Konzeption und Entwicklung eines Expertenmoduls für den Bereich des logischen Entwurfs im Rahmen des intelligenten tutoriellen Systems „eTutor“

Diplomarbeit

Zur Erlangung des akademischen Grades
„Magister der Sozial- und Wirtschaftswissenschaften“
(Mag.rer.soc.oec.)
In der Studienrichtung Witschaftsinformatik

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik-
Data & Knowledge Engineering

Betreuer: o.Univ.-Prof. Dr. Michael Schrefl
Mitbetreuender Assistent: Mag. Christian Eichinger

Verfasst von: Sabine Caroline Roth
Linz, im August 2008

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Diplomarbeit wurde von mir weder im Inland, noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt.

Linz, _____ _____

Ort, Datum Unterschrift

Kurzfassung

E-Learning Systeme werden an Universitäten zur Erhöhung von Effizienz und Effektivität eingesetzt. Es kann unter anderem zur praktischen Vertiefung theoretisch erlernter Inhalte eingesetzt werden. Das am Institut für Data & Knowledge Engineering der Johannes Kepler Universität entwickelte System eTutor stellt eines dieser Trainingssysteme dar. Dieses ermöglicht den Studenten die interaktive Ausarbeitung von Übungen des am Institut gelehrten Themenbereichs. Zusätzlich wird der Lehrende bei der Korrektur der entsprechenden Lösungen der Aufgaben entlastet, die Auswertung erfolgt im Idealfall vollautomatisch. Allerdings steht diese automatische Korrektur derzeit nur bei eingeschränkten Themenbereichen der Lehrveranstaltung zur Verfügung. Eines der derzeit offenen Gebiete ist der logische Datenbankentwurf.

Im Zuge der vorliegenden Arbeit wird ein Modul zur Erweiterung des bestehenden Systems im Bereich des logischen Entwurfs konzipiert und ein entsprechender Prototyp entwickelt. Das Modul unterstützt die Erstellung von Aufgaben, die Abgabe und Analyse der entsprechenden Lösungen sowie deren Bewertung und die entsprechende Vermittlung von Feedback. Bei der Umsetzung des Systems wird ein wissensbasierter Ansatz verfolgt um einen bestmöglichen Lernerfolg zu gewährleisten. Aus diesem Grund ist es notwendig die Semantik der Domäne des logischen Entwurfs zu erfassen. Im konkreten Fall bedeutet dies, dass die Abbildung von konzeptuellen UML-Diagrammen in relationale Datenstrukturen analysiert und erfasst werden müssen. Die daraus resultierenden Abbildungsvarianten müssen schließlich in der Bewertung und bei der Kommunikation mit dem Lernenden berücksichtigt werden. Anhand der Analyse dieser Grundlagen wird ein Konzept zur Auswertung typischer Aufgaben des Datenbankentwurfs erstellt. Zur Evaluierung des Konzeptes wird ein entsprechender Prototyp implementiert. Dieser dient vor allem der Überprüfung der Konzeption und stellt kein fertiges Tool dar.

Abstract

At Universities e-Leanring is used for increasing efficiency and effectiveness. Among other things it may be used to practise theoretically learned subjects. eTutor, which was implemented at the Institute for Data & Knowledge Engineering at the Johannes Kepler University, is one of these training systems. To the student eTutor offers the interactive drawing up of exercises concerning the subjects of the institute. Furthermore the instructor is supported in analyzing the solutions what will be done completely automatic in ideal case. At the moment the automatic analyze is just supported for some of the subjects of the courses. One of the not treated subjects concerns the logical database design.

This Master Thesis designs an extension to the existing version of eTutor and develops a corresponding prototype. The module supports the process of creating exercises, handing in and analyzing of solutions as well as grading and giving necessary Feedback. The development of the system follows a knowledge based approach to guarantee an efficient learning. Because of this it's necessary to collect the semantics of logical design. Within this context this means to analyze the mapping of UML-diagrams in relational structures. The resulting possibilities of mapping have to be considered in grading and in communication with the student. Following the analysis of basics, a concept to evaluate typical exercises of database design will be developed. To proof this concept a corresponding prototype will be implemented. In particular this serves to verify the concept; it won't represent a finished tool.

Inhaltsverzeichnis

1. Einleitung

Durch die raschen Änderungen in der Wirtschaft und Wissenschaft werden zunehmend neue Anforderungen an das Personal gestellt. Die Lernphase eines Menschen ist nicht mehr länger auf die Zeit der schulischen bzw. universitären Ausbildung beschränkt. In zahlreichen Bereichen der Wirtschaft besteht die Notwendigkeit sich laufend auf neue Gegebenheiten und Technologien einzustellen und somit auch neues Wissen zu erwerben. Neue wissenschaftliche Erkenntnisse sollen so schnell und so kostengünstig wie möglich eingesetzt werden, da sie innerhalb kürzester Zeit neuen Entwicklungen unterliegen können. [Ditt02, Wiep06]

Sowohl in der innerbetrieblichen Weiterbildung in Unternehmen als auch im schulischen bzw. universitären Bereich stößt man im Falle des klassischen Präsenzunterrichts auf diverse Probleme, zum Teil aufgrund der hohen Kosten des Unterrichts aber auch wegen zeitlicher, räumlicher und inhaltlicher Diskrepanzen zwischen Lehrendem und Lernenden. Auf diese Probleme wird später genauer eingegangen. Man versucht diese Probleme durch individuelle, zeitunabhängige Weiterbildungsmöglichkeiten wie elektronische Lernumgebungen zu umgehen. [Ditt02]

Im Zuge der technischen Entwicklung wurden sowohl in der Wirtschaft als auch an Universitäten verschiedene elektronische Systeme zur Unterstützung eingesetzt. In eingeschränkten Bereichen wird der Präsenzunterricht durch den Einsatz elektronischer Lernumgebungen sogar gänzlich überflüssig. Auch am Institut für Data & Knowledge Engineering wurde ein derartiges System, der eTutor, implementiert. Hierbei handelt es sich um ein erweiterbares „Intelligentes Tutorielles System“. Dieses versucht den Lehrenden mit seinen fachlichen Kompetenzen in Bezug auf die Auswertung und Beurteilung von Übungsabgaben zu unterstützen. Derzeit wurde bereits ein Kernsystem zur Erfüllung der administrativen Aufgaben entwickelt. Zusätzlich zu diesem Kern bestehen einzelne Lernmodule, welche jeweils ein abgegrenztes Stoffgebiet umfassen. Durch den Aufbau des Systems ist es möglich das derzeitige inhaltliche Angebot um weitere Module zu erweitern.

Generelles Ziel der vorliegenden Arbeit ist die Erweiterung des am Institut für Data & Knowledge Engineering entwickelten „Intelligenten“ E-Learningsystems eTutor um ein Modul zur Beurteilung, Bewertung und Korrektur sowie der Feedback-Vermittlung im Bereich des logischen Entwurfs.

Nachfolgend wird die Problemstellung konkretisiert, das heißt es wird kurz geschildert von welchen Gegebenheiten diese Arbeit ausgeht. Des Weiteren wird aufgezeigt welche Ziele verfolgt werden sollen und welche Anforderungen gestellt werden (siehe Kapitel 1.1). In Kapitel 1.2 wird ein Beispiel zum logischen Entwurf vorgestellt welches im Verlauf der Arbeit wiederholt zu Erklärungszwecken herangezogen wird. Im Anschluss wird in Kapitel 1.3 auf den Aufbau der Arbeit eingegangen.

1.1 Problemstellung

Im Bereich des logischen Entwurfs wird am Institut derzeit nur der herkömmliche Präsenzunterricht mit Ausgabe von Übungen auf Papier praktiziert. Daraus ergeben sich folgende Probleme, die durch die Entwicklung des neuen Moduls behoben werden sollen:

- Eine Präsenzveranstaltung bindet den Lehrenden und die Lernenden an einen bestimmten Ort und erfordert die Anwesenheit zu einer bestimmten Zeit.
- Durch die Ausgabe von papiergebundenen Aufgaben entstehen administrative Schwierigkeiten: Der Lehrende bzw. die Tutoren müssen bereits vor der Ausgabe der Übungen überprüfen, ob diese dem Studenten bereits bekannt sind. Bei der Anzahl der Studierenden führt dies zu einem hohen Vorbereitungsaufwand, da frühere Antritte zur Übung berücksichtigt werden müssen.
- In weiterer Folge hat die Ausarbeitung der Lösung durch den Lernenden ebenfalls auf Papier zu erfolgen. Sämtliche Abgaben haben zur gleichen Zeit zu erfolgen, da den Studierenden die gleichen Bedingungen vorgegeben werden müssen.
- Die Tutoren müssen die Auswertung manuell durchführen.

- Durch diesen aufwändigen Vorgang ist die Zeitspanne bis zum Erhalt des Feedbacks für den Lernenden relativ lang, wodurch der Lernerfolg beeinträchtigt wird.

Durch das zu konzipierende Expertenmodul sollen die eben genannten Probleme im Bereich des logischen Entwurfs behoben werden. Das bereits bestehende Kernsystem des eTutors erfüllt bereits die Anforderungen bezüglich Zeit- und Ortsunabhängigkeit. Auch die Unterstützung bei der Auswahl und Verwaltung von Aufgaben und Ergebnissen wird bereits umgesetzt. Generell ist zu sagen, dass die administrativen Lehraufgaben durch den bereits bestehenden Kern umgesetzt werden und somit nur die Integration des neu zu entwickelnden Expertenmoduls erfordern.

Das Modul zum logischen Entwurf muss somit folgende Aufgaben unterstützen:

- Übungen in diesem Bereich zur Verfügung stellen
- Die Korrektur der Aufgaben durchführen
- Die Bewertung gewährleisten
- Ein entsprechendes Feedback für den Lernenden generieren

Durch den Lehrenden erstellte Übungen werden dem Lernenden zur Lösung vorgelegt. Das System muss die Möglichkeit bieten, Aufgaben elektronisch abzugeben. Diese Abgabe wird als Grundlage für die Auswertung der Lösung durch das Modul herangezogen.

Der eTutor soll den Lehrenden sowohl bei Übungen, die zu Lernzwecken durchgeführt werden, als auch bei zur Lehrveranstaltungsbewertung relevanten Abgaben unterstützen. Um diese beiden Varianten zu unterstützen muss das Expertenmodul folgende Aufgaben unterstützen:

Auswertung der Studentenlösung

In jedem Fall muss das Expertenmodul die Auswertung der Studentenlösung durchführen. Hierfür ist es irrelevant, ob die Abgabe in die

Lehrveranstaltungsnote einbezogen wird oder die Übung als praktisches Lernen angesehen wird. Es muss festgestellt werden ob die umgesetzte Lösung korrekt ist. Zu beachten ist, dass die Lösung hier nicht als Ganzes richtig oder falsch sein muss. Für eine sinnvolle Bewertung und Feedbackerstellung ist es nötig einzelne Fehler innerhalb der Lösung aufzudecken. Fehler müssen im Zuge der Auswertung erkannt und für die spätere Weiterverarbeitung dokumentiert werden. Um eine korrekte Arbeitsweise zu gewährleisten müssen aber auch korrekte Umsetzungen erfasst werden.

Erstellung des Feedbacks

Eine generelle Anforderung an das eTutor System ist die Unterstützung des Lernenden im Lernprozess. Um ein optimales Lernergebnis zu erzielen, muss der Lernende verstehen, welche Fehler er macht. Das heißt das System muss ihm im Zuge der Auswertung der Übungen vermitteln, welche Fehler erkannt wurden, warum es sich dabei um Fehler handelt und wie diese gelöst werden können. Eine reine Benotung der Lösung ist hierfür nicht ausreichend. Aus diesem Grund ist es erforderlich, dass das zu entwickelnde Modul auch ein verbales bzw. schriftliches Feedback unterstützt. Dem Lernenden soll erklärt werden warum etwas als falsch bewertet wurde.

Bewertung der Lösung

Da das eTutor System zur automatischen Auswertung von beurteilungsrelevanten Übungen vorgesehen ist, müssen im Verlauf der Auswertung auch die zu vergebenden Punkte beachtet werden. Eine mögliche Punktevergabe soll bei der Erstellung der Aufgabe generiert werden. Eine zusätzliche Anforderung stellen Eingriffsmöglichkeiten für den Lehrenden dar. Dieser soll bei der Erstellung der Aufgabe die Möglichkeiten der Umsetzung bewerten bzw. bestimmte Varianten verbieten. Das System soll eine mögliche Punktevergabe vorgeben, welche aber angepasst werden kann, das heißt der Eingriff des Lehrenden in die Beurteilung soll ermöglicht werden.

Natürlich sind diese Aufgaben nicht ohne Schwierigkeiten zu bewältigen. Bei der Auswertung der Lösung müssen komplexe Konstrukte berücksichtigt werden und alternative Umsetzungsmöglichkeiten voneinander abgegrenzt werden. Zusätzlich ist zu beachten, dass korrekte Teile der Lösung als falsch erkannt werden könnten und somit eine fehlerhafte Auswertung nach sich ziehen. Ein weiteres Problem liegt darin, dass in manchen Fällen eine vollautomatische Verarbeitung nicht möglich ist. Das Expertenmodul sollte erkennen wann dies der Fall ist um einen Lehrenden bzw. Tutor zur Unterstützung heranziehen zu können.

Im Bereich der Feedbackerstellung liegt das primäre Problem darin, dass die während der Auswertung aufgedeckten Fehler verständlich kommuniziert werden müssen. Das heißt, der Student sollte nach Erhalt des Feedbacks verstehen können warum es sich um einen Fehler handelt und wie dieser behoben werden kann. Wenn eine exakte Erklärung nicht möglich ist, sollte zumindest ein Hinweis zur Lösung des Problems vermittelt werden. Wichtig ist vor allem, dass vor der Erstellung des Expertenmoduls die möglicherweise auftretenden Fehler bekannt sind und somit entsprechende Erklärungen formuliert werden können die bei Eintreten des spezifischen Fehlers kommuniziert werden.

Zur Bewertung sind generell zwei Varianten möglich. Einerseits können beim Vergleich der Studentenlösung mit der Angabe Punkteabzüge beim Erkennen von Fehlern erfolgen, andererseits können aber auch richtig entworfene Tabellen zur Vergabe von Punkten führen. Durch die automatische Beurteilung können in beiden Fällen Fehler durch eine falsche Erkennung erfolgen. Dies hat jedoch keinen Einfluss auf die Auswahl der Methode. Ob ein irrtümlich als falsch deklariertes Element zu Abzügen der Punkte führt oder bei Punktevergabe keine erhält ist irrelevant. Zu beachten ist in jedem Fall, dass eine bestimmte Obergrenze für die zu vergebenden Punkte existiert. Dadurch ist es erforderlich für die einzelnen Elemente festzulegen wie sie bei der Punktevergabe durch ihre Komponenten beeinflusst werden. Weiters müssen durch den Lehrenden getroffene Einschränkungen zusammen mit der Aufgabenstellung dokumentiert werden um bei der späteren automatischen Auswertung berücksichtigt werden zu können.

Im Zuge dieser Arbeit soll ein Konzept zur Bewältigung der eben genannten Aufgaben unter Berücksichtigung der entstehenden Probleme erstellt werden. Ziel ist es ein Modul zu entwickeln welches die automatische Auswertung von Übungen im Bereich des logischen Entwurfs bewältigt und gegebenenfalls den Eingriff der Lehrenden fordert bzw. ermöglicht. Das erstellte Expertenmodul soll den Lernenden durch die Übermittlung von Feedback im Lernprozess unterstützen. Neben dem lerntechnischen Aspekt muss auch die Bewertung von Übungen in Zusammenhang mit der Lehrveranstaltungsbeurteilung ermöglicht werden.

1.2 Beispiel zum logischen Entwurf

Für den logischen Entwurf am Institut für Data & Knowledge Engineering wird von einer konzeptuellen Darstellung in UML ausgegangen. Die Darstellung erfolgt in einem Klassendiagramm basierend auf Klassen ohne Operationen, da diese in der Datenbank selbst nicht abgebildet werden. In Abbildung 1 wird ein Beispiel eines solchen UML-Diagramms dargestellt. Im Zuge der Präsenzveranstaltung wird diese Übung derzeit mit zusätzlichen textuellen Informationen vorgelegt. Die genaue Bedeutung der verwendeten Diagramm-Elemente wird in Kapitel 3.3 erläutert.

Das hier abgebildete Beispiel zeigt einen Ausschnitt aus dem Universitätsbetrieb. Es werden Studenten, Lehrer und LVAs sowie deren Zusammenhänge dargestellt. Studienpläne werden durch eine eindeutige Nummer identifiziert. Sie beinhalten außerdem eine Studienbezeichnung und das Jahr in dem der Plan erstellt wurde. Studienpläne enthalten beliebig viele LVAs, diese existieren jedoch auch unabhängig von Studienplänen. LVAs werden durch eine Nummer gekennzeichnet, sie haben eine Bezeichnung und geben die Semesterstunden und ECTS an. Eine LVA kann für mehrere andere LVAs Voraussetzung sein, sie kann aber auch mehrere LVAs als Voraussetzung haben. LVAs finden an einem bestimmten Datum in einem bestimmten Hörsaal statt. In einem Hörsaal zu einem Datum kann nur eine LVA stattfinden. Hörsäle werden wiederum durch eine eindeutige Nummer identifiziert und enthalten eine bestimmte Anzahl an Sitzplätzen. Ein Datum hat eine ID und besteht aus Monat, Tag und Zeit. LVAs werden vom Personal abgehalten, wobei pro LVA eine Lehrperson vorgesehen ist. Diese werden durch die

Sozialversicherungsnummer identifiziert. Zusätzlich werden Name und Vorname angegeben sowie die Adresse und die jeweilige Stelle im Universitätsbetrieb. Sie werden unterteilt in Assistenten und Professoren. Assistenten werden durch ihr Fachgebiet näher beschrieben und jeweils einem Professor unterstellt. Professoren werden durch das Beginndatum ihrer Führungsposition näher definiert. Studenten werden wiederum durch die Matrikelnummer identifiziert, sie enthalten den Namen, den Vornamen und das Semester in dem sie das Studium begonnen haben. Sie können an verschiedenen LVAs teilnehmen wobei an jeder LVA mindestens ein Student beteiligt sein muss. Die Teilnahme an einer LVA kann in verschiedenen Semestern erfolgen. Studenten legen Prüfungen ab welche an die LVA-Teilnahme in Bezug auf den zu bewältigenden Stoff gekoppelt sind. Prüfungen werden durch die Prüfungsnummer identifiziert und sind nur in Zusammenhang mit einem bestimmten Studenten vorhanden. Sie enthalten außerdem die erreichte Note und das Datum an dem die Prüfung abgelegt wurde. Prüfungen werden von ein bis drei Lehrpersonen abgehalten. Studenten können neben dem Studium auch als Tutoren arbeiten bzw. die ÖH unterstützen. ÖH-Mitglieder werden durch eine ID identifiziert, sie haben eine Position und eine zu erfüllende Aufgabe. Tutoren sind ebenfalls durch eine ID zu identifizieren und für eine bestimmte Stundenzahl angestellt. Sie betreuen jeweils eine LVA wobei sie an dieser nicht selbst teilnehmen dürfen. Einer LVA können mehrere Tutoren zugeordnet werden. Studenten fertigen außerdem jeweils eine Diplomarbeit an wobei jeder Student ein eigenes Thema bearbeiten muss. Die Diplomarbeiten werden durch eine eindeutige Nummer identifiziert und von einem Professor betreut. Jeder Professor kann jedoch auch mehrere Diplomarbeiten betreuen.

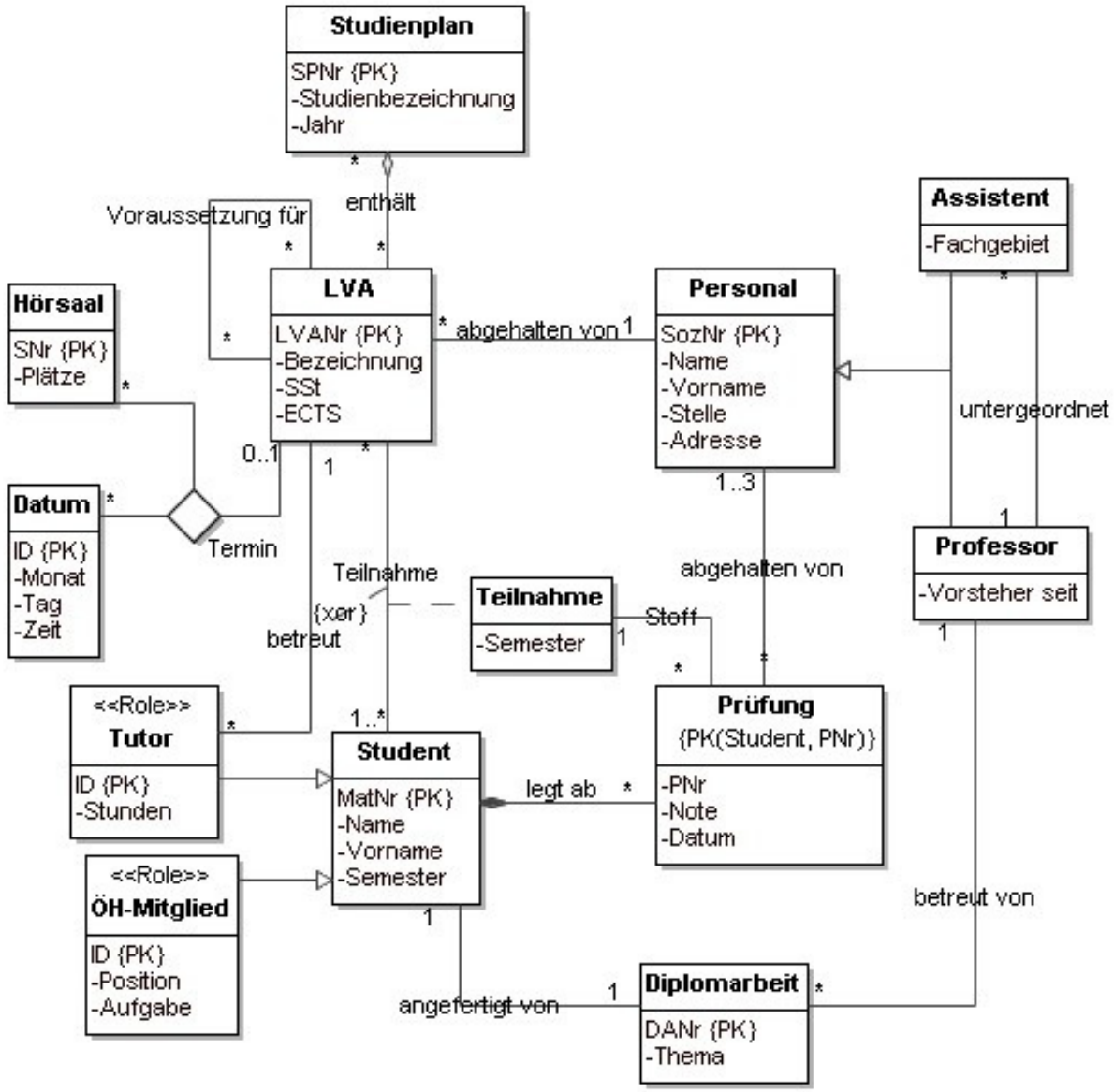
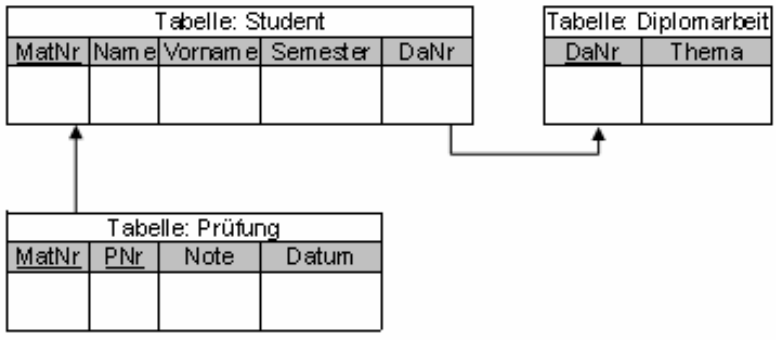


Abbildung 1: Beispiel UML Klassendiagramm

Das logische Modell basiert auf einer relationalen Datenbank, die Grundlagen dafür werden in Kapitel 3.2 genauer beschrieben. Aus dem vorgegebenen Klassendiagramm werden die entsprechenden Tabellen, genauer gesagt das dazugehörige SQL-DDL Skript, abgeleitet. Natürlich bestehen bei verschiedenen Mapping-Alternativen (siehe Kapitel 3.3) auch unterschiedliche Möglichkeiten der Umsetzung. Abbildung 2 zeigt einen Ausschnitt einer möglichen Umsetzung des UML-Diagramms aus Abbildung 1 in Tabellen und einen Teil des dafür erstellten SQL-DDL.




```
CREATE TABLE Diplomarbeit (
        daNr            NUMBER,
        thema           VARCHAR (50),
        PRIMARY KEY (daNr)
);

CREATE TABLE Student (
        matNr           NUMBER,
        name            VARCHAR (20),
        vorname         VARCHAR (20),
        semester        VARCHAR (10),
        PRIMARY KEY (matNr),
        FOREIGN KEY (daNr) REFERENCES Diplomarbeit (daNr)
);

CREATE TABLE Prüfung (
        pNr             NUMBER,
        note            NUMBER,
        datum           DATE,
        FOREIGN KEY (matNr) REFERENCES Student (matNr) ON DELETE CASCAD
        PRIMARY KEY (pNr, matNr)
);
```

Abbildung 2: Ausschnitt SQL DDL zum Beispiel aus Abbildung 2 [Date02]

Aufgabe des logischen Entwurfs ist es somit aus einem UML-Diagramm die entsprechenden relationalen Tabellen abzuleiten. Studenten haben im Zuge der Lehrveranstaltungen Übungen im Sinne des hier gezeigten Beispiels zu lösen.

Dieses Beispiel wird im Zuge der Arbeit immer wieder zu Erklärungszwecken herangezogen. Die einzelnen Elemente werden aus dem hier gezeigten Kontext herausgenommen und unabhängig von anderen Einflüssen betrachtet.

1.3 Aufbau der Arbeit

Wie bereits erläutert ist es das Ziel dieser Arbeit, ein eTutor Modul zur Beurteilung, Bewertung und Korrektur von Aufgabenstellungen im Bereich des logischen Entwurfs zu konzipieren. Die hierfür notwendigen Grundlagen werden in Kapitel 2 behandelt. Hierbei handelt es sich um eine Betrachtung von E-Learningsystemen (siehe Kapitel 2.1), wobei deren geschichtliche Entwicklung sowie der Einfluss von didaktischen Elementen auf die verschiedenen Systeme betrachtet werden. Weiterführend wird in Kapitel 2.2 kurz auf die spezielle Ausprägung, die „Intelligenten Tutoriellen Systeme“, eingegangen. Abschließend wird in Kapitel 2.3 das am Institut für Data & Knowledge Engineering entwickelte und eingesetzte Tool eTutor betrachtet.

In Kapitel 3 werden sämtliche dieses Modul betreffende Standards betrachtet. Das Ausgangsmodell für den logischen Entwurf wird in UML-Darstellung vorgelegt, diese Modellierungssprache wird in Kapitel 3.1 vorgestellt. Des Weiteren wird das Zielsystem, genauer gesagt die relationalen Datenbanken in Kapitel 3.2 erläutert. Da deren Datenstruktur sich weitgehend von derjenigen von UML-Diagrammen unterscheidet, wird anschließend die Überleitung der objektorientierten UML-Elemente in relationale Datenstrukturen behandelt (siehe Kapitel 3.3).

Von diesen Ausführungen ausgehend erfolgt in Kapitel 4 die Konzeption des zu entwickelnden Moduls. Zu Beginn werden die alternativen Umsetzungsvarianten sowie deren Auswahl für die konkrete Umsetzung betrachtet (siehe Kapitel 4.1). Darauf aufbauend wird ein konkretes Lösungssystem entworfen sowie diverse Überlegungen zur Auswertung bzw. zum Feedback bezüglich der Übung angestellt (siehe Kapitel 4.2).

In Kapitel 5 wird die Implementierung eines Prototyps des Moduls vorgestellt. Beginnend mit dem zur Erstellung der Aufgabe notwendigen Tool werden über die entsprechenden Bearbeitungsschritte vom Diagramm zum Zielsystem die nötigen Grundlagen für die Auswertung der Lösung gelegt. Die hierfür nötigen Schritte werden erläutert.

Abschließend werden in Kapitel 6 die erhaltenen Ergebnisse zusammengefasst. Es wird darauf hingewiesen welche Bereiche des logischen Entwurfs außer Acht gelassen wurden. Zusätzlich werden Möglichkeiten zur weiteren Entwicklung des Moduls aufgezeigt.

2 Grundlagen

Wie bereits erwähnt, gewinnen elektronische Lernsysteme in der betrieblichen und universitären Aus- und Weiterbildung zunehmend an Bedeutung. Da es sich bei eTutor ebenfalls um ein derartiges System handelt, werden hier die Grundlagen von E-Learning Systemen erläutert. Erst werden elektronische Lernsysteme im Allgemeinen sowie verschiedene Varianten derselben beschrieben (siehe Kapitel 2.1). Um in den engeren Bereich dieser Arbeit vorzudringen, werden in Kapitel 2.2 wissensbasierte Ausprägungen dieser Systeme, genauer gesagt „Intelligente Tutorielle Systeme“ (ITS), behandelt. Abschließend erfolgt eine Einführung in das eTutor System, welches als Grundlage für das zu entwerfende Modul herangezogen wird (siehe Kapitel 2.3).

2.1 E-Learning

Einleitend wird der Begriff E-Learning definiert und auch verschiedene synonym verwendete Begriffe angeführt. Ein geschichtlicher Rückblick in Kapitel 2.1.1 zeigt die Entwicklung des E-Learning. Des Weiteren wird auf die verschiedenen Merkmale des elektronischen Unterrichts sowie auf daraus resultierende Vor- und Nachteile aus didaktischer und administrativer Sicht eingegangen (siehe Kapitel 2.1.2). Abschließend werden die zu Grunde liegenden Lerntheorien sowie die jeweilige Umsetzung in E-Learning Systemen behandelt (siehe Kapitel 2.1.3).

E-Learning oder Electronic Learning umfasst ein weitläufiges Stoffgebiet. Da der Begriff oft synonym für unterschiedliche Ausprägungen innerhalb des Themenbereichs verwendet wird, unterscheiden sich die Definitionen in der Literatur meist voneinander. Alle Definitionen beinhalten den Einsatz von Computern zur Unterstützung des Lernprozesses. Ob nun Präsenzunterricht ergänzt oder der gesamte Unterricht in ausschließlicher Interaktion mit dem Computer durchgeführt wird ist bedeutungslos. [Ditt02, Wiep06, Bode90] Wiepcke betont zusätzlich, dass der Computer beim E-Learning nicht nur als Hilfsmittel eingesetzt wird, sondern den Lernprozess tatsächlich beeinflusst. Dieser ist abgeschlossen wenn gestellte

Aufgaben entweder richtig gelöst werden oder der Lernende zumindest versteht warum die Lösung nicht korrekt ist. [Wiep06]

In der Literatur werden an Stelle von E-Learning verschiedene englische und deutschsprachige Begriffe verwendet. Einige davon sind tatsächlich gleichbedeutend mit dem Begriff E-Learning, die meisten jedoch beziehen sich auf einen eingeschränkten Bereich des E-Learning bzw. betonen verschiedene Aspekte. Diese eigentlich unterschiedlichen Begriffe werden aber trotzdem oft als Synonyme verwendet. Der Begriff „Lernen mit Software“ kann als gleichbedeutend mit E-Learning angesehen werden. Im Englischsprachigen Raum wird als Synonym der Begriff „Computer Assisted Instruction“ verwendet. Aber es werden auch die verschiedenen Bestandteile des Begriffes Electronic Learning einzeln ersetzt. An Stelle von Learning bzw. Instruction wird im Englischen auch „Training“ verwendet, im deutschsprachigen Raum ist hauptsächlich der Begriff „Unterricht“ in Gebrauch. Um auf die Computerunterstützung einzugehen werden die Begriffe „Computer Aided“ oder „Computer Based“ eingesetzt. Analog werden im Deutschen „Rechnergestützt“, „Computergestützt“ und „Computerunterstützt“ verwendet. Entsprechend der Zusammensetzung der verschiedenen Begriffe werden unterschiedliche Abkürzungen wie CAI, CAL oder CBL verwendet. Um auf wissensbasierte Systeme einzugehen wird „Intelligent“ den restlichen Silben vorangestellt. Dies umfasst Systeme die Fakten und deren Bedeutung in bestimmten Situationen sowie Verknüpfungen zwischen Informationen, wie Analysen, Schlussfolgerungen oder Verallgemeinerungen, abbilden. Diese Darstellung erfolgt meist in Form von Wissensbasen. [Bode90, Wiep06]

In dieser Arbeit wird davon ausgegangen, dass E-Learning Systeme den Lehrenden bei der Vermittlung von Lernstoff ergänzen. Die Unterstützung durch das System eTutor besteht vor allem beim Trainieren der Kompetenzen zum Lösen von Aufgaben in den verschiedenen Themenbereichen des Data & Knowledge Engineering. Somit handelt es sich grundsätzlich um Computer Based Training bzw. um ein Trainingssystem. Zusätzlich werden auch Aufgaben wie die Zuteilung der Übungen oder die Administration der Bewertung bewältig. Unter E-Learning wird somit vor allem die praktische Anwendung des Lernstoffes aus der Präsenzveranstaltung

verstanden sowie administrative Aufgaben im Zuge der Übungsbereitstellung und –bewertung.

2.1.1 Entwicklung

Die Entwicklung des elektronischen Lernens hängt eng mit den allgemeinen technischen Fortschritten zusammen. Durch neue Entwicklungen elektronischer Medien wurden neue Anwendungsfelder eröffnet. Neben der technischen Entwicklung waren aber auch Neuerungen auf dem Gebiet der Lern-Psychologie für die Weiterentwicklung des elektronischen Lernens von Bedeutung. Dieses Zusammenspiel wird ansatzweise in Abbildung 3 gezeigt. Der obere Bereich der Abbildung geht auf die technischen Entwicklungen ein. Der Untere Bereich zeigt die didaktische Entwicklung im Zuge der Lerntheorien und deren Umsetzung in Lernsysteme. Das Zusammenwirken der Bereiche im zeitlichen Verlauf wird im Folgenden erläutert.

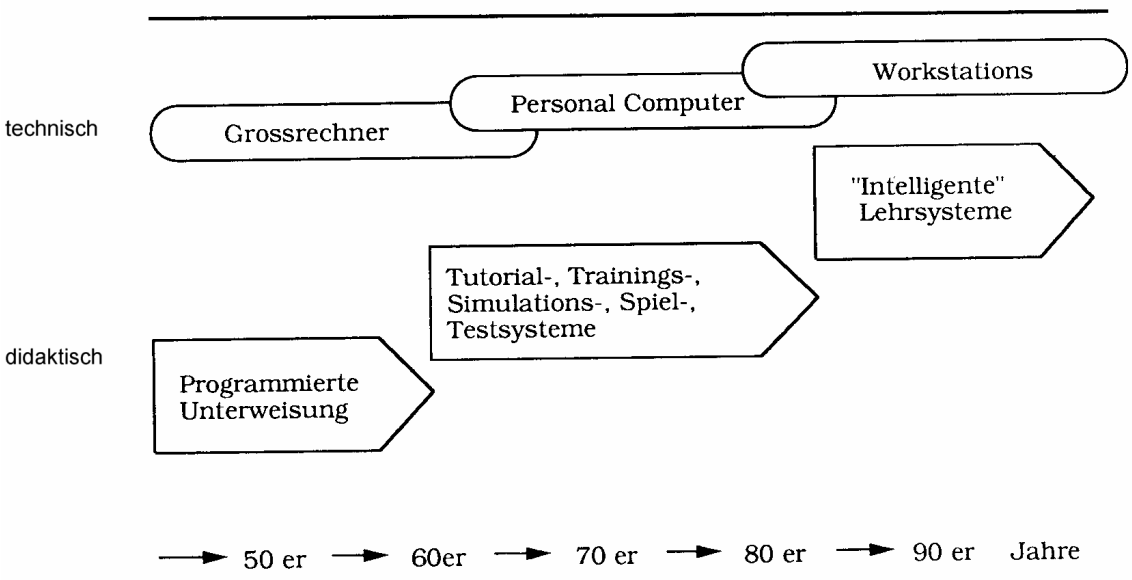


Abbildung 3: Entwicklung Computergestützte Ausbildung [in Anlehnung an Bode90, S 15]

Bereits Ende der 50er Jahre wurden an Universitäten Zentralrechner zur administrativen Unterstützung eingesetzt. Zur selben Zeit wurden die ersten Überlegungen über den Einsatz von EDV im Unterricht angestellt. Allerdings sah man sich zu diesem Zeitpunkt noch enormen Problemen gegenübergestellt: einerseits waren enorme EDV-Kenntnisse nötig um die Computer überhaupt

handhaben zu können, andererseits standen auch die hohen Kosten für Hard- und Software dem Einsatz im Unterricht im W
eg. [Bode90]

Diese Probleme wurden in den 70er Jahren durch den Fortschritt der technischen Entwicklung sowohl im Bereich der Hardware als auch im Bereich der Software gelöst. Dadurch wurde der Einsatz von Computern günstiger, außerdem war dadurch seitens der Anwender keine so umfangreiche Ausbildung nötig. Personal Computer wurden zusammen mit spezieller Lernsoftware hauptsächlich zur Durchführung von Aufgaben wie komplizierten Rechnungen und Simulationen eingesetzt. [Bode90]

Diese Software wurde entweder direkt am Computer oder auf CD's gespeichert. Durch Einsatz dieser Programme wird auch die Zusammenarbeit mehrerer Lernender ermöglicht. Weiters ist ein Angebot der Programme über Internet, das heißt durch Speicherung auf einem zentralen Server, möglich. Verschiedene Lernende greifen somit auf dasselbe Programm zu. [Ditt02]

Zu diesem Zeitpunkt folgte die inhaltliche Umsetzung der Lerntheorie des Behaviorismus. Der Mensch wird durch äußere Reize gesteuert. Die Umsetzung im Bereich des „Computergestützten Lernens" erfolgt durch Textpräsentation, der Lernende wird dazu angehalten einzelne Textteile auswendig zu lernen. Zur Erfolgskontrolle werden Tests durchgeführt. Kritiker dieser Theorie weisen darauf hin, dass Lernende durch diese Methode nicht dazu befähigt werden Probleme eigenständig zu lösen, da sie die Zusammenhänge zwischen den gelernten Teilen nicht vermittelt bekommen. [Ditt02, Wiep06]

Zur selben Zeit wurden Überlegungen angestellt, wie sie Kompetenzen von Lehrkräften durch Computer ersetzt bzw. ergänzt werden könnten. [Bode90] In Lernportalen werden neben den eigentlichen Lernprogrammen auch zusätzliche Funktionen dargeboten. Ergänzende Lernmedien wie Frequently Asked Questions (FAQ), Hilfedateien, etc. unterstützen den direkten Umgang mit dem System. Daneben werden oft auch Möglichkeiten geboten Experten zusätzlich zu den durch das System präsentierten Unterlagen zu kontaktieren, sei es über E-Mail, Foren oder Communities. [Ditt02]

Die Entwicklung wissensbasierter Systeme begann in den 80er Jahren. Durch Einsatz von Methoden der künstlichen Intelligenz wurden „Intelligente Lehrsysteme“ entwickelt. Hierfür wird anstelle von Informationen Wissen und dessen Bedeutung gespeichert. Außerdem soll sich das System dem Benutzer anpassen. Das letzte Ziel ist die Entwicklung in Richtung natürliche Sprache, das heißt eine annähernd menschliche Kommunikation soll realisiert werden. [Bode90]

Lerntheoretisch basieren diese Systeme zu Beginn auf der Theorie des Kognitivismus. Man geht davon aus, dass die Lernenden fähig sind Reize selbständig zu verarbeiten. Laut Wiepcke liegen die Probleme hierbei darin, dass diese Systeme dennoch textlastig sind und der nötige Entwicklungsaufwand nicht zu wesentlichen Verbesserungen des Lernerfolgs führt. [Wiep06 S. 47] Die Weiterentwicklung zum Konstruktivismus beinhaltet die Berücksichtung des individuellen Vorwissens. Darauf basierende Programme bieten den Lernenden Welten, in denen sie Wissen sammeln und anwenden können. [Wiep06]

2.1.2 Merkmale

E-Learning Systeme weisen verschiedene Charakteristika auf, die im klassischen Präsenzunterricht nicht umgesetzt werden können. Unter Berücksichtigung der gewünschten bzw. geforderten Merkmale von E-Learning Systemen erwartet man gewisse Vorteile durch den Einsatz elektronischer Lernmedien. Sie haben sowohl für den Lehrenden als auch für den Lernenden gewisse Auswirkungen. Die Merkmale des elektronischen Lernens werden im Folgenden behandelt. Im Anschluss an diese Ausführungen werden die daraus resultierenden Auswirkungen auf die Qualität des Unterrichts betrachtet.

Reichhaltigkeit

Elektronisch gestützter Unterricht erlaubt einen umfangreicheren Einsatz von Lernmöglichkeiten als herkömmlicher Präsenzunterricht. Sowohl im Bereich der Darstellungsmöglichkeiten als auch beim Methodeneinsatz kann aus einem größeren Pool an Varianten geschöpft werden. Neben den unterschiedlichen Möglichkeiten zur

Stoffvermittlung bieten elektronische Lernsysteme auch verschiedene Mittel zur Leistungsüberwachung. Zusätzlich zu den klassischen Leistungsüberprüfungen wie Tests und Klausuren können durch Beobachtung des Lernverhaltens des Studierenden Rückschlüsse auf dessen Erfolg gezogen werden. [Wiep06]

Interaktivität

Computergestützte Lernsysteme erlauben dem Benutzer in gewissem Maße auf den Unterricht Einfluss zu nehmen. Dadurch kann der Lernende zum Teil bestimmen wann welcher Stoff abgearbeitet wird und wie viel Zeit er dafür aufwendet. Auch Wiederholungen einzelner Themenbereiche werden dadurch ermöglicht. Außerdem kann unter Umständen auf die Darstellungsform Einfluss genommen werden. [Wiep06, Bode90]

Zeit- und Ortsunabhängigkeit

Durch den Einsatz elektronischer Lernmedien sind Studierende nicht länger an Stundenpläne gebunden. Es wird ihnen ermöglicht selbst zu bestimmen wann sie lernen und wie viel Zeit sie für den Stoff benötigen. Außerdem besteht für Studierende die Möglichkeit selbst zu entscheiden wo sie lernen. Einzige Voraussetzung ist die Verfügbarkeit der entsprechenden Medien. Dies sind zum einen Computer aber auch die entsprechenden Datenträger, das heißt die CD mit dem Programm bzw. der Zugang zu Servern. [Wiep06, Bode90]

Adaptivität

Durch das „Computergestützte Lernen“ und die damit verbundene Unabhängigkeit von Lehrenden wird eine individuelle Anpassung des Studienverlaufs an den Studierenden weitgehend ermöglicht. Der Stoff kann sowohl inhaltlich als auch lerntechnisch an den Studierenden angepasst werden. Umfang, Darstellung sowie Hilfen und Rückmeldungen zum Stoff werden individualisiert. [Wiep06, Bode90, Götz91]

Wirtschaftlichkeit

Elektronische Lernmedien führen vor allem zu einer Reduktion der Unterrichtskosten. Die Lehrenden müssen nicht bereitgestellt werden, die Organisation von Präsenzveranstaltungen ist nicht notwendig, somit müssen auch keine Räume reserviert werden. Auf Seite der Lernenden entfallen Anfahrtskosten und mögliche Unterbringungskosten. Außerdem wird durch die gegebene Unabhängigkeit die Ansprache eines größeren Publikums ermöglicht. Der Kostenersparnis in diesem Bereich stehen jedoch die Entwicklungskosten gegenüber. Dies führt zu der Möglichkeit, dass eine höhere Rentabilität auf Kosten der Qualität des Lernprogramms realisiert wird. [Wiep06, Bode90, Götz91]

Resultierende Vor- und Nachteile

Sämtliche Merkmale bergen die Möglichkeit die Effizienz des Unterrichts gegenüber Präsenzveranstaltungen zu steigern, andererseits treten unter Umständen aber auch gewisse Gefahren ans Licht. Die durch die Idee des E-Learning entstehenden Chancen und Risiken werden im Folgenden in Bezug auf die verschiedenen Merkmale erläutert.

Die Reichhaltigkeit des Angebotes kann das Interesse der Lehrenden wecken, da nicht sämtliche Informationen in der gleichen Weise präsentiert werden und der Lernende somit nicht in einen eintönigen Trott verfällt. Andererseits führt das vielfältige Angebot möglicherweise auch dazu, dass der Lernende den Überblick verliert und somit überfordert wird. [Wiep06]

Für den Lernenden besteht die Möglichkeit den Verlauf des Lernprozesses durch die Interaktivität mit dem System zu beeinflussen, somit kann er selbst entscheiden was er wann lernen will und dadurch die eigenen Interessen besser berücksichtigen. Dadurch soll nach Meinung einiger Psychologen der Lernerfolg gesteigert werden. Allerdings setzt dies die Eigeninitiative des Lernenden voraus, da er selbst entscheiden muss was er lernt. Es besteht die Gefahr, dass der Lernende angesichts

der Auswahlmöglichkeiten in eine gewisse Hilflosigkeit verfällt und somit der erwartete Lernerfolg ausbleibt. [Bode90, Wiep06]

Die Vorteile der Zeitunabhängigkeit sind eindeutig: der Lernende kann sich dem Stoff widmen wann immer er dazu die nötige Zeit findet. Demgegenüber steht das Problem, dass sich der Lernende ohne vorgegebenen Zeitplan nicht dazu motivieren kann den Lernprozess zu starten bzw. durchzuführen. Die Unabhängigkeit von Präsenzveranstaltungen führt dazu, dass der Lernende ohne die Kommentare und Blicke anderer arbeiten kann und somit die durch das soziale Umfeld verursachten Unsicherheiten wie etwa Versagensängste überwinden kann. Andererseits führt die Loslösung von Präsenzveranstaltungen unter Umständen zur vollständigen Isolation von anderen Personen. [Bode90, Wiep06]

Es gibt zwei verschiedene Möglichkeiten die Adaptivität von E-Learningsystemen zu gewährleisten. Zum einen kann die Anpassung durch das System selbst erfolgen, zum anderen können dem Lernenden Möglichkeiten zur individuellen Anpassung zur Verfügung gestellt werden. Beide Varianten bergen gewisse Vor- und Nachteile in sich. Die Anpassungsmöglichkeit bietet dem Lernenden in beiden Fällen individuellen Unterricht, der an die speziellen Anforderungen angepasst ist. Allerdings liegen diesem Idealbild gewisse Bedingungen zu Grunde. Durch das System angepasster Unterricht setzt Mindestkenntnisse beim Lernenden voraus. Das heißt der Lernende muss möglicherweise auf einem zu hohen Niveau beginnen. Bietet das System jedoch die Möglichkeit Anpassungen selbst durchzuführen kann der Lernende durch die Variantenvielfalt überfordert werden. [Bode90, Götz91, Wiep06]

Wie aus den eben durchgeführten Beschreibungen ersichtlich, bergen sämtliche Merkmale elektronischer Lernsysteme sowohl Vor- als auch Nachteile in sich. In Tabelle 1 werden die Merkmale und ihre Auswirkungen zusammengefasst. Zu erkennen ist, dass mit E-Learning durchaus positive Ergebnisse erzielt werden können. Einzige Voraussetzung ist eine entsprechende Persönlichkeit des Lernenden bzw. ausreichende Unterstützung durch das System um die Überforderung des Lernenden zu verhindern. In diesem Sinne muss vor allem darauf geachtet werden, dass die Qualität des Systems nicht unter dem Ziel der

Kostenersparnis leidet. Das didaktische Angebot des Systems muss unter Beachtung der verschiedenen Gefahren erstellt werden.

Eigenschaften	Merkmale	Chancen	Risiken
Reichhaltigkeit	• Ermöglichung vielfältiger Darstellungsformen, Navigationsmöglichkeiten, Hilfe-Systeme, Lehrmethoden, Animationen und Bilder	Motivation des Lernenden und Anschaulichkeit	Hilflosigkeit
Interaktivität	• Durch die Gestaltung der Inhalte und Zeitdauer sind Lernende aktiv am Lernprozess beteiligt	Aktives und selbstgesteuertes Lernen	Hilflosigkeit
Zeit- und Ortsunabhängigkeit	• Unmittelbarer und schneller Zugang des Lernstoffes ohne Voranmeldung und lange Wege • Flexible Gestaltung der Lernzeiten sowie des Lerntempos	Just in time Lernen; Learning on demand	Soziale Isolation
Adaptivität	• Anpassung von Stoff und Tiefe an Vorkenntnisse, Erfahrungen und Motive, Interessen, Lerntempo, Lerndauer und Schwierigkeitsniveau der Lernenden • Nutzung differenzierter Hilfen • Speziell auf Lernende zugeschnittene Rückmeldungen • Möglichkeit einer individuellen Erfolgskontrolle	Individuelles und ökonomisches Lernen	Soziale Isolation und Hilflosigkeit
Globalität und Wirtschaftlichkeit	• Bereitstellung der Information zu jeder Zeit an jedem Ort • Neue Inhalte können in kürzeren Zeiträumen aktualisiert und bereitgestellt werden • Es entfallen Reise-, Unterbringungs- und Seminarstättekosten sowie die des Lehrpersonals	Zeit- und Kostenersparnis	Hohe Entwicklungskosten

Tabelle 1: Übersicht: Merkmale des E-Learning und damit verbundene Chancen und Risiken [in Anlehnung an Wiep06 S. 64]

Ein Großteil der hier erwähnten Merkmale wird bereits durch das Kernsystem des eTutors umgesetzt. Da es sich hierbei um ein elektronisches Lernsystem handelt ist die Orts- und Zeitunabhängigkeit bereits gegeben. Sämtliche administrativen Angelegenheiten werden bereits geregelt. Für das hier zu entwickelnde Modul ist vor allem die Berücksichtigung der Adaptivität und der Interaktivität von Bedeutung. Der Lernende löst eine Aufgabe und wird mit Hilfe des Systems auf den richtigen Weg

geführt, das heißt durch Feedback wird ihm vermittelt in welchen Bereichen er sich noch verbessern kann bzw. welche Aspekte er berücksichtigen muss.

