine kontinuierliche Wartung erforderlich sein. Mit steigenden Anforderungen an die Bewertung kann es sein, dass der GReQL-Code regelmäßig aktualisiert und angepasst wird. 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 ist nicht, Lehrkräfte von der Notwendigkeit zu befreien, GReQL-Code zu schreiben, zu bearbeiten oder zu ändern. Dies ist auf die quasi unendliche Vielfalt an Möglichkeiten zurückzuführen, wenn es darum geht, GReQL-Code zu verfassen, und darauf, dass es für ein Werkzeug unmöglich ist, diese Aufgabe perfekt zu erfüllen und alle möglichen Optionen und Alternativen ohne menschliches Eingreifen abzudecken. Vielmehr geht es darum, den Prozess der Regeldefinition erheblich zu erleichtern, indem Lehrkräfte in diesem Generierungsprozess von Regeln unterstützt werden. Auf diese Weise können sich Lehrkräfte auf weniger triviale Aufgaben konzentrieren. 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. Bei der Erstellung von Übungen für UML erstellen Lehrkräfte in den meisten Fällen eine spezifische Musterlösung für diese Übungen. Ausgehend von den bereits vorhandenen Musterlösungen wäre deren Annotierung ein äußerst interessanter Ausgangspunkt, da keine zusätzliche Ressource für Lehrkräfte erforderlich ist. Basierend auf diesem spezifischen Kriterium wurde die Entscheidung zur Annahme dieses Ansatzes in Betracht gezogen, um das System zu entwickeln. 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. **Verbesserung des Zeit- und Arbeitsaufwands 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

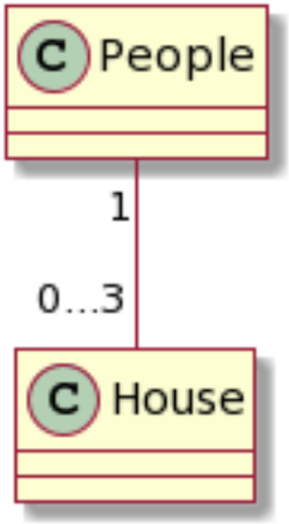
In diesem Kapitel wird der Übergang von der Problemanalyse zur kreativen Konzeptentwicklung im Forschungsprozess betont. Es wird die zentrale Phase markiert, in der theoretisches Wissen und praktische Erkenntnisse zusammengeführt werden, um innovative Lösungsansätze zu formulieren. Theoretische Grundlagen und Modelle aus vorangegangenen Kapiteln werden integriert und die angewandten Methoden zur Entwicklung tragfähiger Lösungen werden erläutert. Dieses Kapitel wird als Brücke zwischen Analyse und Umsetzung genutzt, wobei die kreative Synthese von Ideen und Lösungsansätzen betont wird. Ein detaillierter Einblick in den Prozess der Konzeptentwicklung wird geboten.

4.1 Verschiedene Herangehensweisen

Vor der Ausarbeitung des endgültigen Implementierungskonzepts wurden mehrere methodische Ansätze im Detail untersucht. Das Hauptziel dieser methodischen Ansätze bestand darin, eine Reihe von Regeln zu extrahieren, die anschließend in GReQL-Ausdrücke umgewandelt werden sollten. Diese Regelsets sollten aus den Kommentaren oder Notizen extrahiert werden, die von den Lehrkräften zu einer UML-Übung hinterlassen wurden. Die Wahl des Notizformats ist offen, was viele Möglichkeiten bietet. Zwei Ansätze, die in diesem Rahmen erkundet wurden, werden in diesem Kapitel vorgestellt.

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




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

Abbildung 4.1: Beispiel einer Annotation von UML mit YAML

Regeln, die aus dem Annotationsystem resultierten, verifizierte, konnte er GReQL-Code generieren, welcher anschließend in das JACK-System eingefügt 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 [**skogan1999uml**], UML eXchange Format (UXF) [**suzuki1998making**], Textbasierte [**washizaki2010tcd**] oder JSON-basierte Datenformate [**benson2021uml**], 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 [**chowdhary2020natural**]. 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 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 Integration dieser Funktionalität ermöglicht Lehrkräften, wertvolles Feedback und alle zugehörigen Informationen durch eine einzige Annotation hinzuzufügen, um sowohl die Punktzahlen festzulegen als auch den nächsten Schritt und die anschließende Evaluierung umfassend zu unterstützen.

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 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

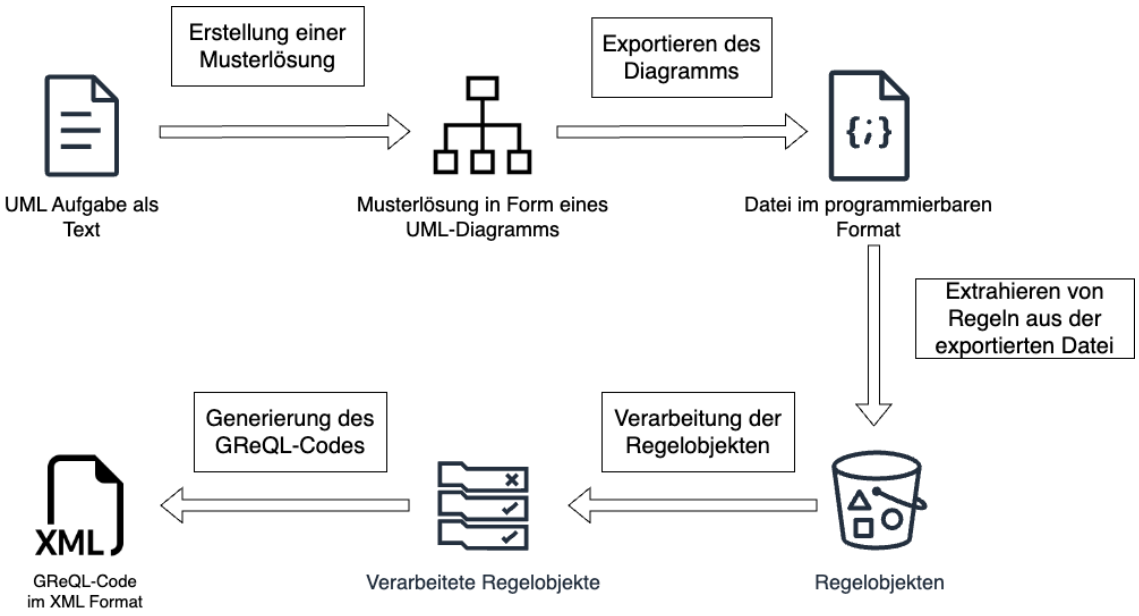


Abbildung 4.2: Repräsentatives schema des konzepts

JACK-Plattform erheblich zu vereinfachen, was das zentrale Ziel dieser Masterarbeit ist. Die tatsächliche Umsetzung dieses 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 [**planttext**] 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 [**planttext**].

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, da sie natürlicher und verständlicher sind als grafische Schnittstellen [**mazanec2012general**].

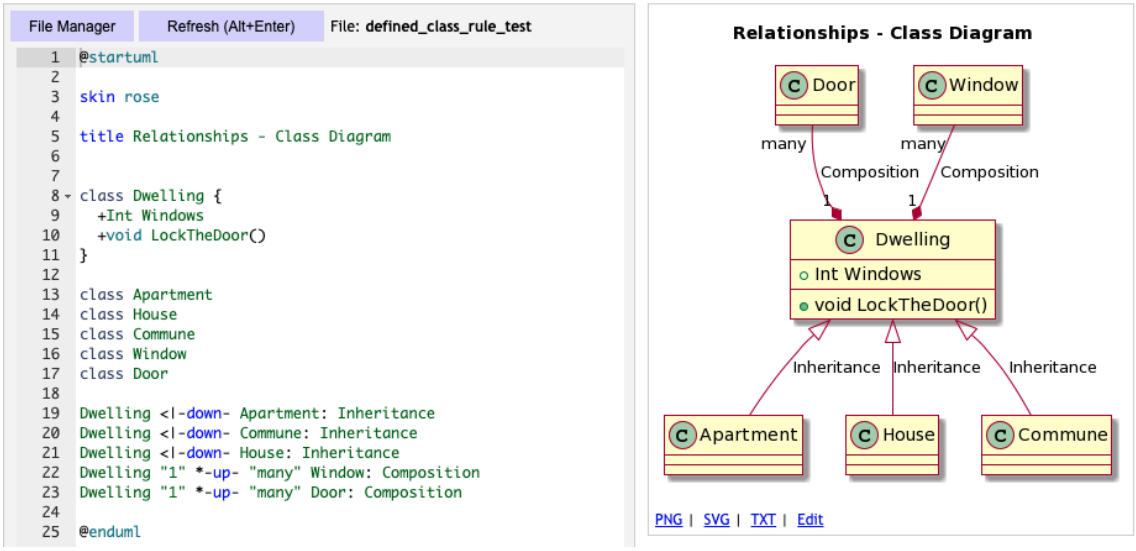


Abbildung 5.1: Grafische Benutzeroberfläche von plantText

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 [**madanayake2017transforming**].

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 [**planttext**].

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 [**planttext**].

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 [**enterarch**], Astah UML [**astah**], MagicDraw [**magic**], Visual Paradigm [**visual**], Umbrello [**umbrello**] und Draw.io [**draw**] 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 Nut-

zerfreundlichkeit zurückzuführen ist. Bedauerlicherweise wiesen sie jedoch einen bedeutenden Mangel auf, der ihre Anwendbarkeit in Bezug auf die 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.

Jedoch, wie im Abschnitt 3.1 erwähnt wurde, gibt es Werkzeuge wie BOUML [**bouml**], Enterprise Architect [**enterarch**] oder Software Ideas [**sim**], die es den Studierenden ermöglichen, den XMI-Code ihres Diagramms zu exportieren, um auf JACK bewertet zu werden. Der von diesen Werkzeugen generierte XMI bietet die Möglichkeit, die Beziehungen zwischen den verschiedenen Elementen eines UML-Diagramms zu bestimmen. JACK verfügt über einen XMI-Parser, der in der Lage ist, diese XMI zu interpretieren und sie entsprechend zu bewerten. Die Integration des XMI-Parsers in den Prozess der Regelextraktion wäre ebenfalls ein möglicher Ansatz gewesen. Allerdings erfordert die Verwendung dieses XMI-Parsers zusätzliche Integrations- und Supportarbeit.

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 diesem 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, "Plant-UML" [**plantUML**] ist. PlantUML ist ein Open-Source-Tool, das von Arnaud Roques entwickelt wurde und erstmals im Jahr 2009 veröffentlicht wurde [**plantUML**]. 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 [**plantUMLParser**]. 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 [**graphViz**] 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 [**plantUMLParser**] 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 [**Node**] 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 Open-Source-JavaScript-Framework, konzipiert und entwickelt von Evan You [**vue**]. Der Ursprung

von Vue.js entsprang der Vision, eine Lösung zur Gestaltung von Benutzeroberflächen in Webanwendungen zu schaffen. In seiner Erstveröffentlichung im Jahr 2014 initiiert, hat Vue.js eine Entfaltung erfahren, die es zu einem Akteur unter den JavaScript-Frameworks [**levlin2020dom**] in der Sphäre der Webentwicklung gemacht hat. Die Ziele und Vorteile von Vue.js bieten eine Lösung für die Schwierigkeiten, die mit der Entwicklung interaktiver Webanwendungen einhergehen, und unterstützen die Verbesserung des Entwicklungsprozesses. Wesentliche Prämissen und Gewinnpunkte von Vue.js offenbaren sich in folgender Weise:

1. **Nahtlose Integration:** Vue.js kann 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. 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 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 [**wohlgethan2018supportingweb**].

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 steigern kann [**wohlgethan2018supportingweb**].

Schließlich bietet Vue.js eine attraktive Option für jene, die ein flexibles und 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 [**wohlgethan2018supportingweb**].

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 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

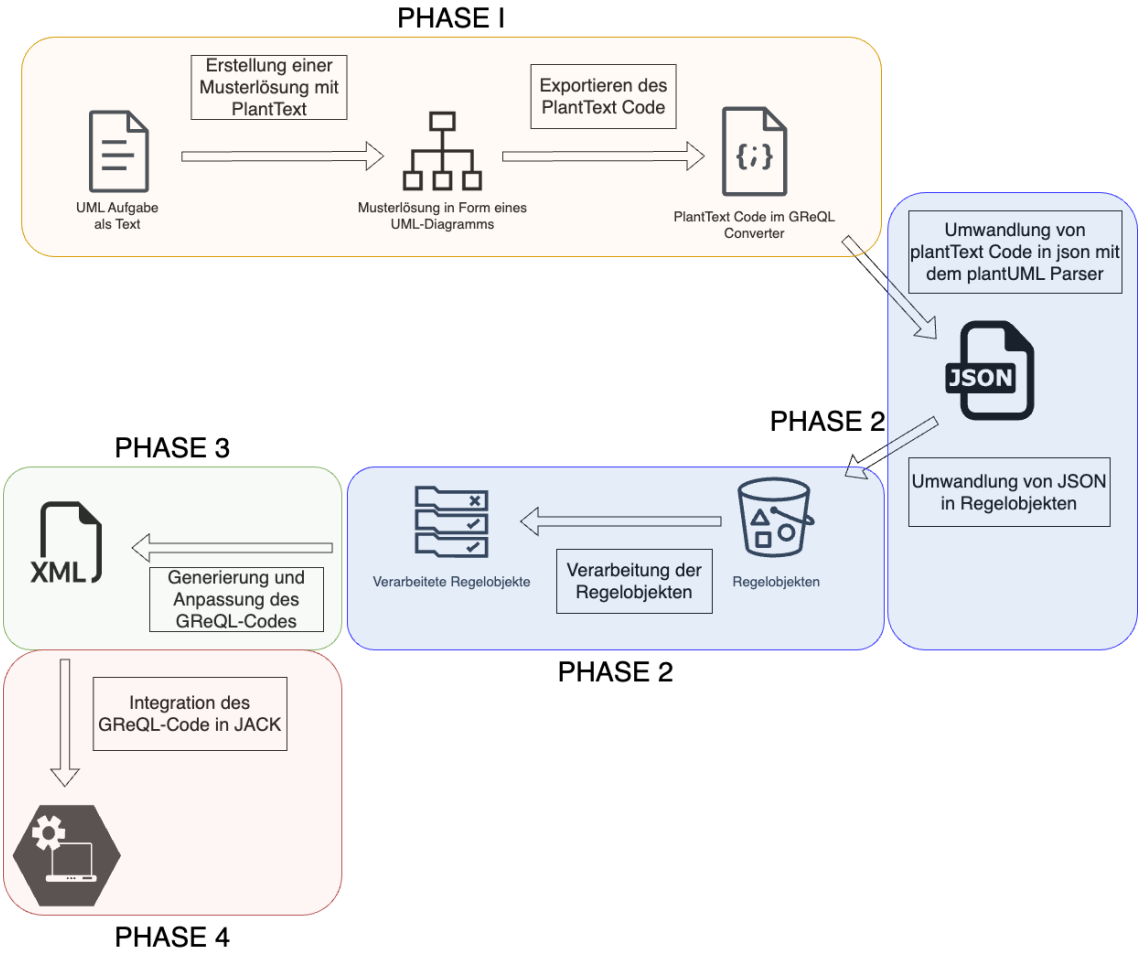


Abbildung 5.2: Repräsentatives Bild des Workflows des GReQL-Converters

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:

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

Tatsächlich gibt es mehrere gültige Formen von PlantUML-Codes, die zur Erzeugung desselben Diagrammtyps dienen können. Insbesondere bei der Darstellung einer Methode innerhalb einer Klasse in PlantUML existieren unterschiedliche Möglichkeiten. Zwei solcher Varianten sind:

- Lock(): void
- void Lock()

Beide genannten Ausdrücke sind in PlantUML gültig, jedoch berücksichtigt der Parser nur eine spezifische Syntax. Daher ist es von entscheidender Bedeutung, die präzise Syntax zu kennen, die verwendet werden sollte, damit der generierte PlantText-Code ordnungsgemäß im GReQL Converter funktioniert. Um diesen Prozess zu erleichtern, bietet die Dokumentation des GReQL Converters eine detaillierte Erläuterung und Begründung für die Verwendung spezifischer Begriffe. 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

Diese Phase beginnt, nachdem der Lehrer seinen PlantText-Code in den GReQL Converter übertragen hat. In dieser Phase entstehen eine Vielzahl von anpassbaren Regelobjekten. Die Anpassung erfolgt gemäß den spezifischen Bewertungskriterien, die die Lehrkraft verfolgt. Es ist hierbei möglich, diese Regelobjekte zu verarbeiten, je nachdem, welche Aspekte der Übung bewertet werden sollen. Weiterhin besteht die Option, Regelobjekte hinzuzufügen oder zu entfernen. In der Bearbeitung der Regelobjekten ergibt sich auch die Möglichkeit, individuelles Feedback für jede einzelnen Regelobjekte hinzuzufügen und die Punktzahl für jedes Regelobjekt erneut zu justieren.

