UML-Tutor

Konzeption und Entwicklung eines halbautomatisierten Systems zur Bewertung und Beurteilung von konzeptuellen Datenbankschemata im Rahmen des intelligenten tutoriellen Systems E-Tutor

Diplomarbeit

Zur Erlangung des akademischen Grades
„Magister der Sozial- und Wirtschaftswissenschaften"
(Mag.rer.soc.oec.)
im Diplomstudium Wirtschaftsinformatik

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

Betreuer: o.Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl
Mitbetreuender Assistent: Dr. Michael Karlinger

Verfasst von: Bernhard Koenings

Linz, Mai 2011

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, Mai 2011

B. Koenings

Danksagung

An dieser Stelle möchte ich mich zuerst bei meiner Familie bedanken. Meine Frau **Judith** und meine Tochter **Teresa** mussten viele Wochenenden auf Ehemann und Vater verzichten. Für das Verständnis dafür gebührt mein herzlichster Dank.
Mein Dank gilt auch meinem Betreuer **o. Univ.-Prof. DI Dr. Michael Schrefl**, der viel Verständnis für meine Berufstätigkeit und der damit verbundenen Zeitgebundenheit entgegenbrachte.
Natürlich danke ich auch meinem mit-betreuendem Assistenten **Mag. Christian Eichinger**, der mich während der doch recht langen Zeit hervorragend und auch zu nicht alltäglichen Zeiten betreut hat. Viele wertvolle Informationen und Anregungen sind durch die Gespräche mit ihm entstanden.
Herrn **Dr. Michael Karlinger** danke ich für die Übernahme der Betreuung von Mag. Eichinger nach dessen beruflicher Veränderung.
Naturgemäß kann man nicht allen, die mich in irgendeiner Weise bei der Erstellung dieser Arbeit unterstützt haben, danken. Deswegen danke ich auch jenen die hier keine namentliche Erwähnung finden, mich aber trotzdem unterstützt haben.

D A N K E

Kurzfassung

Am Institut für Wirtschaftsinformatik – Data and Knowledge Engineering der Johannes Kepler Universität Linz (DKE), wird die Lehre durch ein webbasiertes elektronisches Tutoring[1] System (E-Tutor) unterstützt. Ein Tutoring System bietet im Unterschied zu E-Learning Systemen auch qualifiziertes Feedback an. Dadurch wird der Lernfortschritt positiv beeinflusst. Derzeit wird nur ein Teil der am DKE Institut unterrichteten Themengebiete durch das E-Tutor System unterstützt.

Im Rahmen dieser Arbeit soll das bestehende E-Tutor System so erweitert werden dass das Themengebiet „konzeptueller Datenbankentwurf mit der UML[2]" der LVA „Datenmodellierung" unterstützt wird. Ausgehend vom derzeitigen Ablauf der LVA Datenmodellierung, soll ein Übungsblatt mit der textuellen Beschreibung der Aufgabe elektronisch zur Verfügung gestellt werden. Der Student erstellt aufgrund dieser Beschreibung ein UML Modell (Studentenmodell) und überträgt es anschließend an das E-Tutor System. Vom E-Tutor System erfolgt ein Vergleich des Studentenmodells mit einem Mustermodell, welches vom Lehrenden zuvor erstellt worden ist. Dieses Mustermodell bietet die Möglichkeit, für einen Sachverhalt mehrere Modellierungsvarianten abzuspeichern. Zusätzliche Informationen wie Punkteabzüge bei Fehlern, oder spezifisches Feedback pro Variante können ebenfalls im Mustermodell erfasst werden. Feedback für häufig auftretende, allgemeine Fehler wird vom System selbst generiert.

Als Basis für die elektronische Repräsentation eines UML Modells wird das standardisierte XML Metadata Interchange (XMI)[3] Format Version 1.3 gewählt. Aus diesem Format wird ein spezielles XML

---

[1] Siehe http://etutor.dke.uni-linz.ac.at/moodle/

[2] UML – http://www.omg.org/gettingstarted/what_is_uml.htm

[3] Siehe http://www.omg.org/spec/XMI/

Speicherformat für Muster- und Studentenmodelle generiert. Durch dieses spezielle XML Speicherformat können einerseits Zusatzinformationen für das Mustermodell gespeichert werden, und andererseits der Vergleich von Muster- und Studentenmodellen einfach realisiert werden. Beim Vergleich der Modelle werden Abweichungen eruiert, und die am wahrscheinlichsten vom Studenten gewählte Modellierungsvariante festgestellt. Ausgehend von der festgestellten Modellierungsvariante wird abschließend Feedback generiert und zusammen mit der Beurteilung an den Studenten zurückgegeben.