2.1.3 Lerntheorien

Unter Lernen wird in der Literatur die dauerhafte Veränderung von Verhaltenspotentialen verstanden. Das heißt die betreffende Person weist nach dem Lernprozess Fähigkeiten und Fertigkeiten auf, die ihr zuvor nicht bekannt waren. [Schw93, Wiep06] Um Wissen erfolgreich vermitteln zu können, sind Kenntnisse über die Vorgänge beim Lernprozess, die diese Veränderungen hervorrufen, von Bedeutung. Im Laufe der Zeit wurden diese sowohl in psychologischer als auch medizinischer Hinsicht erforscht. Dennoch kann man davon ausgehen, dass die tatsächlichen Vorgänge weitgehend unbekannt sind. Die bisherigen Erkenntnisse werden jedoch in unterschiedlichen Lerntheorien zusammengefasst, wobei jede dieser Theorien andere Schwerpunkte setzt bzw. andere Sichtweisen vertritt.

Die Entwicklung von E-Learning Systemen und der künstlichen Intelligenz steht mit der Erforschung des Lernprozesses in Verbindung. Sämtliche Systeme basieren auf der Umsetzung von Lerntheorien, es kann davon ausgegangen werden, dass der menschliche Lehrende immer mehr durch den Computer ersetzt werden kann je genauer die Abläufe des Lernens erforscht werden und somit in den Systemen berücksichtigt werden können. Davon ausgehend werden hier die verschiedenen Theorien betrachtet. Obwohl diverse Ausprägungen bestehen, können die verschiedenen Theorien in drei Hauptgruppen zusammengefasst werden. Hierbei handelt es sich um den Behaviorismus, den Kognitivismus und den Konstruktivismus. Im Folgenden werden diese Lerntheorien erklärt und ihre Berücksichtigung in E-Learningsystemen betrachtet.

2.1.3.1 Behaviorismus

Die Entwickler dieser Theorie gehen davon aus, dass der Mensch ein von Außen gesteuertes passives Wesen ist. Reaktionen werden durch zugeführte Reize ausgelöst. [Bode90, Ditt02, Wiep06] Die Vorgänge im Gehirn spielen eine

untergeordnete Rolle und werden daher außer Acht gelassen und als „Black Box“ betrachtet. [Ditt02, Wiep06] Lernen erfolgt im Zuge einer Konditionierung, das heißt durch die Verbindung von Reiz und Reaktion wird der Mensch dazu angehalten ein bestimmtes Verhalten weiter zu verfolgen. Untersucht wird ob ein bestimmter Input das gewünschte Verhalten auslöst bzw. bestärkt. [Bode90, Ditt02, Wiep06]

Innerhalb des Behaviorismus wurden verschiedene Ausprägungen entwickelt. Die klassische Konditionierung nach Pawlow arbeitet mit bestehenden Reiz-Reaktionsmustern. Im Zuge des Lernprozesses wird ein die Reaktion zuverlässig auslösender Reiz wiederholt mit einem anderen neutralen Reiz in Verbindung gebracht und immer mehr durch diesen ersetzt. Ziel ist, dass der neue Reiz alleine dieselbe Reaktion auslöst wie der ursprüngliche Reiz. Die instrumentelle Konditionierung nach Thorndike hingegen geht davon aus, dass Handlungen nicht durch Reize ausgelöst werden sondern zufällig auftreten, das heißt der Mensch experimentiert mit seinem Verhalten. Dies bedeutet gelernt wird durch das Versuch-Irrtum-Prinzip, indem so lange Handlungen ausprobiert werden bis das gewünschte Ergebnis erreicht wird. [Wiep06, Schw93] Im Zuge der operanten Konditionierung von Skinner werden die beiden anderen Theorien verknüpft. Es wird davon ausgegangen, dass Handlungen sowohl durch Reize ausgelöst werden als auch zufällig auftreten können. Sich auf einen der beiden Aspekte zu versteifen betrachtet er als einschränkend. Die Reaktionen auf spontan auftretende Handlungen werden mit diesen in Verbindung gesetzt. Wenn eine positive Reaktion erfolgt wird das Verhalten gefördert und tritt wiederholt auf, eine negative Reaktion führt hingegen zum Vermeiden der Handlung. [Ditt02, Wiep06]

Einsatz in E-Learning Systemen

Innerhalb des Behaviorismus folgen E-Learning Systeme vor allem der Konditionierung durch Verstärkung der Handlungen, das heißt Belohnung bzw. Strafe stehen im Mittelpunkt. In diesem Sinne liegt der Schwerpunkt dieser Systeme auf dem Feedback an den Lernenden und somit auf der Erfolgskontrolle. Der Lernende wird durch ein positives Ergebnis, beispielsweise eine gute Note, belohnt und durch die Rückmeldung von Fehlern bestraft. Das bedeutet gelernt wird um den unangenehmen Folgen einer schlechten Beurteilung auszuweichen. [Bode90, Ditt02]

Das Problem dieser Systeme liegt darin, dass die Lehrinhalte für die Erfolgskontrolle in abgegrenzte Stoffgebiete unterteilt werden und somit die bestehenden Zusammenhänge nicht vermittelt werden. Zusätzlich besteht die Gefahr, dass der Lernende den Stoff nur auswendig lernt, da dies ebenfalls zu einem positiven Ergebnis führt. Das somit fehlende Verständnis für den eingeprägten Stoff kann Probleme bei der realen Anwendung des vermittelten Wissens hervorrufen. [Wiep06] Im Folgenden wird der Systemtyp welcher der Behavioristischen Theorie folgt vorgestellt:

Trainingssysteme werden auch als Übungsprogramme bzw. als Drill and Practice Systeme bezeichnet. Der Benutzer hat in einer Präsenzveranstaltung oder durch ein anderes System ein gewisses Vorwissen erworben. Das Trainingssystem hilft das Wissen zu festigen. Vom System gestellte Aufgaben müssen durch den Benutzer gelöst werden, wenn dieser dazu nicht in der Lage ist wird eine Musterlösung präsentiert. [Bode90, Wiep06]

2.1.3.2 Kognitivismus

Im Gegensatz zum Behaviorismus werden hier innere Denkvorgänge während dem Lernprozess beachtet, die Verarbeitung von Informationen durch den Lernenden wird berücksichtigt. Dem Menschen wird die Fähigkeit Reize aktiv zu verarbeiten zuerkannt. Das heißt Menschen sind fähig Informationen mit dem bereits vorhandenen Wissen zu verknüpfen und Zusammenhänge festzustellen. Im Zuge des Lernprozesses gehen sie die diversen Möglichkeiten auf Gedankenbasis durch und entscheiden so welche Handlungen zielführend sind. [Bode90, Ditt02, Wiep06] Außerdem können sie die Bedeutung von Wissen unter verschiedenen Bedingungen erkennen. Der daraus resultierende Nachteil liegt darin, dass Lernende nicht unbedingt von außen gesteuert werden können. [Wiep06]

Einsatz in E-Learning Systemen

Da die kognitive Lerntheorie die Denkvorgänge des Lernenden stärker berücksichtigt, erfolgt hier eine andere Schwerpunktsetzung als bei behavioristisch geprägten Lernsystemen. Da der Lernende nicht durch Reize gesteuert werden

kann, tritt die Informationsdarbietung in den Mittelpunkt. Auf dem Kognitivismus basierende Lernsysteme unterstützen den Lernenden zusätzlich bei der Erkennung von Zusammenhängen. Sie versuchen im Sinne eines menschlichen Betreuers auf den Lernenden einzugehen. [Ditt02] Probleme liegen hierbei in der Entwicklungsphase der Systeme. Es ist ein hoher Aufwand notwendig um diese Systeme an die verschiedenen Lernenden anzupassen. Aus diesem Grund besteht jedoch auch die Gefahr, dass die Qualität der Lernumgebung außer Acht gelassen wird um das System rascher erstellen zu können. [Wiep06] Dem Kognitivismus folgende E-Learningsysteme werden hier vorgestellt:

Hilfesysteme

Diese Art von Systemen hilft dem Benutzer die Software zu verstehen und die Benutzung zu erleichtern. Die Hilfedateien sind mit den Teilen der Software verlinkt. Es gibt verschiedene Möglichkeiten diese Dateien aufzurufen: zum einen kann dies durch den Benutzer erfolgen, wenn dieser nicht mehr weiter weiß, zum anderen ist dies auch automatisch möglich. [Bode90]

Hypertextsysteme

Wie in Büchern werden hier Verweise von Teilen des Lernstoffes zu verwandten bzw. erklärenden Stoffgebieten verwendet. Allerdings hat hier der Benutzer die Möglichkeit durch betätigen des Links automatisch zu der anderen Textstelle zu gelangen an statt selbst danach suchen zu müssen. Diese Systeme stellen eigentlich keine eigenständige Lernsoftware dar sondern werden in anderen Systemen zur Unterstützung herangezogen. [Wiep06]

Lernergesteuerte Systeme

Der Lernende entscheidet selbständig welche Lerneinheiten er bearbeiten will. Der Aufbau der einzelnen Lerneinheiten wird jedoch durch das System bestimmt. Dies kann zu einem schlechten Aufbau führen, da der Lernende nicht weiß wann welche Informationen wichtig sind. [Bode90]

Tutorielle Systeme

Informationen werden durch das System einem bestimmten Ablauf folgend präsentiert. Dieser Weg wird durch Tests zu Beginn und zwischendurch festgelegt. Schwierigkeiten entstehen hierbei durch die Verwendung natürlicher Sprache. Intelligente Systeme stellen sich auf den Benutzer ein, indem sie dessen Verhalten beobachten. Diese Systeme beinhalten auch Wissen über Lehrmethoden. [Bode90, Wiep06]

2.1.3.3 Konstruktivismus

Diese Theorie stellt eine Weiterentwicklung des Kognitivismus dar. Hier steht die eigenständige Konstruktion von Wissen durch den Lernenden im Mittelpunkt während im Kognitivismus Lösungen für gestellte Probleme gesucht bzw. zugeführte Informationen verarbeitet werden. Lernen wird hier als Kombination von vorhandenem Wissen mit neuen bzw. bestehenden Informationen verstanden. Der Lehrende kann in den Lernprozess maximal durch lenkende Hinweise eingreifen, wobei deren Zahl im Laufe des Lernprozesses abnimmt. Das heißt der Lernende erlangt sein Wissen weitgehend eigenverantwortlich. Dem Lehrenden kommt jedoch die Rolle des Motivators zu. [Ditt02, Wiep06]

Einsatz in E-Learning Systemen

Im Sinne des Konstruktivismus wird dem Lernenden die Möglichkeit geboten eigene Erfahrungen zu sammeln und unvorhergesehene Situationen selbständig zu bewältigen. Als Input wird den Lernenden teilweise der Lernstoff dargeboten, diesen sollen sie in weiterer Folge in den dargebotenen Situationen anwenden. Die Realitätsnähe dieser Systeme kann den Lernerfolg sowohl beschleunigen als auch behindern. Durch die reale Situation wird die Problemlösungsfähigkeit des Lernenden gefördert, er kann sein gesamtes Wissen unter diesen Bedingungen erproben. Andererseits werden dadurch Situationen komplex und verlangen daher relativ gute Kenntnisse im Umgang mit derartigen Systemen. Außerdem kann durch

falsche Lösungswege der Zeitaufwand für das Erlernen bestimmter Sachverhalte steigen. [Wiep06] Systeme die das Lernen durch eigene Erfahrungen ermöglichen werden im Folgenden vorgestellt:

Simulationssysteme

Ursprünglich wurden diese für den Einsatz in der Forschung gedacht, um kostspielige bzw. teure reale Versuche zu ersetzen. Diese Systeme bieten ein Modell eines realen Objekts. Mit diesem Modell kann der Benutzer experimentieren, teilweise ist er auch Handelnder innerhalb des Modells. Es ist jedoch nicht sichergestellt, dass das erworbene Wissen auch in der realen Welt angewandt werden kann. [Bode90, Wiep06]

Spielsysteme

Sie sind eine spezielle Art von Simulationssystemen, die den Benutzer durch Anreize wie Wettkampf animieren. Der Lernstoff wird in Form von Spielen vermittelt. Software dieser Art wird auch als Edutainment bezeichnet, das heißt es handelt sich um Software die mit dem tatsächlichen Stoff auch ein Unterhaltungsprogramm bietet. [Bode90, Wiep06]

Problemlösungssysteme

Der Benutzer löst durch das System begleitet ein gegebenes Problem. Dadurch soll der Benutzer die Fähigkeit zum Problemlösen trainieren. Das System greift ein wenn der Benutzer einen falschen Weg beschreitet. Intelligente Ausprägungen dieser Systeme passen die gestellten Aufgaben an den Benutzer an. [Bode90]

Mikrowelten

Darunter versteht man abstrakte, auf einen engen Bereich eingeschränkte Welten in denen der Benutzer Situationen durch sein Handeln selbst kreiert und diese dann bewältigen muss. Im Modell sind Objekte, ihre Beziehungen und ihr Verhalten

gespeichert. Systeme dieser Art setzen teilweise Methoden der künstlichen Intelligenz ein. [Bode90, Wiep06]

2.2 Intelligente Tutorielle Systeme

Im Folgenden werden „Intelligente Tutorielle Systeme“ näher erläutert. Hierbei handelt es sich um in gewisser Weise besondere E-Learning Systeme. Im Gegensatz zu herkömmlichen Systemen gehen die dem Kognitivismus folgenden Intelligenten Tutoriellen Systeme auf den Lernenden und dessen Vorwissen ein. Sie ermöglichen eine individuellere Behandlung des Einzelnen. Zusätzlich versuchen sie die Aufgaben eines menschlichen Tutors umzusetzen.

Im ersten Abschnitt dieses Kapitels wird der Begriff der Intelligenten Tutoriellen Systeme gegenüber anderen herkömmlichen E-Learning Systemen abgegrenzt (siehe Kapitel 2.2.1). In weiterer Folge wird in Kapitel 2.2.2 auf die wesentlichen Merkmale Intelligenter Systeme eingegangen. Der idealtypische Aufbau dieser Systeme wird in Kapitel 2.2.3 betrachtet.

2.2.1 Begriff

Unter Intelligenten Tutoriellen Systemen (ITS) oder Wissensbasierten Lehrsystemen versteht man E-Learning Systeme welche einen Lehrenden sowohl in fachlicher als auch pädagogischer Hinsicht ersetzen. Dieses ideale Ziel wird in der Praxis bisher jedoch von keinem System tatsächlich erreicht. Systeme, die diesem Ziel nahe kommen, werden bereits als ITS bezeichnet, die Grenze zwischen herkömmlichen E-Learning Systemen und „Intelligenten Tutoriellen Systemen“ ist somit fließend. [Lust92]

Intelligente E-Learning Systeme weisen bestimmte Charakteristika auf, die sie gegenüber herkömmlichen E-Learning Systemen abgrenzen. Intelligente Systeme sind flexibler und passen sich an den Benutzer an, das heißt es gibt keine vorgefertigte Dialogfolge. Bei herkömmlichen Systemen wird davon ausgegangen was man unter den gegebenen Bedingungen erwartet. Um dieses Problem zu lösen werden Methoden der künstlichen Intelligenz eingesetzt, dadurch wird das System

befähigt sich an den Benutzer anzupassen. Alle bisherigen intelligenten Systeme arbeiten mit Wissensbasen. Sie haben Wissen an Stelle von losen Informationen gespeichert und setzen dies sinnvoll ein. Zu diesen intelligenten Systemen zählen neben ITS auch Expertensysteme und Mikrowelten. Auf diese wird hier jedoch nicht näher eingegangen. Eine kurze Beschreibung erfolgte bereits in Kapitel 2.1.4. [Bode90]

2.2.2 Merkmale

ITS weisen verschiedene Merkmale auf, die teilweise auch von anderen E-Learning Systemen in Ansätzen oder vollständig umgesetzt werden. Diese umfassen Flexibilität in jeder Hinsicht: bezüglich Benutzereinwirkung, Stoffauswahl, Zeit und Raum. Letztere können wie bei jedem anderen elektronisch gestützten System frei gewählt werden.

Durch Interaktivität zwischen Benutzer und System wird aktives Lernen ermöglicht. Der Lernende kann im Grunde genommen selbst bestimmen, was er lernen will. Er wird jedoch unter Umständen vom System gelenkt. ITS stellen unter anderem Möglichkeiten zur Kommunikation zur Verfügung, das heißt der Lernende kann Fragen stellen, die auch beantwortet werden. Übungen und Tests helfen dem System den Lernenden zu leiten. Durch Hilfedateien wird der Umgang mit dem System erleichtert. [Bode90]

Flexibilität bezüglich Stoffauswahl wird beispielsweise durch die Bereitstellung eines Menüs verwirklicht. Der Lernende kann selbst entscheiden welches Stoffgebiet er bearbeiten will. Das System nimmt hier Einfluss indem die Auswahl beschränkt wird. Durch Analysen erkennt das System welchen Stoff der Benutzer leicht bewältigt und womit er sich intensiver beschäftigen muss. [Bode90] Um das Ziel des effektiven Unterrichts zu erreichen werden Simulationstechniken und Hypermedia eingesetzt. Dadurch bestimmt das System wann welcher Stoff in welcher Weise präsentiert werden soll. [Wool92]

2.2.3 Aufbau ITS

Intelligente Tutorielle Systeme unterscheiden verschiedene Aspekte des Unterrichts. Dem entsprechend ist ihr Aufbau in verschiedene Teile gegliedert: die Fachkompetenz, die pädagogische und kommunikationstechnische Kompetenz des Experten. Die beiden letzteren werden teilweise zu einem Abschnitt zusammengefasst. [Lust92] Außerdem beschäftigt sich ein Teil des Systems näher mit dem Lernenden. Das System umfasst eine Steuerungskomponente die ein optimales Zusammenspiel der einzelnen Bereiche gewährleisten soll. Sämtliche Module des Systems arbeiten mit Informationen aus den jeweils anderen. Welche Module miteinander kommunizieren ist in Abbildung 4 skizziert. Da der tatsächliche Ablauf innerhalb des Systems schwer darzustellen ist, wird hier nur der Aufbau näher betrachtet.

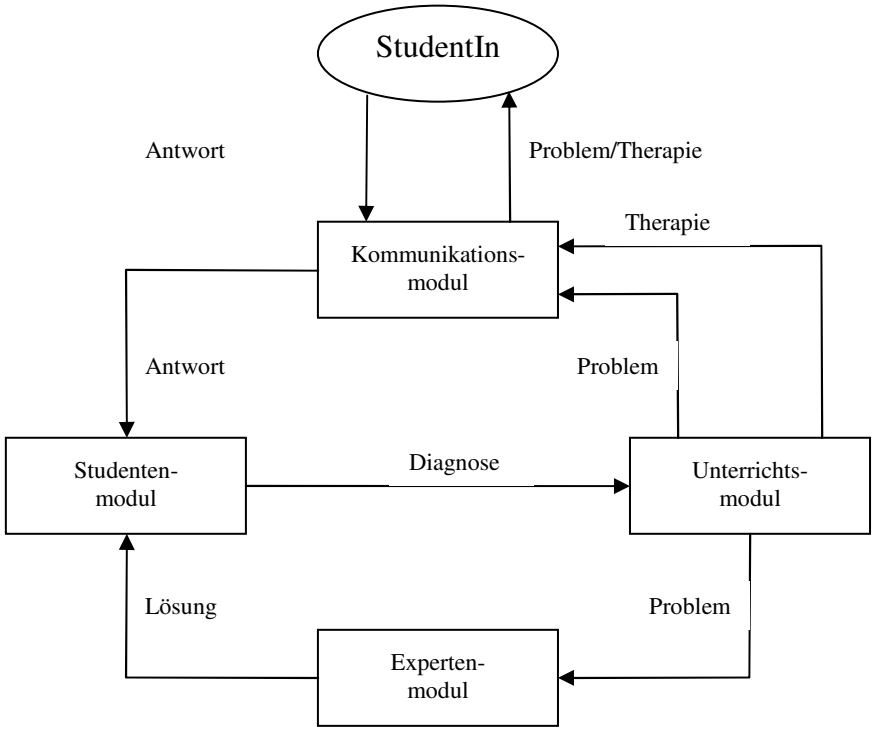


Abbildung 4: Aufbau Intelligenter Tutorieller Systeme [Lust92 S.26]

Expertenmodul

Dieser Teil des Systems enthält das Wissen über den Lehrstoff. Dies dient dazu einen menschlichen Experten in Bezug auf seine Fachkompetenz in dem betroffenen Stoffgebiet zu ersetzen. Dem ITS ist es dadurch möglich Wissen auszuwählen und verschiedene Teilbereiche zu verbinden. [Bode90, Davi89] Das Expertenmodul kreiert Aufgaben und ist zu deren Lösung fähig. Wie ein Experte ist es dazu befähigt, den Lösungsweg zu begründen und die Auswahl eines bestimmten Weges zu erklären. Je nach Aufgabe ist es auch möglich die Aufgaben zur Laufzeit zu generieren. Außerdem stellt das System dem Studenten entsprechende Aufgaben zur Verfügung. Auf Basis der Informationen aus den anderen Modulen legt das System fest welchen Schwierigkeitsgrad der Student bewältigen kann. [Lust92, Grev89]

Studentenmodul

Es ist dafür verantwortlich Informationen über den Benutzer zu erfassen und diese den anderen Modulen zur Verfügung zu stellen. Es werden sowohl langfristige als auch kurzfristige Eigenschaften festgehalten. Dies sind statische Informationen wie Ausbildungsgrad, Sprachkenntnisse und allgemeine Persönlichkeitsmerkmale aber auch dynamische Aspekte wie das aktuelle Lernverhalten des Benutzers. [Bode90, Davi89] Hierfür beobachtet das System den Benutzer und analysiert sein Verhalten. Ständig auftretende Fehler werden von zufälligen abgegrenzt. Durch die erhaltenen Informationen wird den anderen Modulen ermöglicht den Unterricht an den Benutzer anzupassen. [Lust92, Grev89]

Unterrichtsmodul

Hier ist das Wissen über Lehrmethoden enthalten, das heißt das Modul entscheidet welche Lehrstrategien wann eingesetzt werden und welche Aktionen dafür nötig sind. [Bode90, Davi89] Genauer gesagt wird hier festgelegt wann welche Informationen in welcher Form dargeboten werden müssen um den Unterricht so effizient wie möglich zu gestalten. Außerdem legt es fest wann der Benutzer unterbrochen werden soll um ihn auf den richtigen Weg zurück zu führen. [Bode90; Lust92, Grev89]. Diese

Entscheidungen sind von Informationen aus dem Studentenmodul sowie dem Studienfortschritt und der Schwierigkeit des Stoffs abhängig. [Lust92] Wenn der Student von dem im Expertenmodul vorgegebenen Idealen Verlauf abweicht, wird er unterbrochen und auf den richtigen Weg zurückgeführt. [Grev89]

Kommunikationsmodul

Zur Darstellung der Informationen können verschiedene Dialogformen eingesetzt werden. Ziel ist es den Unterricht so weit als möglich in natürlicher Sprache abzuhalten. [Bode90] Unter anderem werden auch Verknüpfungen zwischen den Stoffgebieten ermöglicht. Der Einsatz der Kommunikationsformen wird an den Benutzer angepasst [Lust92]

Modul	Hauptaufgaben
Expertenmodul	Modelliert einen Fachexperten: a) bietet Informationen an b) löst Aufgaben c) begründet Lösungen
Studentenmodul	Modelliert einen Benutzer: a) eruiert Fähigkeits- und Motivationsstand b) erkennt Interaktions- und Lernstil c) stellt Fehler fest d) analysiert Fehler (Diagnose)
Unterrichtsmodul	Modelliert einen didaktisch-methodischen Experten: a) wählt Inhalt und Ordnung der Information und bestimmt Fragen b) wählt Interaktionsstil (z.B. Interventionshäufigkeit und –zeitpunkt)
Kommunikationsmodul	Modelliert einen Kommunikationsexperten: wählt Kommunikationsform (z.B. natürlichsprachlich oder menügesteuert)

Tabelle 2: Übersicht Aufgaben-Teilung der ITS Module [nach Lust92 S. 19]

Die hier erläuterten Module stellen den idealtypischen Aufbau Intelligenter Tutorieller Systeme dar. In Tabelle 2 werden die Aufgaben der einzelnen Module zusammengefasst. Zu beachten ist, dass diese nicht unbedingt voneinander abgegrenzt sein müssen. Teilweise werden Module zusammengefasst, außerdem sind auch Erweiterungen um andere Komponenten durchaus möglich.

2.2.4 E-Learning im Data & Knowledge Engineering

Neben eTutor existieren auch andere E-Learning Systeme welche die Inhalte des Data & Knowledge Engineering abdecken, daher wurde auch in diesem Bereich nach möglichen Lösungsansätzen gesucht. Im Bereich des konzeptuellen Entwurfs existieren beispielsweise KERMIT, ERM-VLE und COLER, diese vermitteln Wissen in Bezug auf die Erstellung von Entity-Relationship Diagrammen. [Sura02] Im Bereich des logischen Entwurfs waren jedoch keine Systeme zu finden, allerdings bestehen einige Systeme deren Inhalte nicht evaluiert werden konnten.

Neben E-Learning Systemen bestehen auch diverse Tools zur automatischen Erstellung von relationalen Tabellen aus UML-Diagrammen. Einige dieser Tools werden in Kapitel 5.1 vorgestellt. Abgesehen von diversen Softwarelösungen wurden auch wie an der Universität von Utah [Mok-01] Algorithmen für das objekt-relationale Mapping veröffentlicht. Auch hier sind keine exakten Informationen bezüglich der konkreten Umsetzung zu erlangen. Aus diesem Grund werden im folgenden Abschnitt die zu Grunde liegenden Konzepte erläutert.

2.3 eTutor

eTutor ist ein am Institut für Data & Knowledge Engineering entwickeltes intelligentes tutorielles System das laufend weiter entwickelt wird. eTutor dient dazu den am Institut gelernten Stoff zu vertiefen und anhand von Beispielen zu praktizieren. Das System präsentiert Kursunterlagen aber hauptsächlich die dazugehörige Übungen, anhand derer der Student selbst überprüfen kann ob der gelernte Stoff tatsächlich verstanden wurde. Vom Studenten übermittelte Übungen werden automatisch durch

das System auf Fehler analysiert und benotet. Zusätzlich werden Fehler und mögliche Lösungen an den Studenten übermittelt. Durch das System sollen Studierende beim praktischen Üben unterstützt werden und die Lehrenden beim Benoten entlastet.

Im Folgenden wird der generelle Aufbau des Systems beschrieben, das durch diese Arbeit erweitert werden soll. Die technischen Grundlagen werden in Kapitel 2.3.1 erläutert und der Ablauf der Übungsauswertung und Übungsadministration, das heißt das Zusammenspiel der einzelnen Komponenten wird gezeigt (siehe Kapitel 2.3.2). Weiters wird in Kapitel 2.3.3 auf die inhaltlichen Grundlagen des Systems eingegangen. Hier erfolgt jedoch keine umfassende Beschreibung des Systems, dieses ist anhand anderer Quellen dokumentiert.

2.3.1 Technische Grundlagen

Das eTutor System ist als Intelligentes Tutorielles System aufgebaut (siehe Kapitel 2.2.3), das in Form einer web-basierten Anwendung konzipiert wurde, das heißt den Studenten wird der Zugriff über Internet ermöglicht. Auf Studentenseite ist nur ein Webzugang nötig um die Lehrmaterialien abarbeiten zu können. In Abbildung 5 ist die generelle Systemarchitektur des eTutors skizziert.

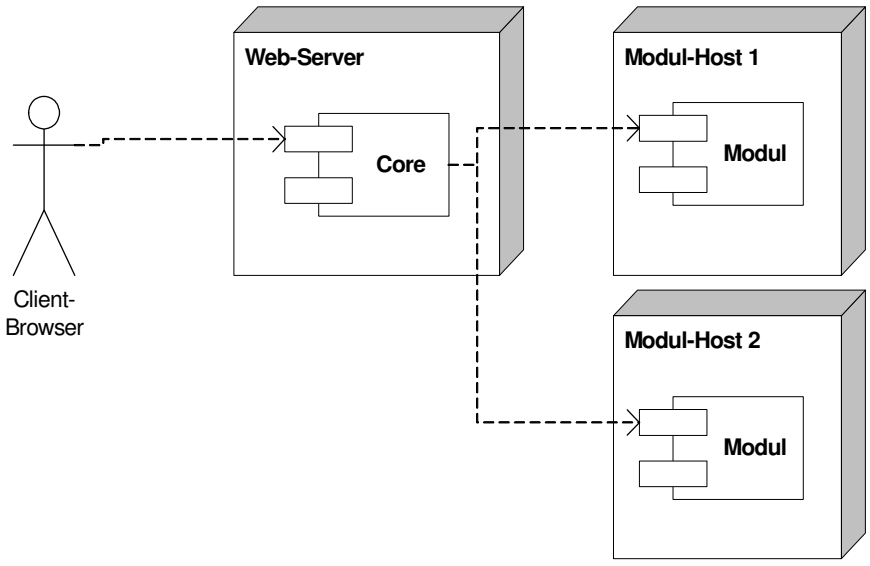


Abbildung 5: eTutor Systemarchitektur [Eich06, S. 4]

Auf einem Web-Server läuft das Kernsystem des eTutors, der eTutor Core. Dieser Teil des Systems umfasst das Kommunikations-, das Studenten- und das Unterrichtsmodul im Sinne des idealtypischen Aufbaus Intelligenter Tutorieller Systeme (siehe Kapitel 2.2.3). Der eTutor Core ist für sämtliche administrative Aufgaben wie zum Beispiel die Studentenverwaltung und die damit verbundenen Benutzerprofile oder die Erstellung neuer Beispiele durch die Assistenten zuständig. Außerdem wird sämtliche Kommunikation zwischen dem System und dem Studenten bzw. dem Assistenten oder Tutor über das Kernsystem abgewickelt. [Eich06]

Neben dem Kernsystem bestehen auch verschiedene Expertenmodule. Jedes davon umfasst ein bestimmtes „Kapitel“ des Lernstoffes. Diese Module stellen Aufgabenspezifische Informationen zur Verfügung und sind für Bewertung und Rückmeldungen zu den einzelnen Aufgaben verantwortlich. Diese kommunizieren über vorgegebene Schnittstellen mit dem eTutor Core. [Eich06]

Neben den idealtypischen Elementen eines Intelligenten Tutoriellen Systems werden im eTutor Konzept auch noch zusätzliche Komponenten vorgesehen. In einer eigenen Datenbank werden sämtliche für die Lehre und die Funktionsweise des Systems notwendigen Informationen gesammelt. Benutzerdaten, sowohl von Studenten als auch von Lehrenden, sowie die Merkmale zur Unterscheidung dieser Gruppen sind enthalten. Zur Abwicklung des Unterrichts werden Lehrveranstaltungen und dazu gehörige Studienleitfäden und Aufgaben verwaltet. Die Bedeutung dieser Elemente wird in Kapitel 2.3.3 gezeigt. Da es sich beim eTutor System um eine Web-Anwendung handelt wird ein Web-Server zur Darbietung des Systems benötigt. Außerdem existiert eine Konfigurationsdatei zur Initialisierung des Systems. [Eich06]

2.3.2 Zusammenwirken der Komponenten

Ablauf der Übungsauswertung

Die Expertenmodule können gemeinsam mit dem Kernsystem situiert sein oder auf einem anderen Rechner abgelegt werden. Wie bereits erwähnt sind sie für die Auswertung der vom Studenten abgegebenen Beispiele verantwortlich. Die Auswertung und deren Präsentation erfolgt durch modulspezifische Views und

Komponenten. Diese werden durch das Kernsystem koordiniert. Der Ablauf der Beispielauswertung erfolgt in nachstehenden Schritten: [Eich06]

- Aktivierung des Moduls durch das Kernsystem: Dies wird durch betätigen des Links zu einer bestimmten Aufgabe initialisiert. Parameter zur Identifizierung der Aufgabe und des Studenten werden festgelegt.
- Erstellung der Studentenlösung: In einer Modulspezifischen Eingabemaske kann der Student seine Lösung entwickeln.
- Auswahl des Bewertungsmodus und Abgabe: Der Student kann zwischen vier verschiedenen Aktionen auswählen:
    - run: die Lösung wird ohne Fehleranalyse durchgeführt
    - check: auf die Durchführung der Lösung erfolgt eine Meldung ob diese korrekt ist oder nicht
    - diagnose: Fehler werden rückgemeldet
    - submit: die Lösung wird abgegeben und durch das System bewertet
- Auswertung der Abgabe durch das System: die übergebene Studentenlösung wird je nach gewähltem Modus durch das entsprechende Modul abgearbeitet
- Feedback an den Studenten: das Modul leitet die gewünschten Informationen über das Kernsystem an den Studenten weiter

Ablauf der Beispieladministration

Neben der Erstellung von Aufgaben müssen auch Funktionen zu deren Aktualisierung bzw. Änderung zur Verfügung gestellt werden. Außerdem muss es ermöglicht werden, dass veraltete Beispiele aus dem System entfernt werden. Im Folgenden werden die für die Erstellung der Aufgaben notwendigen Schritte angeführt: [Eich06]

- View zum Erstellen eines Beispiels aufrufen: Dadurch wird ein Objekt erzeugt in dem sämtliche nachfolgenden Angaben abgelegt werden
- Aktivierung des Moduls: Dies erfolgt durch die Auswahl des Beispieltyps
- Modulspezifische Angaben erstellen: Für jedes Modul können spezifische Angaben gefordert werden, diese werden durch das Modul selbst gefordert

- Festlegen des Angabetextes
- Bestätigung der Angaben

Die Abläufe beim Erstellen und Ändern von Aufgaben sind zwar nicht vollkommen identisch, stimmen jedoch weitgehend überein. Aus diesem Grund wird die Änderung von Aufgaben hier nicht gesondert behandelt. Das Löschen von Aufgaben erfolgt unabhängig von den Modulen und wird hier daher ebenfalls außer Acht gelassen.

Wie aus diesem Erläuterungen ersichtlich werden die generellen Funktionen bei der Erstellung und Auswertung von Aufgaben bereits durch den eTutor Kernel erfüllt. Somit beschränken sich die Anforderungen an das Modul rein auf die für den logischen Entwurf spezifischen Aspekte. Diese müssen dann in das bestehende System eingebunden werden. Hauptaufgabe dieser Arbeit ist es daher die Auswertung von Aufgaben des logischen Entwurfs durchzuführen.

3. Stand der Technik

Der Konzeptuelle Entwurf kann normalerweise nicht als Grundlage für die Implementierung von spezifischen Datenbanken verwendet werden. Er beschreibt nur die Struktur der Daten, dies meist in datenbankunabhängigen Notationen. Als Grundlage für die Umsetzung der Strukturen in der Datenbank wird daher das logische Modell herangezogen welches aus dem konzeptuellen Entwurf überführt bzw. entwickelt wird. In einem Datenbankschema werden die zu speichernden Daten festgelegt. Ausgangslage für den logischen Entwurf ist ein konzeptionelles Modell. [Kemp04]

Dieser Abschnitt der Arbeit beschäftigt sich mit den theoretischen Grundlagen des zu entwerfenden Moduls und entwickelt daraus ein Konzept zur Erstellung der Lösungsüberprüfung. Erst wird in Kapitel 3.1 auf die Unified Modeling Language UML eingegangen, welche am Institut für Data & Knowledge Engineering zur konzeptuellen Modellierung und somit als Angabe für die Aufgaben zum logischen Entwurf verwendet wird. In weiterer Folge werden relationale Datenbanken behandelt, welche das Zielsystem für den logischen Entwurf darstellen (siehe Kapitel 3.2). Darauf aufbauend werden Möglichkeiten zur Überleitung der UML Strukturen in relationale Strukturen analysiert (siehe Kapitel 3.3).

3.1 UML

Dieser Abschnitt der Arbeit beschäftigt sich mit der am Institut für Data & Knowledge Engineering verwendeten Modellierungssprache UML. Diese wird als Basis für den logischen Entwurf eingesetzt. Das Modell wird durch die Mitarbeiter des Institutes erstellt und als Angabe für die Übung zum logischen Entwurf verwendet.

Erst wird in Kapitel 3.1.1 ein Überblick über UML an sich und die Einsatzmöglichkeiten im Bereich des logischen Entwurfs gegeben. Anschließend werden Modellelemente die in Klassendiagrammen verwendet werden können

angeführt (siehe Kapitel 3.1.2). Im Zuge dieser Betrachtungen wird in einem Exkurs die textuelle Darstellung der Diagramme mittels XMI behandelt.

3.1.1 Überblick

Die Unified Modelling Language (UML) ist eine von der OMG (Object Managament Group) standardisierte visuelle Notation. Diese Modellierungssprache für Softwaresysteme wurde in den 90ern von Grady Booch, Jim Rumbaugh und Ivar Jacobson entwickelt. [Hein04] Sie kann in allen Phasen der Software-Entwicklung, sowohl zum Design und zur Spezifikation als auch zur Dokumentation verwendet werden. Die verschiedenen Diagrammtypen und Modellelemente ermöglichen die Darstellung sämtlicher Aspekte von Programmen. [Oest01]

UML bietet verschiedene Diagrammtypen und Konstrukte, wobei die Anwendung jedes einzelnen davon nur in bestimmten Bereichen Sinn macht in anderen aber nicht zielführend ist. Im Bereich des logischen Entwurfs ist nur das Klassendiagramm von Bedeutung. Ein Beispiel dafür wurde bereits in Abbildung 1 in Kapitel 1 gezeigt. Ein Klassendiagramm stellt die interne Struktur des Systems dar. Objekte mit gleichen Eigenschaften und gleichem Verhalten werden zu Klassen zusammengefasst. Zwischen diesen Klassen werden mögliche Beziehungen dargestellt. [Oest01]

Um die Semantik der graphischen Elemente zu erweitern wird unter anderem die von der OMG entwickelte Object Constraint Language (OCL) verwendet. Hierbei handelt es sich um eine formale auf der Mengentheorie basierende Sprache mit deren Hilfe graphisch nicht darstellbare Einschränkungen in UML Diagrammen integriert werden können. Durch den formalen Aufbau wird eine klare Definition der Einschränkungen erreicht. Nachteil dieses Formalismus ist allerdings, dass ein relativ umfassendes mathematisches Verständnis notwendig ist um die Bedeutung zu erfassen. [Oest01, OCL-06]

3.1.2 Modellelemente

Im Folgenden wird auf die verschiedenen Modellelemente eingegangen, es wird jedoch vorausgesetzt, dass ein Leser dieser Arbeit mit den Verwendungszwecken der einzelnen Elemente vertraut ist. Für eine Einführung in die Thematik wird unter Anderem auf [Hitz05, Oest01] verwiesen. UML-Elemente, die im Bereich des logischen Entwurfs am Institut für Data & Knowledge Engineering eingesetzt werden, werden in Tabelle 3 abgegrenzt, ihre Bedeutung wird in Kapitel 3.3 behandelt. In UML werden weitere Elemente definiert, die für die Zwecke des logischen Entwurfs am Institut für Data & Knowledge Engineering aus folgenden Gründen bedeutungslos sind.

1. Einige dieser Elemente werden zur Beschreibung des Verhaltens von Objekten benötigt, was für den Datenbankentwurf irrelevant ist.
2. Andere spezielle Elemente werden für den Unterricht nicht benötigt.

Elemente, die aus einem der hier angeführten Gründe nicht berücksichtigt werden, werden in der Tabelle unter Angabe des entsprechenden Grundes ausgeschlossen. Hierfür wird definiert ob das jeweilige Element nicht verwendet werden kann, weil dessen Bedeutung nicht in der Datenbank umgesetzt werden kann oder ob das Element aus welchem Grund auch immer am Institut für Data & Knowledge Engineering nicht verwendet wird.

UML-Elemente	**Verwendung**
Klasse	Betrachtet
Binäre Assoziation	Betrachtet
N-äre Assoziation	Betrachtet
Rekursive Assoziation	Betrachtet
Assoziationsklasse	Betrachtet
Aggregation	Betrachtet
Komposition	Betrachtet
Generalisierung	Betrachtet
Einschränkungen	Betrachtet
Geschachtelte Klassen	Betrachtet
Rollen	Betrachtet

Aktive Klassen	Kein Einfluss auf Datenstruktur
Qualifizierte Assoziationen	Im Unterricht nicht verwendet
Klassifikationstyp	Kein Einfluss auf Datenstruktur
Redefinition von Attributen, Assoziationsenden, Operationen	Im Unterricht nicht verwendet
Assoziationsgeneralisierung	Im Unterricht nicht verwendet
Abhängigkeit	Kein Einfluss auf Datenstruktur
Interface	Kein Einfluss auf Datenstruktur
Variationen des Klassenkonzepts	Im Unterricht nicht verwendet
Parametrisierte Klasse	Kein Einfluss auf Datenstruktur
Signal	Kein Einfluss auf Datenstruktur

Tabelle 3: UML-Elemente und deren Relevanz

Exkurs: XMI

XML Metadata Interchange (XMI) ist ein ebenfalls von der OMG entwickelter Standard, zurzeit in Version 2.1 veröffentlicht. Hierbei handelt es sich um ein auf XML basierendes Austauschformat für Objekte. Es dient zur textuellen Darstellung von Analyse-Objekten wie beispielsweise UML, aber auch für objektorientierte Software wie Java. Außerdem können auch Komponenten und Datenbanken damit veranschaulicht werden. Da es sich hier um ein XML Dokument handelt, lassen sich XMI Dokumente durch Abfragesprachen wie XPath oder Stylesheets (XSLT) manipulieren. [XMI-03]

Der Standard definiert die Darstellung von Objekten in XML-Konstrukten sowie die Repräsentation von Beziehungen zwischen den Objekten mittels Objekt-IDs. Zusätzlich werden Regeln bzw. Möglichkeiten zur Validierung der erstellten XMI Dokumente vorgestellt. Um diese Aufgaben erfüllen zu können werden Regeln zur Erstellung von XML DTDs, Schemata und Dokumenten eingeführt. Außerdem wird der Weg des Reverse Engineering definiert um XMI Dokumente in die objekthafte Form überführen zu können. [XMI-03]

Auf die Details der Spezifikation wird hier nicht eingegangen, weitere Informationen können durch die auf der Homepage der OMG erhältlichen Spezifikationen eingeholt werden. Für die vorliegende Arbeit ist vor allem das durch das gewählte UML-Tool erstellte Dokument relevant da XMI im Verlauf

der Arbeit zur Speicherung und Weiterverarbeitung des grundlegenden UML-Diagramms verwendet wird.

3.2 Relationale Datenbanken

Das relationale Datenmodell wurde Anfang der 70er Jahre von Codd entwickelt. Zu Beginn waren jedoch die Hardwarebedingungen noch nicht ausgereift genug, deswegen begann der praktische Einsatz erst Ende der 70er. Im Gegensatz zu den bis dahin verwendeten Datenmodellen erlaubt das relationale Datenmodell mengenorientierte Datenabfragen. Wie die meisten anderen Datenbanken arbeiten auch relationale Datenbanken mit Datenbankmanagementsystemen. Durch den Aufbau dieser Systeme ist es möglich die Datenstruktur der Datenbank zu ändern ohne dass der Zugriff auf die Daten geändert werden muss. Das relationale Datenmodell ist derzeit marktbeherrschend, das heißt es ist im Prinzip der gängige Standard im Bereich von Datenbankanwendungen. [Kemp04, Epst89, Meie01]

Das relationale Datenmodell ist ein einfach strukturiertes Modell, es besteht im Wesentlichen nur aus Relationen, welche als Tabelle dargestellt werden können. Beziehungen müssen über Umwege realisiert werden. [Kemp04] In Kapitel 3.2.1 werden der Aufbau und die verwendeten Konzepte der Tabellen vorgestellt. Anschließend werden Methoden zur Beziehungsdarstellung angeführt (siehe Kapitel 3.2.2). Abschließend wird auf die standardisierte Sprache SQL eingegangen (siehe Kapitel 3.2.3).

3.2.1 Relationen

Tabelle: Personal				
SozNr	Name	Vorname	Stelle	AdNr
1255010677	Maier	Andreas	NA	5
5732161069	Huber	Maria	Assistent am Institut ...	7
2876050872	Gruber	Monika	Assistent am Institut ...	2
3486110460	Hofer	Sandra	Professor am Institut ...	8
4512280354	Sommer	Martin	Professor am Institut ...	4

Abbildung 6: relationale Datentabelle

Relationale Datenbanken beruhen auf Tabellen bzw. Relationen, einer zweidimensionalen Matrix wie sie in Abbildung 6 gezeigt wird. Diese bestehen aus einem Namen und Spalten, die die Attribute repräsentieren. Diese können Werte aus einem bestimmten Wertebereich (Integer, …) enthalten oder durch den Platzhalter NULL belegt sein. Die Anordnung der Spalten ist unwesentlich. [Kemp04, Rumb91, Flat90, Moos97, Jaro02] Der Wertebereich der einzelnen Attribute kann in der Tabellendefinition eingeschränkt werden. Der Wertebereich kann durch Zuordnung des Datentyps begrenzt werden. Durch NOT NULL wird festgelegt, dass das Attribut in jeder Instanz einen Wert enthalten muss. UNIQUE verbietet gleiche Werte in unterschiedlichen Instanzen. [Klei97, Rumb91]

Eine Zeile in der Tabelle entspricht einem Tupel, einer spezifischen Ausprägung der Relation. [Kemp04, Rumb91, Flat90, Moos97] Um die Zeilen eindeutig zu identifizieren gibt es das Konstrukt des Primary Key. Dies ist ein Schlüssel aus einem oder mehreren Attributen, der die Zeile der Tabelle eindeutig kennzeichnet. In den Spalten des Schlüssels dürfen keine NULL Werte enthalten sein. Der Schlüssel der Tabelle muss minimal sein, das heißt keines der Attribute des Schlüssels kann gestrichen werden ohne die Identifikation zu verlieren. [Klei97, Rumb91, Jaro02]

Aufgrund des Aufbaus dieser Tabellen gibt es gegenüber dem objektorientierten Entwurf mit UML gewisse Einschränkungen. Zum einen können in den Tabellen keine Operationen repräsentiert werden, das heißt das Verhalten der Objekte muss außer Acht gelassen werden. Zum anderen können nur primitive Datentypen repräsentiert werden. Um geschachtelte bzw. komplexe Datentypen abbilden zu können, müssen diese auf mehrere Tabellen aufgeteilt werden. Außerdem lässt sich

in relationalen Datenbanken das Konzept der Vererbung nicht direkt umsetzen. [Berg97, Kemp04, Rumb91, Jaro02]

3.2.2 Beziehungen

Beziehungen werden in relationalen Datenbanken über das Fremdschlüsselkonzept realisiert. Dies sind Attribute in einer Tabelle, die auf den Primärschlüssel einer anderen Tabelle verweisen. Der Fremdschlüssel kann nur auf bestehende Einträge verweisen oder wenn erlaubt den Wert NULL enthalten. Abbildung 7 zeigt die Zusammenhänge zwischen Primärschlüssel und Fremdschlüssel. [Rumb91, Flat90, Geis05, Jaro02]

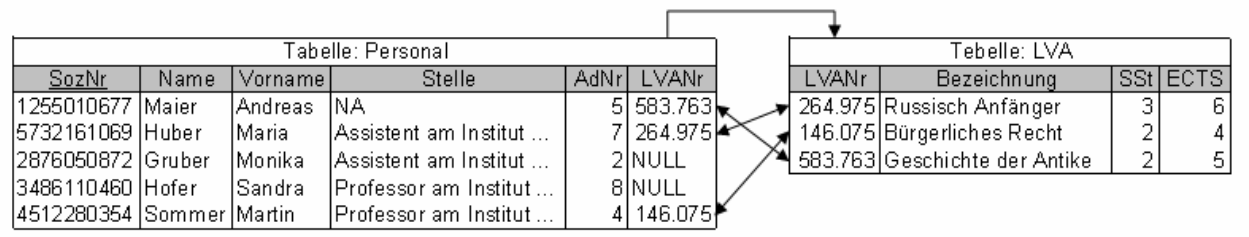


Tabelle: Personal

SozNr	Name	Vorname	Stelle	AdNr	LVANr
1255010677	Maier	Andreas	NA	5	583.763
5732161069	Huber	Maria	Assistent am Institut ...	7	264.975
2876050872	Gruber	Monika	Assistent am Institut ...	2	NULL
3486110460	Hofer	Sandra	Professor am Institut ...	8	NULL
4512280354	Sommer	Martin	Professor am Institut ...	4	146.075

Tebelle: LVA

LVANr	Bezeichnung	SSt	ECTS
264.975	Russisch Anfänger	3	6
146.075	Bürgerliches Recht	2	4
583.763	Geschichte der Antike	2	5

Abbildung 7: Verbindung zwischen Tabellen

Um die korrekte Funktionsweise der Beziehungen und Abhängigkeiten zu gewährleisten, bieten relationale Datenbanken bestimmte Integritätsregeln. Dadurch wird abgesichert, dass Elemente auf die Verweise bestehen nicht gelöscht oder verändert werden können. Dies wird als referentielle Integrität bezeichnet. [Klei97, Moos97, Kemp04] Im Folgenden werden diese Konstrukte näher erklärt.

NO ACTION: Ein Tabelleneintrag kann nicht gelöscht werden, wenn ein Fremdschlüssel aus einer anderen Tabelle auf den entsprechenden Primärschlüssel des zu löschenden Eintrags verweist.

CASCADE: Jeder Tabelleneintrag, dessen Fremdschlüssel auf den zu löschenden Eintrag verweist, wird mitgelöscht.