Phase 3: Generierung und Anpassung des GReQL-Codes

Sobald die Phase der Regelbearbeitung (Phase 2) abgeschlossen ist, kann die Lehrkraft den GReQL-Code auf der Plattform generieren. Die Generierung des GReQL-Codes erfolgt aus den zuvor definierten Regelobjekten. 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.

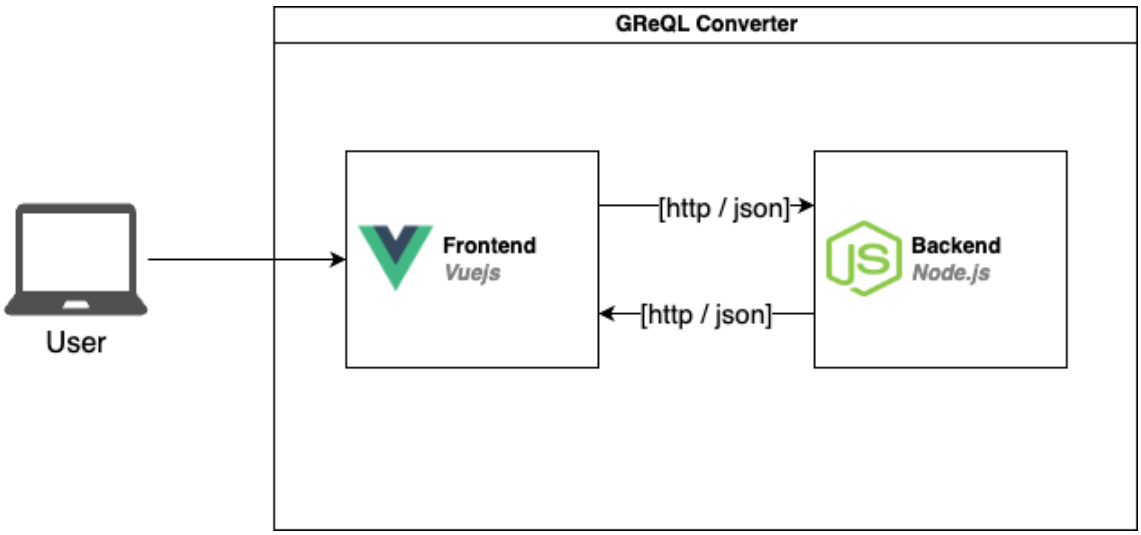


Abbildung 5.3: GReQL-Converter global infrastructure

5.3.1 Einrichtung des PlantUML-Parsers

Der erste Entwicklungsschritt ist die 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 (siehe Code 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 benutzer-

freundliche 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 [**guntupalli2008user**]. 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:

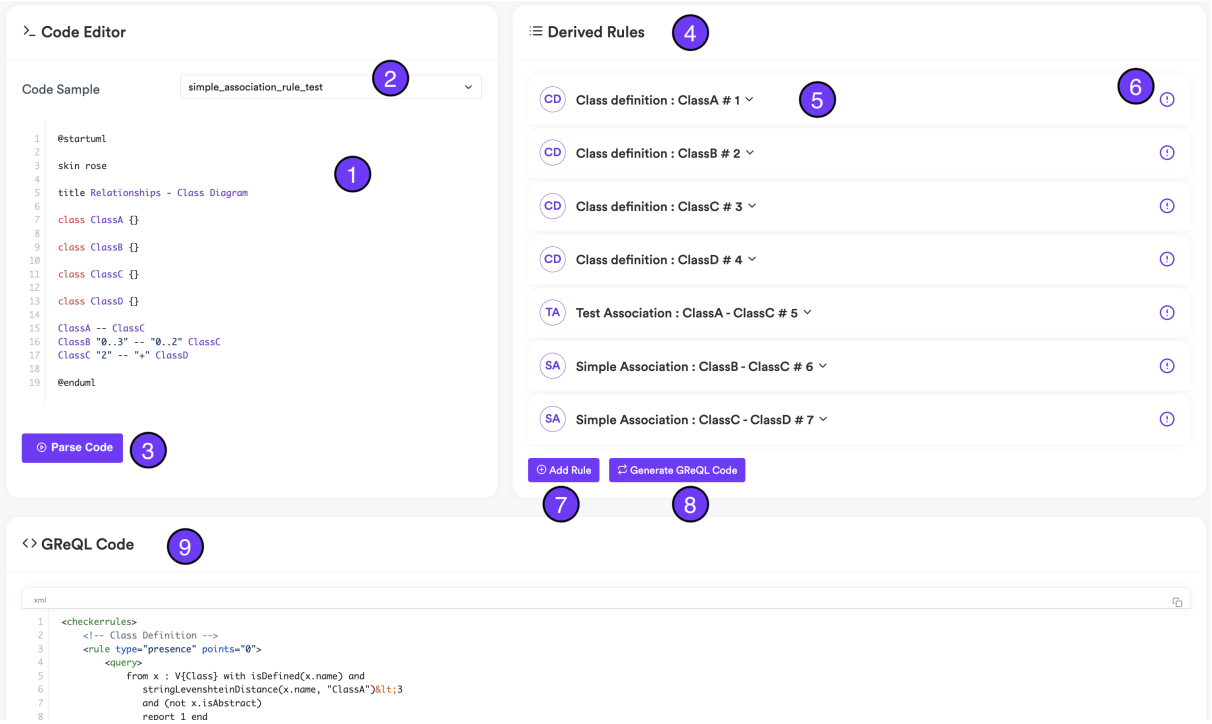


Abbildung 5.4: Dashboard im Überblick

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 Abbildung 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 Abbildung 5.4) dargestellt ist. Da der geschriebene 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 Abbildung 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 Abbildung 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 Abbildung 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 Abbildung 5.4), öffnet sich ein Menü, das es ihm ermöglicht, die in der Regel vorhandenen Informationen nach Belieben zu ändern (siehe Abbildung 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.

Dann gibt es der Button “Add Rule” (Nummer 7 - siehe Abbildung 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 Abbildung 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 Abbildung 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

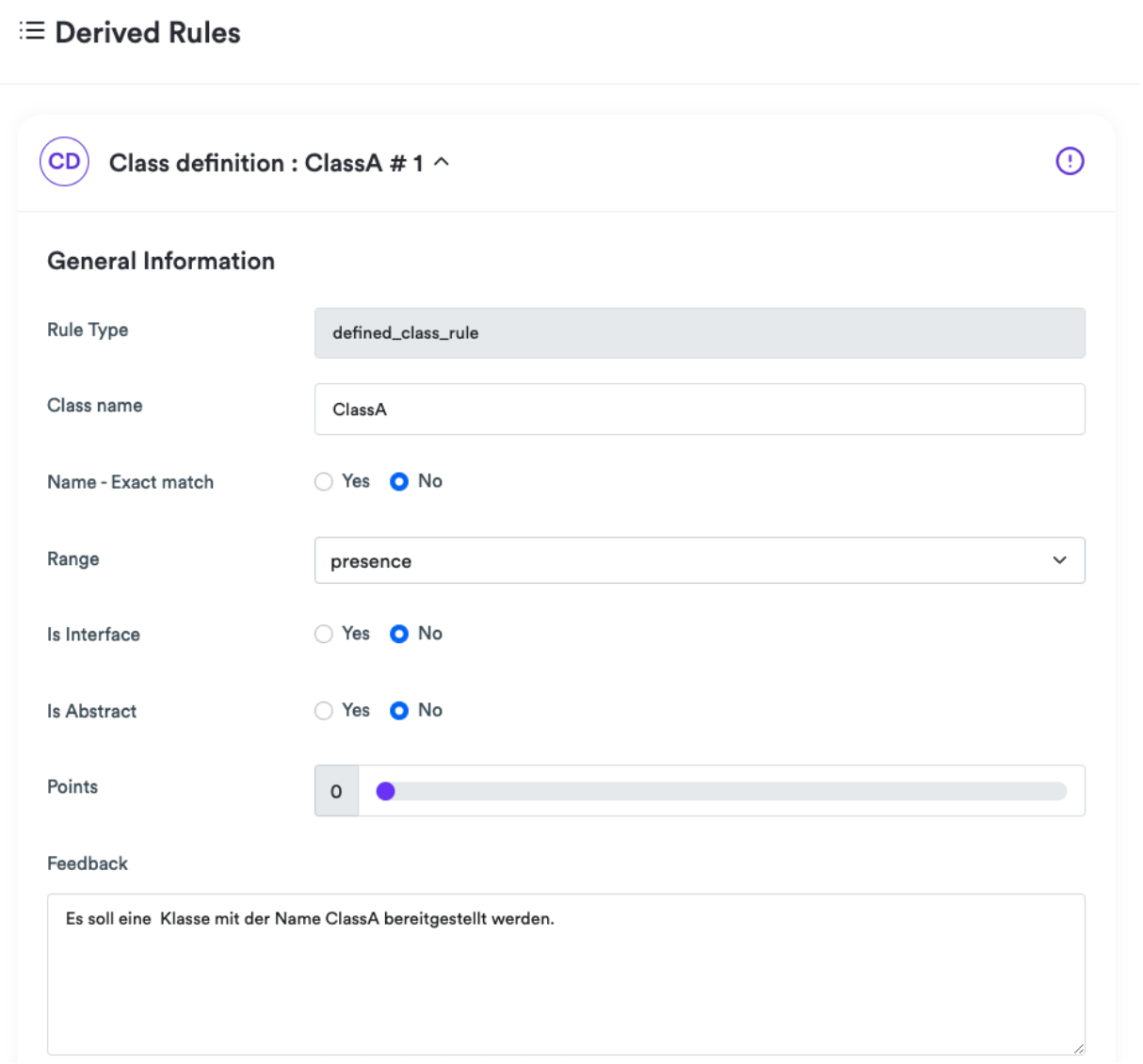


Abbildung 5.5: Beispiel eines Regelobjekts

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 Abbildung 5.6).

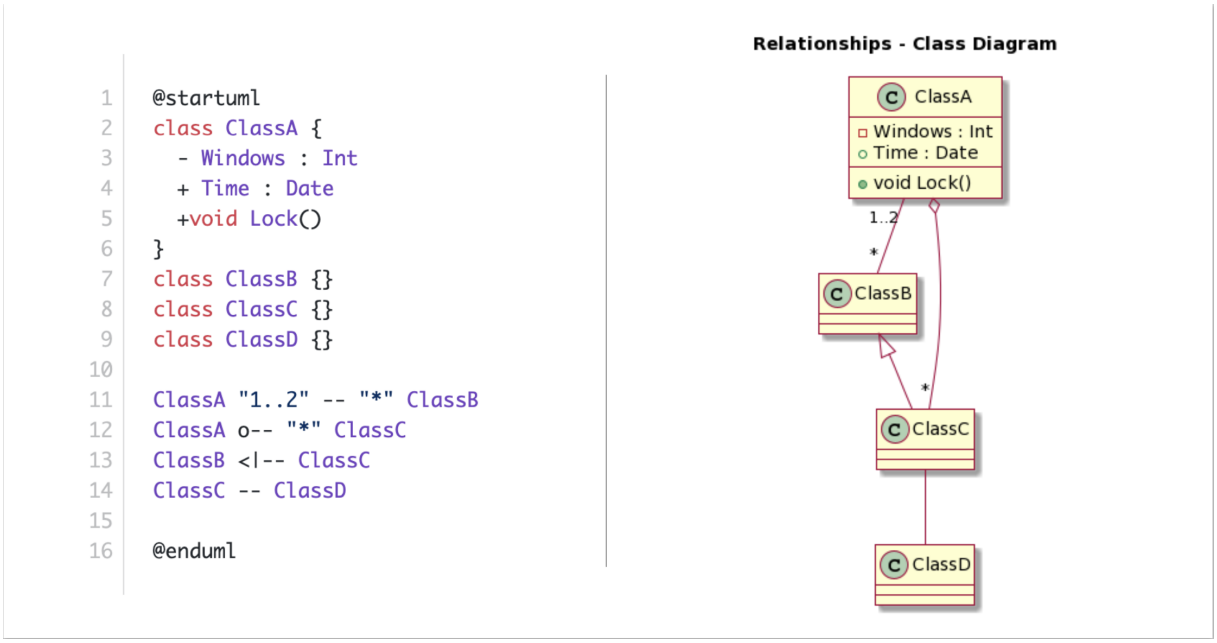


Abbildung 5.6: PlantText-Code mit der grafischen Darstellung

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.
2. Es besteht eine Kompositionsbeziehung zwischen ClassA und ClassC, die durch ein Rautensymbol auf der Seite von ClassA gekennzeich-

net 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 Abbildung 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"
          },
          {
```

```
22              "name": "Lock",
23              "isStatic": false,
24              "accessor": "+",
25              "returnType": "void",
26              "_arguments": ""
27            }
28          ],
29          ...
30        },
31        ...
32        {
33          "left": "ClassA",
34          "right": "ClassB",
35          "leftType": "Unknown",
36          "rightType": "Unknown",
37          "leftArrowHead": "",
38          "rightArrowHead": "",
39          "leftArrowBody": "-",
40          "rightArrowBody": "-",
41          "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 Code 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 Code 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 Code 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 Code 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 Code 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 Code 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 [**albert2003implementing**]. 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 Code 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 Code 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 Code 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 Code 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 Abbildung 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