Abstract

At the "Institut für Wirtschaftsinformatik - Data & Knowledge Engineering" of the Johannes Kepler University Linz (DKE) teaching is supported by a web based electronic tutoring system (E-Tutor). A tutoring system offers also qualified feedback in difference to E-Learning systems. Feedback influences the learning progress in a positive way. Currently only parts of the topic that is taught at DKE are supported by the E-Tutor system.

This thesis describes the extension of the existing system with the subject "conceptual database modeling with UML" from the course "Datenmodellierung". Based on the current course situation a textual description of the lesson is presented electronically. The student has to create a UML Model (student model) in response to the given description and transfers it to the E-Tutor system. The E-Tutor system compares the student model to a master model that was created previously by the lecturer. The master model can store different modeling variants of an issue. Additional information, like penalties on errors or specific feedback at variant level, can also be stored within the master model. Feedback for common errors is automatically generated by the system.

XML Metadata Interchange (XMI) format version 1.3 has been chosen as a basis for the electronic representation of the UML models. From this XMI format a more specific XML storage schema is generated. On the one hand additional information can be stored with the XML storage schema, and on the other hand comparison of student and master model can be done easy. During comparison of the models differences are recognized and the system finds out the modeling variant most likely taken by the student. Finally a feedback text is generated and given back to the student together with the grading based on the taken modeling variant.

Abbildungsverzeichnis

Inhaltsverzeichnis

1 Einleitung

Neben den Präsenzlehrveranstaltungen wird am Institut „Wirtschaftsinformatik – Data & Knowledge Engineering" der Johannes Kepler Universität[4] auch ein breites Spektrum an Lehrangeboten via E-Learning bereitgestellt. Die E-Learning Inhalte werden über das intelligente tutorielle System (ITS)[5] „E-Tutor" bereitgestellt. Dieses System wurde im Rahmen einer Diplomarbeit (Hofer, 2005) konzipiert und ein Prototyp entwickelt. Das E-Tutor System wurde inzwischen in das Kurssystem „Moodle"[6] integriert. Das E-Tutor System ist modular aufgebaut, so dass es erweitert werden kann. Abgegrenzte Stoffgebiete werden jeweils als Expertenmodul zur Verfügung gestellt. Expertenmodule verwalten Aufgaben zum Stoffgebiet und verfügen über Methoden zur Analyse, Bewertung und zum Feedback einer abgegebenen Lösung. Aufgrund der Vielfalt an unterschiedlichen Themen wurden nicht für alle Stoffgebiete Expertenmodule entwickelt.

Im Rahmen der LVA „Datenmodellierung" im Studiengang Wirtschaftsinformatik, wird auch der konzeptuelle Entwurf von Datenbank Schemata (konzeptuelle Modellierung) geübt. Die konzeptuelle Modellierung stellt einen wesentlichen Teil des Datenbank Entwurfs dar. Fehler bei der konzeptuellen Modellierung wirken sich auf alle weiteren Entwurfsschritte aus. Daher ist es sinnvoll, die Ausbildung in diesem Bereich zu intensivieren.

„Blended Learning"[7], also die Kombination von Präsenzlehrveranstaltungen und E-Learning soll auch für die Ausbildung im Bereich Datenbankentwurf genutzt werden. Damit können die allgemeinen

---

[4] Siehe http://www.dke.jku.at

[5] Zum Begriff ITS siehe (Lusti, 1992)

[6] Siehe http://moodle.org/

[7] Zum Begriff „Blended Learning" siehe (Villar Angulo & Alegre de la Rosa, 2008)

Vorteile eines ITS, wie Orts- und Zeitunabhängigkeit, beim Aufbau von Wissen über das Stoffgebiet genutzt werden.

Diese Arbeit befasst sich mit der Entwicklung eines E-Tutor Expertenmoduls für „Konzeptuellen Datenbankentwurf mit UML". Mithilfe des Expertenmoduls werden Übungsaufgaben für die Studenten zur Verfügung gestellt. Die Lernenden versuchen die Lösung zu erarbeiten. Während der Übungsphase können sie ihre Lösung an das Expertenmodul senden, und dieses überprüft die Lösung. Als Ergebnis bekommt der Lernende eine Rückmeldung in Form von qualifiziertem Feedback und einer Bewertung. Qualifiziert bedeutet in diesem Zusammenhang, dass die erkannten Fehler so kommentiert sind, dass sie dem Lernenden helfen in Zukunft Fehler gleicher Art zu vermeiden. Darüber hinaus bietet das Expertenmodul auch Unterstützung beim Erstellen der Aufgaben durch den Lehrenden.

1.1 Motivation

Die LVA „Datenmodellierung" besteht aus einer Vorlesung, in der die theoretischen Grundlagen gelehrt werden, sowie Übungen um das theoretische Wissen in der Praxis erproben zu können. Die Studenten bekommen von ihrem Übungsleiter eine Übungsaufgabe zugeteilt, die in Hausarbeit, in der Regel innerhalb einer Woche, gelöst werden muss. Ausgehend von einer verbalen Problembeschreibung sollen die Studenten ein konzeptuelles Datenmodell mit der UML erstellen.