SET NULL: Wenn ein Tabelleneintrag gelöscht wird, werden sämtliche Fremdschlüssel anderer Tabellen, die auf den zu löschenden Eintrag verweisen, auf NULL gesetzt.

Neben den durch die Datenbank gebotenen Möglichkeiten zur Integritätskontrolle können auch benutzerdefinierte Regeln erstellt werden. Diese CHECK-Regeln können beim Einfügen oder Ändern von Tabelleneinträgen aufgerufen werden. Für die ganze Tabelle können CONSTRAINTS festgelegt werden, das sind benutzerdefinierte Integritätsbedingungen die auch einen Namen erhalten. Diese geben Bedingungen vor die jeder Eintrag in der Tabelle erfüllen muss. Zur Sicherung der Datenkonsistenz in Datenbanken können auch eigene Datenbankprozeduren, so genannte Trigger, erstellt werden. Diese führen je nach SQL-Operation verschiedene Kontrollen durch und können unter Umständen auch Werte ändern. [Moos97, Klei97]

3.2.3 Structured Query Language

Die Daten in relationalen Datenbanken werden durch eigene Sprachen abgefragt. Die gängigste dieser Abfragesprachen ist SQL (Structured Query Language). Sie wurde Mitte der 70er Jahre entwickelt, 1986 erstmals zum ANSI-Standard ernannt, die neueste Version wurde 2006 entwickelt. SQL ist eine deklarative, nicht prozedurale Programmiersprache, der Benutzer bestimmt welches Ergebnis er erhalten will, das DBMS entscheidet wie diese Anforderung erfüllt wird. [Geis05, Klei97, Kemp04, Moos97] Um SQL um prozedurale Funktionen zu erweitern kann es mit diversen Programmiersprachen kombiniert werden, dies wird als embedded SQL bezeichnet. [Geis05]

SQL setzt sich aus drei verschiedenen Teilsprachen zusammen, der Data Definition Language (DDL), der Data Manipulation Language (DML) und der Data Control Language (DCL). Auf die genauen Befehle und optionalen Möglichkeiten wird hier jedoch nicht eingegangen. Die genaue Funktionsweise von SQL sowie die Aufteilung in die verschiedenen Teilsprachen sind an anderer Stelle nachzulesen. [Geis05, Klei97, Flat90, Moos97] Für den logischen Entwurf ist nur die Data Definition Language von Bedeutung, diese wird für den Aufbau der Datenbank-Schemata der Datenbank verwendet. Sie umfasst Befehle zum:

- Erstellen (CREATE TABLE)
- Ändern (ALTER Table)
- und Löschen (DROP TABLE) von Tabellen

Datentypen

Neben den generellen Konstrukten sind aber auch die gebotenen SQL-Datentypen für die Entwicklung des Auswertungsmoduls relevant. Denn auch hier bestehen Diskrepanzen zwischen UML und SQL. Die meisten Datenbankanwendungen haben jedoch spezifische Datentypen in Anlehnung an den SQL-Standard. [Moos97] Hier wird SQL-92 behandelt und auf Unterschiede in Oracle Datenbanken eingegangen. Die Gegenüberstellung der unterschiedlichen Datentypen erfolgt in Kapitel 3.3.1.

3.3 Objekt-Relationales Mapping

Die für den konzeptionellen Entwurf und somit für die Angabe verwendete Modellierungssprache UML wurde primär für den Software-Entwurf entwickelt. Im Bereich des Datenbankentwurfs trifft man daher auf das Problem, dass in UML und relationalen Datenbanken teilweise keine einander entsprechenden Konzepte bestehen. Daher sind bestimmte Regeln notwendig um UML-Elemente in relationalen Strukturen abzubilden. Diese Vorgänge werden als Objekt-relationales Mapping bezeichnet. Hierbei handelt es sich um eine spezielle Form des logischen Entwurfs wobei das konzeptuelle Modell auf eine Objekt-orientierte Form bezogen wird und das logische Modell auf relationale Strukturen. Das Mapping bzw. die Überleitung ist notwendig da manche UML-Konstrukte wie bereits erwähnt wegen der unterschiedlichen Datenstruktur nicht direkt in die Datenbank übertragen werden können. In UML bestehen viele Darstellungselemente wogegen relationale Datenbanken nur Tabellen und das Fremdschlüsselkonzept zur Darstellung von Beziehungen aufweisen.

Teilweise existieren jedoch auch verschiedene Möglichkeiten ein bestimmtes Element in UML Notation in relationale Tabellen zu überführen. Die verschiedenen

Alternativen zur Abbildung der UML-Elemente in relationalen Datenbanken stellen die Basis für das zu entwickelnde Modul zur Auswertung der Aufgaben zum logischen Entwurf dar. In Kombination mit der Vorstellung der einzelnen UML-Elemente werden nun die Möglichkeiten zu deren Überleitung in die relationale Datenstruktur erläutert. Als erstes werden die Basiselemente aus UML in relationale Datenbanken übertragen (siehe Kapitel 3.3.1). Anschließend werden alle Arten von Beziehungen behandelt, bezogen auf einzelne Beziehungselemente in Kapitel 3.3.2 und unter Betrachtung mehrerer eine Klasse betreffende Beziehungen in Kapitel 3.3.3.

3.3.1 Überleitung von Basiselementen

In diesem ersten Abschnitt bezüglich des Mapping von konzeptuellen Konstrukten in eine relationale Datenbank, werden nur Elemente berücksichtigt die unabhängig von anderen Elementen dargestellt werden können. Dies betrifft in erster Linie Klassen aus UML. Änderungen die durch die Einbringung von Beziehungen hervorgerufen werden, werden hier vorerst nicht berücksichtigt, diese werden in weiterer Folge behandelt.

Abbildung 8: Schema für eine Klasse [Oestereich, S. 210]

Eine Klasse definiert Eigenschaften, Merkmale und Funktionen einer homogenen Objektmenge. [Oest01, UML-07] Für die Überleitung der Klassen in ein relationales Datenmodell sind die definierten Funktionen jedoch irrelevant, da das Verhalten der Elemente nicht in der Datenstruktur abgebildet wird. Die Struktur einer Klasse wird in Abbildung 8 dargestellt, einige der Elemente müssen vorhanden sein, andere wiederum sind optional. Eine Klasse wird durch ein Rechteck abgebildet, welches in verschiedene Abschnitte unterteilt werden kann. Notwendig ist der erste Abschnitt der den Klassennamen enthält, zusätzlich kann ein Bereich für Attribute und einer für

Operationen hinzugefügt werden, für diese besteht auch die Möglichkeit der Unterteilung in mehrere Abschnitte. [Hitz05, Oest01]

Im Bereich des logischen Entwurfs sind nur der Klassenname und die Attribute sowie sie betreffende Eigenschaften und Einschränkungen relevant. Operationen werden daher in dieser Ausführung nicht berücksichtigt. Die gesamte Klasse betreffende Eigenschaften werden im Bereich des Klassennamens notiert, Stereotype oberhalb des Namens, Eigenschaftswerte unterhalb. Stereotype wie <<Role>> spezifizieren ein zusätzliches Modellelement während Eigenschaften wie {Abstract} bestehende Modellelemente erweitern. Klassen mit der Eigenschaft {Abstract} können nicht initialisiert werden und bilden daher nur den Ausgangspunkt für Spezialisierungen welche später erläutert werden. [Oest01, UML-07, Hitz05] Am Institut für Data & Knowledge Engineering wird derzeit nur der Stereotyp <<Role>> verwendet. Dieser dient zur Identifizierung von Rollen welche später erläutert werden. Im zu entwickelnden Modul sind daher keine weiteren Stereotype zu berücksichtigen, allerdings soll eine einfache Erweiterung ermöglicht werden.

Attribute sind Elemente die in jeder Instanz einer Klasse enthalten sind, sie haben nur innerhalb dieser Klasse eine Bedeutung. Notiert werden sie im zweiten Abschnitt der Darstellung. Attribute können allein durch ihren Namen definiert werden, zusätzlich können aber auch der Datentyp, Initialwerte und Zusicherungen wie Einschränkungen des Wertebereichs angegeben werden. Wie bei der gesamten Klasse besteht auch hier die Möglichkeit Eigenschaftswerte und Stereotype zuzuordnen. Zusätzlich können auch Initialwerte angegeben werden, diese werden standardmäßig eingesetzt wenn keine anderen Werte spezifiziert werden. Attribute können auch mehrfach in einer Klasse enthalten sein, dies wird durch Mengenangaben gekennzeichnet. Hier handelt es sich Beispielsweise um Arrays. Die Mengenangaben entfallen wenn nur ein einmaliges Vorkommen erlaubt ist. [Oest01, UML-07, Hitz05]

Am Institut für Data & Knowledge Engineering wird ein eigens definiertes Konzept verwendet um eine Identifizierung der einzelnen Objektinstanzen zu ermöglichen. Objektorientierte Programmiersprachen verwenden zwar eigene implizite Objekt-ID’s um die exakte Identifizierung der einzelnen Objekte zu ermöglichen, in relationalen

Datenbanken ist jedoch die Angabe eines expliziten Schlüssels erforderlich. Um einen sinnvollen Schlüssel zu gewährleisten wird dieser vom Lehrenden bereits in der Angabe vorgegeben. Dies erfolgt durch die eigens definierte Eigenschaft {PK} – Primary Key – bzw. durch {ID} – Identifier. Um einen einfachen aus einem Attribut bestehenden Schlüssel zu definieren wird diese Eigenschaft beim entsprechenden Attribut notiert. Ein zusammengesetzter Schlüssel wird durch die Angabe eines Constraints nach dem Muster {ID(Attribut1, Attribut2, …)} festgelegt.

Generell kann davon ausgegangen werden, dass sämtliche Klassen als eigene Tabelle dargestellt werden. Aufgrund der Beziehungen können jedoch unter Umständen Änderungen hervorgerufen werden. Für jeden einfachen Datentyp wird eine Spalte in der Tabelle vorgesehen. Komplexe Datentypen können in der betreffenden Tabelle in mehrere Spalten aufgeteilt oder in eine eigene Tabelle ausgegliedert werden. Attribute mit Mehrfachwerten, wie zum Beispiel Arrays, werden in eine eigene Tabelle ausgegliedert, die nur solange besteht wie die eigentliche Klasse. Wenn Attribute ausgelagert werden, werden diese im Sinne einer existenzabhängigen 1:1 Beziehung behandelt, das heißt der Primärschlüssel der ursprünglichen Klasse wird auch in den zusätzlich entstandenen Tabellen als Primärschlüssel verwendet. [Berg97, Jaro02, Flat90, Meie01] Dies wird in Kapitel 3.4.2 näher erläutert. Das Konzept des {PK} bzw. {ID} wird durch die entsprechende Angabe des Primary Key in den relationalen Tabellen umgesetzt.

Beispiel

Abbildung 9 zeigt die Klasse Personal aus dem eingangs vorgestellten Klassendiagramm. Das Attribut SozNr wird als Primary Key deklariert, Adresse ist als komplexes Attribut zu betrachten. In der implementierungsnahen Sicht (a) wird das Attribut in der Klasse selbst dargestellt. Die Variante (b) zeigt die Ausgliederung des Attributs in eine eigene Klasse. Die Beziehungen die in dem Klassendiagramm von der Klasse Personal ausgehen werden hier nicht berücksichtigt. Stereotype und Eigenschaften werden in diesem einfachen Beispiel nicht berücksichtigt.

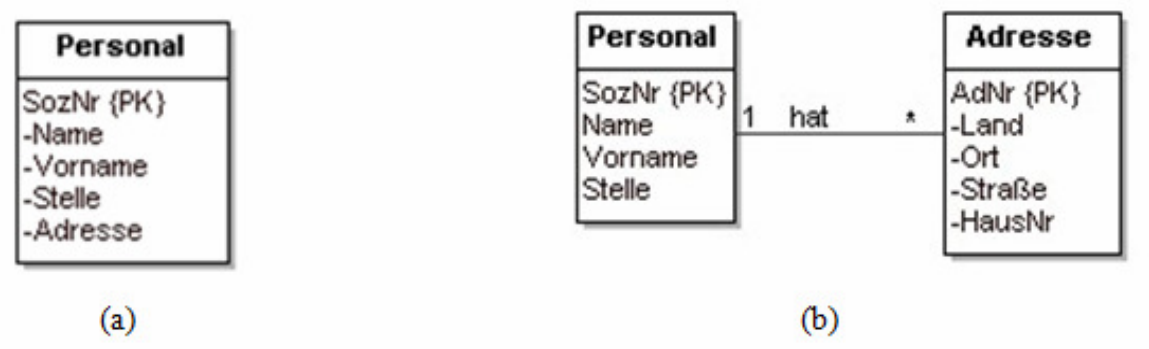


Abbildung 9: Beispiel Klasse Personal

Die auf Grund der vorherigen Betrachtungen entstehenden Mapping-Alternativen für die Klasse Personal werden in Abbildung 10 abgebildet. Hier sieht man, dass bei Auslagerung der Attribute mit zusammengesetztem Datentyp zwar zusätzliche Tabellen entstehen, diese sind jedoch nicht so umfangreich wie bei der Aufspaltung der Komplexen Typen in mehrere einfache Datentypen. Davon abgesehen ist die geplante Verwendung der Typen für die Entscheidung maßgeblich. In dem Beispiel wäre es auch möglich, dass eine Person mehrere Adressen hat, hierfür wäre jedoch eine andere Wahl des Fremdschlüssels erforderlich. Ein zusätzliches Entscheidungskriterium liegt in der Art der geplanten Abfragen. Wenn das Komplexe Attribut oft im Zusammenhang mit den anderen Daten der Tabelle benötigt wird, ist es sinnvoll diese gemeinsam zu speichern, da sonst jedes Mal ein Join der Tabellen durchgeführt werden muss.

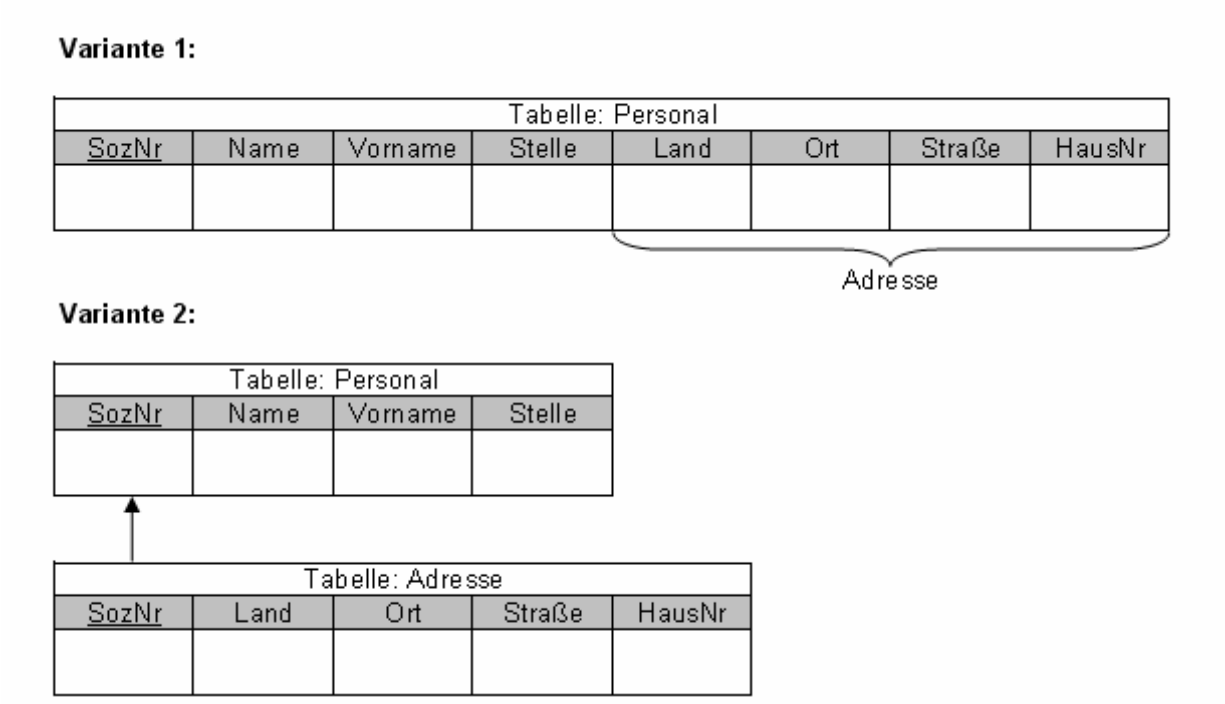


Abbildung 10: Mapping von Klassen

Obwohl in der Literatur komplexe Datentypen berücksichtigt werden, geht diese Arbeit davon aus, dass zusammengesetzte Typen bereits in der Angabe in einer eigenen Klasse dargestellt werden, das heißt, dass die Darstellungsform (b) in Abbildung 9 verwendet wird.

Neben den bisherigen Betrachtungen sind auch die verwendeten Datentypen von Bedeutung. Da in UML und SQL unterschiedliche Typen verwendet werden, wobei jedoch verschiedene Bezeichnungen die gleiche Bedeutung haben, müssen die in der Angabe vorgegebenen Typen in die entsprechenden SQL-Datentypen übergeleitet werden. In UML können die verwendeten Datentypen an den Zweck angepasst werden, bei der Erstellung der Angabe ist man jedoch von den durch das verwendete Tool gebotenen Möglichkeiten abhängig.

Die in SQL Definierten Datentypen ermöglichen die Darstellung von Zahlen, Zeichenketten bis zu ganzen Textdokumenten, Binärdaten und Datums- bzw. Zeitangaben. Sämtliche Bereiche bieten verschiedene auf bestimmte Zwecke angepasste Datentypen. Diese sowie die Äquivalente in Oracle Datenbanken und in Java werden in Tabelle 4 angeführt. Außerdem wird deren möglicher Einsatzbereich betrachtet. Zu beachten ist, dass NULL-Werte bei jedem Datentyp verwendet werden

können. Diese kennzeichnen, dass kein Wert vorhanden ist bzw. dieser unbekannt ist. Hierbei handelt es sich um eine von dem numerischen Wert null oder dem Leerstring abweichende Bedeutung.

SQL-92 Datentyp	Oracle Datentyp	Java Datentyp	Verwendung
Zahlenformate			
Integer	Number	Int	Ganze Zahl (-2.147.483.648 bis 2.147.483.647)
Smallint	Number	Short	Kleine Ganze Zahl (-32.768 bis 32.767)
Numeric (p,s) Decimal (p,s)	Number (p,s)	Java.math.BigDecimal	Festkommazahl wobei angegeben wird wie viele Stellen insgesamt vorhanden sind und wie viele nach dem Komma
Double	Number	Double	Gleitkommazahlen
Float	Number	Double	Gleitkommazahlen mit 38 Stellen
Real	Number	Float	Gleitkommazahlen mit 63 Stellen
Zeichenketten			
Char	Char	Java.lang.String	Ein Zeichen
Char [Länge]	Char (Länge)	Java.lang.String	Zeichenkette fixer Länge, wird durch Leerzeichen aufgefüllt, max 255 Zeichen
Varchar (max. Länge)	Varchar2 (Länge)	Java.lang.String	Zeichenkette mit einer maximalen Länge, kann auch kürzer sein Max. 32700 Zeichen, in Oracle max. 4000 Zeichen
Long Varchar (Länge)	Long	Java.lang.String	Zeichenkette mit einer maximalen Länge, kann auch kürzer sein Max. 2147483647 Zeichen
Binärdaten			
Bit (Länge)	Number	Boolean	Binärdatenkette fixer Länge
Bit Varying (max. Länge)	Number		Binärdatenkette variabler Länge
Binary	Raw	Byte	
Datumsangaben			

Date	Date	Java.sql.Date	Datum: Jahr, Monat, Tag
Time	Date	Java.sql.Time	Uhrzeit: Stunde, Minute, Sekunde
Timestamp	Timestamp	Java.sql.Timestamp	Zeitstempel: Jahr, Monat, Tag, Stunde, Minute, Sekunde, Mikrosekunde

Tabelle 4: Überleitung UML-Datentyp in SQL-Datentypen [in Anlehnung an Moos97, Klei97, Geis05, Ault03, Das-07]

3.3.2 Überleitung einzelner Beziehungen

Dieser Teil der Arbeit geht darauf ein welche Möglichkeiten bestehen um ein Konstrukt aus dem konzeptuellen Entwurf, in diesem Fall einen Beziehungstyp aus UML, in eine relationale Datenbank zu überführen. Zur Vereinfachung wird nur eine von anderen Elementen unabhängige Beziehung betrachtet, das heißt jeweils ein Beziehungstyp.

Binäre Assoziationen

Assoziationen wie die in Abbildung 11 dargestellte binäre Assoziation beschreiben eine Gruppe von Beziehungen welche Instanzen der jeweiligen Klassen eingehen können. In der Abbildung werden die einzelnen Elemente der Assoziation bezeichnet. Die Art der Beziehung kann durch die Vergabe eines Namens, einer Bezeichnung definiert werden. Jedes Objekt einer Klasse weist je nach Kardinalität der Beziehung die entsprechende Assoziation auf. Kardinalitäten bzw. Multiplizitäten beschreiben wie viele Beziehungen ein Objekt mit Instanzen der Klasse, mit der eine Beziehung besteht, eingehen kann. Ein optionaler Richtungspfeil bei der Bezeichnung bestimmt wie die Assoziation zu lesen ist. Das heißt für beide Navigationsrichtungen kann jeweils ein Name angegeben werden. Zusätzlich besteht die Möglichkeit die Enden zu bezeichnen, dadurch wird festgelegt welche Rolle die jeweilige Klasse in der Beziehung spielt. [Hitz05, UML-07, Oest01] Wie bei Klassen und Attributen besteht auch bei Assoziationen die Möglichkeit diese durch Stereotype, Eigenschaftswerte und Zusicherungen näher zu beschreiben. [Oest01]

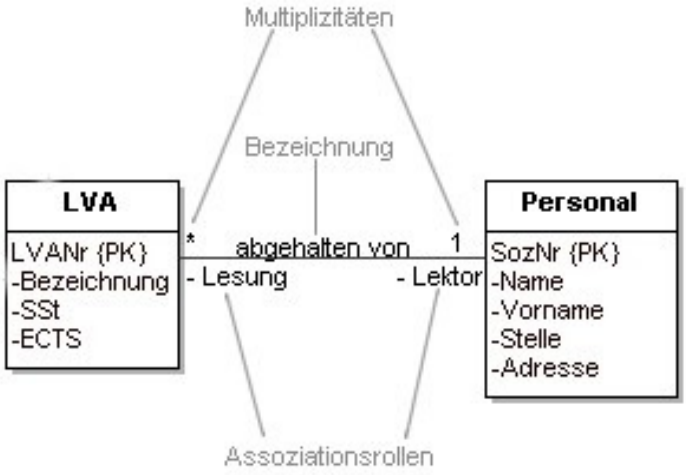


Abbildung 11: Beispiel binäre Assoziation

Unter den in Abbildung 11 dargestellten binären Assoziationen versteht man Beziehungen zwischen genau zwei verschiedenen Klassen. Diese Beziehungen können uni- oder bidirektional sein. Unidirektionale bzw. gerichtete Assoziationen können nur in eine Richtung gelesen werden, bidirektionale jedoch in beide. [Oest01] Wenn eine gerichtete Beziehung von Klasse A nach Klasse B verläuft kann der Primärschlüssel von Klasse B zur Repräsentation dieses Umstands in Klasse A aufgenommen werden. [Jaro02] Diese Unterscheidung ist jedoch irrelevant, da Beziehungen in relationalen Datenbanken durch Joins ausgewertet werden und daher keine Leserichtung ermöglichen.

Bei bidirektionalen Relationen bestehen je nach Kardinalität der beteiligten Klassen verschiedene Möglichkeiten die Beziehungen aus UML in der relationalen Datenbank abzubilden. Im Folgenden werden die Möglichkeiten für alle Kombinationen der Kardinalitäten erklärt. Welche dieser Möglichkeiten besser geeignet ist, hängt oft davon ab, in welchem Kontext das Konstrukt verwendet wird.

Hier wird zugrunde gelegt, dass „Klasse A“ für die erste Klasse der Beziehung steht und „Klasse B“ für die zweite. Die Bezeichnung x:y Beziehung definiert die Kardinalitäten der beiden an der Assoziation teilnehmenden Relationen, wobei x Klasse A betrifft und y Klasse B. Eine 1:1 Beziehung bedeutet, dass ein Objekt der Klasse A genau mit einem Objekt der Klasse B eine Beziehung eingehen kann. Gleiches gilt für ein Objekt der Klasse B. Bei einer 1:n Beziehung kann ein Objekt der Klasse A mit mehreren Objekten der Klasse B eine Beziehung eingehen, ein Objekt

der Klasse B jedoch nur mit genau einem der Klasse A. Es besteht auch die Möglichkeit, dass Klassen optionale Beziehungen aufweisen. Diese Beziehungen können eingegangen werden, sie müssen aber nicht. Derartige Kardinalitäten werden beispielsweise durch 0..1 dargestellt werden, dies bedeutet, dass die Beziehung einmal oder gar nicht eingegangen wird. Hier kann jede beliebige Obergrenze definiert werden.

1:1 Beziehung:

- Klasse A, Klasse B und die Beziehung werden zu einer Tabelle zusammengefügt. Als Primärschlüssel kann jeder der Schlüssel der beteiligten Klassen verwendet werden oder beide gemeinsam. [Jaro02, Kemp04, Jack89] Diese Variante ist von Vorteil wenn eine Klasse von der anderen existenzabhängig ist. [Rumb91] (Die Bedeutung dieses Begriffs wird bei der Betrachtung von Kompositionen erläutert)
- Klasse A und Klasse B werden in jeweils einer Tabelle dargestellt, beide verfügen über einen eigenen Primärschlüssel und enthalten einen eingabepflichtigen Fremdschlüssel der auf die jeweils andere Tabelle verweist. [Jaro02]
- Klasse A und Klasse B werden in jeweils einer Tabelle dargestellt, eine der beiden Tabellen enthält einen eingabepflichtigen Fremdschlüssel der auf den Primärschlüssel der anderen Tabelle verweist. [Rumb91, Flat90, Moos97, Meie01]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Relation dargestellt, der Primärschlüssel der Beziehungstabelle kann aus der Kombination der Primärschlüssel der beteiligten Klassen bestehen oder einen eigenen Primärschlüssel besitzen. In jedem Fall müssen die Primärschlüssel der beteiligten Tabellen als Fremdschlüssel enthalten sein. [Meie01]

1:0..1 Beziehung

- Klasse A, Klasse B und die Beziehung werden zu einer Klasse zusammengefügt, wobei die Attribute von Klasse B und der Beziehung nicht eingabepflichtig sind, der Schlüssel von Klasse B ist als eindeutig (UNIQUE) zu deklarieren. [Jaro02]

- Klasse A und Klasse B werden in jeweils einer Tabelle dargestellt, der Primärschlüssel von Klasse A wird als unikaler Fremdschlüssel in Klasse B aufgenommen, dadurch werden überflüssige NULL werte verhindert. [Jaro02, Moos97, Jack89]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Relation dargestellt, die Primärschlüssel der beteiligten Klassen werden in die Beziehungstabelle als unikale Fremdschlüssel aufgenommen. [Meie01]

0..1:0..1 Beziehung

- Klasse A und Klasse B werden jeweils in einer Tabelle abgebildet, ein nicht-eingabepflichtiger Fremdschlüssel wird in eine der beiden Relationen aufgenommen. [Jaro02, Moos97, Meie01]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle repräsentiert, die Primärschlüssel der beteiligten Klassen werden als unikaler Fremdschlüssel in die Beziehungstabelle aufgenommen. Diese Umsetzung ist von Vorteil wenn auf Instanzebene wenige Beziehungen bestehen. [Jaro02, Jack89]

1:0..N Beziehung

- Klasse A und Klasse B werden in jeweils einer Tabelle dargestellt, der Primärschlüssel von Klasse A wird als eingabepflichtiger Fremdschlüssel in Klasse B aufgenommen. [Jaro02, Moos97, Jack89]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle repräsentiert, die Primärschlüssel der Klassen A und B werden als Fremdschlüssel in die Beziehungstabelle aufgenommen wobei nur der auf die Klasse B verweisende Fremdschlüssel unikal ist. [Meie01]

0..1:0..N Beziehung

- Klasse A und Klasse B werden in jeweils einer Tabelle abgebildet, der Primärschlüssel von Klasse A wird als nicht eingabepflichtiger Fremdschlüssel in Klasse B aufgenommen. [Jaro02, Moos97, Meie01]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle dargestellt, der Primärschlüssel von Klasse A und der von Klasse B werden in die Beziehungstabelle als Fremdschlüssel aufgenommen wobei letzterer unikal ist. [Jaro02, Jack89]

1:N Beziehung

- Klasse A und Klasse B werden in jeweils einer Tabelle repräsentiert, der Primärschlüssel von Klasse A wird als eingabepflichtiger Fremdschlüssel in Klasse B integriert. [Kemp04, Rumb91, Jaro02, Flat90, Moos97, Meie01, Jack89]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle abgebildet, die Primärschlüssel der beiden beteiligten Klassen werden als Fremdschlüssel in die Beziehungstabelle eingefügt, der Fremdschlüssel aus Klasse B ist als UNIQUE zu deklarieren. [Meie01, Rumb91]

0..1:N Beziehung

- Klasse A und Klasse B werden in jeweils einer Tabelle repräsentiert, der Primärschlüssel von Klasse A wird als nicht eingabepflichtiger Fremdschlüssel in Klasse B aufgenommen. [Jaro02, Moos97, Meie01, Jack89]
- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle dargestellt, die Primärschlüssel der beiden Klassen werden als Fremdschlüssel, der aus Klasse B unikal, in die Beziehungstabelle aufgenommen. [Jaro02]

0..M:0..N Beziehung, M:0..N Beziehung und M:N Beziehung

- Klasse A, Klasse B und die Beziehung werden in jeweils einer Tabelle abgebildet, die Primärschlüssel von Klasse A und Klasse B sind in der Beziehungstabelle als Fremdschlüssel enthalten. [Rumb91, Jaro02, Moos97, Meie01]

Beispiel

In Abbildung 11 wird die bereits vorgestellte Klasse Personal zur Klasse LVA in Beziehung gestellt. Hierbei handelt es sich um eine 1:* Beziehung. Um die alleinige Betrachtung der Beziehung zu gewährleisten wird das komplexe Attribut Adresse in Spalten in der Tabelle Personal aufgegliedert. Aus den bisherigen Ausführungen bezüglich der Umsetzung von binären Assoziationen ergeben sich für das hier betrachtete Beispiel zwei verschiedene Umsetzungsalternativen. Variante 1 in Abbildung 12 zeigt die Umsetzung unter Verwendung eines Foreign Key. In Variante 2 wird die Erstellung einer Koppeltabelle verwendet.

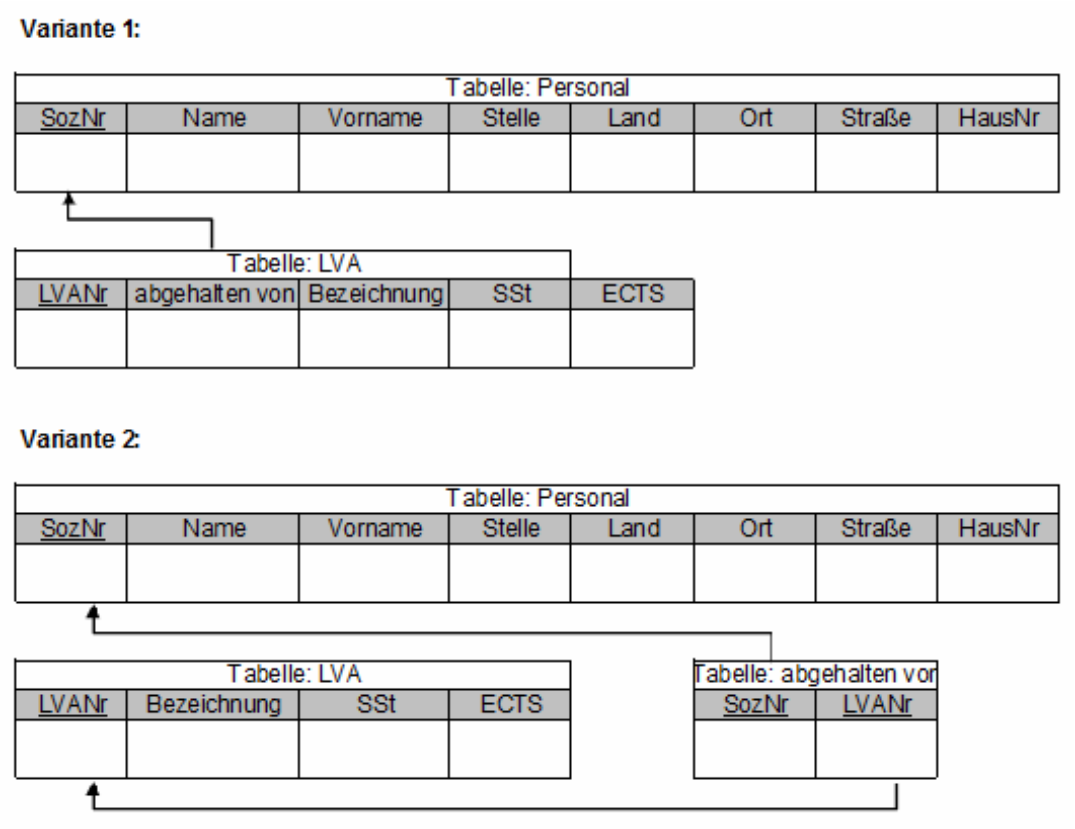


Abbildung 12: Mapping von binären Assoziationen

Durch die Integration des Foreign Key in die Tabelle LVA kann durch eine Kennzeichnung als NOT NULL gewährleistet werden, dass eine Instanz von LVA in jedem Fall von einer Instanz von Personal abgehalten werden muss. Durch die Verwendung einer Koppeltabelle ist dies nicht direkt möglich, hier ist eine Assertion notwendig. Setzt man voraus, dass jede Instanz von Personal mindestens eine LVA abhalten muss, kann dieser Umstand bei beiden Varianten nur durch zusätzliche

Assertions umgesetzt werden. Auch Jarosch weist darauf hin, dass wie aus diesem Beispiel ersichtlich sowohl minimale als auch maximale Kardinalitätsbeschränkungen in den relationalen Strukturen nicht direkt umgesetzt werden können. [Jaro02, S.247] Hierzu ist die Erstellung von speziellen Constraints bzw. Triggern notwendig.

N-äre Assoziation

Manchmal betreffen Beziehungen mehrere Partnerklassen, wobei eine Klasse in mehreren Rollen an der Assoziation teilnehmen kann (Abbildung 13). Hierbei handelt es sich um n-äre Beziehungen. Die Darstellung erfolgt durch eine Raute von der ausgehend die einzelnen beteiligten Klassen assoziiert werden. Multiplizitäten werden wie bei binären Beziehungen dargestellt. Die Bezeichnung der Beziehung wird in der Nähe der Raute angegeben. Im Gegensatz zu Binären Assoziationen sind bei N-ären Assoziationen keine Navigationsrichtungen möglich. [Oest01, Hitz05] In der relationalen Datenstruktur wird jede Beziehung an der mehr als zwei Klassen beteiligt sind über eine Koppeltabelle dargestellt. Die Primärschlüssel der beteiligten Klassen werden als Fremdschlüssel in die Beziehungstabelle aufgenommen. Je nach Multiplizität der beteiligten Klassen, das heißt bei Multiplizität 1, werden diese Fremdschlüssel als eindeutig deklariert. [Rumb91, Flat90, Meie01, Jack89]

Beispiel

Das Beispiel in Abbildung 13 zeigt wiederum einen Ausschnitt aus dem zu Beginn vorgestellten Klassendiagramm. Ein Termin setzt die Klassen LVA, Hörsaal und Datum zueinander in Beziehung. Hierbei handelt es sich um eine ternäre Beziehung, das heißt eine Beziehung die drei Klassen betrifft. Diese Darstellung bedeutet, dass zu einem Datum in einem Hörsaal maximal eine LVA stattfinden kann. Eine LVA hingegen kann zu einem bestimmten Datum in mehreren Hörsälen abgehalten werden, aber auch in einem Hörsaal zu unterschiedlichen Terminen.

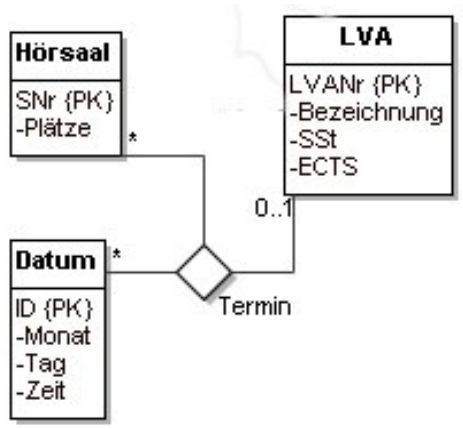


Abbildung 13: Beispiel n-äre Assoziation

Der hier definierten Regel für die Umsetzung von n-ären Beziehungen in relationalen Tabellen folgend werden in Abbildung 14 die zum Beispiel aus Abbildung 13 gehörenden Tabellen dargestellt. Sämtliche Klassen werden nach der vorhin definierten Regel in jeweils eine Tabelle überführt. Die Assoziation Termin wird in eine eigene Tabelle eingetragen wobei die auf die drei Klassen verweisenden Fremdschlüssel gemeinsam den Primärschlüssel dieser Tabelle darstellen.

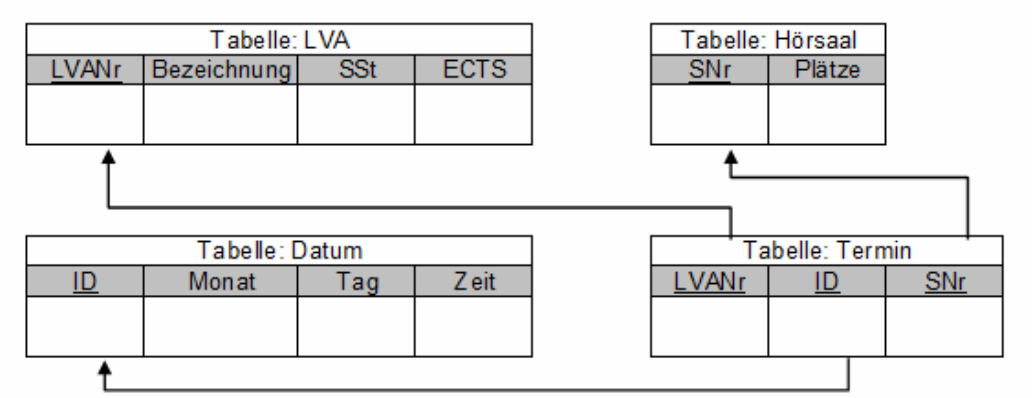


Abbildung 14: Mapping von N-ären Assoziationen

In dieser Darstellung ist zu erkennen, dass zusätzlich zu den bisherigen Einträgen eine Einschränkung der Kardinalität der Klasse LVA von Nöten ist. Da diese hier pro Hörsaal und Datum Kombination nur einmal bzw. gar nicht vorkommen darf kann dies durch eine Einschränkung des Schlüssels auf SNr und ID bewerkstelligt werden. LVANr in der Klasse Termin darf auch NULL sein.

Rekursive Assoziationen

An Stelle von zwei verschiedenen an der Assoziation beteiligten Klassen besteht auch die Möglichkeit, dass eine Klasse in verschiedenen Rollen an der Beziehung beteiligt ist. Je nach Kontext wird hier davon ausgegangen, dass zwei verschiedene Objekte an der Beziehung beteiligt sind. Wie bei jeder anderen Assoziation können auch hier diverse Bezeichnungen wie Rollen und Multiplizitäten angeführt und Eigenschaften näher definiert werden. [Oest01]

Wie bei Assoziationen an denen mehrere Klassen beteiligt sind, gibt es auch bei rekursiven Assoziationen je nach Kardinalität unterschiedliche Umsetzungsmöglichkeiten. Die Unterschiede werden im Folgenden erklärt.

1:1 Beziehung

- Der Primärschlüssel der Klasse ist unter einem anderen Namen als Fremdschlüssel enthalten, dieser ist eingabepflichtig und unikal. [Jaro02]
- Die Beziehung kann auch über eine extra Tabelle dargestellt werden, diese enthält den Primärschlüssel der Klasse zweimal als unikalen Fremdschlüssel. [Meie01] Hierbei sollte die Rolle der jeweiligen Klasse in der Beziehung durch den Namen des Fremdschlüssels identifiziert werden.

0..1:0..1 Beziehung

- Der Primärschlüssel der Klasse ist als nichteingabepflichtiger unikaler Fremdschlüssel unter anderem Namen ein zweites Mal enthalten. [Jaro02]
- Wenn wenige Objekte der Klasse eine Beziehung untereinander eingehen, ist es sinnvoll eine eigene Tabelle für die Beziehung einzurichten. Der Primärschlüssel der Klasse ist unter zwei verschiedenen Namen in der Beziehungstabelle enthalten. [Jaro02, Meie01]

1:0..N Beziehung

- Der Primärschlüssel der Klasse ist unter anderem Namen noch einmal als nicht unikaler eingabepflichtiger Fremdschlüssel enthalten. [Jaro02]

- Eine eigene Tabelle repräsentiert die Beziehung, der Primärschlüssel der Klasse ist zweimal als Fremdschlüssel enthalten, wobei einer der beiden als UNIQUE deklariert werden muss. [Meie01]

0..1:0..N Beziehung

- In der Tabelle der Klasse ist der Primärschlüssel unter anderem Namen noch einmal als nicht eingabepflichtiger nicht unikaler Fremdschlüssel enthalten. [Jaro02]
- Eine extra Tabelle für die Beziehung kann eingeführt werden wenn wenige Beziehungen eingegangen werden. Der Primärschlüssel der Klasse ist in der Beziehungstabelle zweimal unter verschiedenem Namen enthalten, wobei einer unikal ist. [Jaro02]

0..M:0..N Beziehung, M:0..N Beziehung und M:N Beziehung

- Um die Beziehung darzustellen ist eine eigene Tabelle nötig, die den Primärschlüssel der Klasse zweimal unter unterschiedlichen Namen enthält. [Jaro02]

Beispiel

In dem hier verwendeten Beispiel (Abbildung 15) wird die bereits bekannte Klasse LVA gezeigt. Anders als zuvor wird jetzt die rekursive Assoziation „Voraussetzung für“ betrachtet. Aus dem hier betrachteten Zusammenhang ist davon auszugehen, dass jeweils zwei verschiedene Instanzen der Klasse LVA an der Beziehung beteiligt sind. Jede Instanz kann für beliebig viele andere Instanzen Voraussetzung sein, eine Instanz kann auch mehrere Instanzen als Voraussetzung haben.

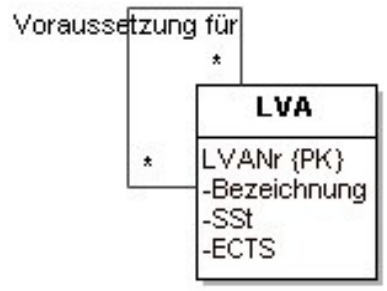


Abbildung 15: Beispiel rekursive Assoziation

In Abbildung 16 werden die Tabellen zum Beispiel aus Abbildung 15 dargestellt. Zusätzlich zu den eigentlichen Tabellen muss durch einen eigenen Constraint gesichert werden, dass die beiden Fremdschlüssel eines Tupels nicht auf denselben Primary Key in der Tabelle LVA verweisen.

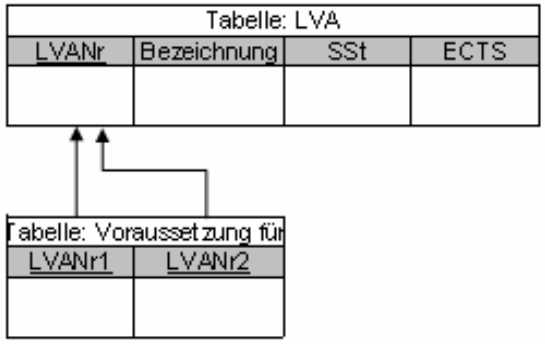


Abbildung 16: Mapping von rekursiven Assoziationen

Wie auch bei binären Assoziationen besteht hier das Problem, dass Kardinalitätsbeschränkungen im Datenschema der Relationen nicht direkt umgesetzt werden können. Dies ist zwar für das hier vorgestellte Beispiel nicht relevant, allerdings kann dieses Problem in einem anderen Kontext auftreten. Wenn eine Eltern-Kind-Beziehung durch eine rekursive Assoziation auf eine Klasse Person dargestellt wird, muss sichergestellt werden, dass eine Instanz welche ein Kind repräsentiert maximal zwei Instanzen als Eltern aufweist.

Assoziationsklasse

Assoziationsklassen wie in Abbildung 17 dienen dazu Beziehungen näher zu beschreiben, die Assoziation kann beispielsweise über eigene Attribute verfügen. Eine Assoziationsklasse ist gleichzeitig eine Beziehung und eine Klasse. Aus diesem Grund können Assoziationsklassen auch Teil einer Beziehung sein. Die Bezeichnung der Beziehung entspricht dem Namen der Assoziationsklasse. [Oest01, Hitz05]

Eine Assoziationsklasse beschreibt die Beziehung näher, das heißt sie dasselbe Löschverhalten wie die Beziehung auf. Daraus ergibt sich das sämtliche Attribute der Assoziationsklasse in der Tabelle integriert werden, die auch die Beziehung beschreibt. Das heißt wenn die Beziehung durch einen Fremdschlüssel in einer der beteiligten Klassen realisiert wird, werden auch die Attribute in diese Klasse aufgenommen. Wenn eine eigene Tabelle für die Beziehung erstellt wird, werden die Attribute der Assoziationsklasse in diese als einfache Attribute eingefügt. [Flat90]

Beispiel

Das hier verwendete Beispiel beleuchtet ebenfalls einen Teil des vorgestellten Klassendiagramms. Dabei handelt es sich um die Klassen LVA und Student und deren Beziehung die Teilnahme. Zu dieser Beziehung besteht eine Assoziationsklasse die ein zusätzliches Attribut enthält. Dieses spezifiziert in welchem Semester ein Student an der LVA teilnimmt.

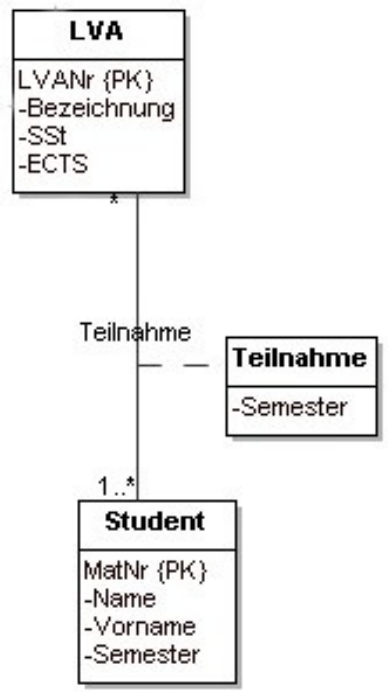


Abbildung 17: Darstellung Assoziationsklasse

Die binäre Assoziation zwischen LVA und Student verlangt die Umsetzung der Beziehung mittels Koppeltabelle. Da die Assoziationsklasse die besagte Assoziation näher beschreibt muss auch die Assoziationsklasse in diese zusätzliche Tabelle integriert werden. Die entsprechenden Tabellen zum Beispiel werden in Abbildung 18 angeführt.

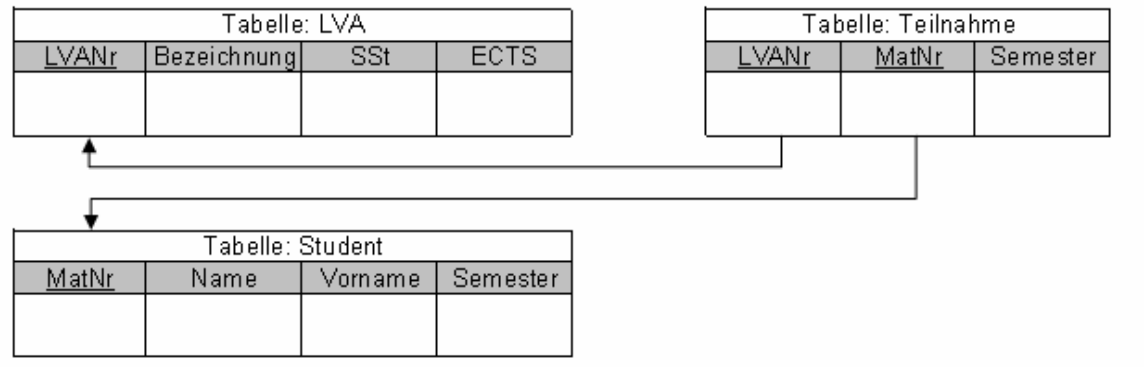


Abbildung 18: Mapping von Assoziationsklassen

Aggregation

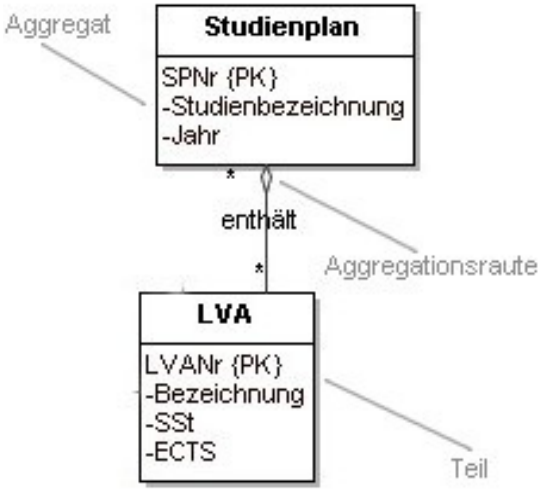


Abbildung 19: Beispiel Aggregation

Eine Aggregation (Abbildung 19) kann nur zwischen jeweils zwei Elementen bestehen. Mit Hilfe einer Aggregation kann beschrieben werden, wie ein Objekt aus Einzelteilen zusammengesetzt wird. Die einzelnen Elemente bestehen jedoch unabhängig voneinander. Eine Aggregation wird wie eine binäre Assoziation durch eine die zwei Klassen verbindende Linie dargestellt, wobei das Aggregat, das Ganze, durch eine leere Raute an dessen Assoziationsende gekennzeichnet wird. Wie bei Assoziationen können auch hier Bezeichnungen für die Aggregation selbst sowie die beiden Enden der Beziehung angegeben werden. Auch die Angabe von Multiplizitäten ist hier vorgesehen da ein Teil im Sinn dieser Beziehung auch mehreren Aggregaten gleichzeitig zugeordnet werden kann. Fehlt eine Multiplizitätsangabe auf Seite des Ganzen ist dies als 1 zu interpretieren. [Oest01, UML-07]

Dieses UML-Element beschreibt zwar eine Teil-Ganzes Beziehung, da jedoch die einzelnen Elemente unabhängig voneinander existieren und jeder Teil in verschiedenen Ganzen enthalten sein kann, kann sie genauso wie eine normale Assoziation behandelt werden. [Rumb91]

Beispiel

Der Ausschnitt aus dem Klassendiagramm zeigt die Klassen Studienplan und LVA sowie die zwischen ihnen bestehende Aggregation. In diesem Fall kann eine LVA mehreren Studienplänen zugeordnet werden, ein Studienplan enthält mehrere LVAs.