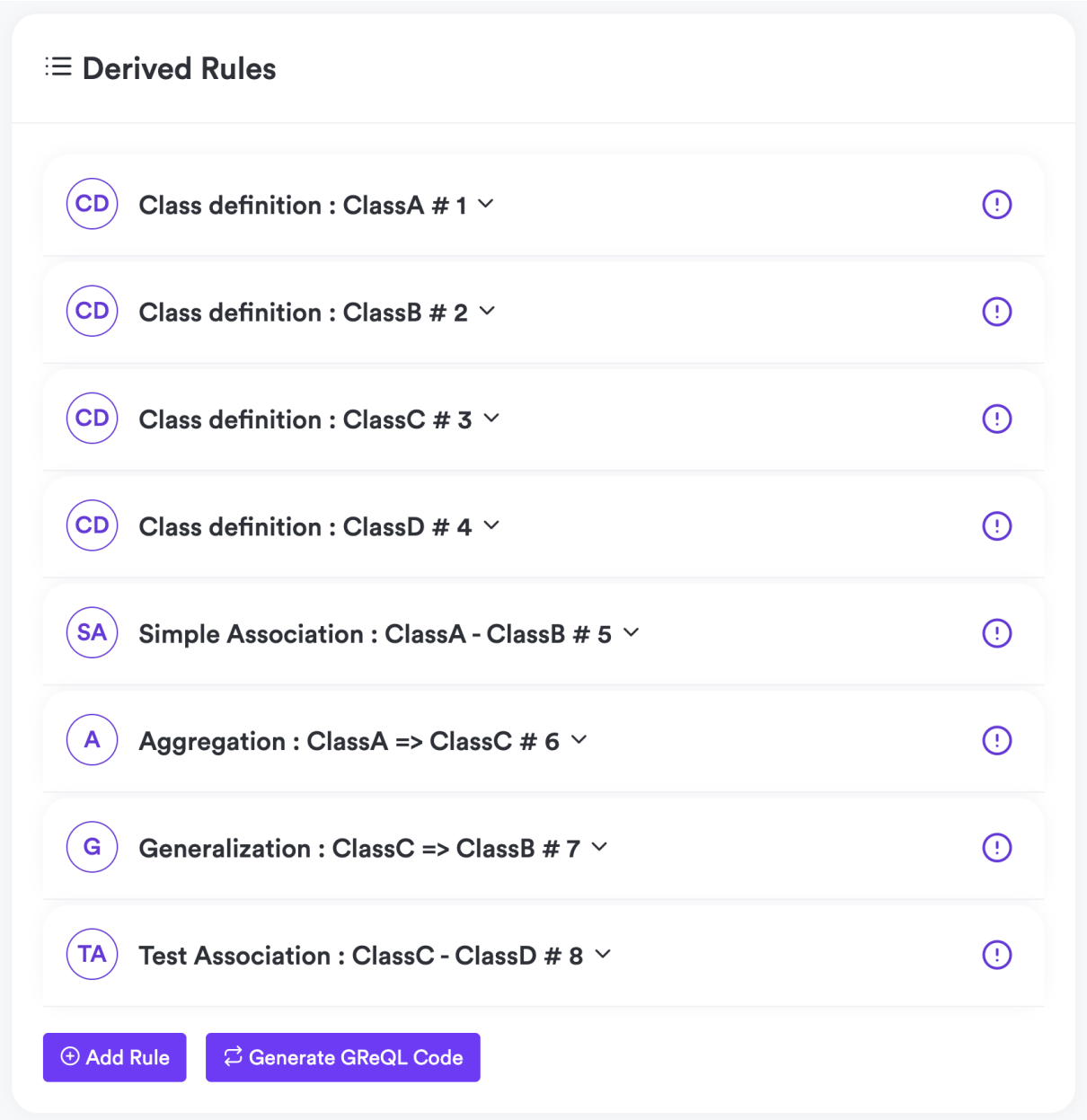


Abbildung 5.7: Regeln, die dem geparsten PlantText-Code entsprechen

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 Code 5.4):

```
<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>
```

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.

```
<checkerrules>
    <!-- Class Definition -->
    <rule type="presence" points="0">
        <query>
            from x : V{Class} with isDefined(x.name) and
               stringLevenshteinDistance(x.name,
               "ClassA")&lt;3
               and (not x.isAbstract)
               report 1 end
        </query>
        <feedback>
            Es soll eine Klasse mit der Name
            ClassA bereitgestellt werden.
        </feedback>
```

```
15      </rule>
16      <!-- Class Definition -->
17      <rule type="presence" points="0">
18          <query>
19              from x : V{Class} with
20                 isDefined(x.name) and
21                 stringLevenshteinDistance(x.name,
22                 "ClassB")&lt;3
23                 and (not x.isAbstract)
24                 report 1 end
25          </query>
26          <feedback>
27              Es soll eine Klasse mit der Name
28              ClassB bereitgestellt werden.
29          </feedback>
30      </rule>
31
32      ...
33
34      <!-- Generalization rule -->
35      <rule type="presence" points="0">
36          <query>
37              from a,b : V{Class}
38                     with
39                        isDefined(a.name) and
40                        a.name="ClassC" and
41                        isDefined(b.name) and
42                        b.name="ClassB" and
43                        a --> V{Generalization} --> b
44                     report 1 end
45          </query>
46          <feedback>
47              Das Diagramm sollte eine Klasse
48              ClassC enthalten, die von einer
49              Oberklasse ClassB erbt.
50          </feedback>
51      </rule>
52      <!-- Test Association Rule -->
53      <rule type="presence" points="0">
54          <query>
55              from x,y : V{Class}
56                             with
57                                isDefined(x.name) and
```

```
58                                  x.name="ClassC" and
59                                  isDefined(y.name) and
60                                  y.name="ClassD" and
61                                  x --> V{Property}
62                                  --> V{Association}
63                                  &lt;-- V{Property}
64                                  &lt;-- y
65                              report 1 end
66          </query>
67          <feedback prefix="Hinweis">
68              Im Diagramm gibt es keine directe Association
69              zwischen die Klasse "ClassD" und
70              die Klasse "ClassC". Das kann durch eine
71              bessere Modellierung vermieden werden.
72          </feedback>
73      </rule>
74  </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 Abbildung 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 A <!class, !attr(0,1,3), !method(1,2),
10 points for rule, 5 points for methods,
3 points for attributes> {
  - 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

!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.
!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.
x points for rule: Bestimmt, wie viele Punkte (X) die Regel der Klassendefinition haben sollte.
y points for attributes: Bestimmt, wie viele Punkte (Y) die Attributregeln haben sollten.
z points for methods: Bestimmt, wie viele Punkte (Z) die Methodenregeln haben sollten.

Ursprünglich wurde in Betracht gezogen, Annotationen direkt vor den relevanten Attributen oder Methoden zu platzieren, um das Exact Matching auf die Attribute oder Methoden zu aktivieren. Jedoch stellte sich heraus, dass dies nicht praktikabel war. Jedes Mal, wenn ein Sonderzeichen zur Unterscheidung und Abfrage der gewünschten Informationen hinzugefügt wurde, löste der PlantUML-Parser einen Fehler aus. Selbst der Versuch, den Namen des Attributs zu modifizieren, beispielsweise durch “number*exact” oder “number!exact”, erwies sich als nicht funktionsfähig. Daher wurde beschlossen, auf Klassenebene zu arbeiten, da dies als der zugänglichste Ansatz für die Umsetzung dieser Funktion erschien. Bisher wurde keine andere Möglichkeit gefunden, diesen Ansatz zu umgehen.

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 Erweiterbarkeit des GReQL-Converters

Der GReQL Converter wurde so entwickelt, dass eine einfache Erweiterung möglich ist. Dies ermöglicht zukünftigen Entwicklern, die Funktionalität des Tools zu erweitern, ohne mehrere Teile des Codes ändern zu müssen, was zu Fehlern führen könnte. Dieser Abschnitt präsentiert die Gesamtarchitektur des Datentransits im GReQL Converter sowie verschiedene Möglichkeiten zur Erweiterung des Tools.

5.5.1 Datentransit im GReQL-Converter

Der Prozess zur Generierung des GReQL-Codes aus dem GReQL Converter durchläuft mehrere Schritte, die bereits in diesem Kapitel beschrieben wurden. Diese Sektion bietet detaillierte Einblicke, die wichtig sind, um zu verstehen, wie die Funktionalitäten des Tools erweitert werden können. Dieser Prozess wird durch Abbildung 5.9 veranschaulicht, die die verschiedenen Schritte des Datentransits zeigt.

Es beginnt zunächst mit dem PlantText-Code der Lehrkraft, der an den PlantUML-Parser gesendet wird, der auf einem Node.js-Server läuft. Der Parser gibt ein JSON-Objekt zurück, das Informationen über die UML-Entitäten sowie deren Beziehungen enthält (siehe Code 5.2). Nach diesem

Schritt wird dieses JSON vom “Class Converter” verarbeitet. Der Class Converter nutzt das JSON des Parsers und enthält Methoden zur Identifizierung von Entitäten und Beziehungen, wodurch Regeln für jede Entität und Beziehung generiert werden. Die Art und Weise dieser Identifizierung wird im Implementierungskapitel beschrieben (siehe Code 5.3).

Der Class Converter verwendet die “Class Rules Definition” zur Generierung der Regeln. Die Class Rules Definition ist eine Art Enum, die als Konfigurationsdatei betrachtet werden kann (siehe Code A.2). Sie enthält alle Definitionen für jedes Objekt des Regeltyps. In dieser Datei ist es möglich, eine neue Regel und ihre Attribute zu definieren. Mithilfe dieser Datei kann der Class Converter Regeln basierend auf dem JSON des Parsers generieren. Die Verwendung der Class Rules Definition spielt eine entscheidende Rolle im GReQL Converter, da viele andere Komponenten von dieser Klasse abhängig sind.

Es wäre möglich gewesen, diese Regelobjekte direkt im Class Converter zu generieren, ohne die Class Rules Definition und unabhängig von anderen Anwendungskomponenten zu nutzen. Diese Objekte wären dann fest codiert gewesen, was eine schlechte Praxis darstellt. Dies hätte jedoch den Prozess des Hinzufügens und der Anpassung von Regeln in anderen Komponenten (die in Kürze erwähnt werden) unnötig komplex gemacht. Dies hätte von zukünftigen Entwicklern verlangt, Änderungen an mehreren Stellen vorzunehmen.

Jedoch ermöglicht die Verwendung der Class Rules Definition eine harmonisierte Regelgenerierung. Es wäre lediglich erforderlich, diese Datei zu ändern und entsprechende Zuordnungen in den sie nutzenden Dateien hinzuzufügen oder zu entfernen, um eine neue Regel hinzuzufügen oder zu entfernen. Die Regeln, die vom Lehrer geändert werden können, erfordern eine grafische Darstellung. Diese grafische Darstellung wird durch Vue.js-Klassen realisiert, die Regeln darstellen. Diese Vue.js-Klassen ermöglichen es dem Lehrer, die generierten Regelobjekte auf der grafischen Benutzeroberfläche zu ändern (siehe Abbildung 5.7). Diese Vue.js-Klassen nutzen natürlich die Class Rules Definition, um die Regelobjekte ihren verschiedenen Platzhaltern zuzuordnen.

Nach diesen Generierungs- und Änderungsschritten wird eine Liste mit allen Regelobjekten erstellt. Diese Liste wird dann an den “Class GReQL Rules Generator” gesendet. Der Class GReQL Rules Generator generiert basierend auf jeder Regel den entsprechenden GReQL-Code. Für jedes Objekt in der Liste identifiziert er zunächst den Regeltyp und generiert dann den entsprechenden GReQL-Code. Der Generierungsprozess wurde im Implementierungskapitel ausführlich beschrieben (siehe Code 5.4). Für jeden Regeltyp ist eine spezifische Methode für die Generierung des

GReQL-Codes verantwortlich.

Es wäre auch möglich gewesen, eine einzige Methode zu schreiben, die die gesamte Liste umwandeln könnte. Dies hätte jedoch den Prozess der Regelgenerierung unnötig komplex gemacht. Es wäre für einen neuen Entwickler schwer gewesen, die Funktionsweise der Methode zu verstehen und die Punkte zu identifizieren, an denen eine neue Regel hinzugefügt oder eine bereits vorhandene geändert werden könnte. Der gewählte Ansatz erleichtert das Hinzufügen neuer Regeln. Diese Vorgehensweise schafft eine Abstraktion, die die Trennung der Generierung verschiedener Regeltypen erleichtert und somit die Entwicklererfahrung verbessert.

Am Ende dieses Prozesses wird der GReQL-Code erhalten, der von bestimmten Bibliotheken leicht modifiziert wird, um ihn auf der Benutzeroberfläche darstellbar zu machen. Diese Sektion beschreibt den Datenverkehr in der Anwendung recht abstrakt.

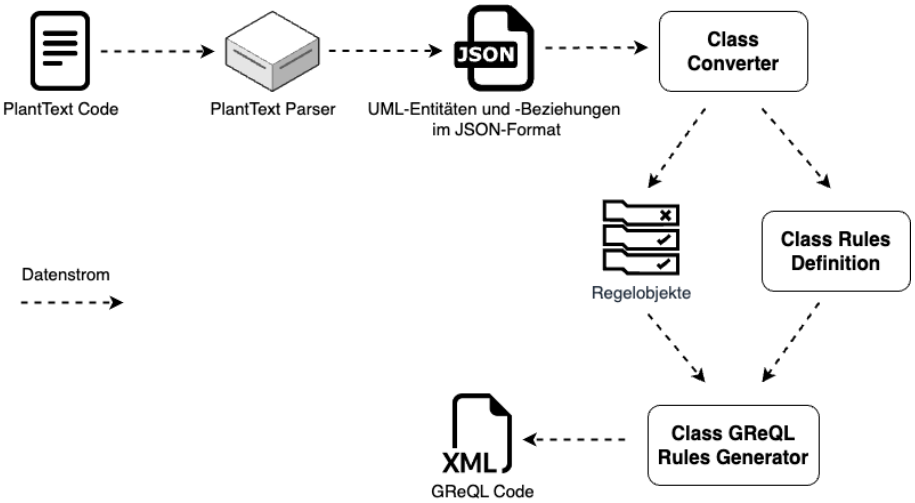


Abbildung 5.9: Schema des Datentransits im GReQL-Converter

5.5.2 Prozess der Integration neuer Regeln

In diesem Abschnitt wird gezeigt, wie man im Code des GReQL Converters neue Regeln hinzuzufügen kann. Bei der Entwicklung wurde stets berücksichtigt, dass das Tool weiter verbessert werden muss. Dies beeinflusste den Entwicklungsprozess, der darauf abzielte, Abstraktionen zu priorisieren, um eine solide Grundlage für die Verbesserung der Tool-Funktionalitäten zu schaffen.

Ein Entwickler möchte eine Regel hinzufügen, die die Anzahl der Methoden in einer Klasse bestimmt: “count_class_methods”. Zunächst muss er die

Parameter bestimmen, die diese Regel ausmachen, und sie in der Class Rules Definition definieren. Im Fall von “count_class_methods” muss er als Attribut den Klassennamen, der untersucht werden soll, sowie die Anzahl der Methoden haben. Nach dieser Definition muss der Entwickler eine neue PlantText-Annotation im “Class Converter” erstellen, um die Identifizierung dieses neuen Regeltyps zu ermöglichen (unter der Annahme, dass die Regel aus dem PlantText-Code generiert werden soll und nicht manuell). Nach der Entwicklung der Annotation soll eine Methode in der “Class Converter” erstellt werden. Diese Methode wird dazu dienen, ein JSON-Objekt basierend auf der hinzugefügten Definition in der “Class Rules Definition” zu generieren. Gleichzeitig wird die Methode Informationen aus der Annotation extrahieren und in das generierte Objekt einfügen. Das erzeugte Objekt wird einer Liste von anderen Regeln hinzugefügt. Darüber hinaus ist es erforderlich, eine Zuordnung zwischen der identifizierten Annotation und der neu erstellten Methode herzustellen. Um diese Regel für den Lehrer sichtbar und bearbeitbar zu machen, muss er eine Vue.js-Klasse erstellen, die die grafische Darstellung der Regel darstellt. Abschließend muss er zum “Class GReQL Rule Generator” gehen, um eine neue Regel und eine neue Methode zur Generierung des entsprechenden GReQL-Codes für die von ihm erstellte neue Regel hinzuzufügen.

Dieser Prozess zur Hinzufügung einer Regel ist für alle Regeln allgemeingültig. Sobald ein Entwickler dies für eine Regel durchgeführt hat, wird es für andere Entwickler offensichtlich. Der Arbeitsablauf bleibt für fast alle Regeln auf der Plattform weitgehend gleich.

5.5.3 Prozess zur Erweiterung der Kompatibilität mit einer neuen Plattform

Aktuell ist BOUML die einzige Plattform, die mit dem GReQL Converter kompatibel ist. Die Hinzufügung der Unterstützung einer neuen Plattform ist jedoch eine Aufgabe, die im GReQL Converter leicht realisierbar ist. Da sich der Regeltyp und die Parameter nicht ändern, sind keine wesentlichen Änderungen im Code erforderlich. Der Entwickler muss lediglich eine neue Query zur Vorlage jeder Regel hinzufügen (siehe Code 5.6). Wenn der GReQL Engine eine Regel auswertet, prüft er lediglich, ob eine der Queries gültig ist. Wenn die Unterstützung einer neuen Plattform hinzugefügt werden soll, muss der Entwickler nur die Queries der bereits vorhandenen Regeln vervollständigen, die eine Erweiterung erfordern. Durch diese Methode muss der Entwickler nur an einer einzigen Stelle im Code Änderungen vornehmen. Der Lehrer muss keine Zielplattform auswählen.

Sobald die Unterstützung hinzugefügt und getestet wurde, kann der Lehrer automatisch GReQL-Code für diese neue Plattform generieren.