Gerade für Anfänger ergeben sich während der Ausarbeitung viele Fragen. Für die Studenten stehen deshalb Tutoren zur Beantwortung von Fragen zur Verfügung. Diese stehen jedoch nur für eine kurze Zeitspanne (meist ca. 2 h / Woche) zur Verfügung. Darüber hinaus müssen sich die Studenten die zur Verfügung stehenden Tutoren und auch die Zeit teilen. Eine individuelle Betreuung ist aus diesem Grund nicht möglich.

Für berufstätige Studenten stellt dies eine zusätzliche Hürde dar, da die Tutoren Sprechstunden in der Regel tagsüber ansetzen.

1.2 Problemstellung

Deshalb soll das bestehende E-Tutor System des DKE[8] um ein Expertenmodul für den Bereich des konzeptuellen Datenbankentwurfs mit UML erweitert werden. Dazu bietet das E-Tutor System Schnittstellen an. Ein Expertenmodul bildet, laut Definition von Lusti (Lusti, 1992), das Wissen eines Experten zum jeweiligen Aufgabengebiet ab. Das Expertenmodul soll die Ausbildung unterstützen, indem es Aufgaben zur Verfügung stellt und Rückmeldungen über Fehler an die Studenten gibt. Es soll aber auch in der Lage sein Prüfungsaufgaben zu verwalten, und abgegebene Lösungen zu bewerten. Aufgaben zu diesem Expertenmodul werden in Form einer textuellen Beschreibung der Anwendungsdomäne gegeben. Aufgrund dieser Beschreibung erstellen die Studenten ein konzeptuelles Datenmodel mit der UML (Studentenmodell). Dieses Studentenmodell soll vom Expertenmodul analysiert werden. Dazu muss das Expertenmodul über das notwendige Expertenwissen verfügen. Expertenwissen kann in Form von UML-Modellen (Mustermodell), die jeweils zuvor gemeinsam mit dem Aufgabentext vom Lehrenden erstellt worden sind, gespeichert werden. Das UML-Model alleine ist jedoch für eine Analyse nicht ausreichend. Meist gibt es für die Modellierung eines Sachverhaltes mehrere Möglichkeiten (Varianten). Diese Varianten sollen im Studentenmodell erkannt, und entsprechend bewertet werden. Wählt der Student eine nicht optimale Modellierungsvariante, so sollen dafür Punkteabzüge vermerkt werden können. Zusätzlich sollen auch unterschiedliche Punkteabzüge bei Fehlern ermöglicht werden. Die Erstellung des Feedbacks soll möglichst automatisch durch das Ex-

[8] Siehe http://etutor.dke.uni-linz.ac.at/moodle/

pertenmodul erfolgen. Es soll jedoch auch die Möglichkeit geschaffen werden spezifisches Feedback bei Verwendung von Modellierungsvarianten zu geben. Die Benennung der Elemente des Studentenmodells muss so erfolgen, dass sie mit den Elementen des Mustermodells verglichen werden können. Um den Lernerfolg zu erhöhen sollen die Rückmeldungen an den Studenten in mehreren Detailierungsstufen erfolgen. Bei der geringsten Detailierungsstufe soll dabei nur der Hinweis auf einen Fehler gemeldet werden, um den Studenten selbst die Möglichkeit zu geben, den Fehler zu entdecken. Bei der höchsten Detailierungsstufe soll eine detaillierte Fehlerbeschreibung bzw. Anleitung zur Fehlerbehebung zurückgegeben werden.

1.3 Beispiel

Abbildung 1-1: Textuelle Aufgabenbeschreibung (Quelle: (DKE, 2008))