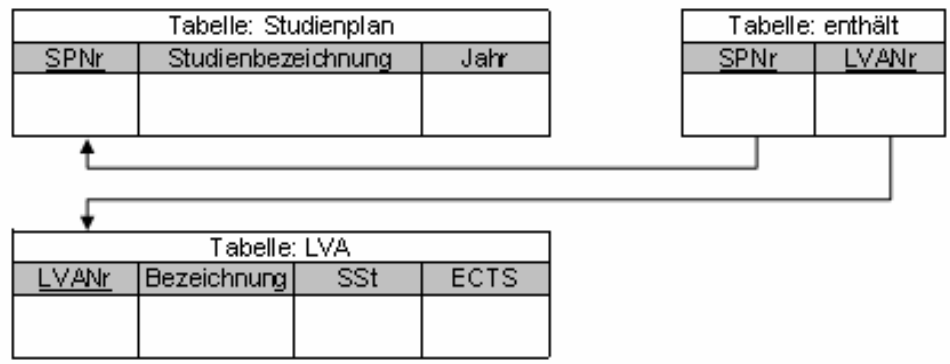


Abbildung 20: Mapping von Aggregationen

Wie in Abbildung 20 ersichtlich wird das Beispiel aus Abbildung 19 wie eine binäre Assoziation mit Kardinalität M:N behandelt. Es wird eine zusätzliche Tabelle mit der Bezeichnung „enthält“ eingeführt. Die Schlüssel der Tabellen LVA und Studienplan werden als Foreign Key in diese Tabelle aufgenommen. Die Begrenzung der Teilezahl ist in diesem Fall zwar nicht notwendig allerdings kann sie auch im Zuge der Aggregation nicht direkt in die Tabellen integriert werden.

Komposition

Wie die Aggregation beschreibt die Komposition ein Ganzes-Teile-Verhältnis (Abbildung 21). Ein Teil darf hier jedoch nur zu einem Ganzen im Sinne einer Komposition gehören, es ist existenzabhängig. Die Darstellung der Komposition unterscheidet sich von einer Aggregation dadurch, dass die Raute die das Ganze markiert ausgefüllt wird. Je nach Multiplizität müssen die Teile mit dem Ganzen gelöscht werden. Sie können auch vor dem Löschen eines Ganzen zu einem anderen zugeordnet werden. [Hitz05, UML-07]

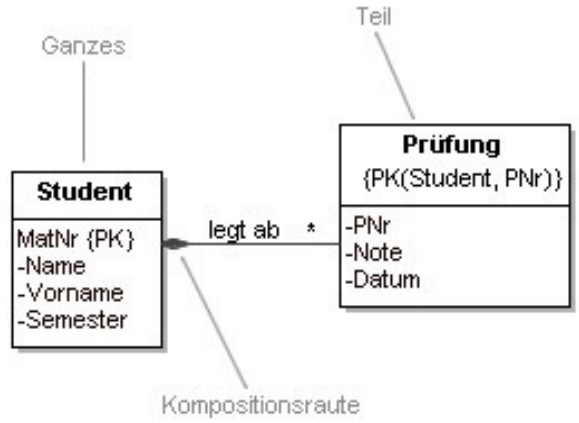


Abbildung 21: Darstellung Komposition

Eine Komposition mit Kardinalität 1 entspricht einer existenzabhängigen Teile-Ganzes Beziehung. Das heißt wenn das Ganze gelöscht wird kann auch das Teil nicht weiter existieren. Das Teil gehört nur zu einem Ganzen. Um diesen Sachverhalt abzubilden wird der Schlüssel des Ganzen in den Primärschlüssel des Teils integriert. Das entsprechende Löschverhalten muss mit Implementiert werden, das heißt wenn das Ganze gelöscht wird muss automatisch das Teil gelöscht werden. [Kemp04]

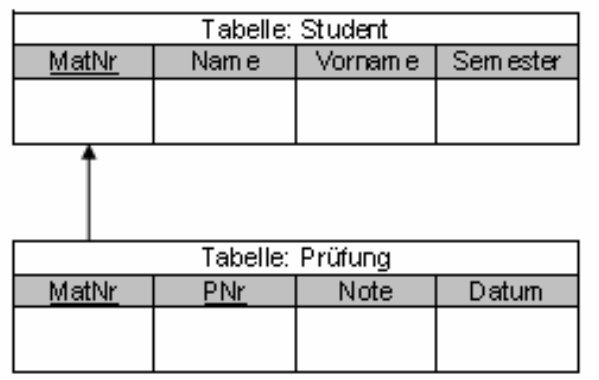


Abbildung 22: Mapping von Kompositionen

Für das in Abbildung 21 präsentierte Beispiel bedeutet das die Darstellung in zwei verschiedenen Tabellen wobei die eine den Fremdschlüssel auf die andere Tabelle in ihren Primärschlüssel aufnimmt. Eine derartige Umsetzung wird in Abbildung 22 gezeigt. Zusätzlich muss beim Fremdschlüssel SerienNr jedoch gesichert werden, dass dieser gelöscht wird wenn der entsprechende Eintrag in der Tabelle Fenster

entfernt wird. Dies wird durch eine zusätzliche Angabe bezüglich des Löschverhaltens bewerkstelligt. (ON DELETE CASCADE).

Generalisierung

Eine Generalisierung ist eine spezielle Beziehung wobei die eine Klasse eine Spezialisierung der anderen ist. Dargestellt wird dies durch einen nicht ausgefüllten Pfeil von der spezialisierten Klasse zu der allgemeinen. Im Gegensatz zu Assoziationen sind hier keine Multiplizitätsangaben möglich. Die spezialisierte Klasse enthält alle Attribute und Operationen der Oberklasse. Die Spezialisierung kann jedoch auch zusätzliche Eigenschaften aufweisen. Aus diesem Konstrukt ergibt sich, dass jede Instanz der Subklasse gleichzeitig auch eine Instanz der Superklasse ist. Die Einteilung der Unterklassen nach einem bestimmten Unterscheidungsmerkmal kann mit Hilfe eines Diskriminators erfolgen, dieser wird entlang der Beziehungslinie angegeben. [Oest01, Hitz05]

Generalisierungen können disjunkt oder überlappend sein, wobei disjunkt als Standard betrachtet wird. Disjunkte Generalisierungen erlauben jedem Objekt nur einer Subklasse anzugehören, wogegen bei überlappenden Generalisierungen jedes Objekt der Superklasse mehreren Subklassen gleichzeitig angehören kann. Ein weiteres Klassifikationsmerkmal von Generalisierungen ist die Vollständigkeit. Vollständig bedeutet hier, dass jedes Objekt einer Subklasse angehören muss, unvollständig hingegen heißt, dass auch reine Instanzen der Superklasse bestehen können. Als Standard wird hier eine unvollständige Generalisierung angenommen. Die Unterscheidung wird durch die Angabe von entsprechenden Eigenschaftswerten bei der Beziehung gewährleistet. [Hitz05]

Es gibt ein paar Möglichkeiten die Generalisierung, also die Vererbung, in relationalen Datenbanken umzusetzen. Durch die Anordnung der Klassen in den Tabellen ist es jedoch nicht möglich die Vererbung im eigentlichen Sinne, mit Ausnahme einer unvollständigen überlappenden Generalisierung, umzusetzen. Eine vollständige Generalisierung kann nur durch zusätzliche Integritätsbedingungen umgesetzt werden.

- Für jede Super- und Subklasse wird eine Tabelle erstellt, in den Subklassen wird derselbe Primärschlüssel verwendet wie in der Superklasse. Für disjunkte Generalisierung entsteht hier jedoch das Problem, dass nicht abgesichert werden kann, dass nur eine Subklasse das Objekt enthält. Zur Lösung kann ein zusätzliches Attribut in die Superklasse integriert werden, welches angibt in welcher Subklasse das jeweilige Element enthalten ist. [Berg97, Kemp04, Rumb91, Meie01]
- Es gibt keine Superklasse, jede Subklasse enthält alle Attribute der Superklasse. Diese Umsetzung ist nur sinnvoll, wenn die Superklasse wenige Attribute enthält, die Subklassen hingegen viele. Hier entstehen zwei Probleme: zum einen ist es schwer die Subklasse zu finden in der das Element enthalten ist, zum anderen ist ein UNIQUE für die Superklassenattribute über alle Subklassen hin gesehen nicht möglich, das heißt eine disjunkte Vererbung kann wiederum nicht ohne zusätzliche Constraints gesichert werden. [Berg97, Rumb91] Es muss überprüft werden ob zwei Subklassen denselben Primary Key enthalten.
- Alle an der Generalisierung beteiligten Klassen werden in der Superklasse zusammengefasst. Die Attribute der Subklassen sind hier nicht eingabepflichtig, es muss jedoch abgesichert werden, dass nur Attribute einer Subklasse verwendet werden. Dies kann beispielsweise über ein zusätzliches Attribut erfolgen. Diese Art der Umsetzung ist nur sinnvoll wenn wenige Subklassen bestehen und diese nur wenige Attribute beinhalten. [Rumb91]

Beispiel

In Abbildung 23 wird die in dem eingangs dargestellten Klassendiagramm enthaltene Generalisierung ohne weitere Einflüsse durch andere Beziehungen dargestellt. Hierbei handelt es sich um die Klasse Personal sowie die dazugehörigen Subklassen Professor und Assistent. Die Subklassen werden durch die Hierarchieebene unterschieden. In dem Beispiel werden die einzelnen Elemente näher bezeichnet.

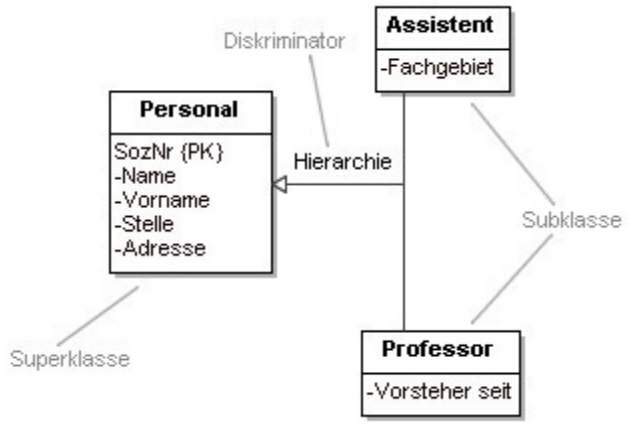
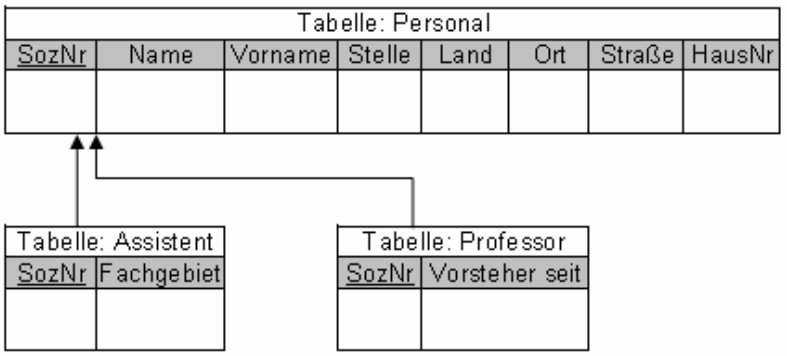


Abbildung 23: Beispiel Generalisierung

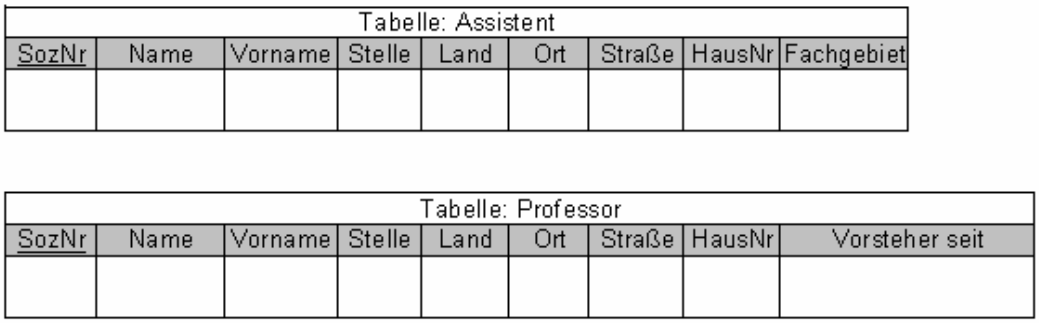
Für das Beispiel aus Abbildung 23 ergeben sich auf Grund der vorigen Betrachtungen drei verschiedene Varianten der Darstellung. Die Unterschiede sind in Abbildung 24 ersichtlich. Variante 1 zeigt die Repräsentation durch Tabellen zu allen Klassen. Die Beschränkung auf die Subklassen wird in Variante 2 umgesetzt. In Variante 3 wird die Integration der Subklassen in die Superklasse gezeigt.

Zu erkennen ist hier, dass in der Darstellung aller Klassen wie in Variante 1 das Problem besteht, dass eine Instanz durch die tabellarische Darstellung in mehreren Subklassen enthalten sein könnte. Dies müsste durch ein zusätzliches Attribut in der Superklasse abgesichert werden. Das Problem der Umsetzung in den Subklassen ist hier eindeutig zu erkennen. Die Superklasse hat hier relativ viele Attribute, die durch diese Umsetzung redundant gespeichert werden. Die Abbildung aller Subklassen in der Superklasse lässt bereits hier erkennen, dass diese Umsetzung relativ große Tabellen nach sich zieht.

Variante 1:



Variante 2:



Variante 3:

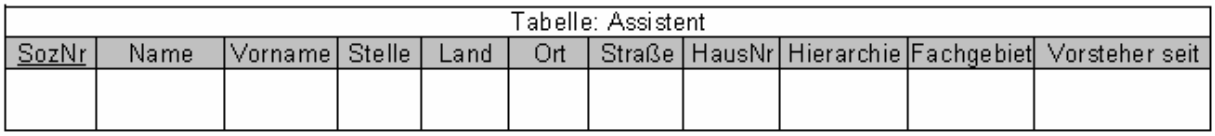


Abbildung 24: Mapping von Generalisierungen

Einschränkungen

Einschränkungen bzw. Zusicherungen werden als Erweiterung zu den herkömmlichen UML-Elementen angesehen. Zu jedem Element können beliebig viele Einschränkungen hinzugefügt werden. Diese können in OCL (siehe Kapitel 3.2.1), natürlicher Sprache oder in jeder anderen Sprache notiert werden. [Hitz05] Aber auch Stereotype und Eigenschaftswerte sind als Einschränkungen zu verstehen. Sämtliche Zusicherungen können die Inhalte und Zustände sowie die Bedeutung von Objekten einschränken. Außerdem besteht die Möglichkeiten Vor- oder Nachbedingungen für Funktionen festzulegen. [Oest01] Dies ist jedoch für den

Bereich des logischen Entwurfs nicht von Bedeutung da hier nur eine statische Struktur generiert wird.

Eine bekannte Zusicherung stellt die XOR-Einschränkung dar. Diese besagt, dass eine Klasse mit mehreren Beziehungen auf Instanzebene nur eine davon eingehen kann. Dargestellt wird dies durch eine die betreffenden Assoziationen verbindende strichlierte Linie auf der {ODER} bzw. {XOR} notiert wird. Bedingung für den Einsatz dieser Einschränkung ist, dass die betroffenen Assoziationen von einer gemeinsamen Klasse ausgehen. [Hitz05]

Am Institut für Data & Knowledge Engineering wird eine spezielle Form an Zusicherungen {ID} zur Identifikation des in den relationalen Tabellen als Primary Key zu verwendenden Attributs. Ein Schlüssel kann aus einem aber auch aus mehreren Attributen bestehen. Wenn nur ein Attribut zur Identifikation der einzelnen Objekte ausreicht wird die Zusicherung {ID} direkt beim Attribut angeführt. Diese Notation ist jedoch nicht möglich wenn es sich um einen zusammengesetzten Schlüssel handelt. Um dies darzustellen wird in der Nähe der Klasse eine Zusicherung in Form {ID(Attribut1,Attribut2,…)} angeführt. Dadurch wird die als Schlüssel zu verwendende Attributkombination definiert.

Diverse Einschränkungen sowie Kardinalitätsbeschränkungen mit einem Minimal- und einem Maximalwert lassen sich in den Tabellen nicht direkt darstellen, da hier nur Bedingungen angegeben werden können die ohne Transaktion realisiert werden können. [Jaro02] Allerdings bietet SQL durch benutzerdefinierte Integritätsregeln und Trigger (diese sind allerdings erst ab SQL:1999 verfügbar) die Möglichkeit einen Großteil der Einschränkungen nachzubilden. Wertebereichseinschränkungen können durch CHECK-Regeln bezüglich des eingeschränkten Attributs umgesetzt werden. In Assertions können auch mehrere Tabellen über normale SQL-Abfragen angesprochen werden. Über Trigger können Berechnungs- und Überprüfungsfunktionen realisiert werden. Beispielsweise kann verhindert werden, dass Werte in falscher Weise abgeändert werden. [Kemp04]

Beispiel

Das in Abbildung 25 gezeigte Beispiel einer XOR-Einschränkung wurde ebenfalls dem Klassendiagramm entnommen. Hier werden die Klassen LVA und Student dargestellt, wobei ein Student auch die Rolle Tutor einnehmen kann. Ein Student kann an einer LVA als Hörer teilnehmen, in der Rolle Tutor kann er die LVA betreuen. Zu berücksichtigen ist, dass nur eine der beiden Beziehungen eingegangen werden kann, dass heißt ein Student kann eine LVA entweder als Tutor betreuen oder daran teilnehmen. Dieser Umstand wird durch die XOR-Einschränkung gekennzeichnet.

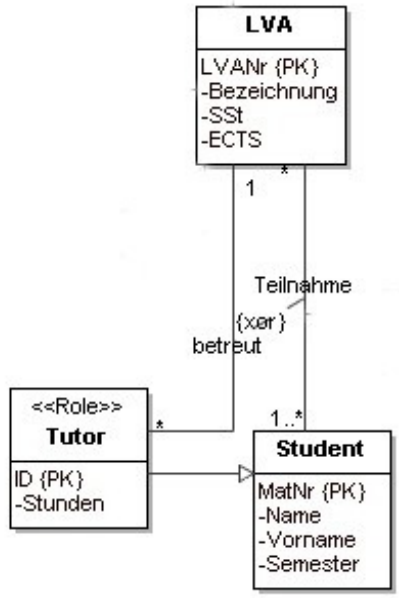


Abbildung 25: Beispiel XOR-Einschränkung

Wie bereits erwähnt kann dieses Konstrukt nicht direkt in die Tabellen überführt werden. Die Tatsache, dass nur eine der beiden Beziehungen eingegangen werden kann muss durch einen eigenen Constraint realisiert werden. Dieser muss absichern, dass entweder eine Beziehung zwischen der Tutoren-Rolle eines Studenten und der LVA besteht oder eine zwischen dem Studenten selbst und der LVA.

Geschachtelte Klassen

Abbildung 26: Darstellung Innere Klasse

Darunter versteht man Klassen, die innerhalb anderer Klassen deklariert werden (Abbildung 26). Im Grunde genommen gibt es jedoch keine definierte Notation zur Darstellung dieses Elements. [Hitz05]

Bezüglich der Abbildung in relationalen Datenbanken ist festzustellen, dass in der Literatur keine expliziten Angaben gemacht werden. Allerdings lässt sich durch das Konstrukt an sich einiges ableiten. Innere Klassen könnten unter Anderem mit einem komplexen Attribut verglichen werden, das heißt die Innere Klasse müsste in einer eigenen existenzabhängigen Tabelle dargestellt werden. Da in einer Tabelle mehrere Einträge enthalten sein können, muss der Primärschlüssel der umschließenden Klasse Teil des Schlüssels der Inneren Klasse sein. Zusätzlich muss ein weiteres Attribut enthalten sein um mehrere Einträge zu ermöglichen. Bei der Abbildung von Inneren Klassen in relationalen Datenbanken ist in jedem Fall ein Informationsverlust zu beobachten. In UML ist eine Innere Klasse nur über die sie umschließende Klasse zu erreichen, in relationalen Datenbanken hingegen ist diese Beschränkung nicht möglich.

Rollen

Dieses Konzept erlaubt einem Objekt mehrere verschiedene Rollen zur Laufzeit anzunehmen und wieder abzulegen. Es können gleichzeitig verschiedene Rollen angenommen werden, auch mehrere Varianten der gleichen Rolle. Die verschiedenen möglichen Rollen, die eine Klasse annehmen kann, müssen nicht eindeutig voneinander abgegrenzt werden. Sie können die Eigenschaften der Klasse genauer definieren. [Hitz05]

Das Konzept der Rollen ermöglicht eine dynamische Vererbung, das heißt zusammen mit den Rollen werden die jeweiligen Eigenschaften und Funktionen angenommen. Um das dynamische Annehmen und Ablegen der Rollen zu ermöglichen müssen alle Klassen des Konstrukts als Tabellen in der Datenbank enthalten sein. Tupel in den Tabellen der Rollen können nur existieren wenn das entsprechende Tupel der Ausgangsklasse vorhanden ist. Daher muss gewährleistet werden, dass Rollen zusammen mit dem Tupel der Ausgangsklasse gelöscht werden. Da ein explizites Element im Gegensatz zu Generalisierungen auch mehrere Rollen annehmen kann, darf hier der Schlüssel der Rolle nicht auf den Schlüssel der Superklasse beschränkt werden, genauer gesagt reicht ein normaler Schlüssel zur Identifikation des Tupels aus, wobei der Fremdschlüssel als normales Attribut angesehen werden kann.

Beispiel

In dem vorgestellten Universitätsbeispiel wird das Rollenkonzept in Bezug auf den Studenten umgesetzt. Der entsprechende Ausschnitt wird in Abbildung 27 dargestellt. Studenten können die Rollen Tutor und ÖH-Mitglied annehmen. Keine dieser Rollen muss angenommen werden, ein Student kann jedoch auch beide Rollen gleichzeitig annehmen. Es ist aber auch möglich, dass eine Rolle öfter angenommen wird, beispielsweise kann ein Student mehrfach die Rolle Tutor einnehmen, das heißt bei verschiedenen LVAs Tutor sein.

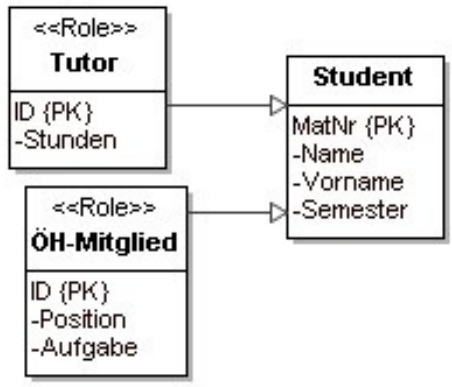


Abbildung 27: Beispiel Rollen

Das hier dargestellte Beispiel ist in drei Tabellen aufzulösen (Abbildung 28) wobei jede dieser Tabellen einen eigenen Primary Key hat. Die beiden die Rollen darstellenden Tabellen enthalten einen Fremdschlüssel der auf die Tabelle Student verweist. Das Löschverhalten dieser beiden Tabellen ist derart zu definieren, dass sie gelöscht werden wenn das entsprechende Tupel in der Studenten Tabelle gelöscht wird.

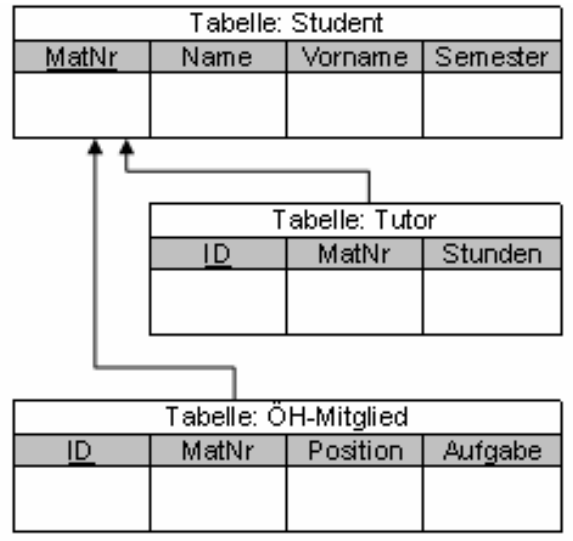


Abbildung 28: Mapping Rollen

3.3.3 Überleitung unter Berücksichtigung mehrerer Beziehungen

Wie man sieht, gibt es wenn nur ein Beziehungstyp betrachtet wird verschiedene Möglichkeiten den objektorientierten Typ aus UML in Relationen zu überführen. Komplexer wird die Überleitung allerdings dann, wenn Kombinationen verschiedener Beziehungselemente aufeinander treffen. Um die daraus resultierenden Auswirkungen festzustellen, werden hier verschiedene Kombinationen anhand der Möglichkeiten bei Betrachtung einzelner Beziehungstypen untersucht.

Generalisierung in mehreren Ebenen

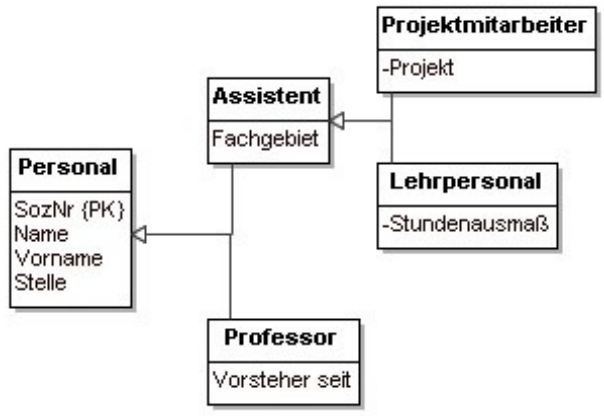


Abbildung 29: Darstellung Generalisierung in mehreren Ebenen

Dieses Beispiel (Abbildung 29) zeigt die bereits bekannte Generalisierung, das Personal allgemein sowie eine Unterteilung in Professoren und Assistenten. Für diese Betrachtung wurde die Klasse Assistent weiter spezialisiert, es gibt Projektmitarbeiter welche durch das jeweilige Projekt genauer definiert werden. Das Lehrpersonal wird zu einem bestimmten Stundenmaß für den Unterricht eingesetzt. Wie bereits erklärt gibt es bei einer Betrachtungsebene verschiedene Möglichkeiten mit einer Generalisierung umzugehen. Diese können theoretisch auch bei mehreren Ebenen angewandt werden. Wie aus diesem Beispiel ersichtlich entstehen bei mehreren Ebenen jedoch gegenüber einer Betrachtungsebene teilweise zusätzliche Probleme bzw. Komplikationen:

Integration der Subklassen in die Superklasse

Wenn man sämtliche Klassen in die Oberklasse integriert bedeutet dies, dass die einzelnen Ausprägungen der untersten Ebene in Assistent integriert werden. In weiterer Folge werden diese sowie die Professoren in die Tabelle für Personal allgemein integriert. Das heißt selbst wenn man nur von den Klassen im gezeigten Beispiel ausgeht, werden bereits 5 verschiedene Klassen in einer Tabelle dargestellt. Die Tabelle (Abbildung 30) wird extrem umfangreich, die Unterscheidung der einzelnen Klassen und die Absicherung, dass nur bestimmte Attribute den einzelnen Subklassen entsprechend verwendet werden, wird extrem kompliziert. Hier wird die Identifizierung der verwendeten Subklassen durch zusätzliche Attribute (Hierarchie, Art) bewerkstelligt. Dadurch wird die Tabelle allerdings noch umfangreicher. Außerdem hat diese Umsetzung zur Folge, dass in den Tabellen sehr viele NULL-Werte enthalten sind. Ein Objekt der Klasse Professor würde generell vier leere Felder aufweisen. Daraus lässt sich schließen, dass diese Umsetzung zwar theoretisch möglich aber praktisch dennoch wenig sinnvoll ist.

Tabelle: Personal													
SozNr	Name	Vorname	Stelle	Land	Ort	Straße	HausNr	Hierarchie	Fachgebiet	Vorsteher seit	Art	Projekt	Stundenausmaß

Abbildung 30: Mapping von Generalisierungen in mehreren Ebenen

Integration der Superklasse in die Subklasse und
Gesonderte Tabellen je Klasse

Hier sind bei mehreren Betrachtungsebenen genau dieselben Auswirkungen zu beobachten wie bei einer Betrachtungsebene. Daher werden diese hier nicht gesondert behandelt.

Generalisierung und Assoziation

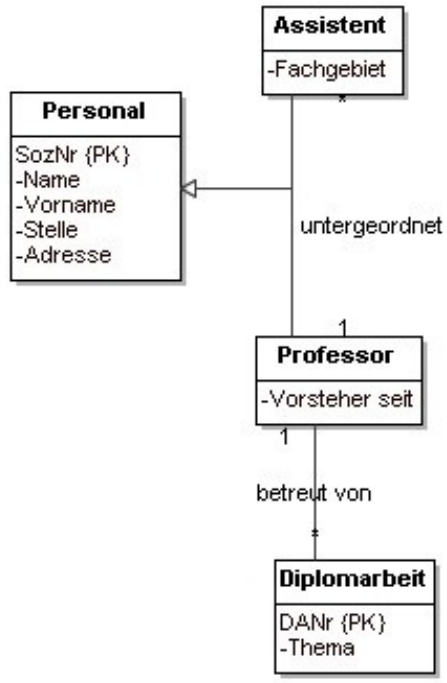


Abbildung 31: Darstellung Generalisierung und Assoziation

Hier wird von dem bereits für die Generalisierung verwendeten Beispiel ausgegangen. Das Personal wird in den Klassen Professor und Assistent spezialisiert. Zusätzlich besteht eine Beziehung zu einer anderen Klasse, der Diplomarbeit, diese Beziehung betrifft jedoch nur die Professoren, das heißt die zweite Subklasse ist nicht geht keine derartigen Beziehungen ein. (Abbildung 31) Hier sind je nach Variante unterschiedliche Auswirkungen zu beobachten.

Integration aller Subklassen in die Superklasse:

Hier sind die gleichen Auswirkungen zu erkennen wie bei der Generalisierung über mehrere Ebenen. Zusätzlich muss die Beziehung zur Klasse Motor umgesetzt werden, da es sich um eine 1:N Beziehung handelt kann sie über eine Koppeltabelle aber auch durch einen Foreign Key in der Tabelle Diplomarbeit umgesetzt werden. In beiden Fällen müsste ein Fremdschlüssel auf die Klasse Professor verweisen, die aber, wie in Abbildung 32 ersichtlich, durch die Integration in die Superklasse nicht mehr existiert. Auch wenn diese Umsetzung theoretisch möglich ist, verlangt sie in der Praxis jedoch die „Umleitung“

der Beziehung bzw. des Foreign Keys sowie zusätzliche Constraints um die korrekte Anwendung zu sichern. Da es sich somit um eine relativ komplexe und Fehleranfällige Umsetzung handelt sollte diese Variante vermieden werden. Eine 1:N Beziehung mit dem höherwertigen Ende an der Klasse Professor wäre einfacher umzusetzen, der Fremdschlüssel müsste mit in die Superklasse integriert werden. Bei komplexeren Systemen führt dies allerdings zu sehr umfangreichen Tabellen.

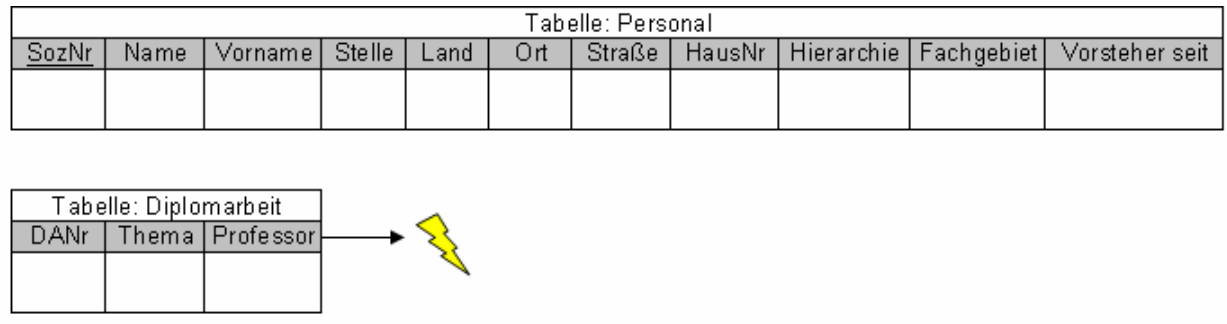


Abbildung 32: Mapping von Generalisierung und Assoziation

Integration der Superklasse in die Subklassen:

Diese Möglichkeit weist dieselben Charakteristika auf wie bei der Betrachtung einzelner Beziehungen. Geht man jedoch von einer Beziehung zur Superklasse aus entstehen dieselben Probleme wie beim gegenwärtigen Beispiel durch die Integration in der Superklasse. Die Tabelle auf die der Fremdschlüssel verweisen sollte existiert nicht. Erfolgt eine Umsetzung durch Umleitung der Beziehung auf die Subklassen müsste der Fremdschlüssel der Beziehung auf alle Subklassen verweisen können, dies ist jedoch nicht möglich.

Gesonderte Tabellen für alle Klassen

Diese Möglichkeit weist dieselben Charakteristika auf wie in einer Betrachtungsebene.

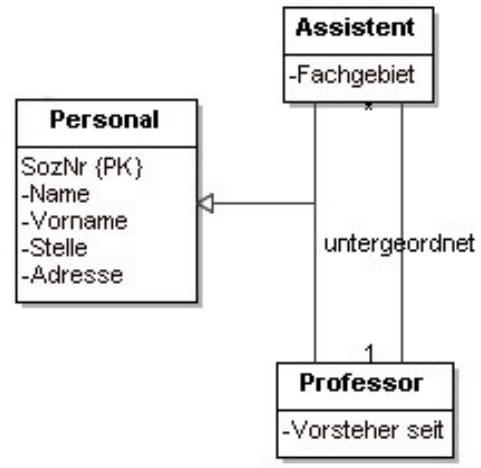


Abbildung 33: Darstellung Generalisierung und Assoziation zwischen den Unterklassen

In diesem Beispiel wird der Mitarbeiterstamm eines Universitätsinstituts dargestellt (Abbildung 33). Sämtliche Mitarbeiter des Instituts sind Personal, diese werden unterteilt in Professoren und Assistenten, wobei Professoren von den Assistenten als Chef angesehen werden. Die daraus resultierenden Änderungen werden hier beschrieben.

Integration der Subklassen in die Superklasse

Durch die Integration der Subklassen müsste die Beziehung zwischen den Klassen geändert werden, da die Klassen Professor und Assistent nicht mehr existieren. Dies würde bedeuten, dass aus der binären Assoziation eine rekursive Assoziation entsteht. Allerdings muss abgesichert werden, dass die dadurch verbundenen Objekte den entsprechenden Klassen angehören müssen. Es darf dadurch keine Beziehung zwischen zwei Assistenten oder zwei Professoren entstehen. Das heißt diese Umsetzung ist zwar theoretische unter Verwendung von Constraints zur Konsistenzsicherung möglich, ist aber sehr komplex und kann daher zu erheblichen Fehlern führen. Aus diesem Grund sollte eine derartige Umsetzung nicht durchgeführt werden. Das Problem des leeren Verweises sowie die theoretische Lösung werden in Abbildung 34 vorgestellt.

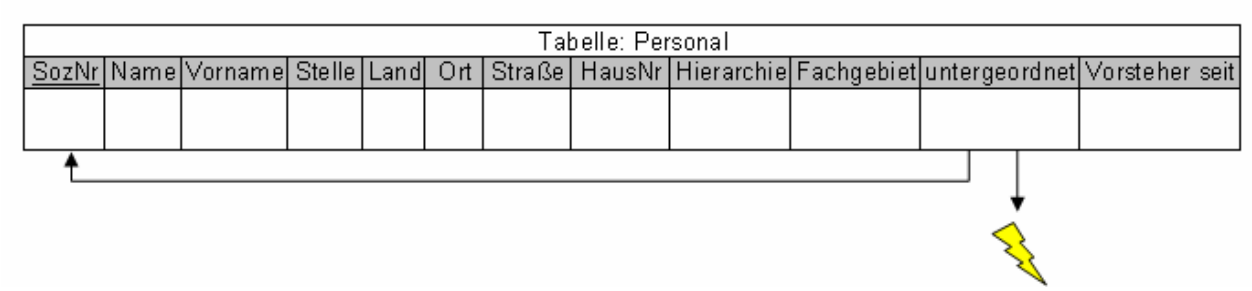


Abbildung 34: Mapping von Generalisierung und Assoziation zwischen den Unterklassen

Integration der Superklasse in die Subklassen

Durch die alleinige Verwendung von Subklassen entstehen keine besonderen Probleme gegenüber der Umsetzung auf einfacher Ebene. Die Assoziation wird wie bei zwei normalen Klassen umgesetzt.

Gesonderte Tabellen für alle Klassen

Hier entstehen ebenfalls keine weiteren Probleme, es erfolgt eine normale Umsetzung der Assoziation.

Mehrere Assoziationen

Abbildung 35 zeigt eine Kombination mehrerer Assoziationen: LVAs werden von jeweils einer Instanz aus Personal abgehalten wobei diese mehrere LVAs abhalten kann. Prüfungen werden von 1-3 Instanzen aus Personal abgehalten wobei diese wiederum mehrere Prüfungen abhalten können.

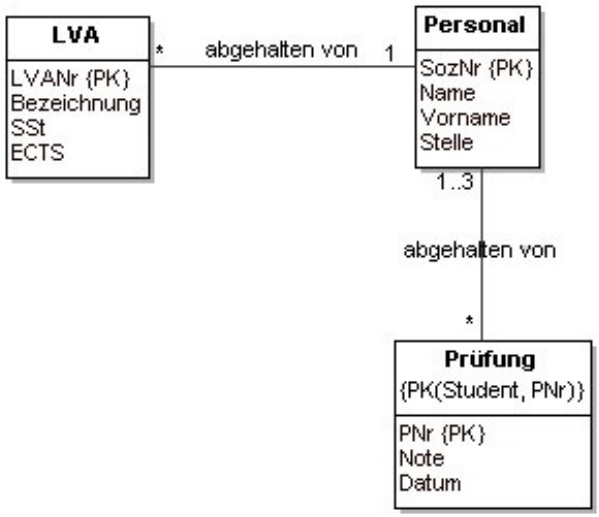


Abbildung 35: Darstellung mehrere Assoziationen

Berücksichtigt man die verschiedenen Möglichkeiten bei einer einzelnen Beziehung ist hier festzustellen, dass trotz Kombination mehrerer Beziehungen mit jeweils zumindest einseitig höherwertigen Multiplizitäten sämtliche Möglichkeiten angewendet werden können.

In dem eingangs geschilderten Beispiel ist jedoch auch eine Kombination von Assoziationen enthalten, wobei eine davon eine 1:1 Multiplizität aufweist (Abbildung 36). Jeder Student schreibt eine Diplomarbeit, diese wird von einem Professor betreut, ein Professor kann auch mehrere Diplomarbeiten parallel betreuen.

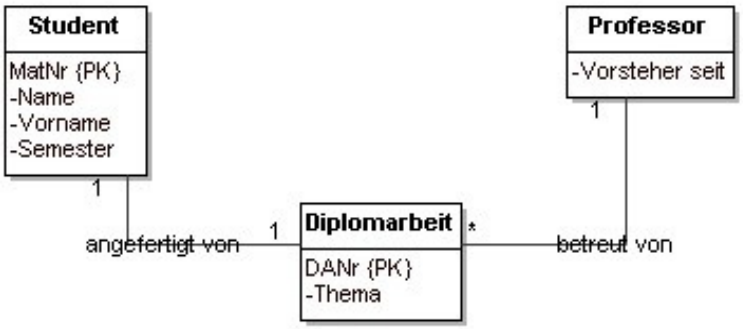


Abbildung 36: Darstellung mehrere Assoziationen mit einer 1:1 Multiplizität

Betrachtet man diese einfache 1:1 Beziehung, wobei von einem der Objekte eine Beziehung zu einem dritten Objekt besteht, ist festzustellen, dass dies tatsächlich Auswirkungen auf die Umsetzungsvarianten hat. Durch die zuvor geschilderten Abbildungsalternativen kann eine der an der 1:1 Beziehung beteiligten Klassen in die

jeweils andere integriert werden. Durch die Bedeutung des hier gezeigten Beispiels ist nur eine Integration der Diplomarbeit in die Studententabelle sinnvoll.

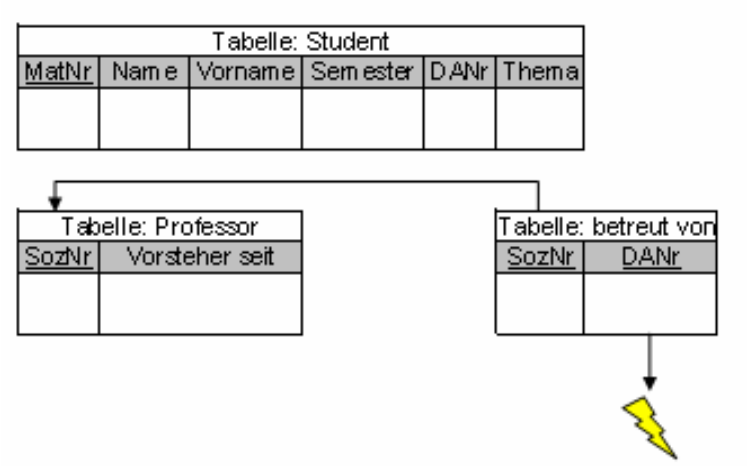


Abbildung 37: Mapping von mehrere Assoziationen mit einer 1:1 Multiplizität

Durch die Umsetzung mittels Integration der Klassen würde die Tabelle Diplomarbeit nicht mehr existieren. Diese geht jedoch auch eine Beziehung mit Professor ein. Durch die in dem Beispiel angeführten Kardinalitäten ist eine Umsetzung mittels Fremdschlüssel in der Tabelle Diplomarbeit möglich, da diese jedoch durch die Integration in Student nicht existiert, muss auch der Fremdschlüssel in die Tabelle Student übertragen werden. In diesem Fall stellt dies eine relativ einfache Umsetzung dar. Würde man jedoch die 1:N Beziehung zwischen Diplomarbeit und Professor mittels Koppeltabelle umsetzen, würde die Tabelle auf die einer der Fremdschlüssel verweisen müsste nicht existieren. In Abbildung 37 ist das eben genannte Problem graphisch dargestellt.

Assoziation und Assoziationsklasse

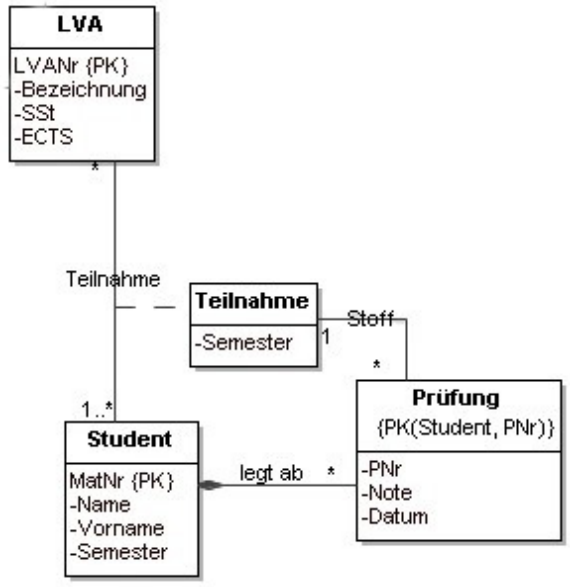


Abbildung 38: Darstellung Assoziation und Assoziationsklasse

Der Ausschnitt aus dem durchgehend betrachteten Klassendiagramm in Abbildung 38 zeigt Studenten die an verschiedenen LVAs teilnehmen und verschiedene Prüfungen ablegen, wobei diese jeweils an die Teilnahme an der entsprechenden LVA gekoppelt sind.

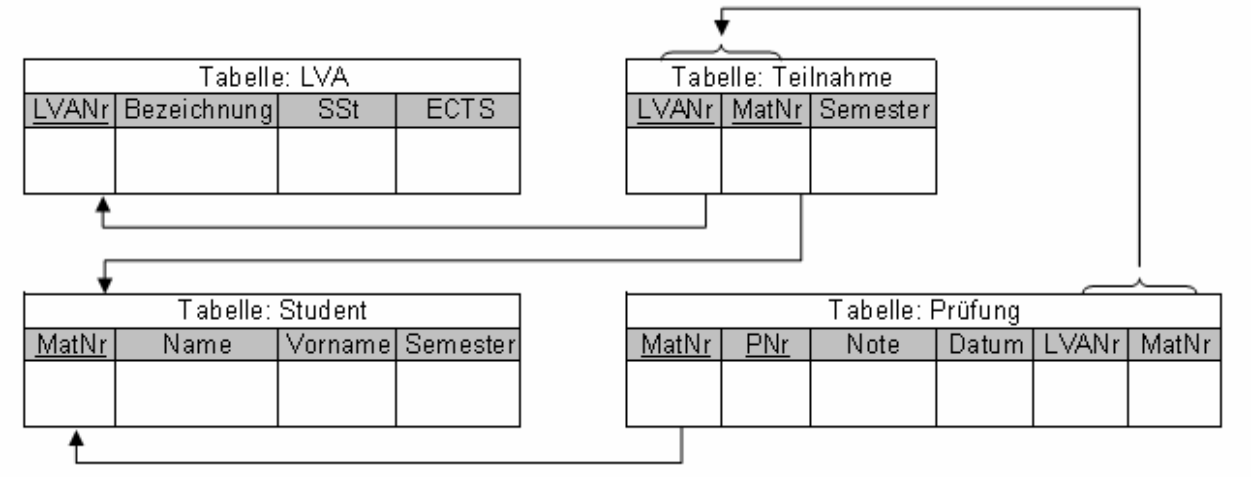


Abbildung 39: Mapping von Assoziation und Assoziationsklasse

Die entsprechenden Tabellen werden in Abbildung 39 dargestellt. Obwohl hier mehrere Beziehungstypen aufeinander treffen, sind gegenüber den Möglichkeiten bei einer Betrachtungsebene keine Änderungen festzustellen. Die Integration der

Foreign Keys wird nach den geltenden Regeln durchgeführt. Auch eine Umsetzung mittels Koppeltabellen würde keinen Einfluss auf die Umsetzung der verschiedenen Beziehungen haben.

Komposition und Assoziation

In dem Beispiel bezüglich der Assoziation und der Assoziationsklasse (Abbildung 38) wird auch eine Kombination von Komposition und Assoziation dargestellt. Laut der zuvor definierten Regel muss der Primary Key der Klasse Student in den der Klasse Prüfung aufgenommen werden. Wie man bei den entsprechenden Tabellen in Abbildung 39 sieht hat die Komposition zwar Auswirkungen auf die Attribute der Klasse, es können jedoch alle anderen Beziehungen unabhängig davon umgesetzt werden.

Generalisierung und Komposition

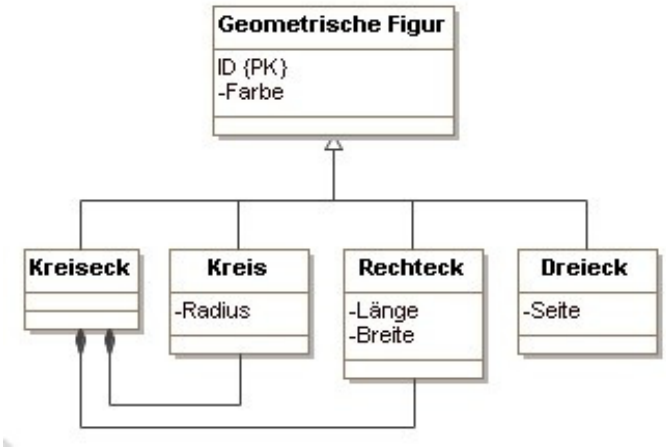


Abbildung 40: Darstellung Generalisierung und Komposition [Oest01, S. 59]

Dieses Beispiel (Abbildung 40) zeigt die Vererbungshierarchie von geometrischen Basisfiguren. Außerdem ist auch ein Kreiseck angeführt. Dieses besteht aus einem Kreis und einem Rechteck, dieser Sachverhalt wird über eine Komposition dargestellt. Aus diesem Beispiel ergeben sich einige Probleme beim Einsatz der Möglichkeiten für eine Betrachtungsebene.