```
<rule type="${rule.existence}" points="${rule.points}">
  <!-- Aggregation Regel -->
  <query>
    <!-- Query for BOUML -->
        from x, y : V{Class}, p: V{Property},
        a,b: V{LiteralString} with
        ${checkName}
        isDefined(p.aggregation) and
        p.aggregation="composite" and
        x --&gt; V{Property} --&gt; V{Association}
        --&gt; p &lt;-- y and
        report 1 end
  </query>
  <query>
    <!-- Enterprise Architect Query hier schreiben -->
  </query>
  <feedback>${rule.feedback}</feedback>
</rule>
```

Quelltext 5.6: Codebeispiel mit mehreren Queries

5.5.4 Prozess der Erweiterung um einen neuen Diagrammtypen

Der GReQL Converter arbeitet nur mit Klassendiagrammen. Es ist jedoch geplant, dass das Tool sich weiterentwickeln kann, indem es andere Arten von Diagrammen integriert. Um die Unterstützung eines neuen Diagrammtyps hinzuzufügen, muss der Entwickler zunächst eine Datei ähnlich dem “Class Converter” erstellen. Für Sequenzdiagramme könnte dies beispielsweise “Sequenz Converter” genannt werden. Es sollte die gleichen Funktionen wie der Class Converter haben, jedoch für Sequenzdiagramme. Es wird eine Anfrage an den PlantUML-Parser gesendet, der ein Objekt zurückgibt, das die Entitäten und Beziehungen zwischen den Elementen des Sequenzdiagramms darstellt, das mit PlantText generiert wurde. Anschließend verwendet der Sequenz Converter eine “Sequenz Rules Definition”-Datei, um Regeln zu generieren, die auf dem gleichen Prinzip basieren, das im Abschnitt zum Datentransit im GReQL-Converter

beschrieben ist (siehe Abschnitt 5.5.1). Danach muss der Entwickler Vue.js-Klassen erstellen, die zur grafischen Modifikation und Visualisierung jeder dieser Regeln dienen. Abschließend muss der Entwickler einen “Sequenz GReQL Rules Generator” erstellen, der den GReQL-Code für jede Regel generiert (basierend auf dem gleichen Prinzip wie der Class GReQL Rules Generator).

Es ist wichtig zu betonen, dass zusätzliche Änderungen an der Startseite und der Tool-Dokumentation erforderlich sind, um dem Toolbenutzer zu ermöglichen zu sehen, dass diese Funktionalität bereits möglich ist. Dieser Prozess zur Hinzufügung eines neuen Diagrammtyps ist generisch. Sobald der Entwickler den Prozess für Klassen verstanden hat, ist es recht einfach, dasselbe für andere Arten von Diagrammen zu tun. Diese Strategie wurde von Anfang an während der Entwicklung des Tools konzipiert, um zukünftige Integrationen zu erleichtern.

5.6 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 Kapitel 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 Abbildung 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.

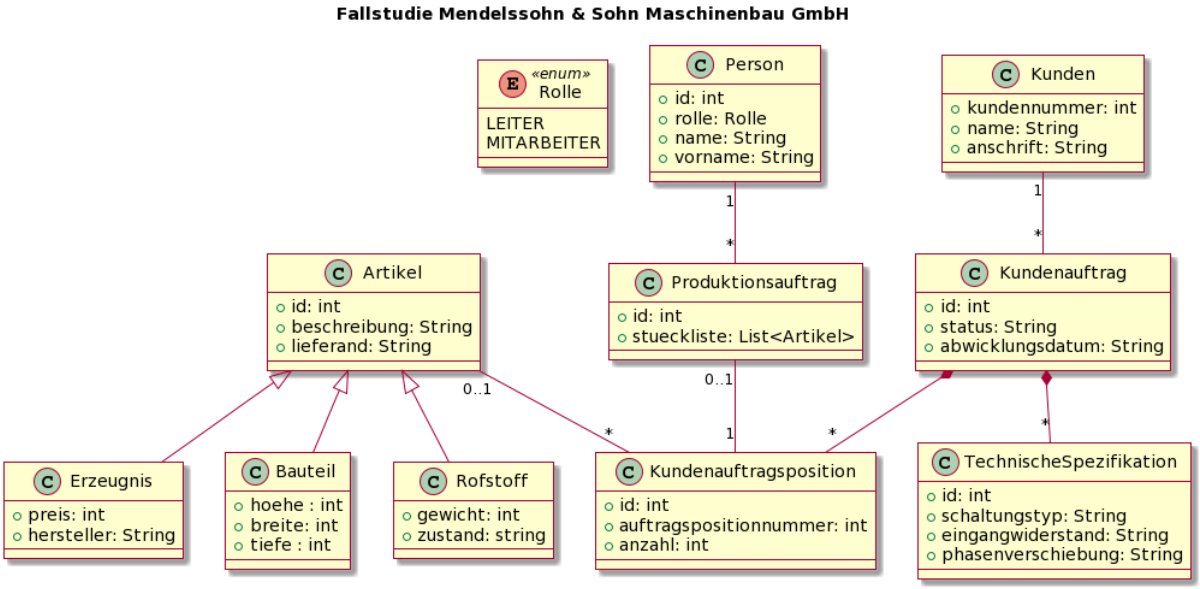


Abbildung 6.1: Fallstudie UML - Klassendiagramm

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 Kapitel 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 Interview zur Bewertung des GReQL Converters

Im vorherigen Kapitel wurde der GReQL Converter auf seine Funktionen hin getestet, um potenzielle Fehler zu erkennen und zu beheben. Es war jedoch von entscheidender Bedeutung, externe Rückmeldungen und Kommenta-

re einzuholen, um weitere Verbesserungsansätze zu identifizieren, neue Ideen zu generieren und die Grenzen des Tools aufzuzeigen. Aus diesem Grund wurde beschlossen, Interviews mit verschiedenen wissenschaftlichen Mitarbeitern durchzuführen, die bereits Erfahrung mit GReQL-Code haben, um das Tool zu testen.

Das Interview erwies sich als das geeignetste Modell, da es bereits die Möglichkeit bietet, direkte Fragen zu stellen und ein lebhafteres Feedback zu erhalten sowie detaillierte und umfassende Antworten vom Befragten zu erhalten. Darüber hinaus ermöglicht das Interviewformat eine gewisse Flexibilität, da zukünftige Fragen je nach den erhaltenen Antworten angepasst werden können. Dieses Format erleichtert auch die Kontextualisierung. Falls eine Frage nicht verstanden wird, kann der Befragte Nachfragen stellen, um im Kontext zu bleiben. Im folgenden Abschnitt wird zunächst die Interviewkonzeption sowie der Prozess und die Ergebnisse des Interviews diskutiert.

6.2.1 Konzeption des Interviews

Das Ziel des Interviews besteht darin, das Feedback verschiedener Lehrer zum GReQL Converter zu sammeln und die Möglichkeiten zur Verbesserung des Tools zu erörtern. Am Ende des Interviews wird es möglich sein festzustellen, ob das Tool einen echten Nutzen für die Benutzer hat und ob es sinnvoll ist, seine Entwicklung fortzusetzen. Um dies zu erreichen, war es zunächst erforderlich, die verschiedenen Lehrer anhand bestimmter Kriterien zu qualifizieren, darunter:

1. Die Lehrer sollten bereits wissen, was GReQL ist.
2. Die Lehrer sollten bereits GReQL-Code geschrieben haben, um ein UML-Klassendiagramm zu bewerten.

Das Interview erstreckt sich tatsächlich über fünf Phasen, von der Vorbereitung bis zum Fragebogen:

PHASE 1: Vorbereitung auf das Interview

Während des Interviews sollte der Schwerpunkt auf dem GReQL Converter liegen. Um zu verhindern, dass die Lehrer während des Interviews mit dem Schreiben von PlantText-Code beschäftigt sind, wurde beschlossen, sicherzustellen, dass die Lehrer vor dem Interview den PlantText-Code schreiben, den sie während des Interviews benötigen werden. Zu diesem

Zweck sollen die interviewten Personen eine Übung durchführen, die sie mit dem PlantText-Tool und plantUML vertraut macht.

Bei der Einladung zum Interview wurde ein Bild ausgewählt und an die Lehrer gesendet, das bereits sein äquivalentes XMI generiert hat, das später verwendet wird, um den generierten GReQL-Code auf JACK zu testen. Da das Hauptziel darin besteht, GReQL-Code aus PlantText zu generieren, sollen die Lehrer versuchen, die Musterlösung auf PlantText zu reproduzieren:

1. Wenn sie dies schaffen, müssen sie ihren PlantText-Code lediglich mit dem GReQL Converter annotieren, die Lösung auf JACK testen und die während des Interviews gestellten Fragen beantworten.

2. Wenn sie dies nicht geschafft haben, wird ihnen der PlantText-Code zur Verfügung gestellt und erklärt, wie dieser Code erstellt wurde, einschließlich einiger Feinheiten. Anschließend sollen sie diesen PlantText-Code mit dem GReQL Converter annotieren und die während des Interviews gestellten Fragen beantworten.

Es ist auch wichtig, den Lehrern die Dokumentation zum GReQL Converter sowie den Zugang zum Tool zur Verfügung zu stellen, damit sie es vor dem Interview testen und sich damit vertraut machen können, wenn ihre Verfügbarkeit dies zulässt.

PHASE 2: Einführung in das Interview

In dieser Phase geht es darum, den GReQL Converter vorzustellen, seine Feinheiten, Funktionen sowie seine Verwendung und Dokumentation zu präsentieren. Das Ziel dieser Phase besteht darin, die Lehrer mit dem Tool vertraut zu machen und ihnen alle notwendigen Informationen für den weiteren Verlauf zur Verfügung zu stellen.

PHASE 3: Annotation des PlantText-Codes

In dieser Phase müssen sich die Lehrer auf die Annotation des PlantText-Codes im GReQL Converter konzentrieren. Sie verwenden das Tool zur Generierung von GReQL-Code.

PHASE 4: Test des generierten GReQL-Codes

In dieser Phase müssen die Lehrer den generierten GReQL-Code auf JACK testen. Zu diesem Zweck wurden entsprechende Übungen bereits

auf der Plattform vorbereitet, um den Prozess zu erleichtern. Sie müssen lediglich ihren GReQL-Code sowie das zuvor vorbereitete XMI eingeben, um ihn zu testen.

PHASE 5: Fragebogen

Am Ende des Interviews werden den Lehrern eine Reihe von vorbereiteten Fragen gestellt, um ihre Erfahrungen, Meinungen und Verbesserungsvorschläge für den GReQL Converter zu bewerten. Die während des Interviews gestellten Fragen sind wie folgt:

1. **Einführung in das Tool:**
   (a) Können Sie kurz erklären, was das Tool GReQL Converter Ihrer Meinung nach ist?
   (b) Haben Sie bereits ähnliche Tools zur Generierung von GReQL-Code aus UML-Diagrammen verwendet?

2. **Benutzererfahrung:**
   (a) Wie empfanden Sie die Benutzeroberfläche des GReQL Converters in Bezug auf Benutzerfreundlichkeit?
   (b) Können Sie die Schritte beschreiben, die Sie unternommen haben, um das Tool zu verwenden, von der Code-Importierung aus PlantText bis zur Generierung von GReQL-Code?
   (c) Ist das Tool intuitiv?
   (d) Konnten Sie auf einen Blick verstehen, was Sie tun mussten?
   (e) Gab es Probleme oder Hindernisse bei der Verwendung des Tools?

3. **Dokumentation:**
   (a) Haben Sie die Dokumentation des GReQL Converters konsultiert?
   (b) Wenn ja, wie fanden Sie sie in Bezug auf Klarheit und Nützlichkeit?
   (c) Gibt es Aspekte des Tools oder seiner Funktionsweise, die Sie in der Dokumentation nicht verstanden haben und die Sie gerne darin gefunden hätten?

4. **Generierung von GReQL-Regeln:**
   (a) Wie würden Sie die Genauigkeit des Tools bei der Generierung von GReQL-Regeln aus mit PlantText erstellten UML-Diagrammen bewerten?
   (b) Gab es Fehler oder Inkonsistenzen in den generierten Regeln?

5. **Vergleich des Tools mit der manuellen Methode:**
   (a) Wie würden Sie die Effizienz des GReQL Converters im Vergleich zur manuellen Erstellung von GReQL-Regeln aus UML-Diagrammen in Bezug auf Zeitersparnis und Genauigkeit bewerten?
   (b) Können Sie konkrete Beispiele nennen, in denen das Tool besonders nützlich oder weniger effektiv als die manuelle Methode war?

6. **Verbesserungsvorschläge:**
   (a) Haben Sie Vorschläge zur Verbesserung der Benutzeroberfläche oder der Funktionen des GReQL Converters?
   (b) Gibt es zusätzliche Funktionen, die Sie gerne im Tool sehen würden?

7. **Nützlichkeit und Relevanz:**
   (a) Halten Sie das Tool GReQL Converter im Kontext der Bewertung von UML-Diagrammen für nützlich?
   (b) Warum oder warum nicht?

8. **Gesamtbewertung:**
   (a) Auf einer Skala von 1 bis 5, wobei 1 “nutzlos” und 5 “äußerst nützlich” bedeutet, wie würden Sie die Nützlichkeit des GReQL Converters bewerten?
   (b) Auf einer Skala von 1 bis 5, wobei 1 “sehr schlecht” und 5 “ausgezeichnet” bedeutet, wie würden Sie das Design des GReQL Converters bewerten?
   (c) Auf einer Skala von 1 bis 5, wobei 1 “sehr schlecht” und 5 “ausgezeichnet” bedeutet, wie würden Sie die Benutzerfreundlichkeit des GReQL Converters bewerten?
   (d) Würden Sie dieses Tool anderen Fachleuten empfehlen, die mit UML-Diagrammen zur Bewertung arbeiten?

9. **Allgemeines Feedback:**
   (a) Haben Sie weitere Kommentare, Vorschläge oder Beobachtungen, die Sie zum GReQL Converter teilen möchten?

Die grundlegende Struktur der Interviews und die Wahl der Fragen wurden maßgeblich von den Erkenntnissen und Ratschlägen des Autors Ian Brace in seinem Werk “Questionnaire Design: How to Plan, Structure and Write Survey Material for Effective Market Research” beein-

Tabelle 6.1: Befragte wissenschaftliche Mitarbeiter

Hochschule	Lehrkraft
Universität Duisburg Essen	Dr. Michael Striewe
Universität Duisburg Essen	Tobias Stottrop
Universität Duisburg Essen	Christoph Olbricht
Technische Hochschule Köln	Paul Hufnagel
Technische Hochschule Köln	Prof. Dr. Mario Winter

flusst [**brace2018questionnaire**]. Dieses Buch bietet umfassende Einblicke in bewährte Methoden zur Gestaltung von Fragen und zur Strukturierung von Umfragen, die sich auch auf die Interviewkonzeption übertragen lassen. Bei der Gestaltung der Fragen zur Benutzererfahrung und Benutzerfreundlichkeit wurde das Buch “Designing and Conducting Survey Research: A Comprehensive Guide” von Louis M. Rea und Richard A. Parker [**rea2014designing**] herangezogen, um die Best Practices für die Benutzerfreundlichkeit von Fragen zu verstehen. Online-Umfrageplattformen wie SurveyMonkey [**monkey**] dienten als Inspiration für die Erstellung des Interviewleitfadens und halfen bei der Identifizierung bewährter Praktiken zur Erstellung von Fragen und zur Strukturierung von Umfragen. Im kommenden Abschnitt steht die Präsentation einer Auswahl der Ergebnisse nach den Interviews im Vordergrund.

6.2.2 Ergebnisse des Interviewverfahrens

Die Interviews wurden mit einer Gruppe von fünf wissenschaftlichen Mitarbeitern von zwei Hochschulen geführt, nämlich der Universität Duisburg-Essen und der Technischen Hochschule Köln (siehe Tabelle 6.1).

Interview mit Christoph Olbricht

Christoph Olbricht, wissenschaftlicher Mitarbeiter an der Universität Duisburg-Essen, hat bereits GReQL-Code für die Auswertung von Java-Übungen geschrieben. Allerdings fehlt ihm Erfahrung in der Auswertung von UML-Diagrammen. Das Interview begann mit einer Vorstellung des GReQL Converters, einschließlich seiner Funktionen und Dokumentation. Vor dem Interview erhielt Herr Olbricht per E-Mail ein UML-Diagramm, für das er im Voraus PlantText-Code schrieb. Während des Interviews übertrug er den Code auf den GReQL Converter, annotierte ihn und führte mehrere Modifikationen durch. Nachfolgend generierte er GReQL-Code, der erfolgreich auf JACK getestet wurde, wobei eine Bewertung von 100% erreicht

wurde. Am Ende des Interviews wurden Fragen aus dem vorherigen Abschnittsfragebogen gestellt, um ein spezifisches Feedback zum Tool zu erhalten.

1. **Einführung in das Tool:**

   Gemäß seinen exakten Worten zum GReQL Converter: “Mit diesem Tool können wir auf einfache und intuitive Weise GReQL-Code aus PlantText generieren und ihn auf sehr angenehme Art und Weise anpassen.” Er hatte zuvor kein ähnliches Tool für die Generierung von GReQL-Code für Klassendiagramme verwendet.

2. **Benutzererfahrung:**

   Entsprechend seiner exakten Worte zur Benutzerfreundlichkeit der Anwendung: “Unheimlich angenehm. Vorher hatten wir ein ähnliches Tool für Java, aber dabei musste man seinen Java-Code einlesen und dann einen GReQL-Code einwerfen, um zu sehen, ob es funktioniert. Es war nur zum Testen gedacht. Bei deinem Tool ist es jedoch besonders angenehm, dass man die komplette Regel direkt erhält und sie nach Bedarf anpassen kann. Dadurch muss ich GReQL eigentlich nicht beherrschen, was für unsere Kunden unheimlich nützlich wäre.”

   Herr Olbricht fand das Tool intuitiv und war überrascht, dass die Regeln direkt generiert wurden, ohne dass eine umfangreiche Vorarbeit mit Annotationen geleistet werden musste. Das Tool macht viel mehr, als er erwartet hatte. Er hätte gerne einen Text gehabt, der den Prozess der Regelgenerierung sowie die Grundregeln klar erklaert und er betont am Ende: “Aber intuitiv genug war das definitiv”.

3. **Dokumentation:**

   Er lobte die Dokumentation als ausgezeichnet, ausgewogen und präzise. Die Menge an Inhalten wurde als angemessen empfunden.

4. **Generierung von GReQL-Regeln:**

   Soweit er sehen konnte, waren die erstellten Regeln sehr genau und frei von potenziellen Fehlern. Er erwähnte auch, dass es ziemlich schwierig sei, Fehler mit einem einzigen Test zu identifizieren, aber dafür müsste man mehrere Varianten des XMI-Dokuments der Musterlösung haben.

5. **Vergleich des Tools mit der manuellen Methode:**

   Entsprechend seiner exakten Worte: “Das ist eine wahnsinnige Zeiter-