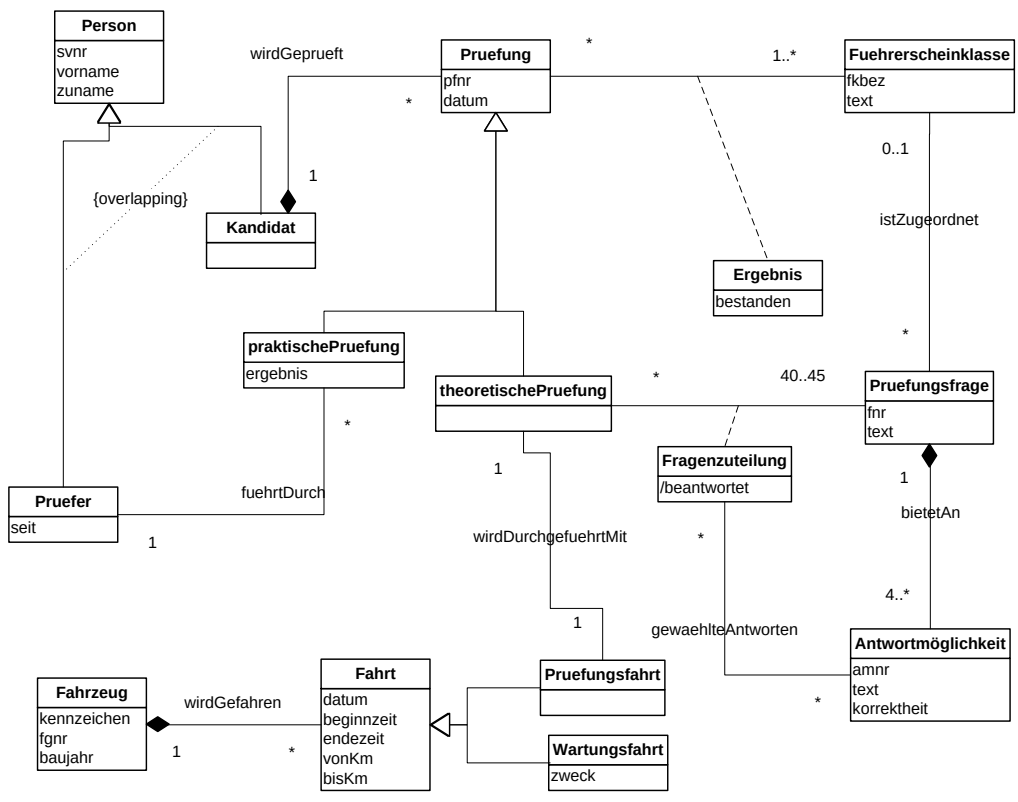
Aufgabe 1: Konzeptueller Datenbankentwurf mit UML (25 Punkte)

Erstellen Sie in UML ein konzeptuelles Datenbankschema für die Verwaltung von Führerscheinprüfungen, das folgenden Sachverhalt repräsentiert: Jeder Prüfungskandidat wird durch seine Sozialversicherungsnummer identifiziert und hat außerdem einen Vor- und Zunamen. Prüfungskandidaten legen Prüfungen ab. Jede Prüfung hat eine eindeutige Nummer sowie ein Prüfungsdatum. Prüfungen werden in theoretische und praktische Prüfungen unterteilt. Prüfungen werden zum Erwerb von mindestens einer Führerscheinklasse abgelegt. Jede Führerscheinklasse hat eine eindeutige Bezeichnung sowie einen beschreibenden Text. Bei jeder Prüfung wird festgehalten, für welche Führerscheinklassen die Prüfung bestanden wurde und für welche nicht. Theoretische Prüfungen umfassen zwischen 40 und 45 Prüfungsfragen. Jede Prüfungsfrage weist eine eindeutige Nummer auf und enthält einen Fragetext. Jede Frage ist maximal einer Führerscheinklasse zugeordnet. Bei jeder Frage gibt es eine Menge von mindestens 4 Antwortmöglichkeiten. Jede Antwortmöglichkeit ist exklusiv einer Frage zugeordnet und kann innerhalb der Prüfungsfrage (nicht aber global) mit Hilfe einer Nummer identifiziert werden. Bei jeder Antwortmöglichkeit ist der Text der Antwort gespeichert sowie vermerkt, ob die Antwortmöglichkeit richtig oder falsch ist. Legt ein Kandidat eine Prüfung ab, beantwortet er jede der zugeteilten Fragen, indem er die seiner Meinung nach richtigen Antwortmöglichkeiten ankreuzt. Es wird vermerkt, welche Antwortmöglichkeiten gewählt wurden; außerdem wird (als abgeleitetes Attribut) vermerkt, ob die Frage als korrekt beantwortet zu werten ist. Praktische Prüfungen werden von einem Prüfer vorgenommen. Von jedem Prüfer sind Sozialversicherungsnummer (eindeutig) sowie Vor- und Zuname bekannt. Außerdem ist bekannt, seit wann jemand als Prüfer tätig ist. Ein Prüfer kann selbst auch als Kandidat einer Prüfung gespeichert sein (da ja jeder Prüfer auch einmal den Führerschein machen musste).Jedes Fahrzeug wird eindeutig durch ein Kennzeichen (und alternativ durch seine Fahrgestellnummer) identifiziert und hat außerdem ein Baujahr.

Als Grundlage für die Erstellung eines konzeptuellen Modells werden vielfach Texte mit der Beschreibung eines Sachverhalts (der Anwendungsdomäne) verwendet. Abbildung 1-1 zeigt ein Beispiel einer Aufgabe für die Erstellung eines konzeptuellen Datenbankmodells. Dieses Beispiel bzw. Teile daraus werden in den einzelnen Kapiteln dieser Arbeit verwendet.