Integration der Subklassen in die Superklasse

Wenn in diesem Fall die Subklassen in die Superklasse aufgenommen werden ist zu beachten, dass die Komposition auf Teile verweist, die nicht als eigene Tabellen bestehen. Dies würde bedeuten, dass die Komposition ebenfalls auf die Superklasse umgelegt werden muss. Das heißt ein Objekt der Superklasse verweist auf zwei andere Objekte derselben Klasse. Da die Komposition einen Fremdschlüssel in den Teilen verlangt der in den eigenen Primärschlüssel aufgenommen wird ist diese Möglichkeit nicht durchführbar. Durch diese Umsetzung würden in diesem Beispiel einige Objekte der Superklasse einen Primärschlüssel mit einem Attribut aufweisen, Objekte der in die Superklasse integrierten Klasse Kreiseck müssen jedoch einen Primärschlüssel mit drei Attributen haben. Abbildung 41 zeigt die Umsetzung des Diagramms aus Abbildung 40 erst ohne Berücksichtigung des Kreisecks. Im unteren Abschnitt wird gezeigt wie die Tabelle aussehen müsste um das Kreiseck mit zu berücksichtigen.

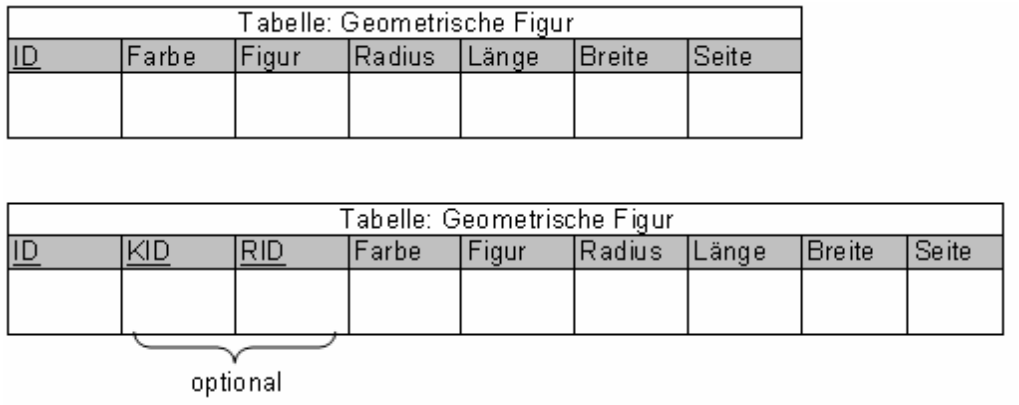


Abbildung 41: Mapping von Generalisierung und Komposition

Neben den Problemen mit dem Primary Key tritt ein zusätzlicher Konflikt auf. Die Attribute bezüglich des Rechtecks und des Kreises werden beim Kreiseck gemeinsam verwendet. Wird jedoch ein Rechteck oder ein Kreis abgebildet muss die Verwendung der jeweils anderen Attribute ausgeschlossen werden.

Integration der Superklasse in die Subklassen und

Gesonderte Tabelle je Klasse

Diese Umsetzungen rufen dieselben Probleme hervor wie bei einer Betrachtungsebene.

Fazit

In der bisherigen Ausführung wurden zwar nicht alle möglichen Kombinationen von Beziehungstypen betrachtet, die fehlenden stellen jedoch nur Substitute der bisher analysierten Elementkombinationen dar. Die grundlegenden Auswirkungen der verschiedenen Beziehungstypen untereinander sind ziemlich klar zu erkennen. Durch die gemachten Beobachtungen lassen sich vier generelle für die vorliegende Arbeit relevante Bedingungen ableiten:

1. Jede Klasse von der mehr als eine Beziehung ausgeht muss in eine relationale Tabelle übergeleitet werden, die Integration in eine andere wird nicht erlaubt.
2. Kompositionen verändern den Primary Key der Teile-Tabellen, in auf diese Teile verweisenden Tabellen muss dies berücksichtigt werden.
3. Integration von Generalisierungen in mehreren Ebenen in eine gemeinsame Tabelle führen zu unverhältnismäßig großen Tabellen, daher soll diese Umsetzungsvariante nur bei einer Generalisierungsebene erlaubt sein.
4. Umsetzungsalternativen ohne Integration der Tabellen beeinflussen einander nicht, das heißt sie können unabhängig voneinander umgesetzt werden.

Die Punkte 2 und 4 beeinflussen die Erstellung der Tabellen in jedem Fall. Die beiden anderen Punkte schließen zwar generell mögliche Alternativen aus, diese werden jedoch in der vorliegenden Arbeit ebenfalls als Basis für die Konzeption des zu entwickelnden Expertenmoduls herangezogen.

4 Konzeption des Expertenmoduls

In diesem Abschnitt der Arbeit werden verschiedene Möglichkeiten zum Vergleich der UML-Angabe mit den vom Studenten erstellten relationalen Tabellen, dem erzeugten SQL-DDL Skript, vorgestellt. Ausgegangen wird von einem UML-Klassendiagramm das als Angabe für die Aufgabe dient. Dieses wird dem Studenten in graphischer Darstellung mit zusätzlichen Informationen präsentiert, allerdings liegt es auch in Form eines XMI-Dokumentes vor. Der Student entwirft das entsprechende SQL-DDL Skript welches die dazugehörigen Tabellen erzeugt. Aufgrund der beiden unterschiedlichen zu Grunde liegenden Dokumente und der verschiedenen Abbildungsalternativen der UML-Elemente in relationalen Tabellen ist auf jeden Fall ein auf bestimmten Regeln basierender Algorithmus zur Unterstützung bzw. Durchführung der Bewertung erforderlich.

Eine besondere Schwierigkeit beim Vergleich der Angabe mit den Tabellen liegt in den frei wählbaren Namen. Bei der Erstellung der SQL-DDL kann der Student von der Namensgebung der Elemente der Angabe abweichen. Einerseits können gleiche Elemente verschiedene Namen aufweisen, andererseits können unterschiedliche Elemente mit der gleichen Bezeichnung angetroffen werden. Bei der Überprüfung der Studentenlösung müssen derartige Möglichkeiten in Betracht gezogen werden.

Eine Lösung dieses Problems ist ein einfacher Hinweis bzw. die Forderung in der Angabe, dass die Tabellennamen mit den Klassennamen und die Attributnamen mit denjenigen der UML-Darstellung übereinstimmen müssen und Abweichungen als fehlen des Elements gewertet werden. Das Problem dieser Umsetzung liegt darin, dass zwar die Namen der Basiselemente der Angabe angepasst werden können, jedoch Beziehungen bei der Umsetzung als extra Tabelle keine vorgegebenen Namen erhalten müssen. Dies kann jedoch durch die Definition von Bezeichnungen für jede verwendete Beziehung umgangen werden. Weiters besteht die Möglichkeit, dass bei der Zusammenführung von Tabellen gleichnamige Attribute aufeinander treffen und keine Vorgaben für die Lösung des Problems getroffen werden können ohne dem Lernenden unter Umständen den Lösungsweg bekannt zu geben. Gleiches gilt für die Bezeichnung von Fremdschlüsseln. Diese Probleme müssen im Verlauf der Konzeption des Expertenmoduls berücksichtigt werden.

Einleitend werden hier in Kapitel 4.1 die verschiedenen Alternativen zur Auswertung der Studentenlösung vorgestellt und die zu verwendende Variante ausgewählt. In weiterer Folge wird der Aufbau des Zielsystems konzipiert (siehe Kapitel 4.2). Hier werden auch Betrachtungen zum Erkennen der einzelnen Elemente angestellt. Außerdem werden sowohl Bewertung als auch Feedback mit einbezogen.

4.1 Alternativen der Auswertung

Aufgabe der Arbeit ist es, die vom Studenten erstellte Lösung mit den theoretischen Lösungsalternativen zu vergleichen und somit die Korrektheit der Lösung zu überprüfen. Zur Auswertung der Studentenlösung können unterschiedliche Alternativen umgesetzt werden. Generell wird eines der Vergleichsobjekte mittels eines bestimmten Algorithmus bzw. mit Hilfe eines Werkzeugs in das jeweils andere Format transformiert und die somit gleich dargestellten Objekte direkt verglichen. Dies bedeutet entweder das Reverse Engineering der Studentenlösung und den anschließenden Vergleich des dem UML-Diagramm zugrunde liegenden XMI-Dokuments oder die Generierung der relationalen Tabellen ausgehend von der UML-Angabe und den Vergleich der erzeugten Tabellen. In beiden Fällen muss die Bewertung unter Berücksichtigung der verschiedenen Abbildungsalternativen mit einbezogen werden.

4.1.1 Reverse Engineering der relationalen Tabellen

Eine Möglichkeit der Auswertung der Studentenlösung besteht in deren Reverse Engineering (siehe Abbildung 42) der Relationen und der damit verbundenen Darstellung in einem UML-Diagramm. Dies kann durch einen eigenen Algorithmus erfolgen bzw. durch ein bestehendes Tool vollzogen werden. Voraussetzung hierfür ist, dass die dahinter liegenden Regeln zur Überleitung bekannt sind. Die beiden UML-Diagramme bzw. die dahinter verborgenen XMI-Dokumente werden in weiterer Folge miteinander verglichen. Diese Variante der Ergebniskontrolle birgt jedoch gravierende Probleme in sich:

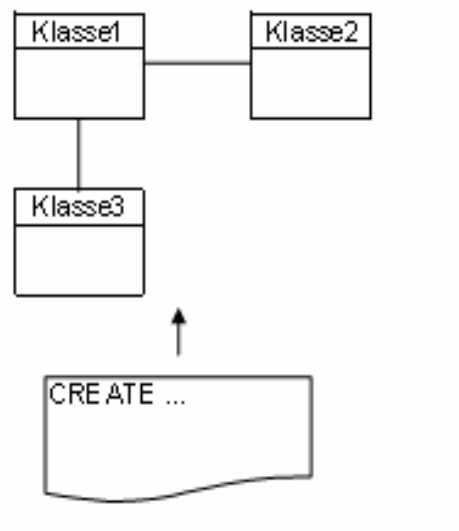


Abbildung 42: Reverse Engineering

Umsetzungsalternativen

Da verschiedene Alternativen zur Überleitung einer Tabelle in Klassen und Beziehungen bestehen, müssen bei korrekter Umsetzung durch den Studenten nicht unbedingt zwei identische Diagramme entstehen. Als Koppeltabelle umgesetzte Beziehungen könnten durch die Beschränkung der Spalten auf Fremdschlüssel als Umsetzung einer Beziehung erkannt werden und somit wieder als reine Beziehung umgesetzt werden. Diese Tatsache könnte aber auch außer Acht gelassen werden und die Beziehung daher auch in dem erzeugten UML-Diagramm als eigene Klasse dargestellt werden.

Aber auch die Anwendung verschiedener Alternativen bei der Erstellung der Relationen kann zu einer unterschiedlichen Umsetzung in dem aus der Studentenlösung generierten Diagramm führen. Beispielsweise werden vom Studenten integrierte Klassen (z. B. auf Grund einer 1:1 Beziehung) nicht externalisiert, das heißt die Attribute der integrierten Klasse bleiben auch in der visuellen Darstellung Teil derselben. Somit sind im Diagramm der Studentenlösung zwei Klassen enthalten wogegen das Diagramm der Aufgabenstellung denselben Sachverhalt durch eine Klasse repräsentiert.

Hier ist zu erkennen, dass die Überleitung der Studentenlösung keine Arbeitsersparnis nach sich zieht. Das Problem der verschiedenen Alternativen Darstellungsformen bleibt nach wie vor bestehen. Zusätzlich zu den Alternativen des relationalen Mappings müssen unter Umständen auch noch verschiedene Alternativen beim Reverse Engineering berücksichtigt werden. Daraus ergibt sich eine erhöhte Zahl an Alternativen die im Grunde genommen alle ein vom Studenten korrekt abgebildetes Konstrukt, sei es die Umsetzung einer Beziehung oder eine Klasse, verbergen. Daraus ergibt sich zwar eine Verschiebung des Problems jedoch erfolgt keine Erleichterung bei der Korrektur der Studentenlösung.

Wegen der hier angeführten Probleme ist diese Ebene zum Vergleich nicht zu empfehlen. Durch den Einsatz von Werkzeugen beim Reverse Engineering besteht eine relativ hohe Wahrscheinlichkeit, dass die tatsächliche Studentenlösung verfälscht wird. Daraus ergibt sich, dass eine an sich korrekte Lösung als falsch interpretiert oder aber auch eine fehlerhafte Lösung als korrekt angesehen werden könnte. Durch diese Betrachtung ist zu erkennen, dass eine derartige Umsetzung den eigentlichen Vergleich zu einem komplexen und fehleranfälligen Vorgang machen würde. Die eigentliche Korrektur der Aufgabe wird dadurch nicht vereinfacht. Diese Lösungsalternative wird daher nicht weiter in Betracht gezogen.

4.1.2 Erzeugung von Muster-Relationen

Da es, wie eben beschrieben, keinen Sinn macht die Studentenangabe durch Reverse Engineering an das Format der Angabe anzupassen wird hier die Überleitung des vorgegebenen UML-Diagramms in ein Relationenschema betrachtet. Diese Variante ist in Abbildung 43 skizziert. Das erstellte Schema ist in gewissem Sinne einer Musterlösung gleichzusetzen. Allerdings ist auch hier beim Vergleich zu berücksichtigen, dass für den Studenten verschiedene Möglichkeiten zur Umsetzung einzelner Elemente bestehen. Dadurch ist es nicht möglich eine einzige Lösung zu definieren, von der nicht abgewichen werden darf.

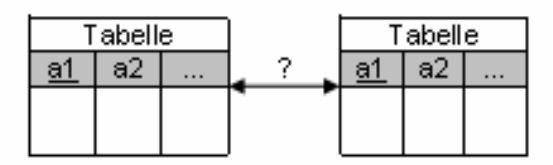


Abbildung 43: Schema Musterrelationen

Das Schema kann mit Hilfe eines Tools erstellt werden, aber auch händisch durch den Lehrenden selbst. Die erstellte Lösung dient in weiterer Folge als Vergleichsobjekt für die Bewertung der Studentenlösung. Wie bereits erwähnt reicht eine einzige Musterlösung hier jedoch nicht aus. Um die unterschiedlichen Alternativen zur Umsetzung der UML-Konstrukte zu berücksichtigen bestehen folgende Möglichkeiten:

Musterlösungen für das gesamte Modell

Um die verschiedenen Mapping-Alternativen zu berücksichtigen könnte man sämtliche erlaubte Alternativen zu unterschiedlichen Musterlösungen kombinieren. Sollten in dem Diagramm jedoch mehrere Elemente enthalten sein, die eine unterschiedliche Interpretation erlauben führt dies zu einer extrem hohen Zahl an möglichen Lösungen. Weiters lässt sich daraus nur ableiten ob das gesamte Schema korrekt ist. Welche Komponenten falsch umgesetzt wurden wäre nicht ersichtlich, somit kann eine Lösung entweder komplett richtig oder komplett falsch sein. Das heißt ein kleiner Fehler könnte dem Studenten die gesamten Punkte kosten. Zusätzlich ergibt sich aus dieser Umsetzungsvariante ein relativ hoher Speicherplatzbedarf pro Aufgabe. Dieser Umstand sollte jedoch nach Möglichkeit umgangen werden.

Teillösungen für die einzelnen Elemente

Die durch eine komplette Musterlösung auftretenden Probleme könnten durch eine gesonderte Betrachtung jedes einzelnen Elements umgangen werden. Dies bedeutet, dass zu jedem Beziehungselement die Darstellungsalternativen gespeichert werden, wenn eine Darstellung in der Studentenabgabe enthalten ist können die Alternativen außer Acht gelassen werden. Diese Variante führt zu einer weitgehend dynamischen Kombination

der Mapping-Alternativen. Somit können auch Punkte für einzelne Ausschnitte des umgesetzten UML-Diagramms vergeben werden. Dadurch, dass beim Speichern der Alternativen die Kombinationen außer Acht gelassen werden, wird auch weniger Speicherplatz benötigt. Die einzelnen Alternativen zu den Elementen können wiederum händisch oder durch Unterstützung eines Tools erstellt werden.

Hier tritt jedoch ein anderes Problem zu Tage. Die verschiedenen Umsetzungsalternativen bewirken teilweise Einschränkungen bei den direkt angrenzenden Elementen. Das heißt bei den Musterteilen muss mitberücksichtigt werden, welche Auswirkungen sie auf andere Elemente haben bzw. durch welche Umsetzungen anderer Elemente sie verändert werden. Beispielsweise kann die Umsetzung einer Beziehung die Zahl der Spalten einer Tabelle verändern da ein Foreign Key eingefügt wird. Zusätzlich muss die Punktevergabe für die einzelnen Umsetzungsvarianten notiert werden. Sämtliche Elemente müssen auf ihr Vorkommen hin überprüft werden und die korrekten Einzelteile bewertet werden.

Aus dieser Betrachtung ergibt sich, dass eine Lösung auf Ebene der relationalen Datenbanken eine exaktere Korrektur der Studentenlösung durch das System zulässt. Allerdings bedeutet diese Variante aber bei der Erstellung der Aufgaben einen relativ hohen Aufwand für den Lehrenden. Immerhin muss dieser sämtliche Alternativen erstellen, was auch mit Unterstützung von Tools einen langwierigen Prozess darstellt. Im Großen und Ganzen bedeutet dies, dass eine derartige Umsetzung zu bevorzugen ist, aber zusätzlich ein Werkzeug bzw. einen Algorithmus zur automatischen Generierung der Alternativen voraussetzt. Ausgehend von dem UML-Diagramm müssen generell alle Möglichkeiten zur Abbildung der Elemente generiert und diese zu einer weiteren Bearbeitung durch den Lehrenden zur Verfügung gestellt werden. Diesem wird die Möglichkeit geboten in die durch das System vorgeschlagene Bewertung der Alternativen einzugreifen und das mögliche Angebot an Umsetzungen aufgabenspezifisch einzuschränken.

4.1.3 Fazit

Wie eben festgestellt ist es von Vorteil ein Zielsystem zu generieren dessen Anforderungen die Studentenlösung erfüllen muss. Bisher wurde davon ausgegangen, dass die UML-Elemente in mögliche Relationen übergeleitet werden. Da überprüft werden soll inwieweit die Studentenlösung dem aufgestellten Regelsystem entspricht muss das relationale Schema, welches der Student erstellt hat, ebenfalls durch bestimmte Ableitungsschritte in eine vergleichbare Form gebracht werden.

Der Student liefert ein DDL-Skript welches die dem UML-Diagramm entsprechenden Tabellen in der Datenbank erzeugt. Diese Tabellen können für die Auswertung der Lösung aus der Datenbank ausgelesen werden, was allerdings einen doppelten Aufwand bedeutet. Ein weiteres Problem liegt darin, dass nicht alle Constraints aus der Datenbank ausgelesen werden können da Assertions nicht unterstützt werden. Da der Student eigentlich ein DDL-SQL Skript abgibt, ist daher zu überlegen ob die Auswertung der Lösung nicht von diesem Skript ausgehen sollte anstatt die in der Datenbank enthaltenen Tabellen zu betrachten. Durch diese Umsetzung bleibt die Erstellung der Relationen sowie deren erneutes auslesen zur Auswertung erspart. Daraus ergibt sich, dass die tatsächliche Auswertung nicht in Form eines Vergleichs der in der Datenbank enthaltenen Relationen erfolgt, sondern ein anders Zielsystem für den Vergleich generiert werden muss.

Wie in dem XMI-Dokument zur UML-Angabe sind auch in dem DDL-Skript sämtliche Informationen gemischt. Klassen und ihre Attribute werden gemeinsam mit den für die Beziehungen relevanten Foreign Keys gespeichert. Zusatzinformationen wie Beschränkungen des Wertebereichs der Attribute werden im Gegensatz zur UML-Darstellung nicht bei den Attributen notiert sondern in eigene Constraints verpackt. Im zu erstellenden Regelsystem müssen diese Unterschiede berücksichtigt werden. Es sollte vermerkt werden welche Konstrukte zusammen gehören. Das heißt zur Auswertung der Lösung wäre im Grunde genommen nur das Auslesen der einzelnen Abschnitte bzw. Tabellen und Constraints notwendig.

Die einzelnen Elemente werden im Anschluss auf ihre Übereinstimmung mit dem Zielsystem hin überprüft. Im Zielsystem muss vermerkt werden welche Alternativen angewendet wurden, welche Teile erfüllt sind und welche Elemente in der Lösung fehlen. Unter Berücksichtigung der zu vergebenden Punkte für die einzelnen Elemente wird dann die erreichte Punktezahl ermittelt. Dies könnte durch einen gesonderten Eintrag von erreichbaren Punkten und erreichten Punkten erfolgen. Wenn eines der Elemente der Studentenlösung nicht zugeordnet werden kann, muss dieser Umstand vermerkt und zur weiteren Betrachtung an den Lehrenden weiter geleitet werden. So bald sämtliche Elemente abgearbeitet wurden werden die erreichten Punkte hochgerechnet und als Gesamtpunktezahl zurückgegeben.

4.2 Aufbau des Zielsystems

Die Erstellung des Zielsystems geht von dem der Aufgabenstellung zu Grunde liegenden UML-Diagramm aus. Die graphische Darstellung wird durch Export mittels XMI in eine verarbeitbare textuelle Form gebracht. Da es sich bei XMI um ein XML-Format handelt können sämtliche Bearbeitungsmittel wie XPath, XSLT, etc. eingesetzt werden. Spezifikationen hierfür sind auf der Homepage des W3-Konsortiums erhältlich (http://www.w3.org). Das generierte Dokument könnte etwa mit Hilfe von XSLT-Templates formatiert und für nachfolgende Arbeitsschritte vorbereitet werden. Beispielsweise könnten die Elemente zur Weiterverarbeitung in bestimmter Weise sortiert werden. Für das weitere Vorgehen können alle Elemente ausgelesen und zur Verarbeitung bereitgestellt werden.

Die Wünsche des Lehrenden sowie Informationen für die spätere Bewertung müssen ebenfalls in dieses Zielsystem einfließen. Wichtig ist vor allem, dass sämtliche Informationen enthalten sind, die zur Überprüfung der möglichen Fehler benötigt werden. Zu beachten ist, dass die erstellte Datenbank dann korrekt ist wenn sämtliche Informationen, die in der Angabe enthalten sind, in die erstellten Tabellen übertragen werden. Dies ist dann der Fall, wenn die Informationen der entsprechenden Klasse enthalten sind sowie die durch Beziehungen hervorgerufenen Änderungen berücksichtigt wurden. Um zu gewährleisten, dass im Zielsystem sämtliche Anforderungen der Angabe enthalten sind, werden in Kapitel

4.2.1 die bei der Lösung der Aufgabe auftretenden Fehlerquellen erarbeitet. Der aus diesen Betrachtungen resultierende Aufbau des Zielsystems wird in Kapitel 4.2.2 vorgestellt.

4.2.1 Betrachtung der Fehlerquellen

Um ein entsprechendes Zielsystem zu erstellen ist es notwendig zu wissen welche Informationen enthalten sein müssen. In dem durch das UML-Tool generierten XMI-Dokument werden teilweise Informationen geboten die für die Umsetzung in das relationale Datenmodell bedeutungslos sind. Außerdem ist zu bedenken, dass die Struktur dieses Dokuments nicht unbedingt optimal für die tatsächliche Auswertung der Lösung gegliedert ist. Aus diesem Grund werden im Folgenden mögliche Fehlerquellen bei der Umsetzung der UML-Elemente in relationale Strukturen betrachtet. Um eine weitgehende Unabhängigkeit der verschiedenen Elemente voneinander zu gewährleisten werden hier Klassen und Beziehungen getrennt voneinander betrachtet.

Klassen

Um die Bedingungen für die Überprüfung der Klassen aufstellen zu können müssen die enthaltenen Elemente analysiert werden. Dem logischen Entwurf zu Grunde liegende Klassen enthalten wie in Kapitel 3.3.1 gezeigt einen Abschnitt mit dem Klassennamen und den die gesamte Klasse betreffenden Eigenschaften sowie einen weiteren Abschnitt mit den enthaltenen Attributen und ihren Zusatzinformationen. Neben einfachen Ausprägungen können auch Arrays, Listen oder komplexe Typen enthalten sein. Eines bzw. eine Kombination dieser Attribute wird als Primary Key definiert, welcher die enthaltenen Tupel eindeutig identifiziert.

Eine Klasse ist richtig umgesetzt wenn die enthaltenen Attribute sowie die gesamte Klasse betreffende Eigenschaften und Stereotype korrekt umgesetzt wurden und die Tabelle entsprechend der Vorgaben bezeichnet wurde. Die Klasse betreffende Constraints wie beispielsweise zusammengesetzte Primary Keys müssen in der Umsetzung berücksichtigt werden. Attribute sind dann in Ordnung wenn deren UML-Datentyp in den entsprechenden relationalen Datentyp übergeleitet wird sowie die

Initialwerte und Eigenschaften berücksichtigt wurden. Mehrwertige Attribute müssen entweder in verschiedene Spalten aufgeteilt oder in eigene Tabellen ausgelagert werden, diese beiden Alternativen müssen einander ausschließen damit ein Attribut nicht doppelt in der Datenbank enthalten ist. Aus dieser Betrachtung ergibt sich der in Abbildung 44 dargestellte Ableitungsbaum. Wichtig ist, dass hier noch keine Beziehungen berücksichtigt werden. Natürlich müssen nicht in jeder Klasse sämtliche hier genannten Elemente enthalten sein. Für einzelne Klassen sind daher nur die für sie relevanten Bereiche zu betrachten.

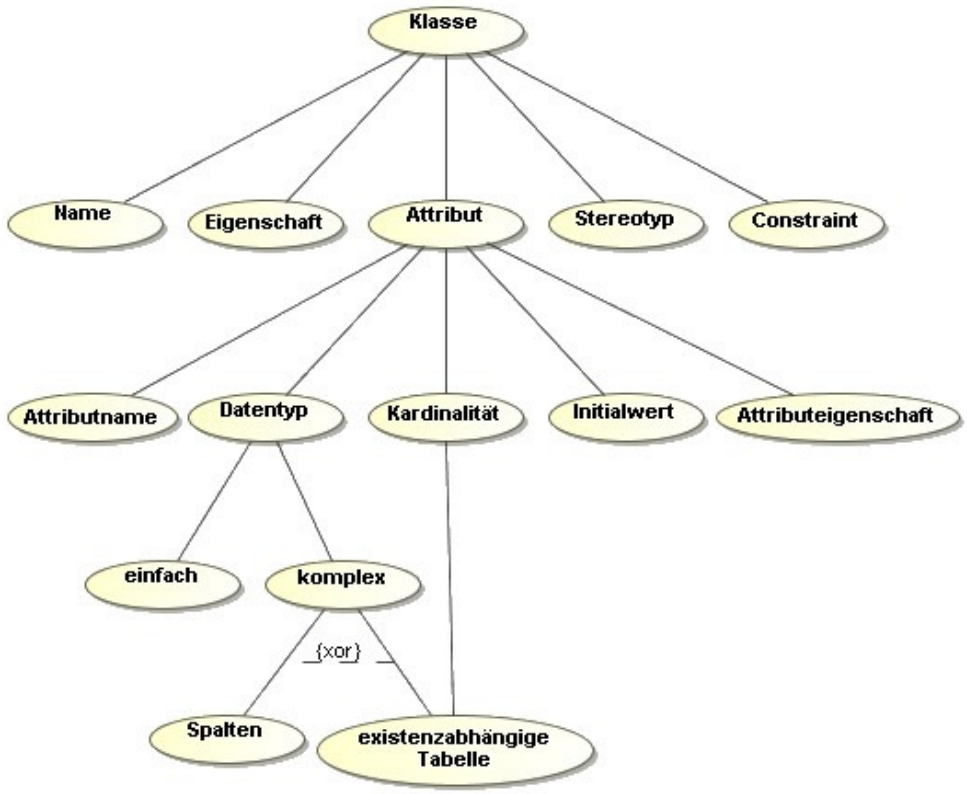


Abbildung 44: Ableitungsbaum zur Korrektur von Klassen

Beispiel

In Abbildung 45 wird daher anhand der Klasse Hörsaal beispielhaft gezeigt wie die jeweilige Auswertung zu erfolgen hat. Diese Klasse ist dann richtig umgesetzt wenn sie mit „Hörsaal“ bezeichnet wurde, keine Eigenschaften und Stereotype berücksichtigt wurden sowie die Attribute richtig umgesetzt wurden und der Primary Key SNr korrekt deklariert wurde. Für beide enthaltenen Attribute gelten die gleichen Anforderungen: dies ist zwar im Klassendiagramm nicht explizit angegeben sie

werden jedoch beide als Datentyp Number deklariert. Beide weisen eine Kardinalität 1 auf und werden daher jeweils in einer Spalte dargestellt. Sie haben keinen Initialwert und keine Eigenschaft. Der einzige Unterschied bezüglich der beiden Attribute liegt in der Bezeichnung des Attributs.

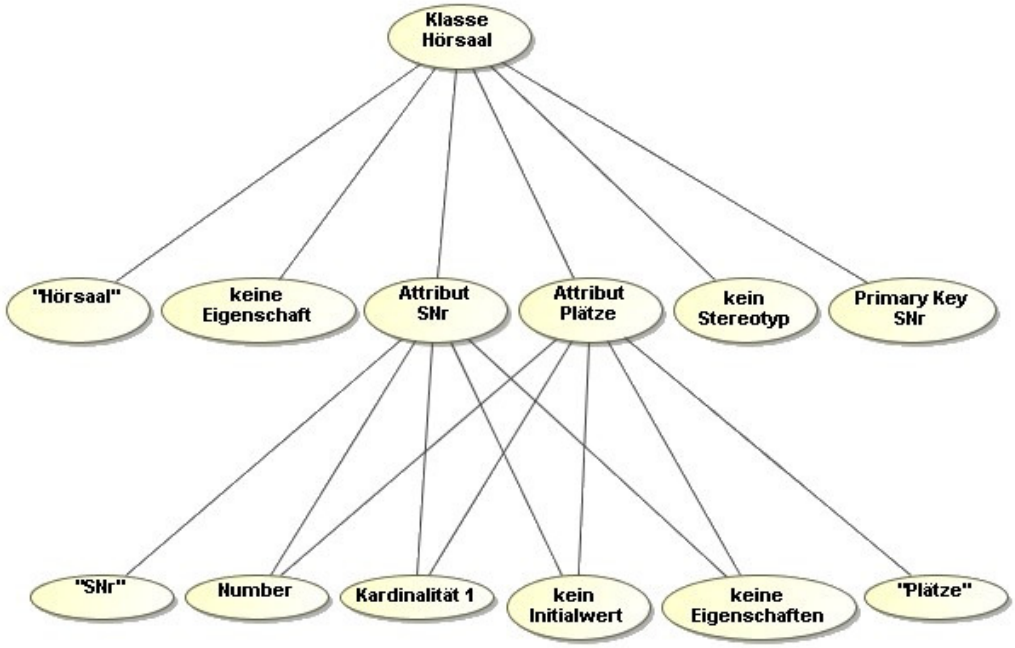


Abbildung 45: Beispiel Auswertung Klasse Hörsaal

Bei der Überprüfung von Klassen können somit verschiedene Fehler auftreten deren Auswirkungen auf die Gesamtbewertung der Klasse definiert werden müssen. Die gesamten Punkte der Klasse beispielsweise wegen einem fehlenden Initialwert bei einem Attribut abzuziehen wäre unverhältnismäßig hart. Aus der vorigen Betrachtung ergeben sich folgende Fehlerpotenziale:

- Klasse
    - Name

      Der Studierende hat die Bezeichnung aus dem UML-Diagramm nicht übernommen: da in der Angabe der Aufgabe darauf hingewiesen wird, dass die Bezeichnungen zu übernehmen sind wird die gesamte Klasse als falsch gewertet.

- Eigenschaften

  Der Studierende hat die geforderten Eigenschaften nicht umgesetzt. Gerade bei abstrakten Klassen führt dies zu einem gravierenden Unterschied. Daher muss der Fehler zu entsprechenden Abzügen bei der Punktevergabe führen.
- Stereotype

  Da von den standardisierten UML-Stereotypen keiner Einfluss auf die Datenstruktur der Datenbank hat wird keiner davon verwendet. Allerdings wird der Stereotyp <<role>> eingesetzt, dieser hat jedoch hauptsächlich Auswirkungen bei der Beziehungsumsetzung.
- Attribute

  Die gesondert angeführten Fehlerquellen bei der Umsetzung der Attribute müssen in die Bewertung der gesamten Klasse mit einfließen.
- Constraints

  Der Studierende hat den Primary Key nicht deklariert: Hierbei handelt es sich um einen schwerwiegenderen Fehler als bei den anderen Teilelementen. Das heißt dieser muss entsprechend größere Auswirkungen auf die Bewertung der Klasse haben. Auch wenn nur eines der Attribute den Schlüssel bildet ist dies als Fehler der Klasse zu werten.

- Attibute
  - Bezeichnung

    Der Studierende hat die Bezeichnung aus dem UML-Diagramm nicht übernommen: da in der Angabe der Aufgabe darauf hingewiesen wird, dass die Bezeichnungen zu übernehmen sind wird das gesamte Attribut als falsch gewertet.
  - Datentyp

    Der Studierende hat bei der Überleitung der einfachen Elemente einen falschen Datentyp gewählt. Dieser Fehler bezieht sich nur auf einen Teil des Attributes, das heißt die restlichen Anforderungen werden auf ihre Erfüllung hin überprüft. Komplexe Datentypen müssen in eine existenzabhängige Tabelle ausgelagert oder in mehrere Spalten aufgeteilt werden, wobei eine ausreichende Anzahl vorhanden sein muss. Bei Auslagerung in eine Tabelle muss der auf die eigentliche

Klasse verweisende Fremdschlüssel der Primärschlüssel der neuen Klasse sein.

- Initialwert

  Der Studierende hat den geforderten Initialwert nicht berücksichtigt. Die restlichen Teile des Attributs werden normal überprüft.

- Zusicherung

  Der Studierende hat beispielsweise die Einschränkung des Wertebereichs nicht berücksichtigt. Dies ist ebenfalls ein von den anderen Elementen der Attribute unabhängiger Fehler.

- Kardinalität

  Mehrwertige Attribute müssen in eine andere existenzabhängige Tabelle ausgelagert werden. Bei der Ausgliederung kann in der externen Tabelle durch den Studenten ein falscher Schlüssel vorgesehen werden. Beschränkungen der Kardinalität müssen durch zusätzliche Constraints umgesetzt werden

Beziehungen

Die Überprüfung der Beziehungen ist durch die verschiedenen Umsetzungsalternativen komplexer als die Auswertung der Klassenumsetzung. Wie bereits in Kapitel 3.3.2 gezeigt bestehen je Beziehungstyp bis zu vier verschiedene Möglichkeiten diesen in den relationalen Tabellen abzubilden. Die Überprüfung der Möglichkeiten ist relativ einfach solange nur eine Beziehung im Spiel ist, wenn jedoch eine Klasse von mehreren Beziehungen betroffen ist wird das Problem umfassender. Generell kann davon ausgegangen werden, dass Klassen von denen mehr als eine Beziehung ausgeht existieren müssen, das heißt sie können nicht auf Grund einer der Beziehungen in eine andere Klasse integriert werden. Dies würde zu Problemen bei der Umsetzung der Fremdschlüssel führen.

Für die Überprüfung des Mappings von Beziehungen werden die im SQL-DDL Skript vorhandenen Fremdschlüssel herangezogen. Diese können je nach Beziehungstyp in einer der beteiligten Tabellen integriert sein bzw. in einer Koppeltabelle aufscheinen. Es könnte aber auch eine der Klassen in der anderen integriert werden wodurch bei der Umsetzung der Beziehung kein spezifischer Fremdschlüssel

entsteht. Die Auflösung der im UML-Diagramm enthaltenen Beziehungen muss somit unter Berücksichtigung der verschiedenen Alternativen erfolgen. Das heißt wenn mehrere Alternativen bestehen eine Beziehung umzusetzen müssen alle diese Varianten im Zielsystem berücksichtigt werden. Um eine doppelte Umsetzung der Beziehungen zu verhindern müssen sich die verschiedenen Möglichkeiten gegenseitig ausschließen.

Bei der Überprüfung der Beziehungen entstehen jedoch auf Grund der frei wählbaren Bezeichnungen beim Mapping gewisse Schwierigkeiten. Eine Möglichkeit zur Bewältigung des Problems liegt in der Einschränkung bzw. Vorgabe der Bezeichnungen. Ohne die Bekanntgabe des Lösungsweges zu riskieren ist eine eindeutige Vorgabe unter Umständen schwierig. Koppeltabellen an sich könnten mit Hilfe des Beziehungsnamens identifiziert werden, dazu ist ein weiterer Hinweis in der Angabe notwendig. Es kann generell verlangt werden, dass bei Anwendung dieser Umsetzungsalternative der Beziehungsname als Tabellenbezeichnung verwendet wird. Die Vereinheitlichung der Namen der Foreign Keys ist allerdings nicht so einfach. Diese könnten den Rollennamen des Beziehungsendes verwenden, aber auch den Klassennamen der an der Beziehung beteiligten Tabelle. Somit ist eine andere Möglichkeit zur Kontrolle der Foreign Keys von Nöten. Eine Alternative liegt darin zu überprüfen auf welche Primary Keys die jeweiligen Fremdschlüssel verweisen. Das heißt wenn in einer Koppeltabelle die Beziehung x zwischen den Klassen A und B dargestellt wird muss diese Tabelle einerseits mit x bezeichnet werden. Andererseits müssen Foreign Keys auf die Tabellen zu den Klassen A und B vorhanden sein, das heißt ein Fremdschlüssel der den Primary Key der Klasse A referenziert und einer der auf die Klasse B verweist.

Die Integration der einen Klasse in die andere ist ein Sonderfall der Beziehungsumsetzung. Im Zielsystem muss festgelegt werden, dass diese Klasse fehlen kann, bzw. dass mindestens eine der beteiligten Klassen im Datenmodell enthalten sein muss. Dies ist die einzige Umsetzungsalternative die gravierende Änderungen in den Klassen hervorruft. Bei allen anderen Alternativen werden die Basisklassen nur durch Foreign Keys erweitert, hier werden jedoch zusätzliche Attribute in die Klasse aufgenommen. Das heißt eine der Klassen fehlt wogegen die andere umfangreicher ist, als sie durch die unabhängige Betrachtung sein sollte. Das

heißt die Klasse muss entweder nur mit ihren eigenen Attributen existieren oder ihre sowie die der einbezogenen Klasse enthalten.

Einen weiteren speziellen Fall stellen Kompositionen dar. Diese haben Einfluss auf die Attribute bzw. den Primary Key der Teile-Klassen. Wie bereits in Kapitel 3.4.2 erwähnt wird der Schlüssel der Ganzes-Klasse in den Schlüssel der Teile aufgenommen. Zusätzlich erfolgt eine Angabe bezüglich des Löschverhaltens der Klasse. Diese Aspekte müssen bei der Erstellung des Zielsystems berücksichtigt werden. Die zusätzlichen Attribute sowie deren Verhalten müssen für die Überprüfung der Klassen zur Verfügung gestellt werden. Das heißt die korrekte Umsetzung der Komposition erfolgt durch die Überprüfung der beteiligten Klassen. Nur Einschränkungen der Teile-Zahl müssen durch Constraints bewerkstelligt werden.

Wie bei der beschränkten Betrachtung von Klassen gibt es auch hier für den Studenten bestehende Fehlerquellen. Diese sind zum Teil auch von der Kardinalität der Beziehung her gegeben. Es wird jedoch auch der Einfluss anderer Beziehungen auf das betrachtete Element berücksichtigt:

- Assoziation und Aggregation:
    - Eine der Klassen wird in die andere integriert obwohl dies durch den Beziehungstyp nicht möglich ist.
    - Es erfolgt die Integration einer der Klassen obwohl dies durch den Lehrenden verboten ist.
    - Die Beziehung wird durch einen Fremdschlüssel in der falschen Klasse repräsentiert, z.B. bei einer 1:N Beziehung in der Klasse mit Multiplizität 1, das heißt ein Objekt müsste unter Umständen durch ein einziges Attribut mehrere Objekte der anderen Klasse referenzieren.
    - Der Fremdschlüssel einer zu 1 Beziehung wird nicht als UNIQUE deklariert.
    - Ein optionaler Fremdschlüssel wird als NOT NULL beschrieben.
    - Die eingeschränkte Multiplizität der Beziehung (z.B. 2..5) wurde in der Umsetzung nicht berücksichtigt, das heißt der entsprechende Constraint fehlt.

- In der erstellten Koppeltabelle liegt ein Fehler vor
  - Der Fremdschlüssel verweist auf eine falsche Klasse.
  - Die Bezeichnung der Beziehung wurde nicht berücksichtigt, die Koppeltabelle trägt einen anderen Namen.
- Bezüglich einer Beziehung wird eine Koppeltabelle erstellt aber auch ein Fremdschlüssel in einer der Basisklassen eingesetzt.
- Obwohl durch eine M:N Multiplizität die Erstellung einer Koppeltabelle gefordert wird, setzt der Studierende die Beziehung durch einen Fremdschlüssel um.

- Assoziationsklasse:
  - Der Name der Assoziationsklasse wurde im Datenmodell nicht berücksichtigt, dieser sollte die Koppeltabelle bzw. den Foreign Key bezeichnen.
  - Für die Umsetzung der Beziehung bestehen dieselben Fehlerquellen wie bei normalen Assoziationen.
  - Die Umsetzung der Attribute folgt denselben Regeln wie diejenige der Attribute normaler Klassen.
- Komposition:
  - In der Teile-Klasse wird ein anderer Primärschlüssel verwendet als in der Ganzes-Klasse.
  - Das Löschverhalten wird nicht berücksichtigt, das heißt ON DELETE CASCADE fehlt.
  - Die eingeschränkte Kardinalität wurde nicht berücksichtigt.
- Generalisierung:
  - Es erfolgt die Integration der einen Ebene in die andere obwohl eine zusätzliche Beziehung zu der integrierten Ebene besteht.
  - Die durchgeführte Integration wurde durch den Lehrenden verboten.
  - Die Klassen werden in der Superklasse zusammengefasst ohne abzusichern, dass sich die Attribute der Subklassen gegenseitig ausschließen.
  - Die Subklassen werden in die Superklasse integriert, es sind jedoch nicht alle Attribute vorhanden.
  - In den Subklassen wird nicht der Schlüssel der Superklasse verwendet.

- Die Eigenschaft {abstract} der Superklasse wird nicht berücksichtigt, es können auch Instanzen der Superklasse erstellt werden.
- Die Eigenschaft {disjunkt} der Generalisierung wird nicht berücksichtigt, es wird kein zusätzlicher Constraint, der die Verwendung desselben Primary Key in den verschiedenen Klassen ausschließt, erstellt.
- Die Eigenschaft {vollständig} wird in der Umsetzung nicht berücksichtigt, das heißt der entsprechende Constraint ist nicht vorhanden.
- In den drei letzten Punkten wurden fehlende Umsetzungen der Eigenschaften berücksichtigt, es ist jedoch auch möglich, dass eine in der Angabe nicht vorhandene Eigenschaft in das Datenmodell integriert wird.
- Die Superklasse wird in die Subklassen integriert obwohl es sich um eine unvollständige Generalisierung handelt und somit auch erine Superklassen-Instanzen bestehen könnten.
- Es werden sowohl Super- als auch Subklassen in eigenen Tabellen dargestellt, diese haben jedoch unterschiedliche Primary Keys.

- Rollen:
  - Die Rollen werden in die „Superklasse“ aufgenommen wodurch die Mehrfachbelegung verhindert wird.
  - Die „Superklasse“ wird in die Rollen integriert, dadurch muss jedes Objekt eine der Rollen annehmen, was durch das Konzept nicht vorgesehen ist.
  - In den Rollen wird der Primary Key der „Superklasse“ verwendet, das heißt ein Objekt kann eine Rolle nur einmal annehmen.

Wie bei den Klassen kann auch für die Beziehungen ein Ableitungsbaum (Abbildung 46) erstellt werden, der die Fehlerquellen bei der Umsetzung angibt. Hier ist die Struktur jedoch komplizierter da die verschiedenen Beziehungstypen zu unterschiedlichen Möglichkeiten führen. Das heißt bei einer Beziehung muss als erstes der Typ betrachtet werden, erst danach können die Einzelheiten definiert werden.

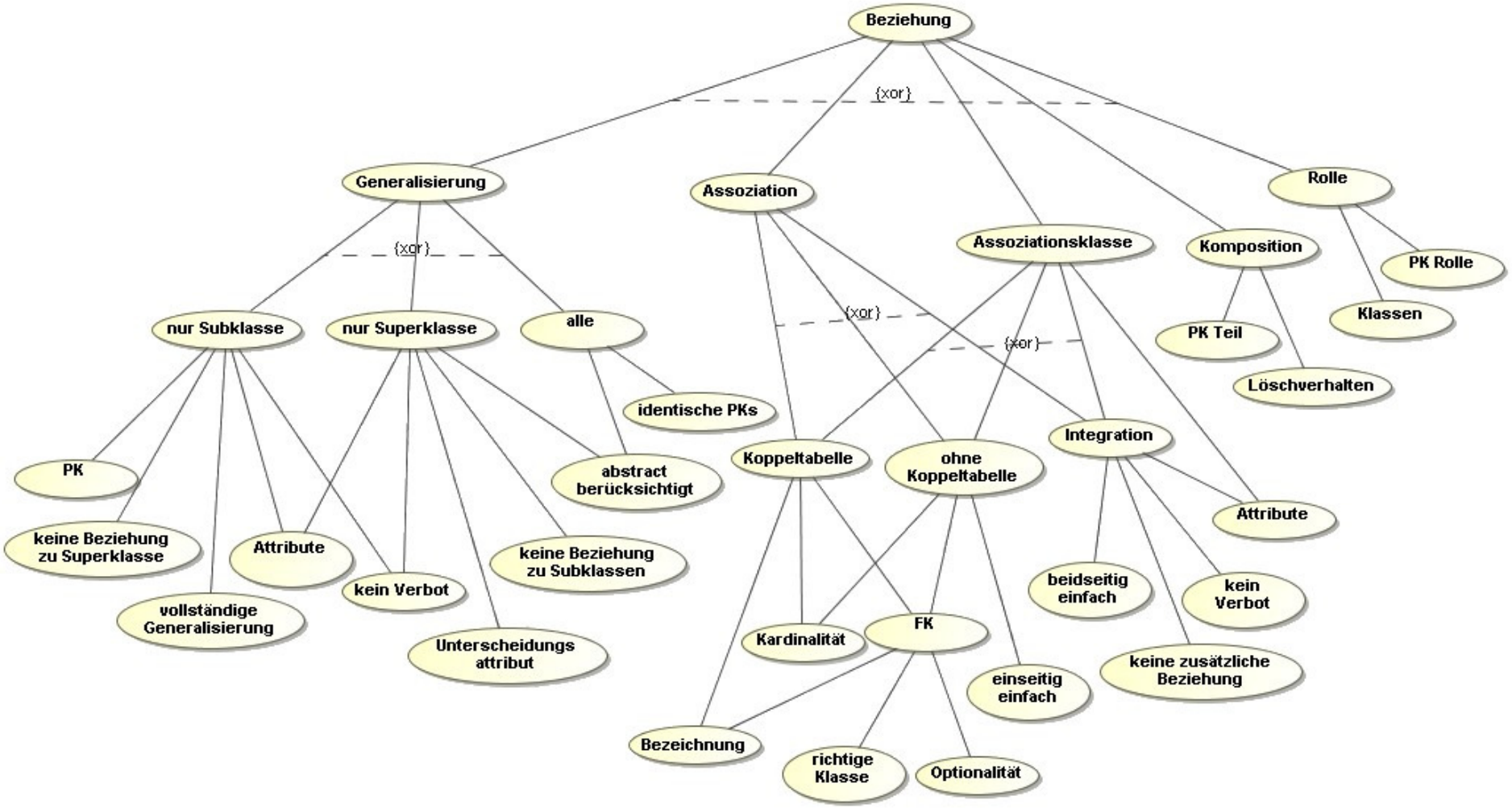


Abbildung 46: Ableitungsbaum zur Korrektur von Beziehungen

Beispiel

Wie bei den Klassen müssen auch hier nicht sämtliche Elemente enthalten sein. Vor allem dadurch dass hier verschiedene Beziehungstypen unterschieden werden muss je nach konkretem Beispiel nur ein begrenzter Pfad betrachtet werden. Dies wird in Abbildung 47 anhand der 1:1 Beziehung „angefertigt von“ zwischen Student und Diplomarbeit erläutert.