sparnis. Wenn ich diesen GReQL-Code selbst geschrieben hätte, hätte ich ihn 5, 6, 7 Mal testen müssen, um zu überprüfen, ob ich keine Fehler gemacht hätte, und ich hätte mich bestimmt irgendwann mal vertippt. Ich hätte vermutlich ungefähr 2-3 Stunden gebraucht, um alle Regeln zu schreiben. Aber mit dem Tool kann man einfach die Musterlösung importieren und los geht's."

6. **Verbesserungsvorschläge:**

   Herr Olbricht schlug vor, dass es nützlich wäre, unabhängiges Scrollen im PlantText-Code und den generierten GReQL-Regeln zu ermöglichen, um einen direkten Vergleich zu erleichtern. Er erwähnte auch, dass man auf dem Bildschirm einen Fehler anzeigen kann, wenn man sich bei einer Annotation in der Syntax vertan hat (z. B. wenn man vor der Annotation bestimmter Regeln die ":" vergisst).

7. **Nützlichkeit und Relevanz:**

   Er fand das Tool sehr nützlich, da es vor allem eine enorme Zeitersparnis mit sich bringt.

8. **Gesamtbewertung:**

   Er bewertete den GReQL Converter insgesamt mit **5 von 5 Punkten** in den Kategorien **Nützlichkeit**, **Design** und **Benutzerfreundlichkeit**. Das Tool wurde als äußerst hilfreich und empfehlenswert für Fachleute eingestuft. Er sagte, wenn das Tool in Produktion sei, könne er es tatsächlich anderen Fachleuten empfehlen. Nach seinen Worten: "Nicht krakeln zu müssen, um welche Regel erzeugt werden soll, ist einfach nur nützlich."

9. **Allgemeines Feedback:**

   Herr Olbricht äußerte kein weiteres Feedback und bezeichnete das Tool als "super".

Interview mit Dr. Michael Striewe

Dr. Michael Striewe ist ein wissenschaftlicher Mitarbeiter an der Universität Duisburg-Essen. Er ist eines der Teammitglieder, das die Plattform JACK pflegt und weiterentwickelt. Zusätzlich dazu verfügt er über umfangreiche Erfahrung mit GReQL, da er die Idee einbrachte und umsetzte, GReQL-Code zur Auswertung von Klassendiagrammen zu nutzen [41]. Daher verfügt er über Fachwissen im Schreiben von GReQL-Code und

in der Auswertung von Diagrammen. Wie bei Christoph Olbricht begann das Interview mit einer Vorstellung des GReQL Converters sowie seiner Dokumentation. Anschließend sollte Dr. Michael Striewe dieselbe Aufgabe wie herr Olbricht durchführen: Code in PlantText für ein Klassendiagramm generieren. Danach annotierte er das Diagramm und testete es auf der Plattform JACK, wo er zunächst ein Ergebnis von 96% erzielte, aufgrund von zwei Fehlern. Insbesondere wurde die Syntax zum Schreiben eines Enums gemäß der Dokumentation nicht eingehalten und es gab einen Tippfehler bei der Eingabe eines Klassennamens in der XMI-Musterlösung. Genauer gesagt handelte es sich um den Namen einer Klasse “Rofstoff”, der in der Standardlösung falsch geschrieben war. Der korrekte Name sollte “Rohstoff” sein. Da er den Fehler bei der Erstellung der Regeln behob und einen exakten Abgleich für den Klassennamen verwendete, erkannte JACK den Fehler. Dieses Verhalten ist keine Programmierfehler oder ein Problem mit dem Tool, sondern vielmehr ein Beweis dafür, dass das Tool zuverlässig arbeitet und wie erwartet funktioniert. Am Ende des Interviews wurden Fragen aus dem vorherigen Abschnittsfragebogen gestellt, um ein spezifisches Feedback zum Tool zu erhalten.

1. **Einführung in das Tool:**

   Dr. Michael Striewe hat das Wesen des GReQL Converters erfasst. Er beschrieb ihn als ein Werkzeug, das es ermöglicht, aus dem PlantUML-Code eines Klassendiagramms GReQL-Code zu generieren. Er hatte zuvor keine Erfahrung mit einem solchen Tool in Bezug auf die Auswertung von UML-Diagrammen mittels GReQL.

2. **Benutzererfahrung:**

   Bezüglich Benutzererfahrung äußerte er: “Es war relativ einfach zu bedienen. Allerdings muss man viel hin und her scrollen. Ansonsten ist es relativ intuitiv und ich bin ziemlich gut damit klargekommen.” Jedoch hatte er zu Beginn einige Probleme bei der Verwendung des GReQL Converters. Als er seinen PlantText-Code auf die Plattform kopierte, konnten nur Klassendefinitionen (“Class Definition”) Regeln generiert werden, während die Assoziationsregeln nicht erkannt wurden. Dies lag daran, dass die Klassennamen in den Assoziationen in Anführungszeichen standen, was der Parser nicht erkennen konnte. Um dieses Problem zu beheben, musste der Benutzer sich an die in der Tool-Dokumentation definierte Syntax halten, und auf Entwicklerseite war es wichtig, den Benutzer zu benachrichtigen, wenn solche Fälle auftreten, damit er das Problem ohne weitere Untersuchungen verstehen kann.

3. **Dokumentation:**

   Die Dokumentation fand er leicht verständlich und nützlich. Er wünschte sich jedoch, dass bestimmte Funktionen, wie die Definition des Enums, stärker hervorgehoben würden.

4. **Generierung von GReQL-Regeln:**

   Hinsichtlich der Präzision des Tools glaubt Dr. Michael Striewe, dass ein einzelnes Beispiel nicht ausreicht, um diesen Aspekt zu testen. Es wäre notwendig, es mit mehreren Lösungen und unterschiedlichen Diagrammen mit mehr oder weniger Fehlern zu testen, um zu sehen, ob die vom GReQL Converter generierten Regeln die Feinheiten erkennen. Er erwähnte auch die Möglichkeit, dass verschiedene Diagramme, die auf PlantText identisch sind, unterschiedliche Regeln generieren könnten, je nach verwendeter Syntax. Seiner Meinung nach sollte das Tool in verschiedenen Varianten getestet werden. Ein einzelnes Beispiel reicht nicht aus, um diese Frage zu beantworten.

   Hinsichtlich der Unstimmigkeiten im generierten GReQL-Code hatte er bis zum Zeitpunkt des Interviews nichts Auffälliges bemerkt. Er betonte jedoch auch, dass er die Regeln nicht einzeln überprüft hatte, was ebenfalls weitere Untersuchungen erfordert.

5. **Vergleich des Tools mit der manuellen Methode:**

   Dr. Michael Striewe ist der Ansicht, dass durch die Verwendung des GReQL Converters viel Zeit gespart werden kann. Um das zu annotierende PlantText-Diagramm zu generieren, benötigte er nur 15 Minuten und konnte von dort aus mehr als zwanzig Regeln erstellen. Das empfand er als äußerst schnell und er könnte sich nicht vorstellen, die gleiche Aufgabe manuell genauso schnell zu erledigen. Er findet das Tool sehr effektiv bei Regeln, die viel Copy-and-Paste erfordern (zum Beispiel bei Attributregeln, Klassendefinitionen und Methoden).

   Dort, wo das Tool weniger effektiv sein könnte, wären spezielle Varianten, insbesondere Fälle, in denen spezifische Varianten von Klassennamen gefordert sind. Zum Beispiel, wenn Regeln erstellt werden müssen, bei denen die Klassennamen genau “Kunde” oder “Kunden” sein müssen, wäre es einfacher, dies manuell zu tun als den GReQL Converter zu nutzen.

6. **Verbesserungsvorschläge:**

- Die Bezeichnung des “Delete”-Buttons sollte in “Disable” geändert werden, um nicht mehr für die Generierung des GReQL-Codes in Betracht gezogen zu werden. Dadurch könnte die Möglichkeit geschaffen werden, bestimmte Codevarianten zu testen. Mit “Delete” wird die Regel gelöscht und der Code muss erneut geparst werden, alle zuvor vorgenommenen Änderungen gehen verloren. “Disable” könnte mehr Flexibilität bei der Generierung der Regeln bieten.

- Der GReQL Converter sollte in der Lage sein, verschiedene Modellierungsvarianten von Regeln zu ermöglichen. Nicht nur Varianten in Bezug auf die Klassennamen, sondern auch die Möglichkeit, mehrere Modellierungsvarianten einer Regel zu haben. Derzeit generiert der GReQL Converter nur Regeln für einen einzigen Diagrammtyp. Es ist jedoch möglich, dass Studenten verschiedene Diagrammtypen erstellen, die alle korrekt sind. Im Fall von “Sohn Maschinenbau GmbH” (siehe Abschnitt 6.1) könnte es möglich sein, “stueckliste” entweder als Attribut oder als Assoziation zu modellieren. Es gibt auch Fälle, in denen Student eine einfache Assoziation anstelle einer Aggregation verwenden könnten, und die Modellierung wäre immer noch korrekt. Der GReQL Converter sollte diese Funktionalität hinzufügen können.

- Der GReQL Converter sollte auch die Möglichkeit bieten, GReQL-Code nur für eine einzelne Regel zu generieren. Das könnte die Änderung von Regeln erheblich erleichtern.

7. **Nützlichkeit und Relevanz:**

   In Bezug auf die Nützlichkeit und Relevanz des Tools sagte Dr. Michael Striewe: “Ja, definitiv, es ist nützlich. Es erleichtert mir zunächst einmal das schnelle Erstellen einer großen Anzahl von Regeln. Danach muss ich jedoch vieles manuell nacharbeiten. Aber der Aufwand, zunächst einmal die Regeln zu erstellen, ist bereits erledigt.”

8. **Gesamtbewertung:**

   Er bewertete den GReQL Converter insgesamt mit **5 von 5 Punkten** in den Kategorien **Design** und **Benutzerfreundlichkeit**, sowie mit **4 von 5 Punkten** für die **Nützlichkeit**, da er noch Erwartungen und Wünsche für Verbesserungen hat. Das Tool wurde als äußerst hilfreich

und empfehlenswert für Fachleute eingestuft.

Interview mit Tobias Stottrop

Tobias Stottrop, wissenschaftlicher Mitarbeiter an der Universität Duisburg-Essen, hat bereits GReQL-Code für die Auswertung von UML-Klassendiagrammen im Rahmen des Vorlesung “Software Systeme” verwendet. Tobias hatte das Interview nicht vorbereitet, da er sich nicht auf der JACK-Plattform registriert hatte und auch nicht den erforderlichen PlantText-Code für die Diagrammbewertung geschrieben hatte. Daher wurde während des Interviews ein anderer Ansatz verfolgt. Basierend auf seiner Erfahrung mit GReQL und einer kurzen Einführung und Demonstration von PlantText ging es darum, verschiedene Anwendungsfälle des GReQL Converters zu testen, um zu analysieren, was möglich ist und welche Punkte noch verbessert werden müssen. Das Interviewmodell war anders als zuvor durchgeführte.

1. **Einführung in das Tool:**

   Tobias Stottrop hat verstanden, was der GReQL Converter ist. In seinen eigenen Worten erklärte er: “Es handelt sich um ein Hilfswerkzeug zur Generierung von GReQL-Code für UML-Diagramme. Es basiert auf einer Beschreibungssprache namens PlantUML und dient der Generierung von XMI-Code.” Er hatte zuvor kein solches Tool verwendet und glaubt, dass der GReQL Converter das erste seiner Art ist.

2. **Benutzererfahrung:**

   Er fand die Syntax-Hervorhebung sehr angenehm und meinte, dass dies die Benutzererfahrung mit dem GReQL Converter verbessert. Er erwähnte auch, dass es schwierig sein kann, bei einem sehr umfangreichen Diagramm den Überblick zu behalten, mit Code auf der einen Seite und mehreren generierten Regeln auf der anderen Seite. Er betonte jedoch, dass er darüber nicht urteilen könne, da ihm kein solches Beispiel vorliegt.

   Er empfand das Tool insgesamt als sehr intuitiv. Aber er erwähnte, dass bestimmte Funktionen, wie das Hinzufügen von manuellen Regeln, nicht auf Anhieb zu finden waren. Außerdem sagte er, dass es keine Hindernisse bei der Nutzung des Tools gibt, aber dass es manchmal für bestimmte Regeln manuelle Eingriffe erfordert, um genau das zu erreichen, was man möchte.

3. **Dokumentation:**

Er fand die Dokumentation verständlich und ausreichend, um das zu erreichen, was er wollte. Er betonte jedoch bedauerlicherweise, dass er nicht mehr dazu sagen könne, da er sich vor dem Interview nicht ausreichend mit dem Tool vertraut gemacht hatte.

4. **Generierung von GReQL-Regeln:**

   Tobias Stottrop betonte, dass das Feedback, das vom GReQL Converter generiert wird, sehr präzise ist, sodass der Studierende nach dem Nichtbestehen eines Teils der Übung genau weiß, was zu tun ist, nachdem er das Feedback gelesen hat (was tatsächlich die Rolle des Feedbacks ist). Allerdings äußerte er Unzufriedenheit mit dieser Funktion. Es wurde daher vorgeschlagen, das Feedback manuell zu ändern, um weniger Informationen zu liefern. Er betonte, dass er sich wünschen würde, dass bei Nichterfüllung einer Regel das Programm das Feedback der anderen Regeln nicht mehr anzeigen sollte. Diese Funktionalität liegt jedoch leider nicht im Bereich des GReQL Converters, sondern eher daran, wie GReQL interpretiert wird oder sogar am JACK-System selbst.

5. **Vergleich des Tools mit der manuellen Methode:**

   Er erwähnte, dass das Tool tatsächlich viel Zeit spart, wenn man schnell Regeln für ein Diagramm generieren möchte. Aber er betonte, dass manuelle Intervention notwendig ist, um die generierten Regeln zu vervollständigen und anzupassen. Er erwähnte auch, dass das Tool viel Copy-and-Paste spart und Fehler aufgrund von schlechten Kopien oder falscher Eingaben vermeidet.

6. **Verbesserungsvorschläge:**

   - Die Möglichkeit, basierend auf einer Annotation den Bereich einer Regel zu wählen (Abwesenheit, Vorhandensein)

   - Die Möglichkeit, Standardwerte für Attribute festzulegen (nicht sehr üblich in UML, aber eine nette Ergänzung)

   - Um das Tool zu vervollständigen, könnte auch eine Sichtbarkeitskomponente auf Paketebene für Attribute hinzugefügt werden.

   - Es ist nicht möglich, Regeln mit einer negativen Punktzahl direkt über das Annotationssystem oder die grafische Darstellung zu erstellen. Dies muss unbedingt direkt nach der Generierung des

GReQL-Codes erfolgen. Es sollte dem Benutzer möglich sein, negative Punkte hinzuzufügen.

- Hinzufügen von Regeln wie “Anzahl Attribute” und “Anzahl Methoden”, um die Anzahl der Attribute bzw. Methoden einer Klasse zu zählen und nicht direkt des gesamten Diagramms.

- Die Möglichkeit, den Typ der Parameter in Methoden zu überprüfen. Diese Funktionalität funktioniert nur für primitive Typen (dieses Problem wird im nächsten Kapitel ausführlich erläutert).

- Die Möglichkeit, den Code vorübergehend zu speichern, während Änderungen vorgenommen werden (ähnlich wie bei der Funktionsweise von PlantText).

7. **Nützlichkeit und Relevanz:**

   Er fand, dass das Tool sehr nützlich ist und Lehrern viel Zeit sparen wird.

8. **Gesamtbewertung:**

   Für die **Nützlichkeit** des Tools vergab er dem GReQL Converter eine Bewertung von **4 von 5** Punkten. Er sagte, dass es noch einige Funktionen gibt, die fehlen und die er gerne in Zukunft sehen würde. In Bezug auf das Design fand er, dass alles viel zu groß war (Text, Komponenten, Symbole). Er sagte, dass es schwierig sein wird, sich zurechtzufinden, wenn die zu generierenden Diagramme sehr groß sind. Während des Interviews wurde besprochen, dass es möglich ist, aus dem Browser heraus zu zoomen, um eine bessere Ansicht zu erhalten, was eine Option ist. Er klagte auch darüber, dass man in der Anwendung viel scrollen muss, und aus diesen Gründen vergab er eine Bewertung von **4 von 5** für das **Design**. In Bezug auf die **Benutzerfreundlichkeit** vergab er ebenfalls **3 von 5** Punkten aus denselben Gründen wie für das Design.

   Er empfiehlt das Tool seinen Kollegen und findet, dass das Tool großartig ist und eine beträchtliche Hilfe sein kann.

Interview mit Paul Hufnagel

Paul Hufnagel ist ein Masterstudent an der Technischen Hochschule Köln. Während seiner Bachelorarbeit arbeitete er mit Dr. Michael Striewe an

einem Thema, das die Verwendung von GReQL-Code zur Auswertung von UML-Diagrammen erforderte. Paul verfügt daher über eine gewisse Expertise in diesem Bereich und hat ein solides Verständnis dafür. Zu Beginn des Interviews erhielt Paul eine Präsentation des GReQL Converters, die seine Funktionalitäten und die dazugehörige Dokumentation umfasste. Anschließend wurde er gebeten, das generierte Diagramm auszuwerten, das er im Rahmen einer per E-Mail zugesandten Übung erstellt hatte. Er bewältigte die Aufgabe ohne Schwierigkeiten (da er bereits Zeit für die Vorbereitung des Interviews sowie für Tests des GReQL Converters aufgewendet hatte). Danach überprüfte er den generierten Code auf JACK und erzielte eine Erfolgsquote von 100%. Im Anschluss beantwortete er die verschiedenen Fragen der Phase 5 des Interviews.