Die Lösung zur Aufgabenstellung (Abbildung 1-2) beinhaltet viele wesentliche Elemente eines UML-Klassenmodels. Neben Klassen finden sich auch Assoziationsklassen, Generalisierungen, sowie auch Aggregationen bzw. Kompositionen als spezielle Form von Assoziationen. Dieses UML-Modell wird als Musterlösung für die Aufgabe verwendet.

Abbildung 1-2: Konzeptuelles UML-Modell zur Textaufgabe (Quelle: (DKE, 2008))



1.4 Aufbau und Gliederung der Arbeit

Im Kapitel 2 werden zuerst die allgemeinen Lerntheorien und anschließend, Intelligente Tutorielle Systeme vorgestellt. Abschließend werden mit dem Entity-Relationship-Model (ER-Model) und der Unified Modeling Language (UML), die theoretischen Grundlagen des Themengebietes „konzeptueller Datenbankentwurf" beschrieben.

Im Kapitel 3 wird auf den Stand der Technik für ITS im Bereich konzeptuelle Datenmodellierung eingegangen. Zusätzlich werden Vergleichskriterien zur Beurteilung dieser Systeme erstellt. Ein Vergleich derzeitiger ITS mit Unterstützung für das Themengebiet „Konzeptueller Datenbankentwurf" wird gegeben.

Kapitel 4 beschreibt aus technischer Sicht das bestehende E-Tutor System und die damit verbundenen Voraussetzungen für die Integration eines neuen Expertenmoduls.

Kapitel 5 beschreibt das Konzept der Umsetzung. Es beschäftigt sich vor allem mit der Erkennung und Bewertung von Abweichungen, der gewählten Darstellungs- und Speicherform des Mustermodells und mit Modellierungsvarianten.

Kapitel 6 widmet sich der Umsetzung des UML-Expertenmoduls. Es gibt einen Überblick über die wichtigsten Algorithmen und Klassen. Wichtige Einschränkungen und noch nicht gelöste Probleme werden aufgezeigt.

Kapitel 7 fasst die wesentlichsten Punkte und Erkenntnisse dieser Arbeit nochmals zusammen.

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen behandelt. Zuerst wird ein Überblick über die wichtigsten Lerntheorien gegeben, anschließend wird das Thema „Intelligente Tutorielle Systeme" behandelt. Abschließend wird auf das eigentliche Fachgebiet, den „konzeptuellen Datenbankentwurf" eingegangen. Begleitend zum konzeptuellen Entwurf werden das Entity-Relationship-Model (ER-Modell) sowie die Unified Modeling Language (UML) und hier besonders das UML-Klassenmodell beschrieben.

2.1 Lerntheorien

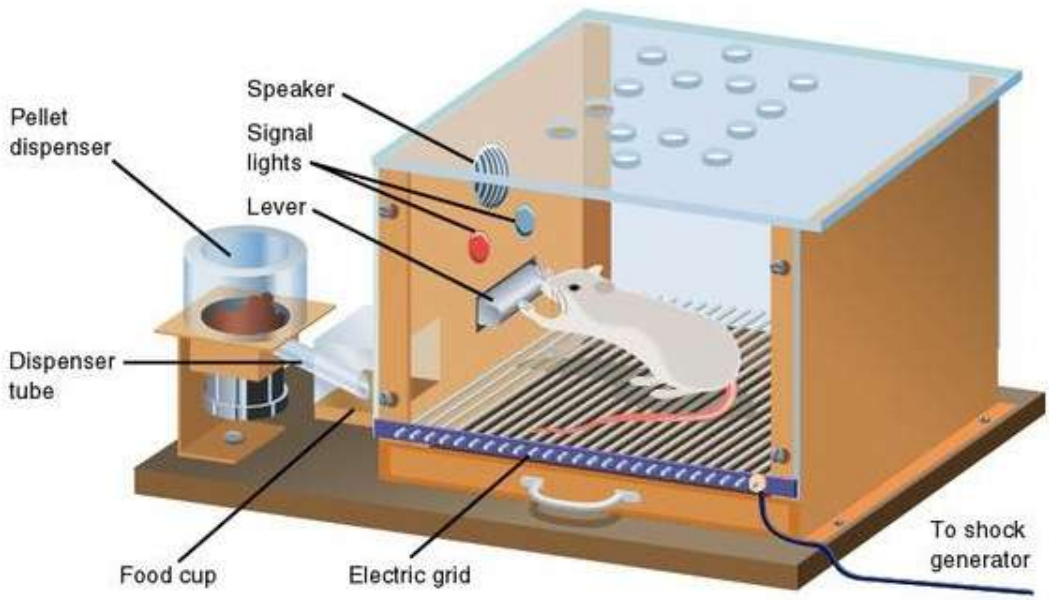
Die Forschung auf dem Gebiet der Theorie des Lernens bzw. des Verständnisses über das Lernen wird vor allem von der Psychologie betrieben. Die Begründer einer Lerntheorie sind meist Wissenschafter aus dem Bereich der Psychologie. Die bekanntesten Lerntheorien sind Behaviorismus, Kognitivismus und Konstruktivismus, die im Folgenden kurz vorgestellt werden.