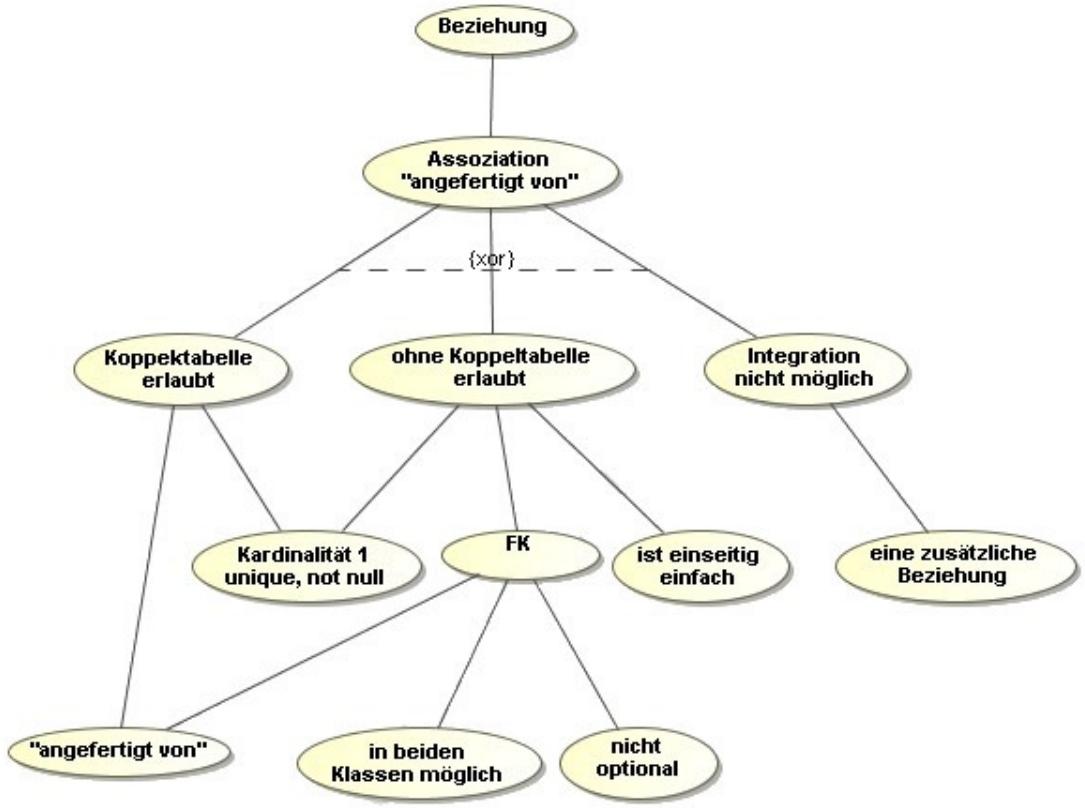


Abbildung 47: Ableitung angefertigt von

Wie bereits erwähnt können von einer Klasse auch mehrere Beziehungen ausgehen. Sobald zwei Beziehungen bestehen muss die Klasse existieren (eine Ausnahme bilden Spezialisierungen im Zuge einer Generalisierung), das heißt für die jeweilige Beziehung wird zumindest die eine Variante ausgeschlossen. Da dennoch für jede Beziehung verschiedene Möglichkeiten zur Umsetzung in den relationalen Tabellen existieren und die Ausprägungsmöglichkeiten einer Klasse mit der Zahl der von ihr ausgehenden Beziehungen steigt, sollte im Zielsystem abgesichert werden, dass die Beziehungen getrennt voneinander betrachtet werden. Beispielsweise können bei zwei binäre Beziehungen vier Möglichkeiten für die konkrete Darstellung der Klasse

bestehen: zwei integrierte Fremdschlüssel, ein integrierter und ein nicht integrierter (dadurch entstehen zwei Möglichkeiten) und kein integrierter Fremdschlüssel. Eine getrennte Betrachtung der Beziehungen bedeutet hier zwar noch keine Einsparung, wenn jedoch weitere Beziehungen hinzukommen wird der dadurch anfallende Aufwand eingeschränkt. Außerdem wird durch getrennte Betrachtung verhindert, dass durch die falsche Umsetzung einer Beziehung die anderen ebenfalls als falsch gewertet werden.

4.2.2 Entwicklung eines Regelsystems

Unabhängig von der tatsächlichen Implementierung ist es notwendig bestimmte Regeln aufzustellen nach denen die Auswertung der Lösung zu erfolgen hat. Hier müssen die Klassen und Beziehungen unter Berücksichtigung der verschiedenen Mapping-Alternativen einfließen. Einander ausschließende Möglichkeiten müssen gekennzeichnet werden. Sämtliche der zuvor angeführten Betrachtungen müssen nun in das Zielsystem einfließen. Es muss sichergestellt werden, dass die Kontrolle sämtlicher Fehlerquellen ermöglicht wird. Weiters soll die Unabhängigkeit der Umsetzung der verschiedenen Elemente gewährleistet werden.

Generell werden zu sämtlichen im UML-Diagramm vorhandenen Klassen die entsprechenden Tabellen in der Lösung gesucht. Die Überprüfung aller Elemente erfolgt entsprechend den zuvor klassifizierten Fehlerquellen. Das heißt für die Auswertung der Klassen sind die entsprechenden Elemente notwendig, sie müssen daher im Zielsystem enthalten sein. Klassen die von der Umsetzung der Beziehung gravierend beeinflusst werden können, müssen gekennzeichnet werden. Derartige Fälle liegen vor wenn eine der an der Beziehung teilnehmenden Klassen in eine andere integriert werden kann. Wenn dies der Fall ist muss überprüft werden ob die Attribute der Klasse selbst enthalten sind, diese betreffen die Bewertung der Klasse. Die Attribute der integrierten Klasse repräsentieren die Beziehung, daher sind hier auftretende Fehler der Beziehung zuzuornen.

Anschließend werden die unterschiedlichen Beziehungen betrachtet. In einem ersten Schritt werden die zuvor außer Acht gelassenen Klassen abgearbeitet. Hier wird überprüft ob die auf Grund der vorhandenen Beziehungen möglichen Integrationen vorgenommen wurden. Wichtig ist dass die integrierte Klasse nicht eigenständig existiert. Die auf Grund der Beziehung integrierten Attribute sowie die Attribute der Klasse selbst werden nach den angeführten Fehlerquellen analysiert. In einem weiteren Schritt werden die vorhandenen Fremdschlüssel bzw. die Koppeltabellen in die Betrachtung mit einbezogen. Für sämtliche Beziehungen muss gesichert werden, dass nur eine der Alternativen umgesetzt wird.

Bereits vor der automatischen Erstellung des Zielsystems muss der Lehrende die erlaubten Alternativen festlegen um unnötigen Arbeitsaufwand zu vermeiden. Das heißt, er muss folgende Alternativen explizit erlauben bzw. verbieten:

- Integration von A in B
- Integration von B in A
- Foreign Key auf B in A
- Foreign Key auf A in B
- Koppeltabelle

Anhand dieser Informationen muss der Lehrende festlegen welche Alternativen er erlaubt bzw. wie er diese bewertet. Die Angaben des Lehrenden werden bei der Erstellung des Zielsystems berücksichtigt, Alternativen die nicht angewendet werden dürfen werden nicht generiert.

Aus diesen Betrachtungen sind nun die entsprechenden Schritte zur Erstellung eines Regelsystems herzuleiten. Sämtliche relevanten Informationen müssen enthalten sein sowie Alternativen der Beziehungsumsetzung und spezielle Einschränkungen des Lehrenden berücksichtigt werden. Die notwendigen Erstellungsschritte für das Zielsystem werden im Folgenden angegeben:

1. Erstellung der Klassen ohne Teile-Klassen von Kompositionen und Assoziationsklassen
    a. Festlegen des Klassennamens

   b. Angabe von Eigenschaften und Stereotypen
   c. Angabe von Attributen
      i. Attributname
      ii. Datentyp
      iii. Kardinalität
         1. Auslagerung der Mehrwertigen Attribute
      iv. Initialwert
      v. Einschränkungen
   d. Festlegen des Primary Key
      i. Bei einem einfachen Schlüssel Angabe bei dem betroffenen Attribut bzw. gesondert
      ii. Bei einem zusammengesetzten Schlüssel gesonderte Angabe
2. Erstellung der Kompositionsteile
   a. Schritte a, b und c von Punkt 1
   b. Festlegen der auf das Ganze verweisenden Foreign Keys
   c. Gesonderte Angabe des aus den Foreign Keys und eventuellen eigenen Attributen zusammengesetzten Primary Keys
   d. Festlegen des Löschverhaltens ON DELETE CASCADE
   e. Erstellung von Constraints zur Beschränkung der Kardinalität
3. Identifikation der Muss-Klassen
   a. Markieren der Klassen die (trotz Integrationsmöglichkeiten auf Grund einzelner Beziehungen) vorhanden sein müssen
4. Identifikation der optionalen Klassen
   a. Markieren der Klassen die auf Grund der vorhandenen Beziehungen in eine andere Klasse integriert werden können
   b. Markieren der Klassen die eine andere Klasse aufnehmen können
5. Integration von Klassen auf Grund von Assoziationen und Aggregationen
   a. Angabe der optionalen Darstellung der aufnehmenden Klasse
   b. Sichern, dass bei Integration die integrierte Klasse nicht gesondert existiert
   c. Sichern, dass bei nicht erfolgter Integration die andere Klasse existiert
6. Aufbau der Alternativen für Assoziationen und Aggregationen

a. Erstellen von Koppeltabellen
    i. Tabellenname festlegen
    ii. Fremdschlüssel festlegen
b. Kreieren von Foreign Keys in den bestehenden Klassen
c. Erstellung von Constraints zur Beschränkung der Kardinalität
d. Sichern, dass die Alternativen einander ausschließen

7. Aufbau der Alternativen für Assoziationsklassen
    a. Wenn sie eine Beziehung zu einer anderen Klasse hat
        i. Punkte a und c von Punkt 6
        ii. Attribute einfügen wie in Punkt 1c
    b. Wenn keine zusätzlichen Beziehungen bestehen
        i. Assoziation wie Punkt 6
        ii. Attribute nach Punkt 1c dort integrieren wo die Beziehung aufgenommen wird
8. Aufbau der Alternativen für Generalisierungen
    a. Wenn die Subklassen keine Muss-Klassen sind
        i. Integration der Attribute der Subklassen in die Superklasse
        ii. Bei disjunkter Generalisierung Erstellung eines Attributs zur Festlegung der verwendeten Subklasse und Angabe von Constraints zur Absicherung, dass nur Attribute einer Subklasse verwendet werden können
        iii. Bei vollständiger Generalisierung sicherstellen, dass mindestens eine der Subklassen mit ihren zugehörigen Attributen verwendet wird
        iv. Sicherstellen, dass keine Subklasse gesondert existiert
    b. Wenn die Superklasse keine Muss-Klasse ist
        i. Integration der Attribute der Superklasse in alle Subklassen
        ii. Bei disjunkter Generalisierung Erstellung eines Constraints zur Absicherung, dass nur eine der Subklassen verwendet werden kann
        iii. Eine unvollständige Generalisierung ist nicht möglich
        iv. Sicherstellen, dass keine Subklasse gesondert existiert

c. Unabhängig von der Deklaration als Muss-Klasse
    i. Erstellung von jeweils einer Tabelle für Super- sowie jede Subklasse
    ii. Integration des Primary Key der Superklasse in die Subklassen
    iii. Festlegen des Löschverhaltens ON DELETE CASCADE
    iv. Disjunkte Generalisierung durch ein Attribut in der Superklasse bzw. durch einen Constraint absichern
    v. Vollständige Generalisierung durch setzen von NOT INSTANTIABLE für die Superklasse
d. Sicherstellen, dass nur eine der Varianten umgesetzt werden kann

9. Anpassung von Rollen
    a. Erstellung des Foreign Key der Rollen-Tabellen auf den Primary Key der „Superklasse“
    b. Festlegen des Löschverhaltens ON DELETE CASCADE

Hier wurden die generellen Schritte zur Erstellung des Regelsystems angeführt. Diese sind von einer tatsächlichen Implementierung weitgehend unabhängig. Dennoch wurde hier abgegrenzt welche konkreten Informationen aus dem XMI Dokument extrahiert werden müssen.

4.2.3 Erkennen Gleichheit/Ungleichheit

Eine besondere Anforderung an das zu entwickelnde Modul liegt darin, die Übereinstimmung von Studentenlösung und Angabe bzw. Zielsystem zu erkennen. Die hierbei auftretenden Probleme und deren Lösungsmöglichkeiten werden im Folgenden erläutert.

Klassen

Wie bereits erwähnt liegt die Schwierigkeit darin, dass der Student im Grunde genommen die Namen frei wählen kann. Für die Entwicklung des Moduls wird davon ausgegangen, dass dieses Problem weitgehend durch eine geforderte Namensgleichheit mit der UML-Angabe gelöst wird. Das bedeutet der Student muss

Tabellen mit dem Namen der äquivalenten Klasse bezeichnen, gleiches gilt für Attribute.

Beziehungen

In Bezug auf die Beziehungen wird vom Lehrenden gefordert Bezeichnungen zu vergeben, eine Ausnahme stellen Generalisierungen dar da hier der Beziehungsname für die Umsetzung in Tabellen irrelevant ist. Der Student muss diese Bezeichnung für die Koppeltabelle bzw. den Foreign Key verwenden.

Dennoch bestehen einige Elemente für die keine konkreten Vorgaben getroffen werden können. Bei der Umsetzung von 1:1 Beziehungen mittels Koppeltabelle könnten beispielsweise zwei gleichnamige Attribute aufeinander treffen. Eine mögliche Lösung für dieses Problem liegt ebenfalls darin, es auf den Lehrenden abzuwälzen, das heißt dieser muss bei derartigen Beziehungen dafür sorgen, dass in den betroffenen Klassen keine gleichnamigen Attribute enthalten sind. Durch die hier festgelegten Namenskonventionen lassen sich die Probleme beim erkennen von Klassen, Attributen und Beziehungen umgehen. Ein weiteres Problem tritt bei der Verwendung von Koppeltabellen zur Beziehungsumsetzung zu Tage. Hier müssen Foreign Keys auf die entsprechenden Klassen verweisen. Diese Fremdschlüssel könnten verschiedene Namen erhalten, welche ebenfalls durch Namenskonventionen gelöst werden könnten. Allerdings tritt bei rekursiven Assoziationen das Problem auf, dass ein Fremdschlüssel in einer Koppeltabelle zweimal mit demselben Namen enthalten sein müsste. Daher muss kontrolliert werden ob der Fremdschlüssel auf die entsprechende Klasse verweist.

Unter Einhaltung dieser Namenskonventionen ist es relativ einfach die Gleichheit bzw. Ungleichheit von Studentenlösung und Zielsystem festzustellen. Klassen sind vorhanden wenn der entsprechende Name bei einer Tabelle vergeben wurde. Sie sind korrekt wenn die Eigenschaft {abstract}, falls vorhanden, umgesetzt wurde und die Attribute nach dem Schema in Kapitel 4.2.1 umgesetzt wurden, gleiches gilt für Beziehungen. Im Zuge der Auswertung müssen aber neben den Gleichheiten auch Ungleichheiten und somit Fehler dokumentiert werden. Es können Tabellen fehlen bzw. falsch bezeichnete Tabellen vorhanden sein, gleiches gilt wiederum für Attribute

und Beziehungen. Bei Beziehungen muss auch festgestellt werden ob eine der „verbotenen“ Umsetzungsalternativen gewählt wurde falls es sich nicht um eine korrekte Alternative handelt. Die auf dieser Auswertung beruhende Dokumentation wird in weiterer Folge als Grundlage für Feedback und Bewertung herangezogen.

Eine besondere Schwierigkeit dieses Problems liegt darin, dass die tatsächlichen Fehler erkannt werden müssen. Genauer gesagt ist dies ein Problem der geforderten Genauigkeit des Fehlerreports. Es kann unterschieden werden ob beispielsweise Beziehungen gar nicht oder falsch berücksichtigt wurden. Diese beiden Varianten könnten natürlich auch generell als fehlende Umsetzung der Beziehung interpretiert werden. Es besteht aber auch die Gefahr, dass Fehler aufgedeckt werden obwohl die eigentliche Ursache in einem anderen Element liegt. Beispielsweise kann eine Beziehung durch eine Zusammenführung der beiden Klassen aufgelöst werden obwohl dies nicht möglich ist. Dadurch kann sich ein Fremdschlüssel auf die falsche Klasse beziehen und dies als Fehler rückgemeldet werden. Der eigentliche Fehler liegt jedoch in der Umsetzung der Beziehung zwischen der zu referenzierenden Klasse und der tatsächlich referenzierten Klasse.

Beispiel

Ein Beispiel für das eben genannte Problem kann aus dem Eingangs gezeigten Klassendiagramm entnommen werden (siehe Abbildung 48). Die 1:1 Beziehung zwischen Student und Diplomarbeit erlaubt die Zusammenlegung der einen Klasse in die andere. Aus dem hier gezeigten Kontext erscheint nur die Integration der Klasse Diplomarbeit in die Tabelle Student als sinnvoll.

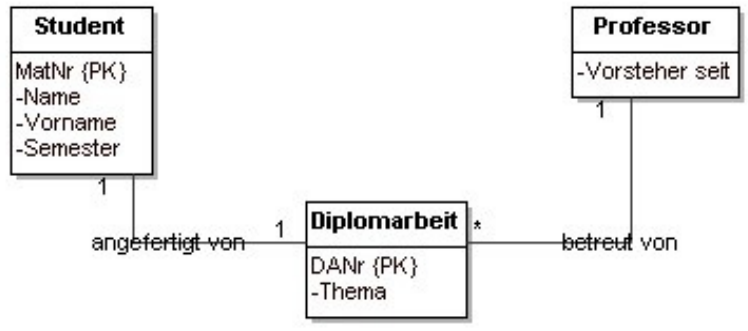


Abbildung 48: Beispiel falsche Fehlererkennung

Da die Klasse Diplomarbeit nicht mehr existiert muss der Fremdschlüssel in der erstellten Koppeltabelle auf die Tabelle Student umgelegt werden. Durch die zuvor definierten Ableitungsregeln darf eine Integration in diesem Fall allerdings nicht erfolgen, das heißt der Fremdschlüssel in der Zwischentabelle müsste eigentlich auf die Tabelle Diplomarbeit verweisen.

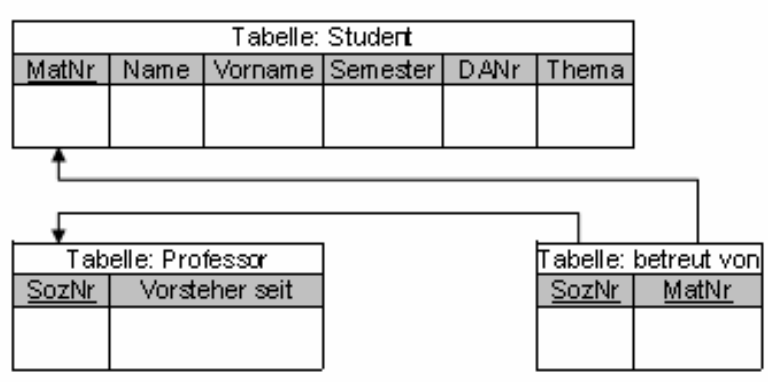


Abbildung 49: Tabellen zur falschen Fehlererkennung

Wie aus Abbildung 49 ersichtlich entstehen Komplikationen bei der Auswertung. Hier kann der umgeleitete Fremdschlüssel als Fehler erkannt werden. Dieser ist in diesem Fall allerdings nur ein Folgefehler der fälschlicherweise integrierten Klasse Diplomarbeit.

Check

Schwierig wird die Feststellung der Gleichheit bzw. Ungleichheit bei frei definierbaren Elementen wie Zusicherungen und der Kardinalität von Attributen und Beziehungen. Zur Umsetzung dieser Elemente müssen CHECK-Regeln bzw. Assertions definiert werden. Da Zusicherungen frei definierbar sind müssen die verschiedenen Möglichkeiten einzeln betrachtet werden.

Eine einfache Zusicherung liegt in der Einschränkung des Wertebereichs von Attributen. Ein Beispiel hierfür ist {1<x<3}, das heißt das Attribut x kann nur einen Wert zwischen 1 und 3 annehmen. Die Umsetzung dieser Einschränkung kann durch „CHECK (x>1 AND x<3)“ umgesetzt werden bzw. durch „CHECK (1<x<3)“. Gleiches gilt für Character-Attribute, nur dass hier der Aufbau des Statements geändert

werden muss (z.B. CHECK (x IN (‚JA', ‚NEIN')). Es wird davon ausgegangen, dass die Darstellungsform der Angabe übernommen wird.

Assertions

Die Komplexität der Auswertung steigt sobald die Absicherungen nur durch SQL-Abfragen durchgeführt werden kann. Ein Beispiel dafür stellt die XOR-Einschränkung dar, welche bereits vorgestellt wurde. Diese betrifft in der Regel zwei von einer Klasse ausgehende Assoziationen wobei nur eine der beiden eingegangen werden kann. Eine Absicherung dafür muss die vorhandenen Fremdschlüssel mit einbeziehen. Hier kann die Abfrage in bestimmten Fällen, genauer gesagt wenn beide Fremdschlüssel in derselben Tabelle enthalten sind, durch eine CHECK-Regel durchgeführt werden, wenn allerdings verschiedene Klassen betroffen sind, muss eine Assertion erstellt werden. Diese beinhalten CHECK-Regeln, die SQL-Abfragen zur Kontrolle der Einschränkung enthalten. Je nach Umsetzung der beiden Assoziationen müssen verschiedene Tabellen und die enthaltenen Fremdschlüssel berücksichtigt werden. Hier ist auch zu beachten, dass die Beziehung betreffende Fremdschlüssel eventuell in erstellten Koppeltabellen enthalten sein können.

Ein wichtiger Punkt bei der Kontrolle der Lösung liegt in den Assertions bezüglich der zusätzlichen Eigenschaften von Generalisierungen. Bei abstrakten Klassen bzw. vollständigen Generalisierungen muss gesichert sein, dass kein Tupel in der Superklasse besteht dessen Primärschlüssel in keiner Subklasse enthalten ist. Bei unvollständigen Generalisierungen ist keine derartige Absicherung nötig. Die Eigenschaft disjoint ist ebenfalls durch eine Assertion zu gewährleisten. Hier muss durch eine SQL-Abfrage überprüft werden ob der Primärschlüssel eines Tupels einer Subklasse auch in keiner anderen enthalten ist.

Wie in dem Klassendiagramm bezüglich der Universität ersichtlich müssen auch beschränkte Multiplizitäten der Beziehungen berücksichtigt werden. Diese sind ebenfalls durch entsprechende Assertions umzusetzen. Hier müssen die Beziehung betreffende Fremdschlüssel betrachtet werden. Liegt eine Beschränkung der Kardinalität vor dürfen bezüglich einer Klasse nur eine beschränkte Anzahl an Fremdschlüsseln vorhanden sein, das heißt wenn eine Klasse A mit 2-5 Instanzen

der Klasse B eine Beziehung eingehen kann, darf in den Instanzen von B maximal 2-5 mal ein Fremdschlüssel auf eine bestimmte Instanz der Klasse A verweisen. Hier ist wiederum zu beachten, dass diese Beziehung je nach Kardinalität verschieden umgesetzt werden kann.

Bei der Überprüfung von Assertions, welche SQL-Abfragen enthalten, ist zu beachten, dass diese SQL-Statements unter Umständen verschieden aufgebaut werden können. Außerdem können die Assertions zwar eine Bezeichnung erhalten, diese kann aber nicht vorgegeben werden. Aus diesem Grund lassen sich Assertions nur schwer überprüfen. Im Zuge dieser Arbeit werden daher nur Wertebereichseinschränkungen und Primary Keys betrachtet, das heißt Zusicherungen welche keine Assertions verlangen. Hier besteht die Möglichkeit einer späteren Erweiterung um zusätzliche Einschränkungen.

4.2.4 Feedback

Wie bereits in Kapitel 1 geschildert wird neben der Auswertung der Lösung auch die Unterstützung des Studenten mittels Feedback gefordert. Dadurch soll der Student auf Fehler hingewiesen und somit im Lernprozess geführt werden. Es ist hier nur ein „einstufiges“ Feedback vorgesehen, das heißt es werden keine Fehler-Patterns erkannt. Das durch das System erstellte Feedback wird auf Basis der durchgeführten Auswertung und Dokumentation aufgebaut. Es wird davon ausgegangen, dass bei korrekter Umsetzung einzelner Elemente keine Rückmeldung bezüglich dieser erfolgt. Sollte das gesamte Diagramm fehlerfrei in Tabellen umgesetzt worden sein, wird diese Information dem Studenten übermittelt.

Natürlich müssen auch Fehler bezüglich der einzelnen Elemente kommuniziert werden. Wie in Kapitel 4.2.1 gezeigt bestehen sowohl bei der Umsetzung von Klassen als auch bei Beziehungen diverse Fehlerquellen. Diese müssen bei der Erstellung des Feedbacks berücksichtigt werden. Bei der Betrachtung von Klassen und Attributen ist das relativ einfach: sie können fehlen, teilweise oder komplett falsch sein oder aber auch zusätzliche Elemente vorhanden sein. Das heißt in Bezug auf Klassen sind folgende Feedback-Varianten relevant:

- „Die Tabelle ‚Name' existiert nicht" wenn eine Klasse gar nicht umgesetzt bzw. ein falscher Name gewählt wurde.
- „Die Tabelle ‚Name' darf nicht enthalten sein" wenn eine Tabelle existiert zu der keine entsprechende Klasse im UML-Diagramm enthalten ist.
- „Der Primary Key ‚Attributname' der Tabelle ‚Name' wurde nicht definiert" wenn der Primary Key gar nicht bzw. falsch deklariert wurde
- „Das Attribut ‚Attributname' existiert nicht" wenn ein Attribut gar nicht umgesetzt bzw. ein falscher Name gewählt wurde.
- „Das Attribut ‚Attributname' darf nicht enthalten sein" wenn ein Attribut existiert zu der keine entsprechende Klasse im UML-Diagramm enthalten ist.
- „Der Datentyp des Attributs ‚Attributname' muss als ‚Typ' definiert werden" wenn der Datentyp fehlt bzw. falsch definiert
- „Die Tabelle bezüglich des komplexen Datentyps ‚Name' muss den Primary Key der Tabelle ‚Name' als Foreign Key enthalten und das Löschverhalten ON DELETE CASCADE definieren"
- „Der Default-Wert des Attributs ‚Attributname' wurde nicht berücksichtigt" wenn der Wert nicht vorhanden bzw. falsch ist.
- „Die Einschränkung des Wertebereichs des Attributs ‚Attributname' wurde nicht berücksichtigt. Dies muss in einer CHECK-Regel umgesetzt werden"

Die Fehlermeldungen für die Klassen an sich lassen sich relativ leicht definieren und sind bei der Auswertung auch ziemlich eindeutig zu erkennen. Fehlermeldungen zu Elementen welche bei der Auswertung nicht berücksichtigt werden sind hier nicht enthalten. Bei Betrachtung der Beziehungen wird das Ganze komplexer. Unter Umständen sind auch umfangreichere Erklärungen zu den Fehlern notwendig. Die möglichen Feedback-Alternativen bezüglich der verschiedenen Beziehungstypen werden im Folgenden erstellt:

Assoziation und Aggregation

- „Die Klassen ‚Name' und ‚Name' können nicht zusammengelegt werden, da dies bei einer 1:N/M:N Beziehung nicht möglich ist"
- „Die Klassen ‚Name' und ‚Name' können nicht zusammengelegt werden, da die integrierte Klasse ‚Name' auf Grund der Beziehung zur Klasse ‚Name' existieren muss"

- „Die Klassen ‚Name' und ‚Name' können nicht zusammengelegt werden, da dies in diesem Fall durch den Lehrenden verboten wurde"
- „Die Beziehung ‚Bezeichnung' kann nicht durch einen Fremdschlüssel in der Tabelle ‚Name' umgesetzt werden, da dieser auf Grund des höherwertigen Beziehungsendes mehrere Tupel der Tabelle ‚Name' gleichzeitig repräsentieren müsste"
- „Der Fremdschlüssel in der Tabelle ‚Name' bezüglich der zu 1 Beziehung ‚Bezeichnung' muss als UNIQUE deklariert werden, da es sich um eine zu 1 Beziehung handelt"
- „Der Fremdschlüssel in der Tabelle ‚Name' bezüglich der Beziehung ‚Bezeichnung' auf die Klasse ‚Name' darf nicht als UNIQUE deklariert werden de es sich um eine zu N Beziehung handelt"
- „Der Fremdschlüssel in der Tabelle ‚Name' bezüglich der Beziehung ‚Bezeichnung' muss als NOT NULL deklariert werden, da es sich um keine optionale Beziehung handelt"
- „Der Fremdschlüssel in der Tabelle ‚Name' bezüglich der Beziehung ‚Bezeichnung' auf die Klasse ‚Name' darf nicht als NOT NULL deklariert werden de es sich um eine optionale Beziehung handelt"
- „Die Beziehung ‚Bezeichnung' unterliegt einer beschränkten Multiplizität, deren Einhaltung muss in einer Assertion berücksichtigt werden"
- „Die Fremdschlüssel in der Koppeltabelle ‚Name' müssten auf die Tabellen ‚Name' und ‚Name' verweisen" wenn ein Fremdschlüssel auf eine falsche Tabelle verweist
- „Die Tabelle ‚Name' zur Umsetzung der Beziehung ‚Bezeichnung' ist nicht vorhanden" wenn eine Koppeltabelle nicht erstellt wurde
- „Wenn eine Koppeltabelle bezüglich der Beziehung ‚Bezeichnung' erstellt wurde darf in die Tabelle ‚Name' kein Fremdschlüssel bezüglich derselben Beziehung erstellt werden"
- „Bei einer M:N Beziehung muss eine Koppeltabelle erstellt werden, daher darf die Beziehung ‚Bezeichnung' nicht mittels Fremdschlüssel umgesetzt werden"

Assoziationsklassen

- „Die Assoziationsklasse ‚Bezeichnung' wurde im Datenmodell nicht umgesetzt" wenn die Beziehung gar nicht bzw. unter falschem Namen vorhanden ist
- Sämtliche Fehlermeldungen für die Attribute „normaler" Klassen gelten
- Sämtliche Fehlermeldungen für die Umsetzung von Assoziationen gelten

Kompositionen

- „Der Primärschlüssel der Tabelle ‚Name' muss im Schlüssel der Kompositionsteil Tabelle ‚Name' enthalten sein"
- „Die Kompositionsteile in der Tabelle ‚Name' müssen zusammen mit dem entsprechenden Ganzen in der Tabelle ‚Name' gelöscht werden, daher muss der Fremdschlüssel in der Teile-Tabelle ‚Name' bezüglich der Tabelle ‚Name' das Löschverhalten ON DELETE CASCADE erfüllen"

Generalisierungen

- „Die Superklasse ‚Name' darf nicht in die Subklassen integriert werden da eine zusätzliche Beziehung zur Superklasse besteht"
- „Die Subklassen dürfen nicht in die Superklasse ‚Name' integriert werden da eine zusätzliche Beziehung zur Subklasse ‚Name' besteht"
- „In diesem Fall wurde die Integration der Superklasse ‚Name' in die Subklassen durch den Lehrenden verboten"
- „In diesem Fall wurde die Integration der Subklassen in die Superklasse ‚Name' durch den Lehrenden verboten"
- „Bei Integration der Subklassen in die Superklasse ‚Name' muss sichergestellt werden, dass ein Tupel nicht in mehreren Subklassen enthalten ist, das heißt dass nur Attribute einer Subklasse belegt werden können"
- „Bei der Integration der Subklasse ‚Name' in die Superklasse ‚Name' wurde das Attribut ‚Attributname' vergessen'
- „In der Subklasse ‚Name' der Superklasse ‚Name' muss der Primärschlüssel der Superklasse verwendet werden"

Rollen

- „Die Rollen dürfen nicht in die Tabelle ‚Name' aufgenommen werden, da dadurch die Mehrfachbelegung einzelner Rollen verhindert wird"
- „Die Klasse ‚Name' darf nicht in die Rollen integriert werden, da dadurch jedes Tupel eine Rolle einnehmen müsste, was auf Grund des Konzeptes nicht vorgesehen ist"
- „In Tabellen bezüglich der Rollen zur Tabelle ‚Name' darf nicht der Primary Key der Tabelle ‚Name' verwendet werden, da dadurch jedes Tupel nur eine Rolle annehmen könnte"

Die hier angeführten Fehlermeldungen stellen Beispiele dar, sie werden nicht unbedingt in dieser Art und Weise umgesetzt.

4.2.5 Bewertung

Eine weitere Anforderung an die Konzeption des Moduls ist die Bewertung von Fehlern sowie deren Rückmeldung an den Lernenden und die Dokumentation für die Endnote. Hier ist zu beachten, dass es für eine Übung eine maximal zu erreichende Punktezahl gibt. Es besteht die Möglichkeit für richtige Umsetzung Punkte bis zu dieser Grenze zu vergeben oder bei Fehlern Punkte von der Gesamtzahl abzuziehen. Welche dieser Varianten angewendet wird macht im Grunde genommen keinen Unterschied. Im Sinne der klassischen Übungsauswertung ist allerdings der Abzug von Punkten sinnvoller. Allerdings ist hierfür festzulegen wie viele Punkte pro Element abgezogen werden können, das heißt wie viele Punkte eine Klasse wert ist, wie viele eine Beziehung usw. Bei Klassen sind wiederum die einzelnen Bestandteile zu berücksichtigen.

Ausgegangen wird von einer Gesamtzahl von beispielsweise 30 Punkten, diese werden bei vollständig korrekt umgesetztem Diagramm vergeben. Von dieser Zahl ausgehend können die Punkte auf die einzelnen Elemente verteilt werden. Beispielsweise kann man bestimmen wie viele Elemente im Sinne von Klassen und Beziehungen in einem Diagramm enthalten sind, und die gesamten Punkte auf diese Elemente verteilen. Dabei kann wiederum jedes Element als gleichwertig angesehen werden oder aber auch Beziehungen und Klassen verschiedene Wertigkeiten aufweisen, so dass beispielsweise eine Klasse doppelt so viel wert ist wie eine

Beziehung. Ein anderer Ansatz hierfür besteht darin, dass für jedes Element, unabhängig von der Anzahl der im Diagramm enthaltenen Elemente, eine vordefinierte Punktezahl abgezogen wird. In diesem Fall ist zu beachten, dass der Punkteabzug mit der Grenze 0 beschränkt ist. Da eine gleichmäßige Aufteilung der Punkte auf die verschiedenen Elemente einen relativ hohen Aufwand bedeutet und zu komplizierten Berechnungen bei der Punktevergabe führen kann, wird hier der Ansatz mit vorgegebenen Punkten pro Element gearbeitet. Das hier verwendete Punktesystem folgt den im Folgenden angeführten Regeln:

- Das gesamte Diagramm kann maximal 30 Punkte erhalten bzw. minimal 0 Punkte
- Wenn kein richtig umgesetztes Element enthalten ist, ist die Punktezahl in jedem Fall 0
- Für eine Klasse können maximal 2 Punkte abgezogen werden, diese werden auf jeden Fall abgezogen wenn ein falscher Name verwendet wurde
- Für einen fehlenden bzw. falschen Primärschlüssel werden 0,5 Punkte abgezogen
- Für die in einer Klasse enthaltenen Attribute kann maximal 1 Punkt abgezogen werden
- Für ein fehlendes bzw. ein falsch benanntes Attribut werden 0,5 Punkte abgezogen
- Für einen falsch gewählten Datentyp werden 0,1 Punkte abgezogen
- Für eine nicht berücksichtige Kardinalität des Attributs werden 0,1 Punkte abgezogen
- Für einen nicht umgesetzten Initialwert werden 0,1 Punkte abgezogen
- Für eine nicht umgesetzte Wertebereichseinschränkung werden 0,1 Punkte abgezogen
- Für eine nicht mögliche bzw. nicht erlaubte Umsetzung einer Beziehung werden 1,5 Punkte abgezogen
- Für eine doppelte Umsetzung einer Beziehung werden 1,5 Punkte abgezogen
- Für eine fehlerhafte Beziehung kann maximal 1 Punkt abgezogen werden
- Für ein nicht berücksichtigtes bzw. falsches Löschverhalten werden 0,2 Punkte abgezogen

Wie hier zu erkennen ist werden beim Feedback exaktere Unterscheidungen getroffen als bei der Punktevergabe. Der Grund dafür liegt darin, dass das Feedback den Lernenden beim Lernprozess unterstützen soll, das heißt es werden nicht nur Fehler definiert sondern auch erklärt was falsch ist. Bei der Punktevergabe ist es jedoch irrelevant welcher Fehler gemacht wurde. Da im Abschnitt zur Auswertung der Lösung bisher nicht alle Elemente, genauer gesagt nicht alle Zusicherungen, beachtet wurden, sind diese auch im Konzept zur Punktevergabe nicht enthalten.

5. Umsetzung

Dieser Abschnitt der Arbeit umfasst die reale Erstellung des Moduls zur Aufgabenbewertung im Bereich des logischen Entwurfs. Der Aufbau bzw. die Schnittstellen des bestehenden Systems wurden ansatzweise in Kapitel 2 betrachtet, die theoretischen Grundlagen in Kapitel 3. Davon ausgehend wurde das zu erstellende Modul konzipiert. Dieses Konzept wird nun anhand eines Prototyps auf dessen Umsetzbarkeit hin überprüft. Das Expertenmodul für den logischen Entwurf an sich muss folgende Aufgaben erfüllen können:

- Es muss dem Lehrenden eine Möglichkeit zur Erstellung der UML-Angabe bieten
- Es muss dem Lehrenden erlauben in die Auswertung und Beurteilung der Lösung einzugreifen (durch Bewertung der verschiedenen Alternativen)
- Es muss das somit zu Grunde liegende Zielsystem erstellen
- Es muss die Studentenlösung auf ihre Übereinstimmung mit dem Regelsystem hin überprüfen
- Es muss darauf aufbauend das Feedback und die Bewertung erstellen

Für die Implementierung werden in Kapitel 5.1 diverse UML-Tools auf ihre Einsetzbarkeit hin überprüft. Der Prototyp selbst wird in Kapitel 5.2 betrachtet.

5.1 Auswahl des UML Werkzeugs

Der erste Schritt für den Lehrenden ist die Erstellung der Aufgabe, das heißt der Entwurf des UML-Diagramms. Hier bestehen zwei Möglichkeiten für die Kommunikation mit dem System: der Lehrende kann die Aufgabe in einem von eTutor unabhängigen Tool erstellen und das erstellte XMI in das System einfügen oder es wird ein im System integriertes Tool zur Erstellung des Diagramms angeboten. Natürlich können auch beide Varianten parallel eingesetzt werden.Um diese Aufgabe durchführen zu können ist in jedem Fall ein entsprechendes Werkzeug nötig.

In diesem Teil der Arbeit werden verschiedene Tools auf ihre Eignung hin überprüft. Als Grundlage wurden verschiedene Kriterien festgelegt, wobei einige davon unumstößlich sind andere aber nur als wünschenswert deklariert wurden. Wichtig ist vor allem, dass sämtliche zur Verwendung vorgesehenen Modellelemente in dem Werkzeug direkt verfügbar bzw. durch andere Konstrukte darstellbar sind. Da die Angabe durch das System weiter verarbeitet werden muss ist ein Export des Modells mittels XMI notwendig. Wünschenswert wäre die Unterstützung von OCL-Constraints sowie die Anpassbarkeit des generierten XMI bzw. die Möglichkeit der Beeinflussung der XMI-Generierung. Außerdem wäre auch die Unterstützung verschiedener Betriebssysteme von Vorteil, wobei Windows als Standard vorausgesetzt wird. Natürlich sollten auch die Kosten der Werkzeuge berücksichtigt werden, diese sind vor allem dann von Bedeutung wenn die anderen Kriterien von verschiedenen Werkzeugen gleichermaßen erfüllt werden.

Da es am Markt sehr viele UML-Werkzeuge gibt, wurde zu Beginn eine erste Einschränkung vorgenommen. Auf der Homepage der OMG wird eine Übersicht mit über 100 verschiedenen UML-Werkzeugen sowie einem Überblick über deren Features referenziert. [Jeck04] Hier wurden in einem ersten Schritt sämtliche Werkzeuge selektiert, die Klassendiagramme sowie XMI-Export unterstützen. Darauf hin wurden diese Werkzeuge in Test- bzw. Vollversionen aus dem Internet bezogen. Da die Seite nicht über sämtliche aktuellen Werkzeuge und Daten verfügt wurden teilweise andere Werkzeuge des Herstellers verwendet. In Tabelle 5 werden sämtliche in Frage kommenden Werkzeuge aufgelistet.

Werkzeug	**Hersteller**	**Plattformen**
Argo UML	University of California	Java
Ameos	Aonix	Linux, Solaris, Windows
ARIS	IDS Scheer	Windows
Artisan Studio	Artisan	Windows
Eclipse UML	Omondo	Java
Enterprise Architekt	Sparx Systems	Linux, Windows
Ideogramic UML	Ideogramic ApS	Linux, Windows
Innovator	MID	Linux, Solaris, Windows
Magic Draw UML	NoMagic Inc.	Java

Meta Mill	Metamill Software	Linux, Windows
Objecteering/UML	Objecteering Software	Linux, Solaris, Windows
ObjectIF	Microtool	Windows
Poseidon	Gentleware	Java
Power Designer	Sybase	Windows
Rational Developer	IBM	Unix, Linux, Windows
Rhapsody	Telelogic	Solaris, Windows
Select Component Architect	Select Business Solutions	Windows
Software through Pictures	Aonix	Solaris, Windows
TAU-UML Suite	Telelogic	Solaris, Windows
Together	Borland	Linux, Mac OS, Solaris, Windows
Umbrello UML Modeler	Umbrello Project Team/Sourceforge	Linux
Umlaut	Irisa	Linux, Solaris, Windows
UMLet	Martin Auer, Thomas Tschurtschenthaler	Java
Visual UML	Visual Object Modelers	Windows

Tabelle 5: Übersicht UML-Werkzeuge

In einem weiteren Schritt wurden sämtliche Werkzeuge in der erhältlichen Version in Betrieb genommen und auf ihre Funktionalität hin getestet. Zu beachten ist, dass einige Programme in der Testversion nur über eingeschränkte Funktionalität verfügen und im Allgemeinen nicht definiert ist, welche Einschränkungen vorgenommen wurden. Außerdem werden nicht alle Werkzeuge in einer Testversion angeboten. Daher konnten ARIS und Software through Pictures nicht getestet werden. Da Umbrello UML nicht für Windows erhältlich ist, wurde dieses ebenfalls außer Acht gelassen. TAU UML Suite wurde in einer Testversion bestellt, dieser Auftrag wurde aber durch die Firma aus unbekannten Gründen nicht erfüllt oder durch ein verborgenes Problem im E-Mail Verkehr verhindert.

Im Folgenden werden die erworbenen Erkenntnisse über die evaluierten UML-Werkzeuge dargestellt. Die angegebenen Preise sind teilweise nicht vollständig, da zu manchen Tools enorm umfangreiche Preislisten für verschiedene Konditionen angeboten werden. Zu einigen Tools sind Preise nur durch Anfrage beim Hersteller selbst zu erfahren, diese sind daher nicht angeführt.

Argo UML

Argo UML ist ein Open-Source Werkzeug, das in der Version 0.24 evaluiert wurde. Von den geforderten Modellelementen werden N-äre Assoziationen und Einschränkungen nicht unterstützt. Stereotype werden zwar vorgegeben, sind jedoch erweiterbar, wodurch auch Rollen dargestellt werden können. Für Assoziationen werden einige Multiplizitäten zur Auswahl angeboten, es können jedoch auch individuelle Eingaben getätigt werden. Argo UML bietet auch Unterstützung für OCL, Operatoren werden durch das Programm angeboten, die erstellten Regeln werden überprüft. Das Programm bietet eine XMI Exportmöglichkeit sowie Code Generatoren für Java, C++, C#, und PHP. Ein zusätzlicher Vorteil besteht in der Unterstützung des Benutzers durch automatisches Zeichnen von Modellelementen.

Ameos

Das früher kostenpflichtige Tool Ameos wird heute in einer Open Source Version über Internet angeboten. Das Produkt wird nun unter dem Namen Open Ameos vertrieben. Dieses Produkt unterstützt die vorgesehenen Elemente bis auf die möglicherweise verwendeten Elemente Geschachtelte Klassen und Rollen. Außerdem sind Stereotype auf Vorgaben durch das Tool beschränkt, bis auf XOR-Einschränkungen können keine Constraints formuliert werden. Multiplizitäten können durch den Benutzer festgelegt werden, das Tool schlägt jedoch einige gängige Auswahlmöglichkeiten vor. Open Ameos bietet neben der XMI Exportmöglichkeit auch die Dokumentation in Rich Text Format (RTF) und Postskript (PS). Außerdem besteht die Möglichkeit einer anpassbaren Codegenerierung und des Reverse Engineering für Java, C, C++, Ada 95 und Raven. Ein Nachteil von Open Ameos liegt in der Benutzeroberfläche da das Projekt und das Diagramm in verschiedenen Fenstern enthalten sind.

Artisan Studio

Dieses Tool wird als Evaluierungsversion für 30 Tage angeboten und wurde in der Version 6.2 getestet. Artisan Studio unterstützt sämtliche geforderten Modellelemente. Für Stereotype und Multiplizitäten sind Vorgaben enthalten, diese

können jedoch erweitert werden. Rollen können durch eigene Stereotype gekennzeichnet werden, Constraints sind individuell definierbar. Das Tool unterstützt den Import bzw. Export von Rational Rose Modellen sowie den Export eines XMI Dokuments. Mit einer Artisan OCS Lizenz werden außerdem Möglichkeiten zur Codegenerierung in Java, C++, Ada und SQL-DDL angeboten. Laut Dokumentation ist zumindest die Anpassung der Codegenerierung möglich, ob auch das XMI Dokument beim Generieren angepasst werden kann ist nicht ersichtlich. Der Preis für den Erwerb einer Lizenz muss bei den entsprechenden Servicestellen erfragt werden, über die Homepage sind diese Informationen nicht zu erhalten.

Eclipse UML

Eclipse UML ist in der derzeit aktuellsten Version 3.0.0 auch in einer freien Edition erhältlich. Hierbei handelt es sich um ein Plug-in für Eclipse. Bis auf n-äre Assoziationen können alle geforderten Modellelemente dargestellt werden. Sowohl Stereotype als auch Constraints sind frei definierbar, Multiplizitäten sind wiederum frei anpassbar. Die freie Version bietet neben dem XMI Import Exportmöglichkeiten in den Formaten GIF, JPG, WMF sowie die Generierung einer Javadoc. Der XMI Export für einzelne Diagramme ist in der freien Version nicht verfügbar, hier können nur die Grundinformationen zum Projekt generiert werden. Codegenerierung und Reverse Engineering sind für Java möglich. Für die lizenzierte Edition sind 2398 € zu zahlen, für ein Paket von 5 Lizenzen 9592 €. Welche zusätzlichen Features diese Edition bietet konnte nicht in Erfahrung gebracht werden.

Enterprise Architekt

Version 7.0 wurde in einer eigenen Evaluierungsversion getestet. Diese kann 30 Tage lang genutzt werden und umfasst sämtliche erhältliche Editionen. Enterprise Architekt unterstützt sämtliche geforderten Modellelemente. Innere Klassen können wie Attribute verwendet werden, die gegebenen Stereotype sind leicht erweiterbar. Constraints können frei definiert werden das Tool unterstützt aber auch OCL, außer bei der Desktop-Edition ist auch eine Validierung der entworfenen Constraints möglich. Neben dem Export des Modells in XMI wird auch die Erstellung von HTML und RTF Dokumentationen ermöglicht. Außer in der Desktop-Edition wird auch die

Codegenerierung und das Reverse-Engineering in C, C++, C#, Java, PHP, Visual Basic, VB.net, Delphi, XML (XLS) und SQL-DDL geboten. Je nach Edition und benötigter Anzahl an Lizenzen variiert der geforderte Preis zwischen 95 $ und 335 $ pro Person. Die Desktop-Edition kostet 135 $ für eine Person. Für Universitäten wird eine eigene Preisliste geboten, zwischen 65 $ für eine einzelne Desktop Edition Lizenz und 12.839 $ für eine unbegrenzte Benutzerzahl der Corporate Edition.

Ideogramic UML

Die Desktop Edition von Ideogramic UML wurde in der 14 Tage Testversion geprüft. Hier werden einige der notwendigen Elemente nicht unterstützt. N-Äre Assoziationen, Assoziationsklassen, Einschränkungen, Rollen, geschachtelte Klassen und Stereotype können zumindest in der Evaluierungsversion nicht dargestellt werden. Laut Homepage besteht in dieser Version zwar eine Einschränkung der Features, diese soll aber nur die Anzahl der Elemente betreffen und trotzdem sämtliche Features für Klassendiagramme anbieten. Neben der XMI Exportmöglichkeit werden auch automatische Dokumentationen in HTML sowie das Speichern des Modells in JPG angeboten. Das Tool bietet Codegenerierung und Reverse-Engineering für Java und C#. Der Preis beläuft sich je nach Version und Anzahl der Lizenzen zwischen 1050 € und 11695 €.

Innovator

Innovator 2007 9.1.03 umfasst mehrere Tools für verschiedene Aufgabenbereiche. Für die Erstellung von Datenbanken ist eigentlich das Tool Data Classix vorgesehen. Dieses bietet jedoch nur eine Darstellung in Form des Entity Relationship Modells. Es bestehen aber auch Tools für den Objektorientierten Entwurf. Getestet wurden Object Classix und Object Excellent welche jedoch weitgehend über dieselben Funktionalitäten verfügen. Dieses Tool verteilt das Projekt selbst und das Klassendiagramm auf wie verschiedene Fenster. Rollen, Geschachtelte Klassen und Stereotype können nicht dargestellt werden, alle anderen Elemente werden angeboten. Die Erstellung von Constraints in OCL wird unterstützt. Object Classix unterstützt den XMI Export, Object Excellent hingegen bietet eine Exportmöglichkeit in XML. Beide Tools ermöglichen die Erstellung einer HTML bzw. Doc

Dokumentation. Codegenerierung wird für Java, C++ bzw. C# und Corba IDL angeboten. Für Universitäten werden freie Lizenzen vergeben. Diese beinhalten einen Lizenzenpool der unter den Studenten und Mitarbeitern verteilt werden kann.

Magic Draw UML

Dieses Tool wurde in der freien Community Edition getestet. Magic Draw unterstützt sämtliche Modellelemente und bietet Außerdem bei jedem Modellelement eine Toolbar mit Elementen zur Verknüpfung sowie automatisches Zeichnen dieser Elemente. Die vorgegebenen Stereotype können wie bei den meisten Tools problemlos erweitert werden. Der XMI Export wird bereits in der freien Edition angeboten, gleiches gilt für die OCL Unterstützung. Magic Draw UML generiert den Constraint-Kopf automatisch, der Benutzer muss nur die tatsächlichen Einschränkungen angeben. Die Validierung der erstellten OCL-Statements ist nur in der Enterprise Edition möglich. Das Werkzeug unterstützt den Import bzw. Export von Rational Rose Modellen. Die kostenpflichtigen Versionen bieten Codegenerierung und Reverse Engineering für Java, C++, C#, SQL, Oracle. Je nach Version sind Lizenzen für Clientanwendungen bis zu einem Preis von 2600 € erhältlich.

MetaMill UML

Dieses Werkzeug wurde in der Version MetaMill UML CASE Tool v4.2 Evaluation getestet. Diese Version ist auf insgesamt 20 Modellelemente beschränkt. Hier werden weder N-Äre Assoziationen noch Assoziationsklassen unterstützt. Stereotype sowie Multiplizitäten sind jedoch leicht erweiterbar, Constraints können individuell definiert werden. Neben dem XMI Export wird auch eine Dokumentation in HTML angeboten. Die Codegenerierung bzw. das Reverse Engineering von C, C++, C# und Java wird unterstützt. Der Preis liegt je nach Anzahl an gewünschten Lizenzen zwischen 125 $ und 1280 $.