1. **Einführung in das Tool:**

   Paul Hufnagel hat das Konzept des GReQL Converters sehr gut verstanden und kurz erläutert, dass es sich um eine Webplattform handelt, die es ermöglicht, mittels PlantUML-Syntax GReQL-Code zu generieren, um UML-Klassendiagramme zu bewerten. Er erwähnte, dass er zuvor noch nie ein ähnliches Werkzeug verwendet hatte. Zudem berichtete er von mehreren Versuchen, mit ChatGPT GReQL zu generieren, die jedoch erfolglos waren.

2. **Benutzererfahrung:**

   Er fand die Benutzeroberfläche des GReQL Converters sehr benutzerfreundlich. Zudem empfand er das Tool als äußerst intuitiv, vor allem aufgrund des vorhandenen Beispielcodes. Dieser Code half ihm, die Syntax zu verstehen, die auf dem GReQL Converter funktioniert.

3. **Dokumentation:**

   In seinen eigenen Worten: “Ich hatte anfangs einige technische Probleme beim Versuch, meinen PlantText-Code zu parsen, weil ich die Dokumentation des GReQL Converters nicht gelesen hatte. Ich hatte mich ausschließlich auf die PlantUML-Dokumentation konzentriert, aber nachdem ich die Dokumentation des GReQL Converters gelesen hatte, wusste ich, was ich tun musste, damit der Code funktioniert.” Er fand die Dokumentation sehr klar und hilfreich.

4. **Generierung von GReQL-Regeln:**

   Paul erwähnte, dass das Tool äußerst nützlich ist, da es ihm enorm viel Zeit spart. Er erwähnte, dass er sich nicht mehr um die Multiplizität

kümmern muss. Er hatte ernsthafte Probleme mit der Multiplizität und der Funktion “checkMultiplicity”, die praktisch nicht auf seinem Code funktionierte. Die Verwendung eines Tools, das all das für ihn erledigt, ist für ihn äußerst hilfreich. Er fand, dass die vom GReQL Converter generierten Regeln präzise sind. Er verglich sie mit dem Code, den er bereits manuell geschrieben hatte, und stellte fest, dass der GReQL Converter viel präziser ist. Er betonte, dass er nach dem Lesen des Codes des GReQL Converters mehrere Alternativen sah, die er für den GReQL-Code nicht unbedingt für möglich gehalten hatte.

5. **Vergleich des Tools mit der manuellen Methode:**

   Er sagte: “An meinem Regelsatz habe ich 4 Monaten gesessen. Der Konverter deckt alle Regeln ab, die wir meistens brauchen. Ich glaube, ich könnte damit meinen kompletten Regelsatz wieder schreiben und ungefähr 80 Stunden sparen.” Er fand, dass der Zeitgewinn enorm ist und die generierte Syntax sehr verständlich ist.

6. **Verbesserungsvorschläge:**

   Er erwähnte, dass der GReQL Converter derzeit nur UML-Klassendiagramme berücksichtigt. Er glaubt, dass der Code auf andere Diagrammtypen wie Sequenzdiagramme, Aktivitätsdiagramme, Use Case-Diagramme usw. erweitert werden könnte. Er sieht hier ein großes Potenzial.

7. **Nützlichkeit und Relevanz:**

   Er findet, dass der GReQL Converter äußerst nützlich ist. Er erwähnt, dass GReQL keine Sprache wie JAVA oder C ist, die leicht verständlich ist, wenn man bereits programmiert hat. Für das Verständnis von GReQL und das Schreiben gültiger Regeln benötigt man eine gewisse Einarbeitungszeit. Der GReQL Converter beseitigt diese Hürde und ermöglicht es jedem, GReQL-Code zu generieren.

8. **Gesamtbewertung:**

   In Bezug auf die **Nützlichkeit**, **Benutzerfreundlichkeit** und das **Design** des Tools vergab er eine Bewertung von **5 von 5 Punkten**. Er sagt, dass es Verbesserungen bei den Funktionen geben könnte, ist jedoch bereits sehr zufrieden mit dem, was er gesehen hat.

Interview mit Prof. Dr. Mario Winter

Prof. Dr. Mario Winter ist ein wissenschaftlicher Mitarbeiter und Professor an der Fakultät für Informatik und Ingenieurwissenschaften der TH Köln. Prof. Mario Winter hat bereits an Projekten gearbeitet, die den Einsatz von GReQL erforderten, sowie an der Plattform JACK und ihrer internen Funktionsweise. Darüber hinaus verfügt er über Erfahrung in der Modellierung von Diagrammen mit PlantUML, was ihm die Vorbereitung auf das Interview erleichterte. Prof. Mario Winter hat sich aktiv darauf vorbereitet, indem er den PlantText-Code aus dem per E-Mail erhaltenen Bild geschrieben hat und einen Vorsprung erlangte, indem er zuvor den GReQL Converter getestet hat. Vor dem Interview hatte er also bereits einen umfassenden Überblick über das Tool und seine Funktionsweise. Während des Interviews bestand seine Aufgabe darin, den von ihm generierten PlantText-Code im GReQL Converter zu testen. Anschließend annotierte er diesen Code, nachdem er den Abschnitt zur Annotation im Dokumentationsmaterial gelesen hatte, und führte dann eine Analyse des generierten GReQL-Codes durch. Nach dieser Testphase kam die Phase 5, in der es darum ging, die Interviewfragen zu beantworten.

1. **Einführung in das Tool:**

   Prof. Mario Winter hat das Wesen des GReQL Converters verstanden. Er beschrieb ihn als ein Werkzeug, das es ermöglicht, aus PlantUML-Code für Klassendiagramme automatisch GReQL-Code zu generieren, der später annotiert wird, um die verschiedenen generierten Regeln anzupassen. Er betonte, dass er zuvor noch nie ein derartiges Tool verwendet hatte, das GReQL-Code generiert.

2. **Benutzererfahrung:**

   Er fand die grafische Benutzeroberfläche des GReQL Converters sehr benutzerfreundlich und selbsterklärend, allerdings nur für Personen, die bereits eine Vorstellung davon haben, was GReQL ist. Er erwähnte, dass er lediglich Probleme mit dem Annotierungssystem hatte, was vermutlich darauf zurückzuführen war, dass er die Dokumentation nicht gründlich gelesen hatte.

3. **Dokumentation:**

   Er sah sich die Dokumentation kurz an, da er sich bereits sehr gut mit PlantUML auskannte. Das Einzige, was er nicht beachtete, war das Annotationssystem.

4. **Generierung von GReQL-Regeln:**

   Prof. Mario Winter betonte, dass die vom GReQL Converter generierten Regeln genau denjenigen entsprachen, die er von einem solchen Tool auf Basis des von ihm eingegebenen Diagramms erwartet hätte. Allerdings testete er nur mit einem einzigen Beispiel und schlug vor, mit mehreren Diagrammen zu experimentieren, um eine fundierte Antwort geben zu können.

5. **Vergleich des Tools mit der manuellen Methode:**

   Er stellte fest, dass der GReQL Converter viel effizienter und schneller ist als die manuelle Methode. Doch in Bezug auf die Genauigkeit der Regeln müssen noch weitere Tests durchgeführt werden, bevor eine konsistente Aussage getroffen werden kann. Er erwähnte jedoch, dass es nicht möglich sei, alternative Regeln zu generieren (genau wie Dr. Michael Striewe es in seinem Interview erwähnte). Der GReQL Converter sei ideal zur Generierung generischer Regeln, aber ab einer gewissen Komplexität sei manuelle Intervention erforderlich.

6. **Verbesserungsvorschläge:**

   - Der GReQL Converter sollte die Möglichkeit bieten, am Anfang des PlantText-Modells Variablen zu definieren und Platzhalter im Modell zu lassen, die von diesen Variablen gefüllt werden. Zum Beispiel könnte eine Variable ClassA = “Auto” sein, und in einer anderen Version ClassA = “Fahrzeug”. Überall im PlantText-Code, wo ClassA erscheint, sollte beim Generieren des GReQL-Codes je nach Definition der Variable entweder “Auto” oder “Fahrzeug” eingesetzt werden. Diese Funktionalität würde es ermöglichen, verschiedene Versionen desselben Diagramms zu haben und unterschiedliche Übungen für verschiedene Studentengruppen zu generieren. Damit wäre es möglich, Übungsfamilien zu erstellen, um verschiedene Schülergruppen zu bewerten.

   - Die Möglichkeit, Regeln mit mehreren Alternativen zu modellieren.

7. **Nützlichkeit und Relevanz:**

   Er fand das Tool äußerst nützlich und sieht ein beträchtliches Potenzial für die Erweiterung des Tools, insbesondere hinsichtlich der Integration anderer Diagrammtypen.

8. **Gesamtbewertung:**

   Hinsichtlich der **Nützlichkeit** vergab er dem GReQL Converter eine Bewertung von **4 von 5 Punkten**, da er noch verschiedene Verbesserungsbereiche sieht. Bezüglich des **Designs** und der **Benutzerfreundlichkeit** vergab er **3 von 5 Punkten**. Er erwähnte, dass es sicherlich Verbesserungspotenzial gibt, aber da er nicht ausgiebig getestet hat, zieht er es vor, zunächst einen Mittelwert zu vergeben.

6.3 Erweiterung von GReQL Converter-Funktionen gemäß der Interviewrückmeldungen

Nach Abschluss der Interviewphase wurden zahlreiche Rückmeldungen gegeben, um zur Verbesserung des GReQL Converters und seiner Funktionalitäten beizutragen. Da es nicht möglich war, alle diese Funktionen nur im Rahmen dieser Masterarbeit zu implementieren, wurde eine Auswahl der Funktionen getroffen, die den größten Einfluss auf das Benutzererlebnis und die Funktionalitäten haben. In diesem Kapitel geht es darum, diese verschiedenen Funktionen vorzustellen.

6.3.1 Feature 1: Doppeltes Scrolling

Während der Interviews äußerte die Mehrheit der Teilnehmer Beschwerden über die Notwendigkeit eines umfangreichen Scrollens. Die Anordnung des PlantText-Codes auf der linken Seite des Bildschirms und der generierten Regeln auf der rechten Seite führte dazu, dass das Scrollen auf dem Bildschirm für beide Teile gleichzeitig erfolgte. Dies erschwerte die Visualisierung erheblich, insbesondere bei komplexeren und umfangreicheren Diagrammen, wenn Vergleiche angestellt oder Regeln im Zusammenhang mit dem PlantText-Code identifiziert werden mussten.

Um dieses Problem anzugehen, wurde eine unabhängige Scrollfunktion für die beiden Bildschirmbereiche implementiert. Dadurch ist es möglich, in den generierten Regeln zu scrollen, ohne dass sich die Ansicht des PlantText-Codes ändert. Dies erleichtert es dem Benutzer, Korrespondenzen zwischen dem PlantText-Code und den generierten Regeln zu suchen. Das Ziel dieser Funktionalität besteht darin, die Benutzererfahrung zu verbessern und das Tool angenehmer in der Anwendung zu gestalten.

6.3.2 Feature 2: Regeln deaktivieren/aktivieren

Während des Interviews mit Dr. Michael Striewe erwähnte er das Problem des “Delete”-Buttons, der verwendet wird, um Regeln zu entfernen, die generiert wurden und die möglicherweise nicht im GReQL-Code benötigt werden. Er zeigte jedoch während des Interviews die Grenzen dieser Funktionalität auf und schlug vor, einen Button zu verwenden, der es ermöglicht, Regeln vorübergehend zu deaktivieren, anstatt sie zu löschen. Falls ein Benutzer beispielsweise versehentlich eine Regel löscht, gäbe es keine Möglichkeit zur Wiederherstellung. Der Benutzer müsste den PlantText-Code erneut parsen, und alle Änderungen an den Regeln wären unwiederbringlich verloren.

Diese neue Funktionalität ermöglicht es dem Benutzer, eine Regel, ein Attribut oder eine Methode vorübergehend zu deaktivieren, ohne sie zu löschen (siehe Abbildung 6.2). Auf diese Weise kann der generierte Code zuvor visualisiert werden, und falls eine Deaktivierung versehentlich erfolgt ist, könnte die Regel einfach wieder aktiviert werden. Diese Funktion wurde implementiert. Das Ergebnis dieser Funktionalität besteht darin, dass sie mehr Flexibilität bei der Generierung von GReQL-Code bietet und die Visualisierung des Codes erleichtert, insbesondere in Kombination mit der nächsten Funktion, dem Rule Viewer.

6.3.3 Feature 3: Rule Viewer

Diese Funktion wurde ebenfalls von Dr. Michael Striewe vorgeschlagen und zielt darauf ab, die Benutzererfahrung zu verbessern. Um den GReQL-Code einer bestimmten Regel zu visualisieren, muss zunächst der GReQL-Code aller Regeln generiert werden, um dann zu untersuchen, welcher Teil der Regel entspricht, die man betrachten möchte. Im Fall der Klassendefinition könnte es vorkommen, dass der Benutzer nur sehen möchte, worauf sich die Regel bezieht, nachdem er Attribute deaktiviert oder bestimmte Parameter geändert hat. Es war zuvor nicht möglich, den Code einer einzelnen Regel zu visualisieren.

Diese Funktion ermöglicht es nun, durch Klicken auf einen Button den entsprechenden GReQL-Code einer bestimmten Regel zu generieren (siehe Abbildung 6.2). Dadurch kann der Benutzer Tests und Änderungen leichter durchführen, was mehr Möglichkeiten bietet und das Benutzererlebnis verbessert.

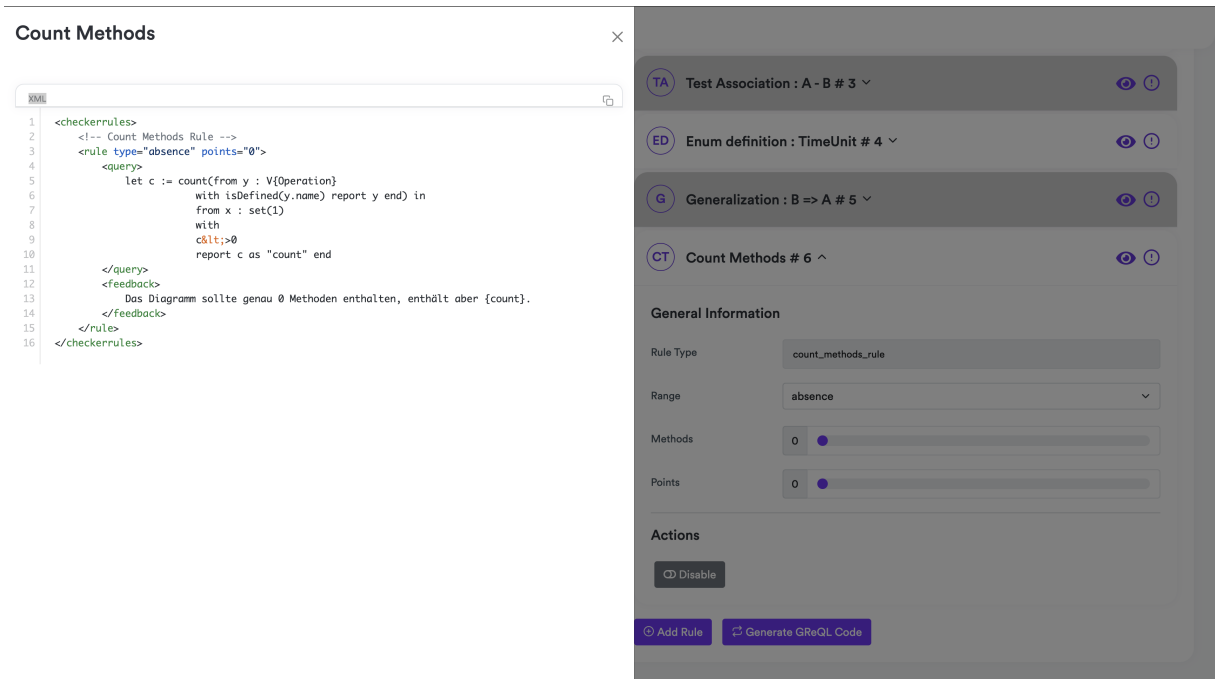


Abbildung 6.2: Feature 2 und 3

Auf der linken Seite des Bildes ist der GReQL-Code für die Regel “Count Methods” zu sehen. Um dieses Canvas anzuzeigen, kann man auf die Schaltfläche in Form eines Auges neben dem Titel der Regel klicken. Auf der linken Seite sind mehrere Regeln in Grau markiert, was bedeutet, dass sie deaktiviert sind. Diese werden daher bei der vollständigen Generierung des GReQL-Codes ignoriert.

6.3.4 Feature 4: Kombinierte Regeln

Die erste Version des GReQL Converters war zweifellos sehr nützlich, da er die Erstellung einer großen Menge von Regeln allein aus einem PlantText-Code ermöglichte. Allerdings blieb seine Nützlichkeit auf die Generierung einfacher und größtenteils generischer Regeln beschränkt. Diese Einschränkung wurde vom Dr. Michael Striewe während seines Interviews aufgezeigt, als er den Wunsch äußerte, komplexere Regeln aus annotiertem PlantText-Code zu generieren. Ein praktisches Beispiel hierfür wäre zum Beispiel eine Klasse “Bibliothek”, die ein oder mehrere Objekte der Klasse “Buch” enthält. Diese einfache Darstellung kann auf verschiedene Arten modelliert werden, sei es durch das Hinzufügen von “Buch” als Attribut zur Klasse “Bibliothek“, durch Aggregation, Komposition oder einfache Beziehung. Die Einschränkung der ersten Version des GReQL Converters bestand darin, dass er keine Möglichkeit bot, Regeln zu kombinieren, um

verschiedene, aber alle korrekte Antworten zu ermöglichen.

Um dieses Problem zu lösen, wurde die Entwicklung der Funktion zur Kombination von Regeln vorangetrieben. Diese Funktion zielt darauf ab, eine gewisse Flexibilität zu bieten und Lehrkräften die Definition von deutlich komplexeren Regeln zu ermöglichen. Damit können Variationen der Antworten definiert werden, die alle korrekt sein können. Um die Funktionsweise dieser Funktionalität zu veranschaulichen, ist es sinnvoll, dies anhand eines Beispiels zu erläutern. Angenommen, ein Lehrer möchte überprüfen, ob ein Schüler die richtige Beziehung zwischen der Klasse “Bibliothek” und der Klasse “Buch” korrekt verwendet hat (siehe Abbildung 6.3). Es gibt jedoch mehrere Möglichkeiten, wie diese Beziehung konzipiert werden kann, wie zuvor erwähnt. Um GReQL-Code für diesen speziellen Fall zu generieren, wurde das Annotationsystem erweitert. Durch das Stichwort “combineID” können Kombinationen mehrerer Regeln mit derselben Kennung erstellt werden. Dies ermöglicht schließlich die Generierung von GReQL-Code (siehe Code 6.1), der eine der drei Bedingungen überprüft. Wenn eine dieser Bedingungen erfüllt ist, erhält der Schüler die Punkte. Details zu diesem Annotationsystem finden sich in der Dokumentation des GReQL Converters.