2.1.1 Behaviorismus

Beim Behaviorismus wird der Lernende selbst als sogenannte „Black Box" bezeichnet. Das heißt die Verarbeitungsvorgänge des Lernenden werden nicht untersucht. Vielmehr wird untersucht wie bestimmte Reize Verhaltensmuster auslösen. Edward Thorndike gilt, gemeinsam mit John B. Watson, als ein Begründer der Theorie. Diese Theorie ist auch als Reiz-Reaktions-Mechanismus (S-R) bekannt. Der Lehrende versucht diesen Mechanismus zu beeinflussen, damit neue Verhaltensmuster erzeugt werden. Es wird zuerst ein Reiz ausgesendet auf den der Lernende mit einem bestimmten Verhalten reagiert. Entspricht das gezeigte Verhalten der gewünschten Reaktion wird der Reiz durch Belohnung verstärkt.

Skinner hat mit seiner Skinner-Box das Verhalten von Ratten untersucht. Er verfeinerte jedoch im Unterschied zu Thorndike den Reiz-Reaktions-Mechanismus indem er nach der erfolgten gewünschten Reaktion eine Belohnung gibt. Wird allerdings nicht die erwünschte Reaktion ausgeführt, so erfolgt anstelle der Belohnung eine Bestrafung (Herrnstein, 1970), (Skinner, 1950).

Abbildung 2-1: Die Skinner-Box (Quelle: (BF Skinner: Operant Conditioning))



2.1.2 Kognitivismus

Während beim Behaviorismus das Gehirn als „Black Box" betrachtet wird, ist gerade beim Kognitivismus diese Box das Zentrum der Erforschung. Es interessieren hier die internen Vorgänge und Verarbeitungsprozesse. Lernen wird als Verarbeitung und auch Aufbereitung von Reizen verstanden. Durch diese „innere" Verarbeitung ist Verhalten nur mehr bedingt von außen steuerbar.

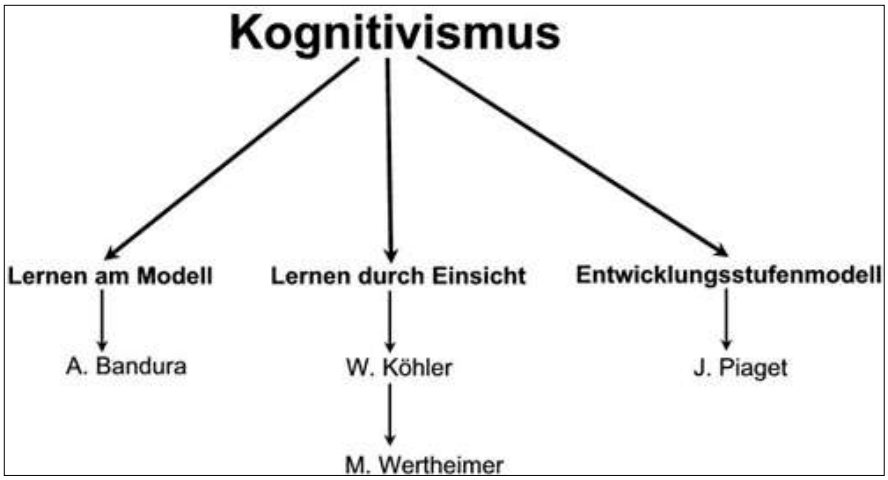
Den Reizen von außen wird mit einem Verhaltensmuster begegnet, das in der Lage ist die Situation mit einzubeziehen. Es entwickeln sich „Landkarten", das heißt der Reiz wird aufgrund von bereits ge-

lernten Verhaltensmustern und durch Erfahrung an die Umgebung angepasst oder die Umgebung wird durch die Interpretation verändert (Baumgartner & Payr, 1994).

Abbildung 2-2: System der kognitiven Lerntheorie (Quelle: (Einführung: Kognitivismus))

Im Rahmen des Kognitivismus gibt es eine Reihe weiterer Theorien, einige davon werden nachfolgend beschrieben.

Abbildung 2-3: Übersicht kognitive Theorien (Quelle: (Einführung: Kognitivismus))



Beim „*Lernen am Modell*" ahmt der Lernende beobachtetes Verhalten nach. Hat er damit Erfolg, wird er auch in Zukunft dieses Verhalten zeigen. Bei Nichterfolg wird das Verhalten nicht weiter angewandt (Angermeier, 1978).