Objecteering/UML

Obejcteering/UML wurde in der Version 5.3.0 SP3 Enterprise Edition Trial getestet. Das Tool bietet sämtliche geforderte Modellelemente an wobei Geschachtelte Klassen nur im Navigationsbaum gezeigt und nicht graphisch dargestellt werden. Obejcteering/UML bietet nur wenige vordefinierte Stereotypen, es gibt jedoch einen eigenen Editor zur Erstellung benutzerspezifischer Stereotypen. Neben dem XMI Export werden auch die Erstellung einer HTML Dokumentation sowie von RTF und Postscript Dokumenten ermöglicht. Zusätzlich bietet das Tool Möglichkeiten zur Integration von Eclipse sowie zum Import von Rational Rose Modellen. Außerdem wird die Codegenerierung in Java, C++, C#, Corba und SQL-DDL für Oracle angeboten. Es wird eine freie Personal Edition angeboten, alle anderen Editionen sind auf eine 30 Tage Evaluierungsphase beschränkt. Der Preis für eine Lizenz muss beim Hersteller erfragt werden.

ObjectIF

ObjectIF wird in verschiedenen Varianten angeboten, das heißt mit Eclipse Integration oder Visual Studio Integration. Das Werkzeug wurde für diese Arbeit in der Version ObjectIF Eclipse Personal Edition getestet. Festzustellen ist, dass weder N-äre Assoziationen noch Assoziationsklassen unterstützt werden. Geschachtelte Klassen werden nur im Navigationsbaum gezeigt jedoch nicht graphisch dargestellt. Stereotype und Constraints sind wie bei den meisten anderen Tools erweiterbar bzw. frei definierbar. ObjectIF bietet im Gegensatz zu den anderen evaluierten Tools keine Multiplizitäten. Neben dem Export als XMI wird hier auch XML angeboten. Dokumentationen können in HTML und DOC erstellt werden. In der freien Edition wird die Codegenerierung für Java, C++ und C# nicht angeboten alle anderen Editionen verfügen über dieses Feature. Lizenzen werden sowohl für einzelne Arbeitsplätze als auch in einer Floating-Version angeboten. Die Preise hierfür liegen zwischen 490 € und 1990 € für eine Arbeitsplatzlizenz und zwischen 1990 € und 3990 € für eine Floating Lizenz.

Poseidon

Dieses Tool wird ebenfalls in verschiedenen Editionen angeboten, getestet wurde die Community Edition. Das einzige Element, das hier nicht dargestellt werden kann sind n-äre Assoziationen. Geschachtelte Klassen können erstellt werden, werden jedoch nicht graphisch dargestellt. Stereotype und Constraints sind wie bei den meisten Tools erweiterbar und frei definierbar. Poseidon ermöglicht den Export des Modells mittels XMI und mit erweiterter Lizenz mittels WMF. Das Modell kann in verschiedenen graphischen Formaten wie GIF, JPG gespeichert bzw. exportiert werden. In der freien Version ist die Codegenerierung nur für Java möglich, sonst werden auch C, C#, C++, Corba, Delphi, Perl, PHP und SQL-DDL unterstützt. Die Codegenerierung kann über Templates angepasst werden, ob dies auch für den Export des XMI Files gilt ist jedoch fraglich. Außerdem besteht die Möglichkeit zur Eclipse Integration sowie zum Import von Rational Rose Modellen. Dieses Tool ist in einer freien Version erhältlich, sonst ist die unentgeltliche Verwendung auf eine 30-tägige Testphase beschränkt. Je nach Edition kostet eine Lizenz für einen einzelnen User bis zu 1990 €, sonst reichen die Preise bis 2990 €. Zusätzlich besteht die Möglichkeit eine erweiterte Lizenz für die freie Version zu erwerben, diese kostet 5 € im Monat.

Power Designer

Dieses Tool wurde in der Version Power Designer 12.5 Evaluation getestet. Hier ist festzustellen, dass wiederum n-äre Assoziationen nicht darstellbar sind. Ein weiteres Problem bestand in der Anwendung des Tools selbst, laut Dokumentation sind Stereotype und Constraints zwar erweiterbar, dies konnte jedoch in der Testversion nicht bewerkstelligt werden. Laut Dokumentation ist auch die Verwendung von OCL in Constraints möglich, da sich jedoch keine Constraints anlegen ließen konnte auch dies nicht überprüft werden. Das Tool bietet Möglichkeiten zum XMI Export und erlaubt die Codegenerierung und das Reverse Engineering für Java, C#, Corba, Visual Basic und XML (DTD, XLS). Laut Dokumentation sind die Mapping Definitionen anpassbar, dies konnte in der getesteten Version durch die eingeschränkte Funktionalität jedoch nicht überprüft werden. Zusätzlich wird der Import von Rational Rose Modellen ermöglicht. Power Designer kann 15 Tage lang

getestet werden über den Preis einer lizenzierten Version sind keine Informationen frei erhältlich.

Rational Developer

Bei diesem Tool ist zu beachten, dass es sich um einen enorm großen Download handelt. Getestet wurde die Version Rational Developer For System z V7.1 Trial for Multiplatform. Bis auf n-äre Assoziationen werden sämtliche Modellelemente angeboten, jedoch lassen sich geschachtelte Klassen nicht direkt im Diagramm darstellen. Die Informationen sind bei den Details der umgebenden Klasse gespeichert. Stereotype sind wiederum leicht erweiterbar und Constraints frei definierbar. Nachteil dieses Werkzeuges ist, dass der XMI Export zwar möglich ist, dazu aber ein weiteres Tool von der Homepage geladen werden muss. Geboten wird die Codegenerierung für C++, Ada, Corba, Java, Oracle 8, XML und Visual Basic. Rational Developer kann für 12 Tage getestet werden, die Lizenz kostet 4500 $.

Rhapsody

Für die Datenerhebung wurde die Version Rhapsody Developer 7.1 Evaluation getestet. Bei diesem Tool fehlen die drei Elemente: n-äre Assoziationen, Assoziationsklassen und geschachtelte Klassen. Stereotype sind erweiterbar, Constraints und Multiplizitäten sind frei definierbar, es werden jedoch für Multiplizitäten einige Beispiele geboten. Der Export mittels XMI wird wie gefordert angeboten. Die Codegenerierung bzw. das Reverse Engineering ist für Java, C, C++ und Ada möglich, wobei für jede Programmiersprache ein eigenes Modul vorhanden ist. Außerdem wird der Import und Export von Rational Rose und Eclipse Modellen angeboten. Rhapsody wird für eine Testphase von 30 Tagen unentgeltlich angeboten, die Preise für Lizenzen müssen direkt beim Hersteller erfragt werden.

Select Component Architect

Dieses Tool wurde in der Version Select Solution Factory 7.0 Evaluation getestet. Es werden sämtliche Modellelemente unterstützt. Zu beachten ist, dass Klassen in User und Business Classes unterteilt werden. Geschachtelte Klassen werden zwar nicht

als vollständige Klasse angezeigt, allerdings wird durch ein Symbol innerhalb der umgebenden Klasse indiziert, dass eine innere Klasse besteht. Stereotype und Constraints sind leicht erweiterbar bzw. defnierbar, Multiplizitäten können unabhängig von den Vorgaben gewählt werden. Neben dem Export des Modells in XMI wird auch die Generierung einer HTML Dokumentation unterstützt. Codegenerierung und Reverse Engineering wird für C, C++, Java, Ada und XML (XLS) angeboten. Der Import und Export von Rational Rose Modellen wird unterstützt. Wie bei einigen anderen Tools sind auch hier keine Preise ohne persönliche Anfrage erhältlich.

Together

Together 2007 wurde in einer Evaluation Edition bezogen und getestet. Generell werden durch dieses Tool alle geforderten Modellelemente unterstützt allerdings ist eine Besonderheit zu beachten. N-äre Assoziationen sind nur durch einen Umweg über Assoziationsklassen zu erstellen. Die Assoziationsklasse verfügt über die Raute die eine mehrwertige Beziehung indiziert. Nach Erstellung der Beziehungen müssen die Beziehung zur Assoziationsklasse und die Assoziationsklasse selbst gelöscht werden. Wie bei den meisten Werkzeugen sind Stereotype leicht erweiterbar, Multiplizitäten frei wählbar. Die Erstellung von OCL konformen Constraints wird unterstützt, allerdings werden in dem dafür bestimmten Editor Fehler zwar angezeigt aber keine Informationen angeboten welcher Fehler gemacht wurde. Der geforderte XMI Export wird geboten, außerdem besteht die Möglichkeit eine HTML Dokumentation zu erstellen bzw. das Modell als Grafik (GIF, JPG) abzuspeichern. Die Codegenerierung von C++, Java, DDL (SQL-Skript) und XSL wird angeboten. Together ist in einer 15 Tage Evaluierungsversion erhältlich, Preise für eine lizenzierte Version müssen beim Hersteller Borland erfragt werden.

UMLet

Dies ist ein sich in der Entwicklung befindendes Plug-in für Eclipse. Hier ist festzustellen, dass sämtliche Elemente dargestellt werden können. Die Bearbeitung dieser Elemente ist allerdings nur in einem eigenen Textfeld möglich. Das heißt sie werden in frei geschriebenem Text definiert. Der Vorteil darin liegt, dass sämtliche

Zusätze frei und unkompliziert definierbar sind. Allerdings wird auch nicht wirklich überwacht was erlaubt ist. In der derzeitigen Version sind nur Exportmöglichkeiten in graphischer Form gegeben, das heißt in JPG, SVG, PDF und EPS. Der XMI Export ist derzeit nicht implementiert, das Diagramm wird jedoch in Form eines XML Files gespeichert. Dieses folgt jedoch nicht den definierten XMI Konventionen.

Visual UML

Dieses Werkzeug wurde in der Version Visual UML 5.3 Developer Edition getestet. Es werden sämtliche Modellelemente unterstützt, allerdings sind Stereotype relativ kompliziert zu erweitern und daher Rollen eher schwer darzustellen. Muliplizitäten können zusätzlichen zu den angebotenen Vorlagen leicht erweitert werden. Constraints sind frei definierbar, der Aufbau im Sinne von OCL wird durch den Editor vorgegeben. Neben dem Export in XMI werden auch diverse andere Formate wie GIF, JPG, WMF und TIFF angeboten. Außer in der Standard-Edition wird die Codegenerierung und das Reverse Engineering für XML-DTD, C++, C#, Java, Visual Basic, VB.net und SQL-DDL für Oracle, SQL-Server und andere unterstützt. Außerdem können verschiedene mit anderen Werkzeugen wie Rational Rose und SQL Server erstellte Modelle importiert und exportiert werden. Sämtliche Editionen des Tools werden in einer 30 Tage Testversion angeboten. Die Standard Edition von Visual UML kostet für einen User 295 $ für 5 User 1295 $, analog die Developer Edition 495 $ und 2195 $, die Plus-Developer Edition 695 $ und 3095 $.

Nachdem sämtliche erhältliche Werkzeuge, die die Basiskriterien Darstellung von Klassendiagrammen und XMI-Export erfüllt haben, getestet wurden erfolgte ein direkter Vergleich in Bezug auf die vordefinierten Kriterien. Tabelle 4 gibt eine Übersicht welche Werkzeuge die benötigten Klassendiagramm-Elemente aus Kapitel 3.2.2 unterstützen und den XMI-Export anbieten. Hierbei handelt es sich um K.O.-Kriterien für die Auswahl des Werkzeuges. Das heißt das sämtliche Werkzeuge, die diese Anforderungen nicht erfüllen, für die weitere Analyse nicht in Betracht gezogen werden. Hierfür ist es irrelevant ob die bisher zur Verwendung geplanten Elemente nicht unterstützt werden oder nur diejenigen für die bisher nur die Möglichkeit der Verwendung besteht. In der Tabelle wird keine Rücksicht darauf genommen, wie die Darstellung der Elemente erfolgt bzw. ob diese leicht bewerkstelligt werden kann.

Tools welche die primären Anforderungen erfüllen werden in der Tabelle färbig hinterlegt.

Auffallend ist, dass vor Allem N-äre Assoziationen und Assoziationsklassen nicht berücksichtigt werden. Außerdem bleiben Elemente zur Erweiterung der Grundsemantik unbeachtet, das heißt Einschränkungen und Stereotype werden nicht unterstützt. Bei einigen Tools werden weder Rollen noch geschachtelte Klassen zur Verfügung gestellt. Weiters ist zu beachten, dass manche Tools zwar XMI unterstützen, dies aber teilweise auf dessen Import beschränkt wird oder eine zusätzliche Lizenz bzw. ein extra Tool erfordert.

	Argo UML 0.24	Open Ameos	Artisan Studio 6.2	Eclipse UML 3.3.0 Frei	Enterprise Architekt 7.0	Ideogramic UML Desktop Edition	Innovator 2007 9.1.03 Object Excellent	Innovator 2007 9.1.03 Object Classic	Magic Draw UML Community Edition	Meta Mill UML CASE Tool v4.2	Objecteering/UML 5.3.0 SP3 Enterprise Edition
Klasse	x	x	x	x	x	x	x	x	x	x	x
Assoziation	x	x	x	x	x	x	x	x	x	x	x
gerichtete Assoziation	x	x	x	x	x	x	x	x	x	x	x
N-äre Assoziation		x	x		x		x	x	x		x
Multiplizitäten	x	x	x		x	x	x	x	x	x	x
Rekursive Assoziation	x	x	x	x	x	x	x	x	x	x	x
Assoziationsklasse	x	x	x		x			x	x		x
Aggregation	x	x	x	x	x	x	x	x	x	x	x
Komposition	x	x	x	x	x	x	x	x	x	x	x
Generalisierung	x	x	x	x	x	x	x	x	x	x	x
Einschränkung		XOR	x	x	x		x	x	x	x	x
Rollen	x		x	x	x				x	x	x
Geschachtelte Klassen	x		x	x	x				x	x	x
Kommentare	x	x	x	x	x	x	x	x	x	x	x
Stereotype	x	fixe	x	x	x			x	x	x	x
XMI	x	x	x	Import	x	x	x	x	x	x	x

Tabelle 6: Erfüllung Muss-Kriterien durch UML-Werkzeuge

	ObjectIF Eclipse Personal Edition	Poseidon Community Edition	Power Designer 12.5	Rational Developer For System z V7.1	Rhapsody Developer 7.1	Select Solution Factory 7.0	Together 2007	UMLet	Visual UML 5.3 Developer Edition
Klasse	x	x	x	x	x	x	x	x	x
Assoziation	x	x	x	x	x	x	x	x	x
gerichtete Assoziation	x	x	x	x	x	x	x	x	x
N-äre Assoziation						x	x	x	x
Multiplizitäten	-	x	x	x	x	x	x	x	x
Rekursive Assoziation	x	x	x	x	x	x	x	x	x
Assoziationsklasse		x	x	x		x	x	x	x
Aggregation	x	x	x	x	x	x	x	x	x
Komposition	x	x	x	x	x	x	x	x	x
Generalisierung	x	x	x	x	x	x	x	x	x
Einschränkung	x	x	x	x	x	x	x	x	x
Rollen	x	x	x	x	x	x	x	x	x
Geschachtelte Klassen	x	x	x	x		x	x	x	x
Kommentare	x	x	x	x	x	x	x	x	x
Stereotype	x	x	x	x	x	x	x	x	x
XMI	x	Lizenz	x	extra Tool	x	x	x	x	x

Forts. Tabelle 6: Erfüllung Muss-Kriterien durch UML-Werkzeuge

In einem weiteren Schritt werden die Tools, die die Pflichtkriterien erfüllen auf zusätzliche Funktionalitäten hin überprüft und verglichen. Die Erfüllung von Wunschkriterien sowie zusätzliche Features und der Preis des Tools werden hier in Betracht gezogen. Tabelle 5 zeigt die relevanten Informationen zum möglichen Werkzeugpool. Wie bereits zu Beginn des Kapitels erläutert wurde, werden OCL und die Anpassbarkeit des generierten XMI-Dokuments als wünschenswert betrachtet. Zusätzliche Informationen werden in den Punkten Export bzw. Dokumentation, Preis und Plattformen zusammengefasst. Manche Tools bieten auch Code-Generatoren für diverse Programmiersprachen und Datenbankschemata. Informationen die nicht in diesen Punkten integriert werden können, werden in den allgemeinen Punkt Extras aufgenommen.

Bei der Auswertung der zusätzlichen Informationen wurde festgestellt, dass viele Werkzeuge dieselben Features bieten. Einige bieten jedoch spezielle Besonderheiten. Allerdings ist zu beachten, dass nicht sämtliche Versionen eines Tools das gleiche Leistungsspektrum umfassen. Im Folgenden werden die verschiedenen Zusatzleistungen näher betrachtet.

Für den Bereich der OCL Unterstützung ist festzustellen, dass nur die Hälfte der Werkzeuge Hilfen für OCL Statements bieten. Der Benutzer wird meist bei der Erstellung der Constraints unterstützt, beispielsweise durch die Vorgabe des Kopfes. Zwei der Instrumente unterstützen laut Dokumentation auch die Validierung der erstellten OCL-Statements, allerdings erst in einer leistungsstärkeren Version. Together gibt zwar an, dass ein Statement fehlerhaft ist, jedoch erhält man keine Hinweise auf den enthaltenen Fehler.

	Artisan Studio 6.2	Enterprise Architekt	Magic Draw UML	Objecteering/UML
OCL		Constraints möglich, lt. Doku auch Validation (nicht in Desktop Edition)	wählbar, Kopf wird automatisch erzeugt, Validierung nur in Enterprise Edition	
Export/Doku	XMI	HTML, RTF, XMI	Xmi auch in Opensource	XMI, RTF, Postsript, HTML
Code anpassbar	über Templates (lt. Doku)			
Preis	30 Tage evaluierungsversion, Preise für die lizenzierte Version müssen beim Hersteller erfragt werden	30 Tage Evaluierung (alle 3 Editionen in einem Paket); je nach Personenzahl und Edition zwischen 95 und 335 $ / Person; Desktop Version für eine Person 135 $; für Unis 65 $ Desktop Edition bis 12839 $ für unbegrenzte Benutzerzahl bei Corporate Edition	je nach Version Opensource bis 2600 für Clientanwendungen	Personal frei, lizenzierte Version 30 Tage Evaluierung, Preise müssen beim Hersteller erfragt werden
Plattformen	Windows	Windows, Linux	Java	Windows, Linux, Solaris
Code Generierung	Java, C++, Ada, SQL-DDL (mit Atrisan OCS Lizenz)	C, C++, C#, Java, PHP, Visual Basic, VB.net, Delphi, XML (XLS), SQL-DDL (nicht in Desktop Edition)	java, c++, c#, SQL, Oracle (nicht in Opensource)	C++, C#, Java, Corba, SQL-DDL (Oracle)
Reverse Engineering		nicht in Desktop Edition	nicht in Opensource	
Extras	Rational Rose Import/Export		Rational Rose Import/Export; automatisches Zeichnen von Modellelementen und automatisch erscheinende Toolbar bei Elementen mit möglichen zusätzlichen Elementen	Eclipse integration, Rational Rose Import

Tabelle 7: Zusatzinformationen zu UML-Werkzeugen

	Select Solution Factory	Together	UMLet	Visual UML
OCL		wird durch Editor unterstützt, allerdings keine Angabe welche Fehler gemacht wurden		Constraints im Sinne von OCL, durch Editor vorgegeben
Export/Doku	XMI, HTML	XMI, HTML, JPG, GIF	JPG, SVG, PDF, EPS, Diagramm wird in XML Form gespeichert, jedoch nicht XMI konform	XMI, GIF,JPG,WMF, TIFF
Code anpassbar				
Preis	auf ein Modell beschränkte Demo-Version, volle Evaluierungsversion für 28 Tage auf Anfrage, Preise für die lizenzierte Version müssen beim Hersteller erfragt werden	15 Tage Evaluierung (funktioniert auch nacher noch), Preise für die lizenzierte Version müssen beim Hersteller erfragt werden	Open Source	Standard/Developer/Plus-Developer: EditionSingle User: 295/495/695 $, 5 User: 1295/2195/3095 $; 30 Tage Testversion
Plattformen	Windows	Solaris, Windows, Linux, Mac OS	Java	Windows
Code Generierung	C,C++, Java, Ada, XML (XLS)	C++, Java, DDL (SQL-Skript), XSL	wird erst entwickelt	XML-DTD, C++, C#, Java, Visual Basic, VB.net, SQL-DDL (Oracle, SQL-Server,…) (nicht in Standard-Version)
Reverse Engineering	x			Nicht in Standard-Edition
Extras	Rational Rose Import/Export		Rudimentäre Oberfläche, Elemente zwar editierbar aber nur handschriftlich	Rational Rose, SQL Server, ... Import/Export

Forts. Tabelle 7: Zusatzinformationen zu UML-Werkzeugen

Fast alle Werkzeuge unterstützen die Ausgabe von XMI Files. Hierfür stellt UMLet eine Ausnahme dar, dieses bietet zwar die Erstellung eines XML Files jedoch ist dieses nicht XMI konform. Neben dem Wunschausgabeformat XMI bieten die Werkzeuge zusätzlich die Erstellung von Dokumentationen in Textformaten wie HTML und RTF sowie die Sicherung der graphischen Darstellung in verschiedenen Bildformaten wie JPG und GIF.

Als weiteres Wunschkriterium wurde die Anpassbarkeit des generierten Codes festgelegt. Hier ist zu beachten, dass nur Artisan Studio eine derartige Möglichkeit zur Verfügung stellt. Allerdings besteht auch hier die Einschränkung auf die Einflussnahme durch Templates. Bei den anderen Werkzeugen liegt eine Beschränkung auf eine nachträgliche Adaption des XMI Dokuments vor, allerdings wird dadurch nicht gewährleistet, dass das Dokument erneut graphisch dargestellt werden kann.

In Bezug auf die Preise trat bei vielen Werkzeugen das Problem auf, dass diese nur durch persönliche Anfrage bei der Firma zu erfahren sind. Einige der Anbieter stellen eine kostenfreie vereinfachte Version des Tools zur Verfügung. Hier fehlen hauptsächlich die zusätzlichen Code-Generatoren. Bei Together wurde festgestellt, dass das Tool auch nach Ablauf der Evaluierungsperiode weiter verwendet werden konnte.

Bis auf UMLet bieten sämtliche Werkzeuge verschiedene Code-Generatoren. Diese erzeugen den Code in unterschiedlichen Programmiersprachen, einige aber auch SQL-DDL Skripte für verschiedene Datenbanksysteme. Die Datenbankunterstützung wird jedoch von keinem der Tools in der freien oder Standard-Version geboten. Zum Teil besteht auch die Möglichkeit des Reverse Engineering, dies ist wiederum auf höherwertige Versionen beschränkt.

Einige Werkzeuge bieten noch zusätzliche Extras: neben dem Rational Rose Import und Export wird auch eine Eclipse Integration zur Verfügung gestellt. In Bezug auf die Benutzeroberfläche ist zu beachten, dass UMLet nur eine relativ rudimentäre Oberfläche bietet, da sich dieses Werkzeug noch in der Entwicklungsphase befindet. Magic Draw hingegen bietet die automatische Erstellung von UML-Elementen. Bei

anvisieren eines Elements mit der Maus wird eine Toolbar mit allen für das Element verfügbaren Beziehungen sichtbar.

Zusammenfassend ist festzuhalten, dass Magic Draw in der Opensource-Version bereits alle geforderten Features bietet. Es können sämtliche benötigte UML-Modellelemente dargestellt werden sowie das Diagramm in XMI exportiert werden. Die Erstellung von OCL-Constraints wird durch einen Editor unterstützt, die Validierung kann jedoch erst in der Enterprise Edition durchgeführt werden. Da Magic Draw auf Java basiert ist es Betriebssystemunabhängig und kann daher umfassend eingesetzt werden. Zusätzlich bietet das Tool eine komfortable Benutzeroberfläche. Daher wäre es von Vorteil dieses Tool zur Erstellung der UML-Angabe für den logischen Entwurf einzusetzen.

Bei der Verwendung des Moduls ist jedoch zu beachten, dass bereits bei der Erstellung des UML-Diagramms auf den nachfolgenden Aufbau des Zielsystems Rücksicht genommen werden muss. Dies ist vor allem bei der Umsetzung einzelner Elemente von Bedeutung. Es muss gesichert werden, dass die Raute von n-Ären Beziehungen den Namen der Beziehung erhält und dass die Assoziationen zu den beteiligten Klassen nicht bezeichnet werden. Zusätzlich muss die Umsetzung der Rollen durch einen kleinen Trick bewerkstelligt werden. An Stelle eines Stereotyps muss ein generalizationSet zur Abbildung der Rollen herangezogen werden. Zusätzlich muss sich auch der Ersteller des UML-Diagramms an gewisse Namenskonventionen halten, es ist wichtig, dass die in dem Diagramm enthaltenen Bezeichnungen keine Leerzeichen enthalten und dass Klassennamen und Beziehungsnamen nur einmal vorkommen. Attribute verschiedener Klassen dürfen denselben Namen erhalten falls sie nicht ineinander integriert werden können.

5.2 Implementierung des Expertenmoduls

Dieses Kapitel beschäftigt sich mit der Erstellung des Prototyps. In Kapitel 5.2.1 wird die Architektur des Moduls betrachtet. Die verschiedenen Umsetzungsalternativen werden in Kapitel 5.2.2 vorgestellt. In weiterer Folge wird das Vorgehen zur Erfüllung der Aufgaben definiert (siehe Kapitel 5.2.3). In Kapitel 5.2.4 steht die Erstellung des Zielsystems im Mittelpunkt während sich Kapitel 5.2.5 mit der Auswertung der Lösung beschäftigt. Anschließend werden in Kapitel 5.2.6 die Einschränkungen des Prototyps definiert. Abschließend wird die Anwendung des Prototyps vorgestellt (siehe Kapitel 5.2.7).

5.2.1 Architektur

Bei dem hier implementierten Modul handelt es sich derzeit nur um einen Prototyp zur Erstellung und Auswertung der Aufgaben im Bereich des logischen Entwurfs. Dennoch lassen sich die Schnittstellen für die Anbindung an das bestehende eTutor System leicht erweitern. Die entsprechenden Klassen und Methoden sind bereits im Prototyp enthalten. Das Modul ist nach der in Abbildung 50 dargestellten Struktur aufgebaut. Zusätzlich sind Klassen zur Darstellung der verschiedenen Modellelemente enthalten. Diese werden in einem späteren Teil der Arbeit vorgestellt. Hier werden die den verschiedenen Funktionen zu Grunde liegenden Klassen betrachtet.

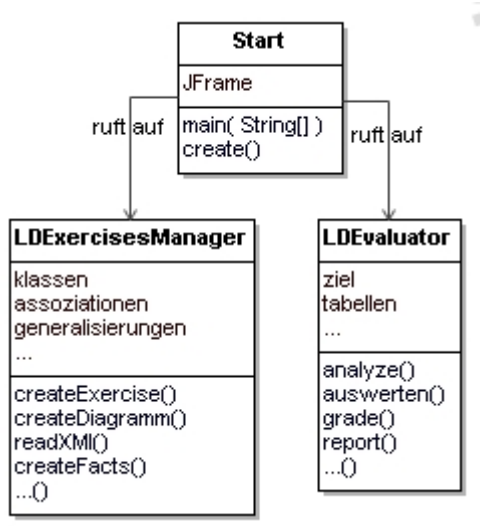


Abbildung 50: Architektur des Expertenmoduls

Die Klasse *Start* erstellt ein einfaches Frame in welchem die verschiedenen Aufgaben des Moduls durchgeführt werden. Die *main(String[])* Methode ruft *create()* auf. Hier wird die grundsätzliche Struktur der Benutzeroberfläche festgelegt. Ausgehend vom Menü des Frames werden die verschiedenen Funktionen des Prototyps aufgerufen. Neben einer Funktion zum Erstellen der Aufgabe werden welche zur Abgabe und Auswertung der Lösung, sowie zur Erteilung von Feedback und zur Bewertung angeboten. Diese Funktionen werden in verschiedenen Klassen dem Schema des eTutor entsprechend zusammengefasst.

In der Klasse *LDExercisesManager* sind sämtliche Methoden zur Erstellung einer Aufgabe zum logischen Entwurf enthalten. Die Klasse stellt Listen zum Zwischenspeichern der aus dem UML-Diagramm bzw. dem XMI-Dokument ausgelesenen Modellelemente zur Verfügung. Die verschiedenen Methoden erweitern zum Teil das aus der Klasse *Start* übergebene Frame. Die Methode *createExercise()* dient als Einstiegspunkt für die Erstellung einer Aufgabe. Sie ruft die Methoden zur Erstellung des Diagramms und zur Bewertung der Alternativen auf. Die Klasse verfügt über Methoden zum Einlesen des XMI-Dokuments, diese werden von der Methode *readXMI()* aufgerufen. Die hierfür notwendigen Packages, *org.w3c.dom.** und *org.xml.sax.**, werden importiert. Die ausgelesenen Elemente werden in den entsprechenden Listen zwischengespeichert. Die Methode *createFacts()* hingegen ruft sämtliche Untermethoden auf welche das Zielsystem erzeugen. Das Zielsystem wird mit Hilfe der Methoden des Packages *java.io.** in

einer eigenen Datei gespeichert und somit zur weiteren Verwendung zur Verfügung gestellt. Neben diesen Funktionen wird durch die Methode *createDiagramm()* auch eine Schnittstelle für eine spätere Integration eines UML-Editors geboten. Diese lädt derzeit nur das dem Diagramm entsprechende Bild.

Zur Abgabe und Auswertung der Lösung wird die Klasse *LDEvaluator* zur Verfügung gestellt. Die Liste *ziel* enthält das erstellte Zielsystem, welches aus dem von *LDExercisesManager* erstellten Dokument ausgelesen und als Grundlage für die Auswertung herangezogen wird. Die Methode *analyze()* erzeugt die Benutzeroberfläche für die Eingabe der Studentenlösung. Bei Abgabe der Studentenlösung wird die Methode *auswerten()* aufgerufen. Deren Untermethoden lesen die einzelnen Elemente des übergebenen DDL-Skripts in die Liste *tabellen* ein. Zusätzlich wird die Auswertung der Lösung durchgeführt und die entsprechenden Anmerkungen in verschiedene Dokumente geschrieben, eines für die generelle Dokumentation, eines für die Bewertung und eines für die Erteilung von Feedback. Durch die Methoden *grade()* und *report()* können die Dokumente für die Bewertung und das Feedback abgerufen werden. Aufgaben zum Lesen und Schreiben von Dokumenten werden wiederum durch die Methoden des Packages java.io.* unterstützt. Sämtliche Methodenaufrufe ändern das von der Klasse Start übergebene Frame.

5.2.2 Umsetzungsalternativen

Für die Auswertung der Aufgabe stehen grundsätzlich zwei verschiedene Varianten der Umsetzung zur Verfügung: einerseits kann ein Regelsystem herangezogen werden, andererseits können diese Regeln aber auch „händisch“ in Java programmiert werden. In beiden Fällen liegt der Hauptaufwand beim Auslesen des XMI-Dokuments und der gegenseitigen Zuordnung der einzelnen Elemente in dem durch Parsen des XMI entstandenen Baums. Auch die Berücksichtigung der verschiedenen Umsetzungsalternativen bleibt derselbe Aufwand, diese müssen sowohl in der Abfrage als auch beim Erstellen des Zielsystems berücksichtigt werden.

Der Vorteil des Regelsystems liegt darin, dass es auch trotz fehlender Informationen zu einer möglichen Lösung bzw. zu mehreren Lösungen führt. Sie sind vor allem dann hervorragend einsetzbar wenn klar definierte Regeln zur Erfüllung der Aufgaben erstellt werden können. Außerdem erlauben sie auch die Integration in Programmierumgebungen wie Eclipse, das heißt man kann auch eine normale GUI mit dem System verknüpfen. Der Nachteil dieser Systeme liegt allerdings darin, dass die Regeln genau definiert werden müssen, da sie einander in ihrer Ausführung beeinflussen können. In der prozeduralen Programmierung ist dieser Umstand einfacher zu berücksichtigen, da man hier genau weiß welche Abfragen aufeinander folgen und die gegenseitige Beeinflussung somit bei der Erstellung berücksichtigt werden kann. Regelsysteme folgen zwar auch einem Algorithmus allerdings ist die Abfolge der Regelaufrufe nicht so einfach zu verfolgen. Es wird zwar definiert was das System zu tun hat, wie es diese Aufgabe bewerkstelligt legt es jedoch selbst bzw. der zu Grunde liegende Algorithmus fest. Da die vorliegende Aufgabe relative klar definierte Schritte verfolgt, wird für diese Arbeit eine Umsetzung der Regeln in Java gewählt.

5.2.3 Vorgehen

Für die Erstellung der Aufgabe und die Auswertung der Lösung müssen bestimmte Schritte abgearbeitet werden. Die Erstellung der Aufgabe hat in jedem Fall vor einer möglichen Auswertung zu erfolgen. Basierend auf dem erstellten Zielsystem können in weiterer Folge mehrere Lösungen ausgewertet werden. Das heißt das Zielsystem muss nur einmal erstellt werden. Daher werden die Erstellung und die Auswertung getrennt voneinander betrachtet.

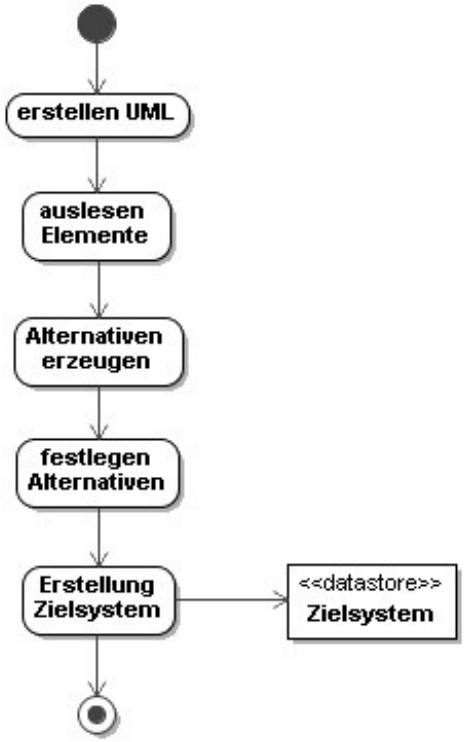


Abbildung 51: Ablauf Erstellung des Zielsystems

Die Erstellung des Zielsystems (Abbildung 51) beginnt mit der Erzeugung des UML-Diagramms. Die einzelnen Elemente werden aus dem zu Grunde liegenden XMI-Dokument ausgelesen. Darauf aufbauend werden dem Lehrenden die verschiedenen alternativen Umsetzungsvarianten vorgelegt, damit dieser festlegen kann welche konkreten Möglichkeiten erlaubt werden und wie die Punktevergabe zu erfolgen hat. In weiterer Folge wird das tatsächliche Zielsystem erstellt. Dieses wird in einer eigenen Datei gespeichert um als Grundlage für die Auswertung der Studentenlösung zu dienen.

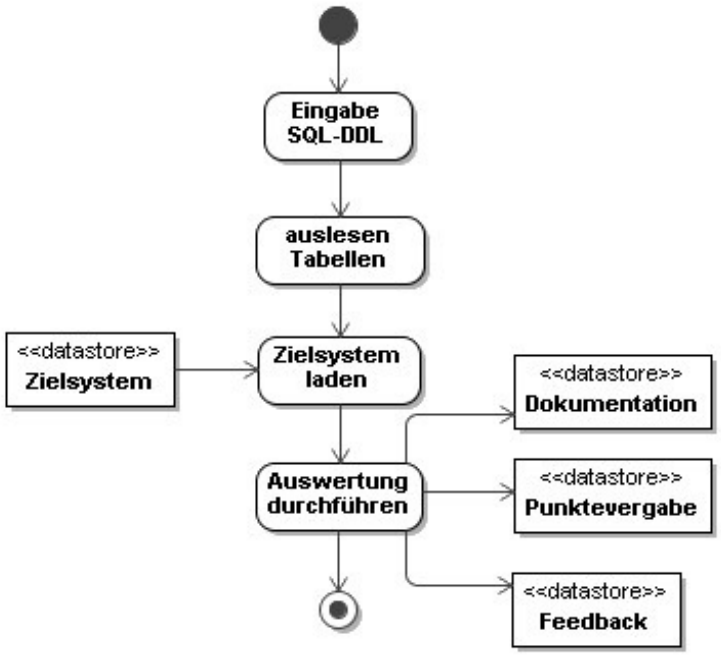


Abbildung 52: Ablauf Auswertung der Lösung

Die Auswertung der Aufgabe (Abbildung 52) startet mit der Eingabe des SQL-DDL Skripts durch den Studenten. Sobald dieser die Aufgabe abgibt werden die erstellten Tabellen ausgelesen und das Zielsystem geladen. Anschließend werden Lösung und Zielsystem verglichen. Die aus dem Vergleich resultierenden Informationen werden in verschiedenen Dokumenten zwischengespeichert. Einerseits werden gefundene Informationen dokumentiert andererseits fließen Fehler in das Feedback und die Bewertung ein. Diese können nach erfolgter Auswertung abgerufen werden.

5.2.4 Erstellung des Zielsystems

Wenn das Diagramm fertig erstellt wurde, wird bei Erstellung in einem externen Editor das XMI durch den Lehrenden an das System übergeben. Falls ein integrierter Editor verwendet wird, muss das System das XMI selbst generieren und Speichern. Im Prototyp wird derzeit nur ein extern erstelltes XMI-Dokument verwendet. Die Quelle muss im Programmcode geändert werden. Im Anschluss werden die verschiedenen Mapping-Alternativen ausgewiesen, zusätzlich werden Vorschläge für die Punktevergabe erzeugt. Diese basieren auf der Auswertung der in der Literatur gängigen Varianten.

Um dies zu bewerkstelligen müssen die einzelnen Elemente aus dem XMI ausgelesen werden. Zu beachten ist hier, dass in dem Dokument etliche Informationen enthalten sind, welche nicht benötigt werden bzw. erst für die Verwertung adaptiert werden müssen. Zum einen ist zu Beginn des Dokuments ein großer Block bezüglich des Erstellers und diverser allgemeiner Informationen über das Diagramm enthalten, wie Packageimports und eigens definierte Zusicherungen. Dieser wird weder bei der Erstellung des Regelsystems noch bei der Auswertung benötigt. Zum anderen sind bei jedem Element Objekt-ID's enthalten, welche zwar innerhalb des Dokuments zur Identifikation von an Beziehungen beteiligten Klassen benötigt werden, innerhalb des Zielsystems ist deren Verwendung allerdings nicht sinnvoll. Hier sollten die Bezeichnungen der einzelnen Klassen angewendet werden.

Aus diesen Betrachtungen ist ersichtlich, dass die Informationen aus dem XMI-Dokument ausgelesen und teilweise in veränderter Form gespeichert werden müssen. Dazu ist es notwendig zu wissen wie die Darstellung der einzelnen Elemente im XMI-Dokument erfolgt. Hierfür wurde das eingangs definierte Beispiel um Datentypen, Zusicherungen und Defaultwerte erweitert. Das somit entstandene Diagramm wird in Abbildung 53 dargestellt.

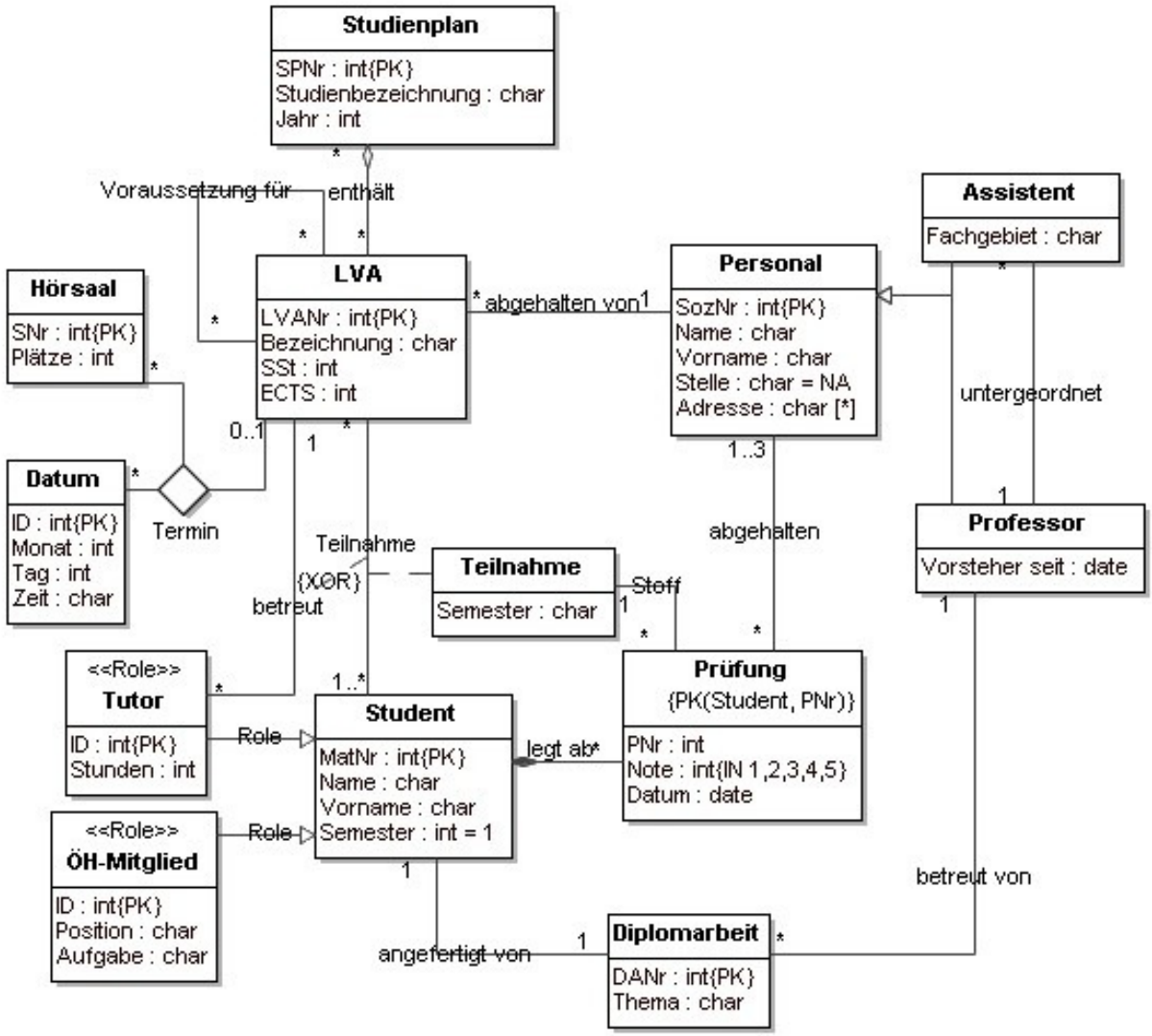


Abbildung 53: Beispiel Uni mit Erweiterungen

Aus diesem Diagramm wird durch das verwendete Tool Magic Draw ein XMI Dokument erstellt (durch Export EMF UML (v2.x) XMI). Da man davon ausgehen muss, dass die verschiedenen Elemente nicht unbedingt in geordneter Reihenfolge in dem Dokument enthalten sind, muss man die entsprechenden Elemente beim Auslesen zuordnen. Das heißt es werden die umschließenden Tags auf ihre Übereinstimmung mit den zu suchenden Elementen hin überprüft. Tags mit dem Beginn *ownedRule* enthalten Zusicherungen, *packagedElement* leitet entweder Klassen oder Assoziationen ein, *ownedAttribute* sind Attribute bzw. Beziehungsenden usw. Die genaue Zuordnung wird in Anhang A beschrieben.

Somit muss nur noch die Frage beantwortet werden, wie diese Elemente ausgelesen werden können. Diese müssen zwischengespeichert und die einzelnen Bestandteile

der Elemente zusammengefügt werden. Die Zusammenführung der Elemente erfolgt über die IDs, außerdem müssen die IDs der Assoziationsenden durch den Namen der entsprechenden Klassen ersetzt werden.

Da es sich bei dem XMI-Dokument um ein spezielles XML-Dokument handelt können hier alle Mittel angewendet werden wie bei normalen XML-Dokumenten. Das heißt die Daten können unter Verwendung eines Parsers ausgelesen werden oder mittels XSLT transformiert werden, sie können auch mit XPath bearbeitet werden. Für die vorliegende Arbeit ist es allerdings nur notwendig die in dem XMI-Dokument enthaltenen Daten auszulesen. Sie müssen ohnehin durch einen Algorithmus in das Zielsystem transformiert werden. Das Dokument wird daher geparst und anhand der Knotentypen ausgewertet, das heißt es wird jeweils eine Liste der Elemente mit bestimmten Starttags erstellt und jeder Knoten abgearbeitet.

Das auslesen der Elemente erfolgt in der Klasse *LDExercisesManager* in der Methode *readXMI()*. Für die einzelnen Elemente werden Listen angelegt welche die entsprechenden Objekte enthalten. Es werden sämtliche zusammengehörige Elemente in den jeweiligen Objekten gespeichert. Dies erfolgt nach dem in Abbildung 54 dargestelltem Muster. Die genaue Erklärung zu den einzelnen Attributen erfolgt im Data-Dictionary in Anhang B.

Aggregationen und Kompositionen werden wie Assoziationen dargestellt, sie werden jedoch in einer eigenen Liste eingetragen. Rollen werden ebenfalls in einer eigenen Liste abgebildet, die Darstellung erfolgt analog zu Generalisierungen. Die enthaltenen Beziehungen müssen dem Lehrenden zur konkreten Bewertung dargeboten werden. Somit besteht die Möglichkeit individuell festzulegen wie welche Möglichkeiten bewertet werden. Hier kann die gesamte Liste der Elemente mit deren Bezeichnung vorgelegt werden oder etwa durch Kombination mit einem integrierten UML-Editor gekennzeichnet werden welche Beziehung gerade betrachtet wird. Wie bereits erwähnt werden nur die Bezeichnungen der Beziehungen vorgelegt und die möglichen Umsetzungsalternativen zur Auswahl angeboten. Da Generalisierungen keine Bezeichnung enthalten wird hier die Superklasse angegeben.

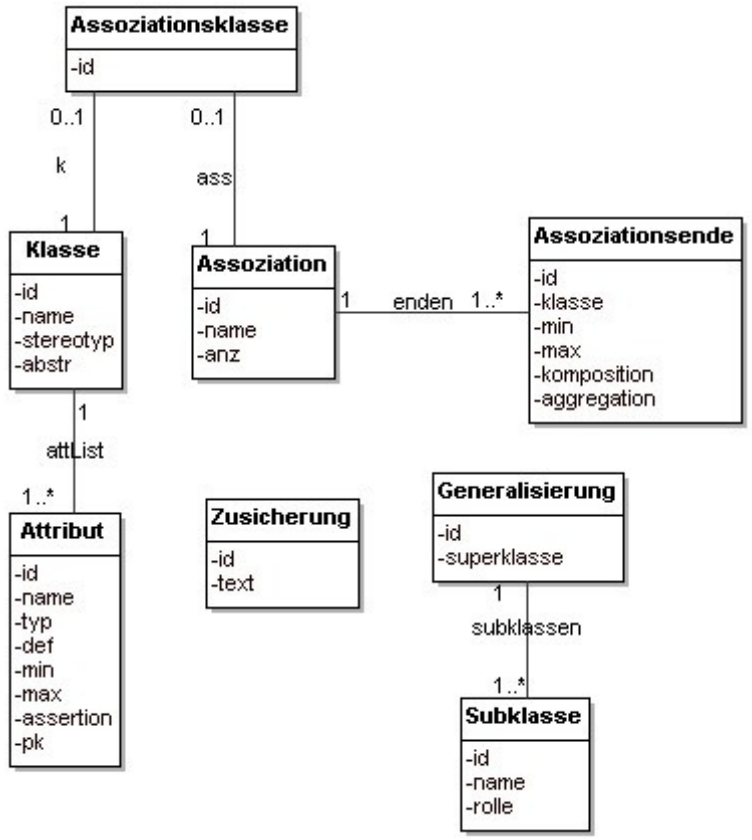


Abbildung 54: Speicherstruktur

Anschließend werden die Alternativen mit den vom Lehrenden definierten Punkten gespeichert. Hier werden wegen effizienter Speichernutzung Varianten, die keine Auswirkungen auf die Punktevergabe haben, nicht berücksichtigt, verbotene Alternativen werden daher nicht in das Zielsystem aufgenommen.

Die Elemente werden systematisch auf ihr Vorhandensein und die Art ihrer Abbildung in Tabellen überprüft (Methode: *createFacts()* in *LDExercisesManager*). Erst werden die Klassen an sich überprüft im Anschluss werden die einzelnen Beziehungen und ihre Auswirkungen auf die Klassen betrachtet. Zu berücksichtigen ist, dass hier wiederum verschiedene Umsetzungsvarianten gewählt werden können.

Bei der Erstellung von einzelnen Klassen gibt es kaum Alternativen, diese werden nur bei der Umsetzung einer Beziehung durch Integration beeinflusst. Das heißt bei der Erstellung des Zielsystems können die aus dem XMI-Dokument ausgelesenen Klassen übernommen werden. Nur Arrays müssen in eine eigene Tabelle

ausgelagert werden. In dieser Tabelle muss dann das Löschverhalten ON DELETE CASCADE berücksichtigt werden.