```
1  @startuml
2
3  Bibliothek o-- "*" Buch : <combineID=1>
4  Bibliothek *-- "*" Buch : <combineID=1>
5  Bibliothek "1" -- "*" Buch : <combineID=1>
6
7  @enduml
```

Abbildung 6.3: Kombinierte Regeln

```
1 <checkerrules>
2     <!-- Combined  Rule -->
3
```

```
4     <rule type="presence" points="0">
5
6         <!-- Aggregation SubRule -->
7         <query>
8             from x, y : V{Class}, p: V{Property},
9             a,b: V{LiteralString}
10                with
11                isDefined(x.name) and
12                stringLevenshteinDistance(x.name,
13                "Bibliothek")&lt;3 and
14                isDefined(y.name) and
15                stringLevenshteinDistance(y.name,
16                "Buch")&lt;3 and
17                isDefined(p.aggregation) and
18                p.aggregation="shared" and
19                x --> V{Property} -->
20                V{Association} --> p &lt;-- y and
21                isDefined(a.value) and
22                a.value="*"  and
23                isDefined(b.value) and
24                b.value="*"  and
25                x --> V{Property} --> a and
26                x --> V{Property} --> b
27                report 1 end
28        </query>
29
30        <!-- Composition SubRule -->
31        <query>
32            from x, y : V{Class}, p: V{Property},
33            a,b: V{LiteralString}
34                with
35                isDefined(x.name) and
36                stringLevenshteinDistance(x.name,
37                "Bibliothek")&lt;3 and
38                isDefined(y.name) and
39                stringLevenshteinDistance(y.name,
40                "Buch")&lt;3 and
41                isDefined(p.aggregation) and
42                p.aggregation="composite" and
43                x --> V{Property} -->
44                V{Association} --> p &lt;-- y and
45                isDefined(a.value) and
46                a.value="*"  and
```