Beim „*Lernen durch Einsicht*" wird einem Lernenden die Lösung eines Problems plötzlich klar. Dieses Erkennen der Lösung, geschieht durch eine Umordnung der Reize. Durch einen neuen geänderten

Blickwinkel können die Probleme in einzelne, bereits bekannte Teilprobleme zerlegt werden. Durch dieses Umstrukturieren werden Lösungsstrategien erarbeitet, und es erfolgt eine plötzliche „Erkenntnis". Die Lösung kann danach jederzeit wiederholt werden (Hilgard & Bower, 1973).

Beim *„Entwicklungsstufenmodell"* beobachtete J. Piaget das Verhalten seiner Kinder. Er konnte beobachten, dass im Laufe der Zeit verschiedene Fähigkeiten erlernt werden. Zwar gab es bei den Kindern unterschiedliche Entwicklungsstufen, trotzdem lassen sich ungefähre Angaben machen, wann ein Kind über welche Fähigkeiten verfügen sollte (Piaget, 1975).

2.1.3 Konstruktivismus

Die Theorie des Konstruktivismus geht davon aus, dass Lernen „konstruiert" wird. Jeder Lernende konstruiert dabei sein Wissen auf Grundlage seiner persönlichen Erfahrungen. Bei dieser „Konstruktion" werden jedoch die kulturellen und gerade aktuellen gesellschaftlichen Werte mit einbezogen.

Lernen wird gleichsam als aktives Handeln begriffen, das aus dem Gehirn selbst entsteht, und auch selbst organisiert wird (Boudouries, 1998). Der Lernende schafft sich eine eigene Repräsentation der Welt, um die Reize zu verarbeiten.

Abbildung 2-4: Aktivitäten während dem Lernen (Quelle: (Siebert, 1998))

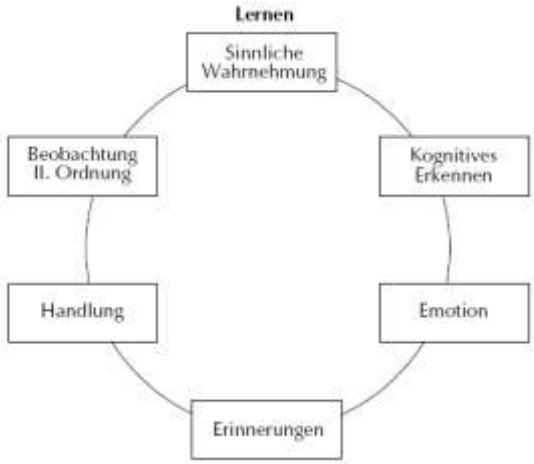
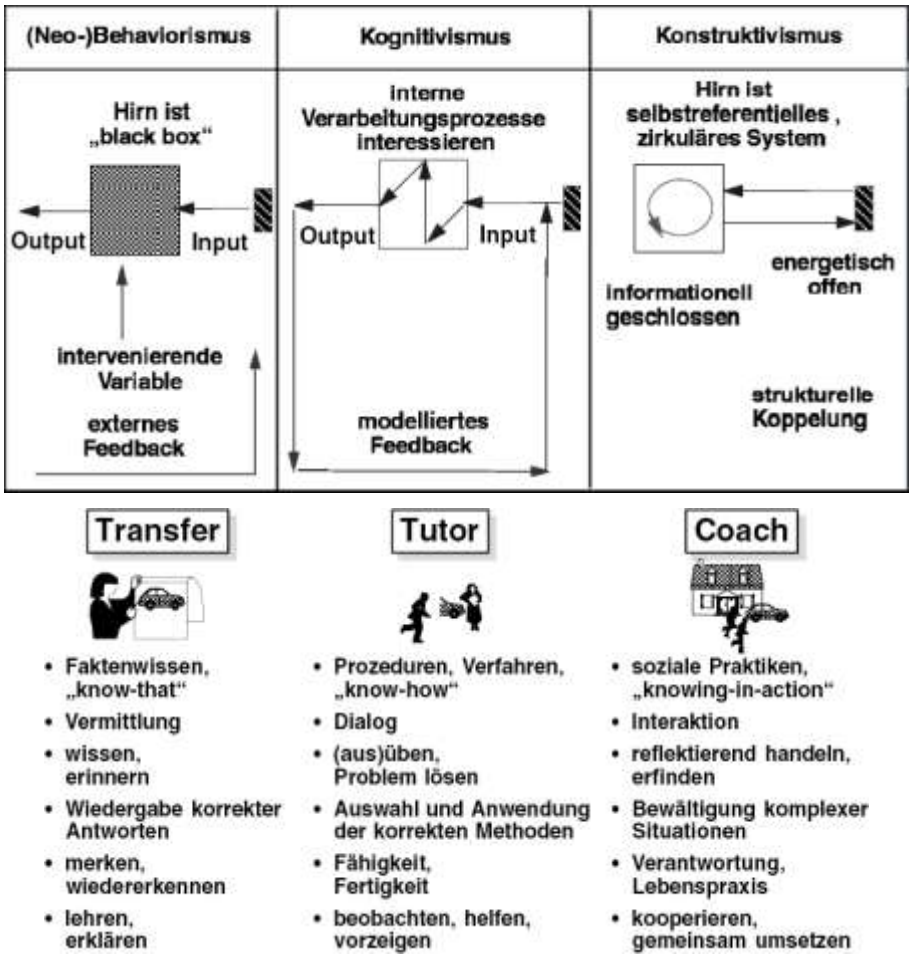