Zu den Klassen hinzugefügt werden jedoch die Attribute *bez* und *muss*. Der Boolean-Wert *bez* gibt an ob eine Tabelle nur als Umsetzung einer Beziehung vorhanden ist. *Muss* ist ein Integer-Wert und gibt die Zahl der Beziehungen an, an der die Klasse beteiligt ist. Wenn dieser Wert größer 1 ist, muss die Klasse in der Lösung vorhanden sein, es handelt sich somit um eine Muss-Klasse. Zu jedem einzelnen Attribut wird ein Objekt der Klasse *ForeignKey* hinzugefügt, dieses wird belegt wenn es sich um einen möglichen oder verpflichtenden Foreign Key in der jeweiligen Tabelle handelt. Das *ForeignKey* Objekt enthält dessen Namen, die ID der Beziehung und einen Wahrheitswert, welcher angibt ob der Foreign Key bei Löschen des entsprechenden Primary Key mitgelöscht werden muss. Sollte ein Foreign Key aus mehreren Attributen bestehen wird dieser Eintrag ebenfalls bei jedem einzelnen Attribut integriert.

Wie bereits erwähnt müssen bei den Klassen nach dem Auslesen aus dem XMI keine gravierenden Änderungen durchgeführt werden (Methode: *createKlassen()* in *LDExercisesManager*). Allerdings werden sie durch die Beziehungen beeinflusst welche sie eingehen. Je nach Art der Beziehung hat dies unterschiedliche Folgen. Die Aufrufe erfolgen durch *createBeziehungen()* in *LDExercisesManager*.

Assoziationen

- Wenn es sich um eine 1:1 Assoziation handelt und die jeweilige Klasse keine Muss-Klasse ist, kann diese in der anderen Klasse integriert werden, das heißt die eine Klasse ist nicht vorhanden, während die andere mehr Attribute enthält. Dieser Umstand wird durch eine zweite Klasse mit diesen Attributen repräsentiert, deren Name ist nach folgendem Muster aufgebaut : Name der zu integrierenden Klasse + „IN“ + Name der Klasse in die integriert wird
- 1:N Assoziationen können außerdem durch einen Foreign Key am höherwertigen Ende umgesetzt werden. Dieser erhält wiederum den Wahrheitswert bez, welcher angibt, dass es sich hier um eine Umsetzungsvariante der Beziehung handelt. Eine derartige Umsetzung ist auch bei 1:1 Assoziationen möglich.

- M:N können nur durch eine Zwischentabelle umgesetzt werden, diese Variante gilt jedoch auch für die beiden anderen Assoziationstypen. Zur Repräsentation dieser Umsetzung wird eine eigene Klasse mit dem Attribut bez=true umgesetzt diese enthält Foreign Keys auf alle an der Beziehung beteiligten Klassen, hier ist der FK nicht zu Löschen wenn das referenzierte Objekt gelöscht wird.

Aggregationen

- Wie Assoziationen

Kompositionen

- Diese Beziehung erfordert einen Foreign Key in der Tabelle des Teils. Hier ist das entsprechende Löschverhalten zu berücksichtigen.

Generalisierungen

- Um die Zusammenlegung der Klassen zu repräsentieren wird wiederum eine eigene Klasse mit den zu integrierenden Attributen erstellt.
- Wenn alle Tabellen bestehen bleiben wird der Primary Key der Superklasse in den Subklassen als Primary Key und gleichzeitig Foreign Key aufgenommen

Rollen

- Hier ist nur der Foreign Key auf die Ausgangsklasse sowie dessen Löschverhalten einzutragen

Zusätzlich zu den hier angeführten Änderungen wird eine eigene Liste mit den Namen und den Möglichkeiten der Beziehungsumsetzung erstellt, diese dient in weiterer Folge zur Abfrage ob alle Beziehungen umgesetzt wurden. Die Liste wird an den Beginn der Klassenliste angehängt. Außerdem wird ein Array mit der gewählten Punktevergabe für die Fehlertypen gespeichert.

5.2.5 Auswertung und Feedback

Für die Lösung der Aufgabe ist als erstes die Angabe von Bedeutung, das heißt das als XMI zu Grunde liegende UML-Diagramm wird dem Studenten vorgelegt. Zusätzlich wird ein integrierter Texteditor zur Erstellung der Lösung geboten. Neben den hier angeführten Funktionen müssen auch die Verschiedenen Abgabe-Modi

wählbar sein. Der Student soll zwischen Übungsmodus und Abgabemodus entscheiden können. Außerdem besteht auch die Möglichkeit die Art des Feedbacks festzulegen. Dies wird im Grunde genommen durch das Kernsystem des eTutor realisiert. Nach erfolgter Ausarbeitung kann der Student die Lösung zur Auswertung abschicken. Sollte die Abgabe nicht durch das System ausgewertet werden können, muss dies dem Studenten mitgeteilt werden.

Beim Auslesen des Textes wird von einem nach SQL-92 Standard korrekt aufgebauten DDL-Skript ausgegangen. Der übermittelte Text wird hier wiederum anhand verschiedener Schlüsselwörter ausgewertet wobei nicht alle Schlüsselwörter des Standards berücksichtigt werden. Die relevanten Schlüsselwörter werden im folgenden Text angesprochen. Jede Tabelle wird durch CREATE TABLE eingeleitet und endet bei „);“. Der Text des gesamten Eintrags wird zeilenweise abgearbeitet. Eine Zeile beginnt mit einem Spaltennamen bzw. mit CHECK, PRIMARY KEY oder FOREIGN KEY. PRIMARY KEY kann auch bei einer Spalte hinzugefügt werden, das heißt es wird nach dem Schlüsselwort gesucht, wenn es am Beginn der Zeile steht müssen die entsprechenden Attribute zugeordnet werden. Gleiches gilt für REFERENCES: wenn in derselben Zeile kein FOREIGN KEY aufzufinden ist handelt es sich um einen bei der Spalte angegebenen Foreign Key. Bei den Foreign Keys wird außerdem überprüft ob das Löschverhalten CASCADE implementiert wird. Diese Daten werden wiederum in einer Liste welche Elemente von Typ Klasse enthält abgespeichert und zur Durchführung der Analyse zur Verfügung gestellt.

Zur Auswertung wurde wie bereits erwähnt ein eigener Java-Algorithmus erstellt. Dieser ist in der Klasse *LDEvaluator* enthalten und startet mit der Methode *analyze()*. In der Liste der Tabellen welche das Zielsystem abbildet, wurde ebenfalls eine Liste mit den in der Angabe enthaltenen Beziehungen abgelegt. Diese wird zu Beginn der Auswertung ausgelesen und aus dem Zielsystem entfernt. Im Anschluss arbeitet der Algorithmus die im Zielsystem enthaltenen Klassen der Reihe nach ab. Nach diesem Durchlauf werden die Elemente in der Lösung auf ihr Vorkommen im Zielsystem hin untersucht. Als letzter Schritt erfolgt eine Abfrage, ob Beziehungen im Ziel sind welche nicht in der Lösung umgesetzt wurden. Die Beziehungen wurden in der Liste während der Abarbeitung der anderen Elemente als umgesetzt markiert wenn eine

Tabelle bzw. ein Foreign Key oder eine Integration im Zuge der Beziehungsumsetzung erkannt wurde.

Jede im Zielsystem enthaltene Tabelle wird in Abhängigkeit ihrer Eigenarten behandelt. Tabellen welche in jedem Fall enthalten sein müssen werden sofort in Bezug auf ihre Spalten hin analysiert, das heißt Primary Key, Defaultwert und eventuelle CHECK-Regeln werden kontrolliert falls sie im Ziel enthalten sind. Wenn Foreign Keys enthalten sind werden diese ebenfalls abgearbeitet. Die von diesen repräsentierten Beziehungen werden als vorhanden markiert. Tabellen welche integriert werden können werden anhand der Attribute welche in der sie aufnehmenden Tabelle enthalten sind identifiziert. Wenn eines der Attribute enthalten ist wird überprüft, ob auch alle anderen Attribute übernommen wurden. Wenn die Tabelle eine Beziehung umsetzt und keine Integration darstellt handelt es sich um eine Koppeltabelle. Wenn diese vorhanden ist, werden die enthaltene Foreign Keys überprüft und die Beziehung als umgesetzt markiert. Sollte eine Beziehungsumsetzung entdeckt werden deren Beziehung bereits als vorhanden markiert ist, wird festgehalten, dass diese doppelt umgesetzt wurde.

Für diese Abfragen stehen die entsprechenden Methoden zur Verfügung. Es wird jeweils abgefragt ob ein Element welches laut Zielsystem vorhanden sein sollte tatsächlich in den vom Studenten erstellten Tabellen enthalten ist. Um zu überprüfen ob Elemente in der Lösung sind welche im Zielsystem nicht vorhanden sind wird die Methode *inZiel (String name)* eingeführt. Diese liefert sowohl bei Klassen als auch bei Assoziationen den Wert „true“ wenn ein fremder Name auftritt.

Die Reaktion auf diese Auswertung wird in drei verschiedenen Dokumenten gespeichert. Als erstes wird im Zuge der Auswertung die Dokumentation erstellt. Diese enthält die Abweichungen und Übereinstimmungen von Lösung und Zielsystem. Sie wird im Grunde genommen nur intern verwendet. Zusammen mit diesem Dokument werden zwei weitere, eines für die Punktevergabe und eines fürs Feeback, generiert. Die Feedbackvermittlung und die Informationen bezüglich der Punktevergabe werden zusammen mit der Dokumentation bei der Auswertung der Lösung erstellt. Im Grunde genommen ist es hier nur mehr nötig das entsprechende Dokument auszulesen und in der GUI auszugeben.

Beispiel

Wenn in der Angabe eine Klasse Personal enthalten ist welche in der Lösung nicht umgesetzt wurde, erzeugt das System folgende Ausgab in den Dokumenten:

Dokumentation: „Personal nicht vorhanden“
Punkte: „Tabelle Personal -2“
Feedback: „Die Klasse Personal muss in die Tabelle Personal umgesetzt werden“

5.2.6 Einschränkungen

Da es sich bei dem bisher implementierten Modul um einen Prototyp handelt sind diverse Einschränkungen zu beachten. Diese entstanden da hier nur die Funktionalität der Auswertung überprüft werden sollte und zusätzliche Funktionen nicht beachtet wurden. Konkret handelt es sich hierbei um folgende Einschränkungen:

- Die Anbindung an das eTutor System wurde nicht vorgenommen, da nicht alle Funktionalitäten enthalten sind und eine eigene Benutzeroberfläche für Testzwecke besser geeignet ist.
- Die Einbindung eines UML-Editors wurde unterlassen, dieser ist für die Überprüfung der Auswertung nicht notwendig. Aus diesem Grund werden die graphische Darstellung und das XMI-Dokument in eigenen Dateien gespeichert und direkt im Programmcode eingelesen.
- Die Einschränkung der Alternativen wurde in dem Prototyp nicht übernommen. Diese müssten bei der Erstellung des Zielsystems berücksichtigt werden. Die Auswertung selbst wird dadurch jedoch nicht beeinflusst, da überprüft wird ob

erlaubte Alternativen umgesetzt wurden. Aus welchem Grund die Alternative nicht erlaubt ist wird in dem derzeitigen System nicht beachtet.

- Assertions wurden bei der Auswertung nicht berücksichtigt da diese in unterschiedlicher Weise umgesetzt werden können und somit sämtliche Möglichkeiten beachtet werden müssten. Dieser Punkt stellt eine Erweiterungsmöglichkeit für das hier entworfene Modul dar.
- Tippfehler in der Lösung führen automatisch zu Fehlern. Bei der Auswertung wird von einem korrekten Aufbau des Dokuments ausgegangen. Die Namenskonventionen müssen exakt eingehalten werden. Bei den Schlüsselwörtern der DDL werden jedoch sowohl Groß- als auch Kleinschreibung toleriert.

5.2.7 Anwendung des Prototyps

Anhand des bereits in Abbildung 53 vorgestellten Beispiels wird nun die Funktionsweise des Prototyps erläutert. Beim Starten des Prototyps wird eine rudimentäre Benutzeroberfläche, genauer gesagt die Menüleiste des Frames, angezeigt (Abbildung 55).

Abbildung 55: Startseite des Prototyps

Der Menüpunkt *Administration* enthält einen Unterpunkt zum Anlagen der Aufgabe. Der Punkt *Auswertung* wird in *Lösung analysieren*, *Lösung bewerten* und *Feedback erteilen* aufgeteilt. Als erstes muss das Zielsystem erstellt werden, das heißt der Menüpunkt *Administration – Aufgabe anlegen* muss gewählt werden. Hier wird die graphische Darstellung des Diagramms geladen, da die Einbindung des UML-Editors unterlassen wurde. Nach betätigen des Buttons *Alternativen bewerten* entspricht der Prototyp der in Abbildung 56 gezeigten Darstellung.

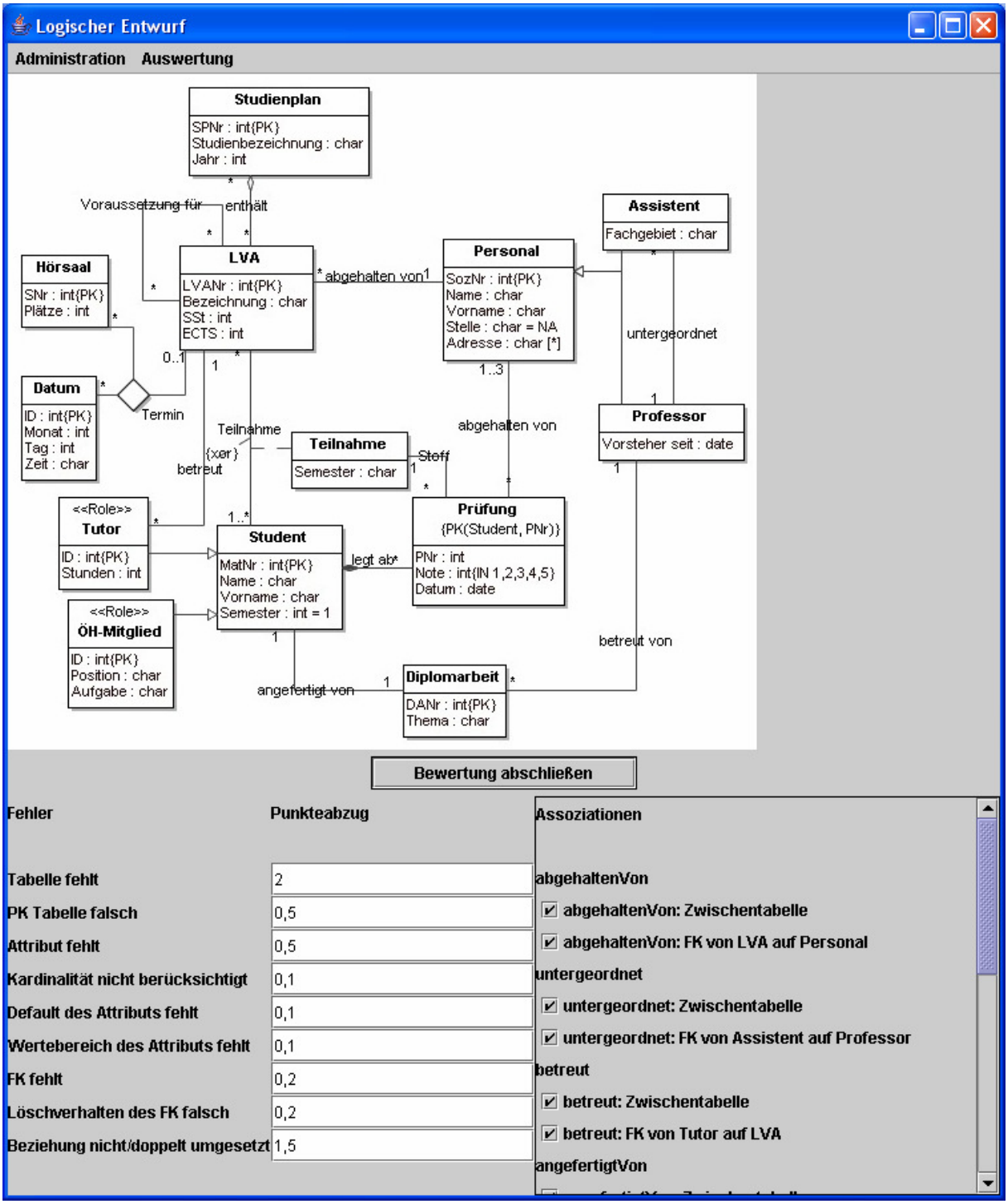


Abbildung 56: Auswählen der Alternativen

In dem hier gezeigten Frame können die verschiedenen Alternativen ausgewählt werden. Die hier vorgenommenen Einschränkungen werden allerdings in dem bisherigen Prototyp nicht übernommen. Die vorgenommene Auswahl würde zu einer Beschränkung auf die hier markierten Alternativen führen. Da in der Auswertung jedoch nicht zwischen nicht möglichen und nicht erlaubten Umsetzungsvarianten

unterschieden wird, würde die tatsächliche Einschränkung der Alternativen nur eine einfache Änderung bei der Erstellung des Zielsystems erfordern. Hier dürften nicht markierte Alternativen nicht generiert werden.

Außerdem können die Punkteabzüge für verschiedene Fehlertypen festgelegt werden. Diese werden bei betätigen des Buttons *Bewertung abschließen* für die Auswertung der Lösung zwischengespeichert. Außerdem erfolgt die Erstellung des Zielsystems, dieses wird ebenfalls gespeichert. Die Erstellung der Aufgabe ist somit abgeschlossen.

Für die Abgabe der Aufgabe muss als erstes der Menüpunkt *Auswertung – Lösung analysieren* ausgewählt werden. Das nun erscheinende Frame wird in Abbildung 57 gezeigt. Die Graphik der zu lösenden Aufgabe wird angezeigt, außerdem werden die zu berücksichtigenden Namenskonventionen vorgegeben. Im unteren Bereich des Frames wird ein Bereich zur Eingabe der SQL-DDL Statements zur Verfügung gestellt. Hier ist die Lösung bezüglich der Aufgabe einzugeben.

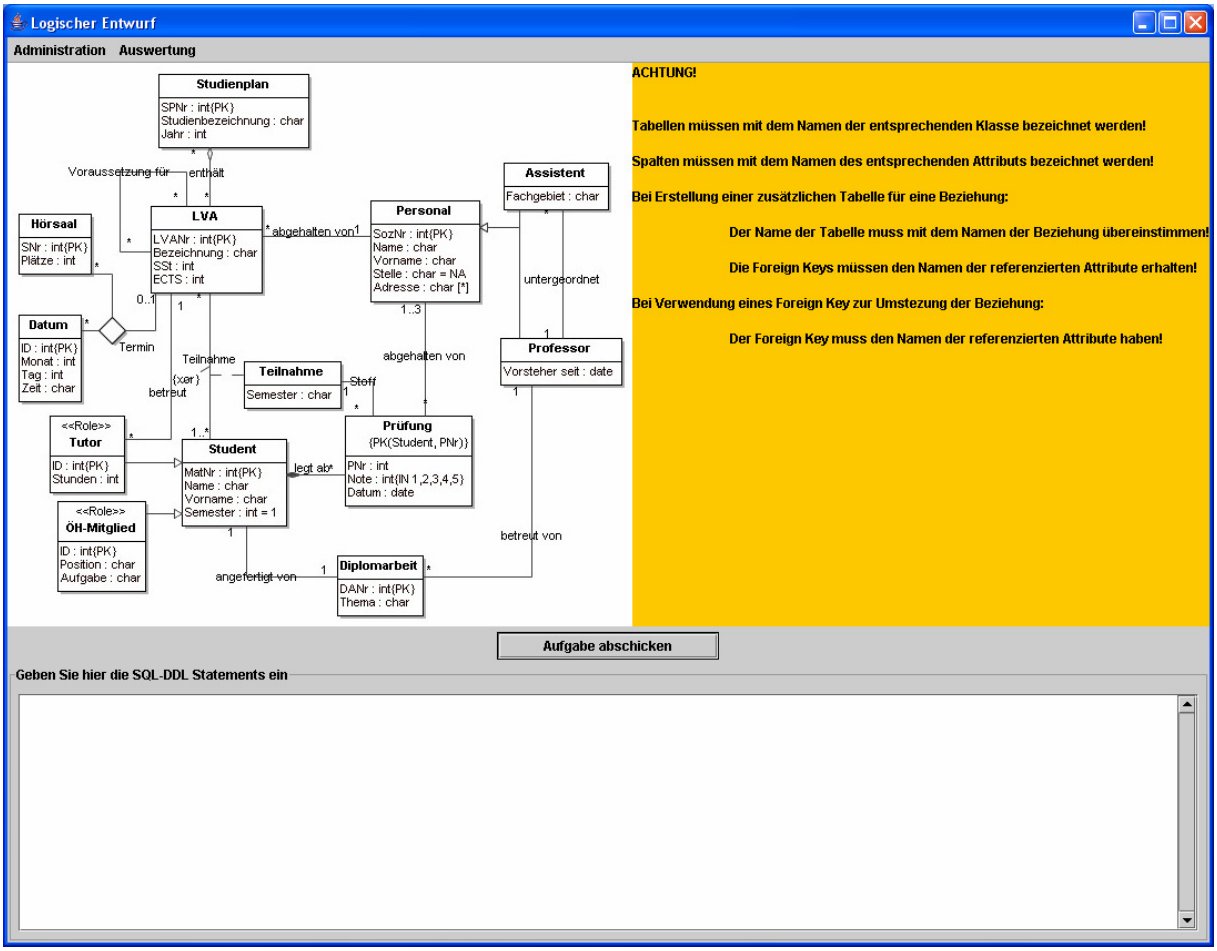


Abbildung 57: Erstellen der Lösung

Nach erfolgter Eingabe der Lösung muss der Button *Aufgabe abschicken* betätigt werden. Nun wertet das System die Lösung aus und speichert die Fehler und Übereinstimmungen in den entsprechenden Dokumenten. Unter dem Menüpunkt Auswertung werden Funktionen zur Anzeige der Punktevergabe (*Lösung bewerten*) und des Feedbacks (*Feedback erteilen*) angeboten. Diese rufen die entsprechenden Dokumente auf und geben den Inhalt wider.

Im derzeitigen auf der beiliegenden CD enthaltenen Prototyp ist das hier vorgestellte Beispiel fix integriert. Informationen bezüglich des Starts des Prototyps und dessen Verwendung werden in der auf der CD enthaltenen Datei „Readme“ zur Verfügung gestellt.

Soll ein anderes als das hier vorgestellte Beispiel angewendet werden, müssen die entsprechenden Dateien im Unterverzeichnis Files des Verzeichnis „Expertenmodul logischer Entwurf“ an Stelle der bestehenden Dateien eingefügt werden. Eine

graphische Darstellung der neuen Aufgabe muss als JPG unter dem Titel „UML“ gespeichert werden, das der Aufgabe entsprechende XMI-Dokument wiederum unter dem Titel „XMI“.

6. Zusammenfassung und Ausblick

Im Zuge dieser Arbeit wurde ein Modul konzipiert und ein Prototyp erstellt welcher die Auswertung von Aufgaben zum logischen Entwurf unterstützt. Das System überprüft die vom Studenten entworfene Lösung, welche als DDL bzw. Text übermittelt wurde, in Relation zu dem bei der Erstellung der Aufgabe definierten Zielsystem. Dieses wird durch das Modul automatisch aus dem in Magic Draw erstellten UML-Diagramm bzw. aus dem zu Grunde liegenden XMI-Dokument (EMF UML (v2.x) XMI) generiert, erlaubt aber das Eingreifen des Lehrenden bei der Punktevergabe. Durch die Konzeption wird aber auch die Verwendung jedes anderen Tools, welches ein demselben Standard folgendes Dokument erzeugt, ermöglicht. Unter Verwendung des generierten Zielsystems wird die Auswertung der Lösung gestartet. Diese kann in einem eigenen Textfeld in das System eingegeben werden. Aus diesem werden die einzelnen Elemente anhand der enthaltenen Schlüsselwörter und unter Annahme eines korrekten Aufbaus des Dokuments ausgelesen. In einem ersten Schritt erfolgt lediglich die Auswertung der übergebenen Lösung, diese wird in einer internen Dokumentation gespeichert. Obwohl die Bewertung und das erstellte Feedback ebenfalls während der Auswertung erstellt werden, werden diese erst nach Anfrage übermittelt.

Das Ziel der vorliegenden Arbeit lag darin, ein neues Modul für eTutor, das E-Learning System des Instituts für Data & Knowledge Engineering, zu konzipieren. Das Modul muss die Abgabe von Studentenlösungen unterstützen was durch die integrierte Texteingabe ermöglicht wird. Die Hauptaufgabe liegt jedoch in der automatischen Auswertung der Lösung und der dadurch entstehenden Entlastung der Tutoren und Assistenten. Dies wurde durch das erstellte Konzept bewerkstelligt. Allerdings ist zu beachten, dass das Konzept unter gewissen Annahmen erstellt wurde, dies betrifft in erster Linie, die durch das Modul vorgeschriebenen Namenskonventionen. Ohne diese würde die automatische Auswertung nicht möglich sein. Bisher werden jedoch nicht alle möglichen UML-Elemente unterstützt, die Auswertung von Assertions ist ein komplexes Gebiet und wurde daher in der aktuellen Version des Moduls außer Acht gelassen. Die Erstellung des Feedbacks wird durch ein eigens erstelltes Dokument bewerkstelligt. Hier werden einzelne Fehler an den Studenten übermittelt. Zusammenhänge der Fehler im Sinne von

Patterns werden allerdings nicht unterstützt. Die Beurteilung erfolgt ebenfalls zusammen mit der Dokumentation der Auswertung. Es wird bei der Erstellung der Aufgabe eine Möglichkeit geboten die generellen Fehlerarten zu bewerten. Die Alternativen der Beziehungsumsetzung können zwar gewählt werden, allerdings ist es nicht möglich gesondert Punkte für eine bestimmte Variante der Umsetzung zu vergeben. Derzeit fehlt jedoch die Verknüpfung der Auswahl mit dem Zielsystem, da es sich dabei um eine reine Datenübertragung handelt. Die Logik der Erstellung des Zielsystems wird dadurch nicht beeinflusst.

Für die weitere Entwicklung des Moduls bestehen daher einige Ansatzpunkte. Da es sich bei dem bisherigen System um einen Prototyp handelt ist eine Weiterentwicklung der Benutzeroberfläche in Betracht zu ziehen. Dies beinhaltet eine Anpassung an das eTutor System. Neben einer Erweiterung der graphischen Oberfläche, können auch die hier fehlenden Elemente wie Assertions ergänzt werden. Diese erfordern eine exakte Betrachtung der verschiedenen Umsetzungsalternativen. Neben diesen einfachen Erweiterungen könnten sowohl bei der Auswertung der Lösung als auch bei der Feedbackerteilung Patterns berücksichtigt werden. Das heißt es besteht die Möglichkeit das System an die Erkennung von Zusammenhängen zu adaptieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ANSI	American National Standards Institute
CAI	Computer Assisted Instruction
CAL	Computer Assisted Learning
CBL	Computer Based Learning
CD	Compact Disc
DBMS	Database Management System
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
DOC	Document, Textdatei
DTD	Dokumenttyp-Definition
E-Learning	Electronic Learning
EDV	Elektronische Datenverarbeitung
EPS	Encapsulated Postscript-Datei
FAQ	Frequently Asked Questions
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
ID	Identifier
ITS	Intelligente Tutorielle Systeme
JPG/JPEG	Joint Photographic Expert Group, File Interchange Format
OCL	Object Constraint Language
OMG	Object Management Group
PDF	Portable Document Format
PK	Primary Key
PS	Postskript
RTF	Rich Text Format
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TIFF	Tagged Image File Format
UML	Unified Modeling Language
WMF	Windows Metafile

XLS	Microsoft Excel Format
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language Transformations

Literaturverzeichnis

Ault03 M. Ault; Oracle – Datenbanken: Administration und Management; mitp-Verlag; Bonn; 2003.

Berg97 S. Bergamaschi, A. Garuti, C. Sartori, A. Venuta; Object Wrapper: an Object-Oriented Interface for Relational Databases; Proceedings of the 23rd EUROMICRO Conference '97 New Frontiers of Information Technology; 1997.

Bode90 F. Bodendorf; Computer in der fachlichen und universitären Ausbildung; Oldenbourg; München, Wien; 1990.

Das-07 T. Das, V. Iyer, E. Hanes Perry, B. Wright, T. Pfaeffle; Oracle Database JDBC Developer's Guide and Reference, 11g Release 1 (11.1); Oracle; September 2007.

Date02 Übungen Datenmodellierung WS 2002/03; Übungsangabe Übung P3.

Davi89 A.A. David, O. Thiery, M. Crehange; Intelligent Hypermedia in Education; in: H. Maurer (Ed.); Computer Assisted Learning – 2nd international Conference, ICCAL'89, Proceedings; Springer; Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio; 1989.

Ditt02 U. Dittler; E-Learning: Erfolgsfaktoren und Einsatzkonzepte mit interaktiven Medien; Oldenbourg; München, Wien; 2002.

Eich06 C. Eichinger, M Karlinger, G. Nitsche; eTutor – Modularchitektur; Version 2; JKU Linz, Institut für Wirtschaftsinformatik – Data & Knowledge Engineering; 2006.

Epst89 R.G. Epstein, R.M. Aiken; The Information Resource Model; in: H. Maurer (Ed.); Computer Assisted Learning – 2nd international Conference, ICCAL'89, Proceedings; Springer; Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio; 1989.

Flat90 R.G. Flatscher; SQL Seminar: Design relationaler Datenbanken; IWT Verlag; Vaterstetten, 1990.

Geis05 F. Geisler; Datenbanken: Grundlagen und Design; mitp-Verlag; Bonn; 2005.

Götz91 K. Götz, P.Häfner; Computerunterstütztes Lernen in der Aus- und Weiterbildung; Deutscher Studien-Verlag; 2. Auflage; Weinheim; 1991.

Grev89 S.H. Greve; A Real-time Coaching Environment for Triangle Congruence Proofs; in: H. Maurer (Ed.); Computer Assisted Learning – 2nd international Conference, ICCAL'89, Proceedings; Springer; Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio; 1989.

Grub04 G. Gruber; Objektrelationaler Datenbankentwurf mit UML und SQL Forward-Engineering in Theorie und Praxis; JKU Linz, Institut für Anwendungsorientierte Wissensverarbeitung; 2004.

Hein04 L.J. Heinrich, A. Heinzl, F. Roithmayr; Wirtschaftsinformatik-Lexikon; 7. Auflage; Oldenbourg Verlag; München, Wien; 2004;

Hitz05 M. Hitz, G. Kappel, E. Kapsammer, W. Retschitzegger; UML@Work – Objektorientierte Modellierung mit UML2; dpunkt.verlag GmbH; 3. Auflage; Heidelberg; 2005.

Hofe05 A. Hofer; E-Exercises in Data & Knowledge Engineering – Konzeption und Entwicklung eines intelligenten tutoriellen Systems; JKU Linz, Institut für Wirtschaftsinformatik – Data & Knowledge Engineering; 2005.

Jack89 G.A. Jackson; Entwurf relationaler Datenbanken mit dBASE-Anwendungsbeispielen; Carl Hanser Verlag; München, Wien; 1989.

Jaro02 H. Jarosch; Datenbankentwurf: Eine beispielorientierte Einführung für Studenten und Praktiker; Vieweg & Sohn Verlagsgesellschaft mbH; Braunschweig/Wiesbaden; 2002.

Jeck04 M. Jeckle; 100 UML Tools; Online im Internet: http://www.jeckle.de/umltools.htm; Stand: 2004.

Kemp04 A. Kemper, A. Eickler; Datanbanksysteme: Eine Einführung; Oldenbourg; 5. Auflage; München, Wien; 2004.

Klei97 P. Kleinschmidt, C. Rank; Relationale Datenbanksysteme: Eine praktische Einführung; Springer; Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio; 1997.

Lust92 M. Lusti; Intelligente tutorielle Systeme: Einführung in wissensbasierte Lernsysteme; Oldenbourg; München, Wien; 1992.

Meie01 A. Meier; Relationale Datenbanken: Leitfaden für die Praxis; Springer; 4. Auflage; Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio; 2001.

Mok-01 W. Y. Mok, D. P.Paper; On Transformations from UML Models to Object-Relational Databases; Proceedings of the 34th Hawaii International Conference on System Sciences; 2001.

Moos97 A. Moos, G. Daues; Datenbank-Engineering: Analyse, Entwurf und Implementierung relationaler Datenbanken mit SQL; Vieweg & Sohn Verlagsgesellschaft mgH; 2. Auflage; Braunschweig/Wiesbaden; 1997.

OCL-06 Object Management Group; Object Constraint Language; Version 2.0; Online im Internet: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML; Stand: Mai 2006; Download am: 5. November 2007.

Oest01 B. Oestereich; Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language; Oldenbourg; 5. Auflage; München, Wien; 2001.

Rumb91 J. Rumbaugh, M. Blaha, W. Premerlani, F, Eddy, W. Lorensen; Object-Oriented Modeling and Design; Prentice Hall; Englewood Cliffs, New Jersey; 1991.

Schä86 G. Schäfer; Entwurf logischer Datenstrukturen für konzeptionelle Schemata von Datenbanken; Verlag Josef Eul; Bergisch Gladbach, Köln; 1986.

Schw93 W. Schwendenwein; Theorie des Unterrichtens und Prüfens; WUV-Universitätsverlag; 5. Auflage; Wien; 1993.

Sura02 P. Suraweera, A. Mitrovic; KERMIT: A Constraint-Based Tutor for Database Modeling; in: S.A. Cerri, G. Gouarderes, F. Paraguacu (Ed.); ITS2002; Springer-Verlag; Berlin, Heidelberg; 2002.

UML-07 Object Management Group; Unified Modeling Language: Superstructure; Version 2.1.1; Online im Internet: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML; Stand: Februar 2007; Download am: 5. November 2007.

Wiep06 C. Wiepcke; Computergestützte Lernkonzepte und deren Evaluation in der Weiterbildung – Blended Learning zur Förderung von Gender Mainstreaming; Kovac; Hamburg; 2006.

Wool92 B. Woolf; Hypermedia in Education and Training; in: D. Kopec, R.B. Thompson; Artificial Intelligence and Intelligent Tutoring Systems – Knowledge-Based Systems for Learning and Teaching; Ellis Horwood Limited; Chichester; 1992.

XMI-03 Object Management Group; XML Metadata Interchange (XMI) Specification; Version 2.0; Online im Internet: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML; Stand: Mai 2003; Download am: 9. Dezember 2007.

Anhang A: Elemente des XMI-Dokuments

Unter Verwendung von Auszügen aus diesem Dokument wird nun die Abbildung der einzelnen Diagrammelemente erarbeitet. Diese wird im Folgenden für jeden Elementtyp betrachtet.

Darstellung von Klassen

Klassen werden durch folgendes Konstrukt eingeleitet:

```
<packagedElement xmi:type="uml:Class" xmi:id="_14_0_7e10253_1210081391323_319176_388"
                 name="Personal">
```

Der erste Teil des Ausdrucks definiert den Elementtyp, die xmi:id identifiziert das Element innerhalb des Diagramms. Diese ID besteht aus einer lückenlosen Kombination aus Ziffern, Buchstaben und Underscores. Abschließend wird der Name der Klasse angegeben. Ziel der ersten Aufarbeitung ist es sämtliche IDs durch die entsprechende Bezeichnung zu ersetzen. Ein weiterer spezifischer Eintrag innerhalb dieses Starttags bezüglich einer Klasse bezieht sich auf die Eigenschaft {abstract}. Diese wird durch das Attribut isAbstract=“true“ welches nach dem Namen der Klasse enthalten ist definiert. Vor dem Endtag der Klasse können sämtliche enthaltenen Elemente eingefügt werden. Attribute werden wie folgt dargestellt:

```
<ownedAttribute xmi:id="_14_0_7e10253_1211275947359_915329_2026" name="SozNr" visibility="private">
  <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/JavaPrimitiveTypes.library.uml#int"/>
</ownedAttribute>
```

Der einleitende Teil des Tags definiert wiederum den Elementtyp, die ID folgt dem Schema der Klassen-IDs. Nach der Angabe des Namens erfolgt die Definition der Sichtbarkeit des Attributes, diese Information ist hier jedoch irrelevant. In einem darunter liegenden Tag wird der Datentyp des Attributes definiert, dieser enthält den gesamten Pfad des gewählten Typs, allerdings ist nur der Teil nach dem Zeichen # von Interesse. Die Datentypen entsprechen generell dem im Diagramm gesetzten Typ, eine Ausnahme hierfür stellt der Datentyp Date dar, dieser wird im XMI-

Dokument als String dargestellt. Zur Festlegung eines Initialwertes wird ein weiterer Tag nach folgendem Muster in ownedAttribute eingefügt:

```
<defaultValue xmi:type="uml:LiteralString" xmi:id="_14_0_7e10253_1211274496609_367815_2015"
    name="" value="NA"/>
```

Innerhalb dieses Tags ist im Grunde genommen nur der Value von Bedeutung. Weiters ist zu beachten, dass Attribute auch verschiedene Kardinalitäten aufweisen können. Zu deren Darstellung wird im XMI-Dokument angegeben welche Maximal- bzw. Minimalzahl an Instanzen zu einem Attribut bestehen dürfen. Dies erfolgt durch eine gesonderte Angabe dieser Werte innerhalb der Attributdefinition. Der Maximalwert wird durch upperValue eingeleitet, der Minimalwert durch lowerValue:

```
<upperValue xmi:type="uml:LiteralUnlimitedNatural"
    xmi:id="_14_0_7e10253_1211274549390_565955_2018" name="" value="*"/>
<lowerValue xmi:type="uml:LiteralUnlimitedNatural"
    xmi:id="_14_0_7e10253_1211274549390_571431_2017" name="" value="*"/>
```

Hier sind wiederum nicht sämtliche Informationen notwendig, nur die Einleitung und der Value haben eine Bedeutung für die Auswertung. Zu beachten ist, dass die gesamten Elemente keiner bestimmten Ordnung unterliegen, das heißt sie können in beliebiger Reihenfolge enthalten sein.

Darstellung von Assoziationen

Die Sammlung der Informationen bezüglich der einzelnen Assoziationen ist schwieriger als die Informationen zu den Klassen aufzudecken. Assoziationen werden zum Teil als eigenes Element dargestellt, zum Teil sind die zugehörigen Informationen allerdings in den beteiligten Klassen enthalten. Das eigene Element wird wie folgt dargestellt:

```
<packagedElement xmi:type="uml:Association" xmi:id="_14_0_7e10253_1210081533455_684262_582"
    name="untergeordnet" memberEnd="_14_0_7e10253_1210081533455_686456_583
    _14_0_7e10253_1210081533455_856345_584"/>
```

Diese Darstellung enthält wiederum ein einleitendes Element sowie eine Objekt-ID und einen Namen. Zusätzlich werden die an der Assoziation beteiligten Klassen

durch memberEnd referenziert. Diese Identifizierung erfolgt durch die Objekt-ID des Assoziationsendes. Dieses wird innerhalb der entsprechenden Klasse angegeben:

```
<ownedAttribute xmi:id="_14_0_7e10253_1210081533455_856345_584" name="" visibility="private"
        type="_14_0_7e10253_1210081446732_588784_432"
        association="_14_0_7e10253_1210081533455_684262_582">
    <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_14_0_7e10253_1210081699055_22526_707"
        name="" value="1"/>
    <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_14_0_7e10253_1210081699055_997953_706" name=""
        value="1"/>
  </ownedAttribute>
```

Wie zu sehen ist entspricht die ID dieses Assoziationsendes einer der unter memberEnd angegebenen Nummern, auch die ID der Assoziation wird hier angegeben. Wie bei den Attributen werden auch hier die Ober- und Untergrenzen der Multiplizitäten angegeben. Die in einer Klasse enthaltenen Werte geben die Multiplizität der anderen an der Beziehung beteiligten Klasse an.

Einen besonderen Fall von Assoziationen stellen n-äre Assoziationen dar. Hier wird die Raute als eigene Klasse dargestellt, diese erhält den Namen der Beziehung. Diese Klasse beinhaltet nur Beziehungsenden welche auf die dazugehörigen Klassen verweisen. In den beteiligten Klassen sind verweise auf die die Raute repräsentierende Klasse enthalten, hier sind jedoch keine Multiplizitäten definiert. Davon abgesehen erfolgt die Umsetzung analog zu binären Assoziationen. Rekursive Assoziationen werden genau wie binäre Assoziationen umgesetzt, außer dass das Assoziationsende auf die eigene Klasse verweist.

In Bezug auf n-äre Assoziationen ist für den Algorithmus zum Auslesen der Assoziationen zu beachten, dass die Namensgebung bei der Raute erfolgt. Die Assoziationslinien zwischen der Raute und den beteiligten Klassen dürfen nicht bezeichnet werden. Da der Name der Raute im Diagramm nicht angezeigt wird, muss die Bezeichnung in einem eigenen Textfeld angegeben werden.

Darstellung von Assoziationsklassen

Assoziationsklassen werden im Grunde genommen wie Klassen dargestellt, nur dass das einleitende Tag anders aufgebaut ist als bei normalen Klassen:

```
<packagedElement xmi:type="uml:AssociationClass"
        xmi:id="_14_0_7e10253_1210085923452_772999_2211" name="Teilnahme"
        memberEnd="_14_0_7e10253_1210085923452_415094_2213
        _14_0_7e10253_1210085923452_693644_2212">
```

Im Anschluss an die Angabe des Namens werden die IDs der an der Assoziation beteiligten Klassen referenziert. Innerhalb der Assoziationsklasse können dieselben Eintragungen gemacht werden wie in normalen Klassen. Die Eintragungen in den Klassen bezüglich der Assoziationsenden entsprechen denjenigen bei binären Assoziationen.

Darstellung von Aggregationen und Kompositionen

Die Darstellung der Aggregationsenden innerhalb der beteiligten Klassen erfolgt in Anlehnung an die Darstellung von Assoziationen. In der Teile-Klasse stimmt die Darstellung mit derer von normalen Assoziationsenden überein. In der Aggregatklasse wird jedoch eine zusätzliche Identifikation eingefügt:

```
<ownedAttribute xmi:id="_14_0_7e10253_1210083432678_59767_1875" name="" visibility="private"
      type="_14_0_7e10253_1210081715102_847297_709" aggregation="shared"
      association="_14_0_7e10253_1210083432678_283890_1874">
  <upperValue xmi:type="uml:LiteralUnlimitedNatural"
      xmi:id="_14_0_7e10253_1210083451335_457241_1925" name="" value="*"/>
  <lowerValue xmi:type="uml:LiteralUnlimitedNatural"
      xmi:id="_14_0_7e10253_1210083451335_641543_1924" name="" value="*"/>
</ownedAttribute>
```

Die Bezeichnung aggregation=”shared” weist auf die Aggregation hin. Die Definition der Beziehung selbst erfolgt hier ebenfalls als Assoziation. Gleiches gilt für Kompositionen, die Unterscheidung gegenüber Aggregationen erfolgt durch die Verwendung von composite anstelle von shared.

Darstellung von Generalisierungen

Generalisierungen werden durch zusätzliche Tags in den Subklassen dargestellt. Diese weisen im Eintrag general eine Referenz auf die Objekt-ID der Superklasse auf. Dieser Tag sieht folgendermaßen aus:

```
<generalization xmi:id="_14_0_7e10253_1210081636240_697890_643" isSubstitutable="false"
    general="_14_0_7e10253_1210081391323_319176_388"/>
```

Darstellung von Zusicherungen

Zusicherungen zu Attributen, Klassen oder Assoziationen werden in einem gänzlich eigenen Element definiert. Die Regel bezüglich der Primary Keys wird folgendermaßen dargestellt:

```
<ownedRule xmi:id="_14_0_7e10253_1210153744656_496236_1108" name="PK">
  <specification xmi:type="uml:OpaqueExpression" xmi:id="_14_0_7e10253_1210153801359_291712_1109"
      name="">
    <body>PK(Student, PNr)</body>
  </specification>
</ownedRule>
```

Zusicherungen werden durch ownedRule eingeleitet, sie verfügen ebenfalls über eine eigene Objekt-ID. Am Ende des einleitenden Tags wird der Name der Zusicherung angegeben. In weiterer Folge wird ein die Spezifikation beginnendes Tag angegeben, dieses ist für die Auswertung jedoch nicht relevant. Innerhalb des Tags Body wird die eigentliche Zusicherung angegeben, in diesem Fall entspricht dies der UML-Darstellung {PK(Student, PNr)}. Wie bereits erwähnt kann die Spezifikation der Zusicherung aber auch in einem eigenen Element definiert werden, dieses ist im Grunde genommen gleich aufgebaut. Der Unterschied besteht darin, dass die Objekt-IDs der Attribute welche diese Zusicherung erfüllen müssen nach dem Namen der Zusicherung angegeben werden. Wenn es sich um mehrere IDs handelt werden diese durch ein Leerzeichen getrennt. Die Darstellung sieht wie folgt aus:

```
<ownedRule xmi:id="_14_0_7e10253_1211274684109_457071_2019" name="Primary Key"
      constrainedElement="_14_0_7e10253_1211275947359_915329_2026
      _14_0_7e10253_1210084220887_963335_1970 ">
  <specification xmi:type="uml:OpaqueExpression" xmi:id="_14_0_7e10253_1211275757468_868530_2024"
      name="">
    <body>PK</body>
  </specification>
</ownedRule>
```

In dem gewählten Werkzeug wird zwar eine eigene XOR-Zusicherung angeboten, allerdings wird diese wiederum nicht in dem XMI-Dokument integriert, daher ist es notwendig eine eigene Zusicherung zu erstellen. Der Algorithmus setzt hier die Bezeichnung „XOR Klasse“ voraus wobei Klasse der Name der Klasse ist von welcher die beiden betroffenen Assoziationen ausgehen. Die Spezifikation muss „XOR“ lauten, diese wird auch im Diagramm angezeigt.

Darstellung von Rollen

Zu den Klassen, welche Rollen repräsentieren können zwar Stereotype angewendet werden, allerdings werden diese im XMI-Dokument nicht zugeordnet. Daher muss diese Verknüpfung durch die Anwendung eines GeneralizationSet gewährleistet werden. Wegen des erstellten Algorithmus muss dieses mit dem Namen „Role“ bezeichnet werden. Diese wird zusammen mit der Definition des Stereotyps angegeben. In dem hier angeführten Beispiel fehlen die für die Implementierung irrelevanten Teile zur Definition des Stereotyps.

```
<packagedElement xmi:type="uml:Package" xmi:id="_14_0_7e10253_1210082063178_607993_1149"
        name="eigeneStereotype">
    <packagedElement xmi:type="uml:Stereotype" xmi:id="_14_0_7e10253_1210082079101_532122_1150"
        name="Role">
    …
    <packagedElement xmi:type="uml:GeneralizationSet" xmi:id="_14_0_7e10253_1211444970046_35949_762"
        name="Role" generalization="_14_0_7e10253_1210082018083_338581_1108"/>
  </packagedElement>
```

Das hier angeführte Package wird benötigt um die Zusammengehörigkeit von Stereotyp und GeneralizationSet zu erkennen. Dieses führt in der Definition der Klassen zur Erweiterung generalizationSet="ID" nach dem XML-Attribut general. Somit kann auch die Verknüpfung des Stereotyps mit der die Rolle repräsentierenden Klasse erfolgen.

Anhang B: Data-Dictionary der Speicherstrukturen

Klassen

id	die formatierte XMI-ID um die einzelnen Elemente bei der Zusammenführung zu identifizieren
name	den Namen der Klasse
stereotyp	falls in weiterer Folge neue Stereotype integriert werden, dieses Element bleibt in der bisherigen Zusammenstellung leer, da Rollen bei den Generalisierungen gespeichert werden
abstr	das Attribut liefert einen Wahrheitswert, der angibt ob die Klasse abstract ist oder nicht
attList	die Attribute der Klasse sind in einer Liste gespeichert, deren Elemente von Typ Attribut sind

Attribute

id	XMI-ID wie bei Klassen
name	gibt den Namen des Attributs an
typ	gibt an von welchem Datentyp das Attribut ist, der Datentyp Datum wird im XMI-Dokument in String umgewandelt
def	dieses Element enthält den Initialwert des Attributs falls vorhanden
min	dieses Element gibt die Mindestanzahl der in der Klasse enthaltenen Attribute dieses Typs an, für Arrays relevant
max	dieses Element gibt die entsprechende Maximalzahl an
assertion	in diesem Element werden alle Assertions gespeichert welche nicht den Primary Key anzeigen
pk	dieses Element enthält einen Wahrheitswert, der angibt ob das Attribut Teil des Primärschlüssels der Klasse ist

Assoziationen

id	XMI-ID wie bei Klassen
name	dieses Element enthält die Bezeichnung der Beziehung

anz	gibt an, wie viele Klassen an der Beziehung beteiligt sind
enden	dies ist eine Liste die Elemente vom Typ AssEnde (Assoziationsende) enthält

Assoziationsenden

id	XMI-ID wie bei Klassen, gibt an zu welcher Klasse das Ende gehört
klasse	gibt den Namen der Klasse an, zu der das Ende gehört
min	gibt die Untergrenze der Kardinalität bezüglich dieser Klasse an
max	gibt die entsprechende Obergrenze an
komposition	enthält einen Wahrheitswert, der angibt ob das Ende das Ganze einer Komposition darstellt
aggregation	enthält einen Wahrheitswert, der angibt ob das Ende das Ganze einer Aggregation darstellt

Assoziationsklassen

id	XMI-ID wie bei Klassen
k	enthält ein Objekt vom Typ Klasse, welches die Attribute der Assoziationsklasse umfasst
ass	enthält ein Objekt vom Typ Assoziation, welches die Informationen bezüglich der Beziehung umfasst

Generalisierungen

id	XMI-ID wie bei Klassen, gibt die ID der Superklasse an
superklasse	gibt den Namen der Superklasse an
subklassen	eine Liste die Elemente vom Typ Subklasse enthält

Subklassen

id	XMI-ID wie bei Klassen, gibt die ID der Subklasse an
name	gibt den Namen der Subklasse an

rolle enthält einen Wahrheitswert, der angibt ob es sich um eine Rolle handelt (true) oder um eine echte Generalisierung

Zusicherungen

id XMI-ID wie bei Klassen, gibt die ID des Elements an zu dem die Zusicherung gehört, wenn mehrere Elemente dazu gehören werden diese in verschiedene Objekte aufgeteilt

text enthält die Spezifikation der Zusicherung, bei XOR-Zusicherungen den Namen