```
47                  isDefined(b.value) and
48                  b.value="*"  and
49                  x --> V{Property} --> a and
50                  x --> V{Property} --> b
51                  report 1 end
52          </query>
53
54          <!-- Simple Association SubRule -->
55          <query>
56              from x,y : V{Class}, ass : V{Association},
57              a,b,c,d  : V{LiteralString}
58                  with
59                  isDefined(x.name) and
60                  stringLevenshteinDistance(x.name,
61                  "Bibliothek")&lt;3 and
62                  isDefined(y.name) and
63                  stringLevenshteinDistance(y.name,
64                  "Buch")&lt;3 and
65                  isDefined(a.value) and
66                  a.value="*"  and
67                  isDefined(b.value) and
68                  b.value="*"  and
69                  x --> V{Property} --> a and
70                  x --> V{Property} --> b and
71                  isDefined(c.value) and
72                  c.value="1"  and
73                  isDefined(d.value) and
74                  d.value="1"  and
75                  y --> V{Property} --> c and
76                  y --> V{Property} --> d and
77                  x --> V{Property} --> ass
78                  &lt;-- V{Property} &lt;-- y
79                  report 1 end
80          </query>
81          <feedback>
82              ... i need a feedback please     .
83          </feedback>
84      </rule>
85  </checkerrules>
```

Quelltext 6.1: Ausschnitt aus dem GReQL-Code, der nach der Konvertierung erhalten wurde

6.4 Zusammenfassung der Evaluation

Abschließend bietet die Analyse und Bewertung des GReQL Converters einen detaillierten Einblick in die Wirksamkeit und Potenziale dieses Tools. Die erörterten Verbesserungsvorschläge sowie die ermittelten Stärken und Schwächen legen den Grundstein für zukünftige Entwicklungen und zeigen auf, wie dieses Instrument im Bildungsumfeld und darüber hinaus noch effektiver genutzt werden könnte. Die Bewertung des Tools, getragen von den Erwartungen, dem Nutzen und den Erkenntnissen aus den Anwendererfahrungen, bildet eine solide Grundlage für weiterführende Studien und Implementierungen zur Steigerung seiner Funktionalitäten und Anwenderfreundlichkeit. Diese umfassende Bewertung legt den Fokus auf die kontinuierliche Weiterentwicklung und Optimierung des GReQL Converters, um seinen Wert und seine Wirksamkeit für zukünftige Anwendungen zu maximieren.

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

Die Interviews ergaben überwiegend positive Ergebnisse. Der GReQL Converter stellt ein innovatives Tool dar, das als erstes seiner Art die Generierung von GReQL-Code ermöglicht, um Klassendiagramme zu bewerten. Alle Befragten äußerten sich äußerst positiv über die Nützlichkeit des GReQL Converters und betonten dessen Potenzial, Lehrern erheblich Zeit beim Verfassen von GReQL-Code zu sparen. Sie sind außerdem bereit, das Tool ihren Kollegen zu empfehlen, die bereits Erfahrung mit GReQL-Code für die Klassendiagrammbewertung haben.

Die grafische Benutzeroberfläche und die Nutzererfahrung stießen bei den meisten Befragten auf Zustimmung. Dennoch wurde auf verschiedene Bereiche hingewiesen, die noch verbessert werden könnten, um das Tool weiter zu optimieren. Obwohl der GReQL Converter die Generierung generischer Regeln stark erleichtert, stellten die Interviews fest, dass bei zunehmender Komplexität manuelle Eingriffe erforderlich sind. Es wurde betont, dass fortlaufende Verbesserungen an dem Tool dazu beitragen könnten, den Bedarf an solchen manuellen Interventionen im Laufe der

Zeit zu verringern. Es ist jedoch wichtig zu beachten, dass der GReQL Converter nicht darauf abzielt, die menschliche Intervention zu ersetzen. Stattdessen sollte er als unterstützendes Instrument für Lehrkräfte betrachtet werden.

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.

PlantUML Parser

Bezüglich des PlantUML 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 [**GReQL-wiki**]). 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 [**OMG˙UML˙23˙Infrastructure**], der von BOUML in Gebrauch genommen wird. Dieser Standard erkennt jedoch lediglich vier primitive Datentypen, nämlich int, bool, string und UnlimitedNatural [**OMG˙UML˙23˙Infrastructure**]. 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 pri-

mitiv 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.

7.3.1 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.

7.3.2 Erweiterung bestehender Regeln

Eine Vielzahl von Regeln könnte von Verbesserungen profitieren, um eine größere Bandbreite von Designvarianten zu unterstützen und kompatibler zu werden. Um dieses Ziel zu erreichen, ist es unerlässlich, den GReQL Converter an einer beträchtlichen Anzahl von Diagrammen zu testen, um

seine Grenzen und Einschränkungen zu identifizieren und umfassend anzugehen. Durch eine breite Testbasis können potenzielle Schwachstellen erkannt und verbessert werden, um eine robustere und flexiblere Anwendung des Tools zu gewährleisten.

7.3.3 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. Eine Möglichkeit, die Kompatibilität mit Enterprise Architect hinzuzufügen, wurde im Abschnitt über die Erweiterbarkeit des GReQL-Converters gezeigt (siehe Abschnitt 5.5).

7.3.4 Erweiterung des PlantUML-Syntaxparsers zur Erkennung von Syntaxfehlern

Während des Interviews mit Dr. Michael Striewe wurde ein Verhalten bemerkt. Als er seinen PlantText-Code auf die Plattform kopierte, konnten nur die Klassendefinitionen Regeln (“Class Definition”) generiert werden, und die Assoziationsregeln wurden nicht erkannt. Dies lag daran, dass die Klassennamen in den Assoziationen in Anführungszeichen standen, was der Parser nicht erkennen konnte. Dieses Problem lässt sich dadurch erklären, dass der Parser solche Ausdrücke nicht als Fehler erkennt. Es gibt mehrere Arten von gültigem PlantText-Code, der jedoch nicht zwangsläufig vom Parser verarbeitet werden kann. Aus diesem Grund sollte man sich an die Dokumentation halten, um die Regeln zu schreiben, und auf der Entwicklungsseite einen Weg finden, diese Fehler zu identifizieren und insbesondere den Benutzer darauf aufmerksam zu machen, dass etwas nicht wie erwartet funktioniert. Das gilt auch für den Fehler, den er bezüglich des Enums erwähnt hat, der eine spezifische Syntax im GReQL Converter erfordert.

Der PlantUML-Parser ist ein Open-Source-Projekt, was bedeutet, dass der Quellcode für jeden zugänglich, modifizierbar und verteilbar ist. Es ist also möglich, direkt auf der Parser-Ebene Änderungen vorzunehmen, um

Funktionen hinzuzufügen und damit seine Fähigkeiten zu erweitern.

7.3.5 Erweiterung auf andere UML-Diagrammtypen

Im Rahmen dieser Masterarbeit wurde der GReQL Converter auf die Generierung von GReQL-Code für UML-Klassendiagramme spezialisiert. Während der Entwicklung wurde jedoch berücksichtigt, dass das Tool erweitert werden kann, um auch andere Arten von Diagrammen wie Aktivitäts-, Sequenz- oder sogar Use-Case-Diagramme zu unterstützen (siehe Abschnitt 5.5). Es ist bereits möglich, mithilfe von GReQL Code zu schreiben, um diese genannten Diagrammtypen zu evaluieren. Zusätzlich sind diese Diagramme mithilfe von PlantUML modellierbar. Folglich könnte der GReQL Converter tatsächlich erweitert werden, um Code für verschiedene Arten von Diagrammen zu generieren. Es wäre also ein Ansatz zur Verbesserung des GReQL Converters, nicht nur auf Klassendiagramme beschränkt zu sein, sondern auch andere Diagrammtypen zu unterstützen. Dies würde das Tool vielseitiger machen und seine Anwendungsfälle deutlich erweitern.

7.4 Zusammenfassung der Diskussion

Die Entwicklung des GReQL Converters basierte auf der grundlegenden Überlegung, ein Tool zu schaffen, das sich kontinuierlich weiterentwickeln kann. Dieser Aspekt fand sich in der Gestaltung des Codes wider, der gemäß etablierter Designprinzipien und bewährter Praktiken verfasst wurde. Der Fokus lag darauf, kommenden Entwicklern, die an diesem Tool arbeiten, eine klare Orientierung zu bieten und ihre Entwicklererfahrung erheblich zu verbessern, wie auch McConnell es erwähnt [**mcconnell2006software**]. Diese strategische Herangehensweise zielt darauf ab, eine solide Grundlage zu schaffen, die zukünftige Erweiterungen und Anpassungen erleichtert und einen nahtlosen Entwicklungsprozess für kommende Versionen des GReQL Converters ermöglicht.

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.[**amraouy2023sentiment**] 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.[**aggarwal2018machine**] rele-

vant, die verschiedene Algorithmen des maschinellen 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] W. Abramowicz.
Business Information Systems: 16th International Conference, BIS 2013, Poznań, Poland, June 19-21, 2013, Proceedings. Bd. 157. Springer, 2013.

[2] 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.

[3] 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.

[4] 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).

[5] 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.

[6] 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.

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

[8] 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.

[9] *Dynexite*. 2023. URL: https://dynexite.rwth-aachen.de/t/login.

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

[11] 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).

[12] 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.

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

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

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

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

[28] *Leapsome*. 2023. URL: https://www.leapsome.com/.

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

[30] LPLUS GmbH. *LPLUS*. 2023. URL: https://lplus.de/home-eng.

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

[32] 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.

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

[34] 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).

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

[36] *Questionmark*. 2023. URL: https://www.questionmark.com/.

[37] 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.

[38] 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.

[39] 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.

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

[41] 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.

[42] 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.

[43] *Turnitin*. 2023. URL: https://www.turnitin.com/.

[44] M. Ullrich et al. „Automated assessment of conceptual models in education“. In: *Emisa Journal (EMISAJ)* (2023). URL: https://emisa-journal.org/emisa/article/view/278ue.

[45] 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.

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

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',

    // OTHERS
    'combined': 'combined_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',
        info_text: rulesExplanations.defined_class_rule,
        info_image: 'assets/rules/defined_class_rule.png',
        feedback: '... no feedback yet',
        points: 0,
        existence: 'presence',
        active: true,
```

```
40          rule_specific: {
41              class_name: "Car",
42              exact_match: false,
43              abstract: false,
44              interface: false,
45              methods: [],
46              attributes: [],
47          }
48      },
49      // ENUM
50      'defined_enum_rule': {
51          rule_type: 'defined_enum_rule',
52          rule_name: 'Enum definition',
53          info_text: rulesExplanations.defined_class_rule,
54          info_image: 'assets/rules/defined_class_rule.png',
55          feedback: '... no feedback yet',
56          points: 0,
57          existence: 'presence',
58          active: true,
59          rule_specific: {
60              enum_class_name: "Car",
61              exact_match: false,
62              attributes: [],
63          }
64      },
65      // GENERALIZATION & SPECIALIZATION
66      'generalization_rule': {
67          rule_type: "generalization_rule",
68          rule_name: "Generalization",
69          existence: "presence",
70          info_text: rulesExplanations.generalization_rule,
71          info_image: "assets/rules/generalization_rule.png",
72          points: 0,
73          feedback: '... no feedback',
74          active: true,
75          rule_specific: {
76              class_child: "Child",
77              class_parent: "Parent",
78              exact_match: false,
79              type: "inheritance" // implementation
80          }
81      },
82      // RELATIONSHIPS
```

```
83      'simple_association_rule': {
84          rule_type: "simple_association_rule",
85          rule_name: "Simple Association",
86          existence: "presence",
87          info_text: rulesExplanations
88              .simple_association_rule,
89          info_image: "assets/rules/relations.png",
90          points: 0,
91          feedback: "... no feedback",
92          active: true,
93          rule_specific: {
94              class_A: "Class A",
95              class_B: "Class B",
96              exact_match: false,
97              A_multiplicity: "1",
98              B_multiplicity: "1"
99          }
100     },
101     'composition_rule': {
102         rule_type: "composition_rule",
103         rule_name: "Composition",
104         existence: "presence",
105         info_text: rulesExplanations.
106             aggregation_composition_rule,
107         info_image: "assets/rules/relations.png",
108         points: 0,
109         feedback: "... no feedback",
110         active: true,
111         rule_specific: {
112             class_composite: "Composite",
113             class_element: "Element",
114             exact_match: false,
115             element_multiplicity: "*",
116         }
117     },
118     'aggregation_rule': {
119         rule_type: "aggregation_rule",
120         rule_name: "Aggregation",
121         existence: "presence",
122         info_text: rulesExplanations
123             .aggregation_composition_rule,
124         info_image: "assets/rules/relations.png",
125         points: 0,
```

```
126         feedback: "... no feedback",
127         active: true,
128         rule_specific: {
129             class_aggregate: "Aggregate",
130             class_element: "Element",
131             exact_match: false,
132             element_multiplicity: "*",
133         }
134     },
135     // ASSOCIATION CLASS
136     'association_class_rule': {
137         rule_type: "association_class_rule",
138         rule_name: "Association Class",
139         existence: "presence",
140         info_text: rulesExplanations.association_class_rule,
141         info_image: "assets/rules/ass_class.png",
142         points: 0,
143         feedback: "... Es muss eine Assoziationsklasse C
144         auf der Beziehung zwischen Klasse A
145         und Klasse B haben.",
146         active: true,
147         rule_specific: {
148             class_A: "Class A",
149             class_B: "Class B",
150             class_C: "Association Class",
151             exact_match: false,
152         }
153     },
154     // OTHERS
155     'combined_rule': {
156         rule_type: "combined_rule",
157         rule_name: "Combined Rule",
158         existence: 'presence',
159         info_text: rulesExplanations.combined_rule,
160         info_image: '',
161         points: 0,
162         active: true,
163         feedback: "... i need a feedback please.",
164         rules:[]
165     },
166     // OPTIONAL
167     'nomination_consistency_rule': {
168         rule_type: "nomination_consistency_rule",
```

```
169         rule_name: "Nomination Consistency",
170         info_text: rulesExplanations.nomination_consistency,
171         info_image: "",
172         active: true,
173     },
174     'count_methods_rule': {
175         rule_type: "count_methods_rule",
176         rule_name: "Count Methods",
177         existence: 'absence',
178         info_text: rulesExplanations.count_methods_rule,
179         info_image: '',
180         points: 0,
181         active: true,
182         rule_specific: {
183             methods: 0,
184         }
185     },
186     'count_attributes_rule': {
187         rule_type: "count_attributes_rule",
188         rule_name: "Count Attributes",
189         existence: 'absence',
190         info_text: rulesExplanations.count_attributes_rule,
191         info_image: '',
192         points: 0,
193         active: true,
194         rule_specific: {
195             attributes: 0,
196         }
197     },
198     'test_association_rule': {
199         rule_type: "test_association_rule",
200         rule_name: "Test Association",
201         existence: "absence",
202         info_text: rulesExplanations.test_association_rule,
203         info_image: "assets/rules/relations.png",
204         points: 0,
205         active: true,
206         rule_specific: {
207             class_A: "Class A",
208             class_B: "Class B",
209             exact_match: false,
210         }
211     },
```

```
},
METHODS_TYPE: {
    name: "public_method_name",
    exact_match: false,
    return_type: "void",
    visibility: "public",
    arguments: "",
    points: 0,
    active: true,
    feedback: '... no feedback',
    is_static: false
},
ATTRIBUTE_TYPE: {
    name: "attribute_name",
    exact_match: false,
    type: "string",
    visibility: "public",
    points: 0,
    active: true,
    feedback: '... no feedback',
    is_static: false
},
ENUM_ATTRIBUTE_TYPE: {
    name: "ENUM_ATTR",
    exact_match: false,
    points: 0,
    active: true,
    feedback: '... no feedback',
},
EXISTENCE_TYPE: {
    'presence': 'presence',
    'absence': 'absence'
},
GENERALIZATION_TYPE: {
    'inheritance': 'inheritance',
    'implementation': 'implementation'
},
COMBINED_RULE_DEFINITION: {
    DEFINE_CLASS : "Class Definition sub-rule",
    METHOD: "Method sub-rule",
    ATTRIBUTE: "Attribute sub-rule",
    GENERALIZATION: "Generalization sub-rule",
    COMPOSITION: "Composition sub-rule",
```

```
255     AGGREGATION: "Aggregation sub-rule",
256     SIMPLE_ASSOCIATION: "Simple_association sub-rule",
257     ASSOCIATION_CLASS: "Association_class sub-rule",
258     TEST_ASSOCIATION: "Test_association sub-rule"
259 },
260 COMBINED_RULE_ELEM: {
261     DEFINE_CLASS: {
262         rule_type: "Class Definition sub-rule",
263         name: "Class name",
264         exact_match: false,
265         abstract: false,
266         interface: false,
267         active: true,
268     },
269     METHOD: {
270         rule_type: "Method sub-rule",
271         name: "Method name",
272         class: "class name",
273         exact_match: false,
274         return_type: "void",
275         visibility: "public",
276         arguments: "",
277         active: true,
278         is_static: false,
279     },
280     ATTRIBUTE: {
281         rule_type: "Attribute sub-rule",
282         name: "Attribute name",
283         class: "class name",
284         exact_match: false,
285         type: "string",
286         visibility: "public",
287         active: true,
288         is_static: false,
289     },
290     GENERALIZATION: {
291         rule_type: "Generalization sub-rule",
292         active: true,
293         class_child: "Child",
294         class_parent: "Parent",
295         exact_match: false,
296         type: "inheritance" // implementation
297     },
```

```
298     AGGREGATION: {
299         rule_type: "Aggregation sub-rule",
300         active: true,
301         class_aggregate: "Aggregate",
302         class_element: "Element",
303         exact_match: false,
304         element_multiplicity: "*",
305     },
306     COMPOSITION: {
307         rule_type: "Composition sub-rule",
308         active: true,
309         class_composite: "Composite",
310         class_element: "Element",
311         exact_match: false,
312         element_multiplicity: "*",
313     },
314     SIMPLE_ASSOCIATION:{
315         rule_type: "Simple_association sub-rule",
316         active: true,
317         class_A: "Class A",
318         class_B: "Class B",
319         exact_match: false,
320         A_multiplicity: "1",
321         B_multiplicity: "1"
322     },
323     ASSOCIATION_CLASS:{
324         rule_type: "Association_class sub-rule",
325         active: true,
326         class_A: "Class A",
327         class_B: "Class B",
328         class_C: "Association Class",
329         exact_match: false,
330     },
331     TEST_ASSOCIATION: {
332         rule_type: "Test_association sub-rule",
333         active: true,
334         class_A: "Class A",
335         class_B: "Class B",
336         exact_match: false,
337     }
338 }
339 }
```

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["p"] = 0
93      result["mdp"] = 0
94      result["adp"] = 0
95      result["cID"] = -1
96
97      result.classMatch =
98          /!class/.test(input);
99      result.ignoreClass =
100         /!ignoreClass/.test(input);
101
102     const attrMatch =
103         /!attr\((.*?)\)/.exec(input);
104     if (attrMatch) {
105         const attrValue = attrMatch[1].trim();
106         result.attr = attrValue
107             === '*' ? '*' :
108             attrValue.split(',').map(Number);
109     } else
110         result.attr = [];
111
112     const methodMatch = /!method\((.*?)\)/.exec(input);
113     if (methodMatch) {
114         const methodValue = methodMatch[1].trim();
115         result.method = methodValue
116             === '*' ? '*' :
117                methodValue.split(',').map(Number);
118     } else
119         result.method = [];
120
121     const combineMatch =
122         /combineID=(\d+)/.exec(input);
123     if (combineMatch) {
124         result.cID = parseInt(combineMatch[1]);
125     } else
126         result.cID = -1;
127
```

```
128
129     const regex = /(\d+)
130         (point|points) for (\w+)/g;
131     let match;
132     while ((match =
133         regex.exec(input)) !== null) {
134         const points = parseInt(match[1]);
135         const name = match[3];
136         if (name === "rule")
137             result.p = points;
138         else if (name === "method"
139             || name === "methods")
140             result.mdp = points;
141         else if (name === "attribute"
142             || name === "attributes")
143             result.adp = points;
144     }
145     return result;
146 },
```

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