Abbildung 2-5: Übersicht über die Lerntheorien (Quelle: (Waba, 2005))



Unterschiede der drei Lerntheorien

Abbildung 2-5 zeigt einen Vergleich der drei Lerntheorien. Dabei wird nochmals die Hauptaussage dargestellt und die jeweiligen Anwendungsformen beschrieben. Während beim Behaviorismus hauptsächlich Faktenwissen aufgenommen wird, ist es beim Kognitivismus hauptsächlich Anwendungswissen, und beim Konstruktivismus Problemlösungskompetenz.

Anwendung der Lerntheorien

Elektronische Lehr- bzw. Lernsysteme sollen die Studenten beim Erlernen eines Stoffgebiets unterstützen, deshalb ist die Beachtung der Lerntheorien von Bedeutung. Elektronische Tutoring Systeme sind am ehesten der Theorie des Kognitivismus zuzuordnen, da durch gezielte Rückmeldungen Hilfestellung angeboten wird. Dadurch bekommt der Student eine andere Sicht auf das Problem, um damit der Lösung näherzukommen. Je mehr Wissen der Student bereits aufgebaut hat, umso eher kommt auch der Konstruktivismus in Frage. Durch verschiedene Lösungsversuche und den unterschiedlichen Hinweisen baut der Student Faktenwissen auf. Mit Hilfe dieses Faktenwissens konstruiert er ein Lösungsverfahren. Letztlich ist es das Ziel, den Studenten zur Problemlösungskompetenz zu führen. Die Erreichung dieses Ziels soll mit Hilfe von qualifiziertem, in mehreren Detailierungsstufen gegebenem, Feedback unterstützt werden.

2.2 Intelligente Tutorielle Systeme (ITS)

Intelligente tutorielle Systeme sind computerbasierte Systeme die versuchen, die Rolle eines Tutors nachzuahmen. In diesen virtuel-

len Lernumgebungen wird das Fachwissen eines Tutors in Modulen gespeichert. Das Feedback und die Bewertung der Übungen müssen aus diesem gespeicherten Wissen erfolgen. Dabei sind diese Systeme so ausgelegt, dass sie in der Lage sind, die Lernfortschritte des Lernenden durch qualifiziertes Feedback zu verbessern.

Der Begriff Intelligent Tutoring Systems wurde 1982 von Sleeman & Brown erstmals verwendet (Brown, 1982). Sie verwendeten diesen Begriff um diese gegen die vorhandenen Computer based Training (CBT) Systeme abzugrenzen. Auch Heinrich/Roithmayr erklären bereits 1998 „intelligentes Lehrsystem" in ihrem Wirtschaftsinformatik Lexikon (Heinrich & Roithmayr, 1998):

„**Intelligentes Lehrsystem**. Eine Weiterentwicklung der traditionellen computerunterstützten Lehrverfahren. Ein i. L. ist in der Lage, im Dialog Wissen über den Lernstoff und über den Lernenden zu verwenden und damit den Lernprozeß zu steuern. Nach der Art der Lernwegsteuerung unterscheidet man rechnergesteuertes Lehrsystem und lerngesteuertes Lehrsystem.

- Rechnergesteuertes Lehrsystem: Der Lernprozeß wird vom Computer gesteuert.
- Lerngesteuertes Lehrsystem: Der Lernprozeß wird vom Lernenden gesteuert.

Den Erkenntnissen der Lernpsychologie und der Didaktik entspricht eine gemischte Lernwegsteuerung am besten. Nach der Vorgehensweise beim Lernprozeß unterscheidet man selektives Lehrsystem und generatives Lehrsystem.

- Selektives Lehrsystem: Zur Fortsetzung des Lernprozesses wird zwischen geplanten, bei Lernbeginn festgelegten Alternativen ausgewählt.
- Generatives Lehrsystem: Die Alternativen werden erst im Lernprozeß bestimmt."

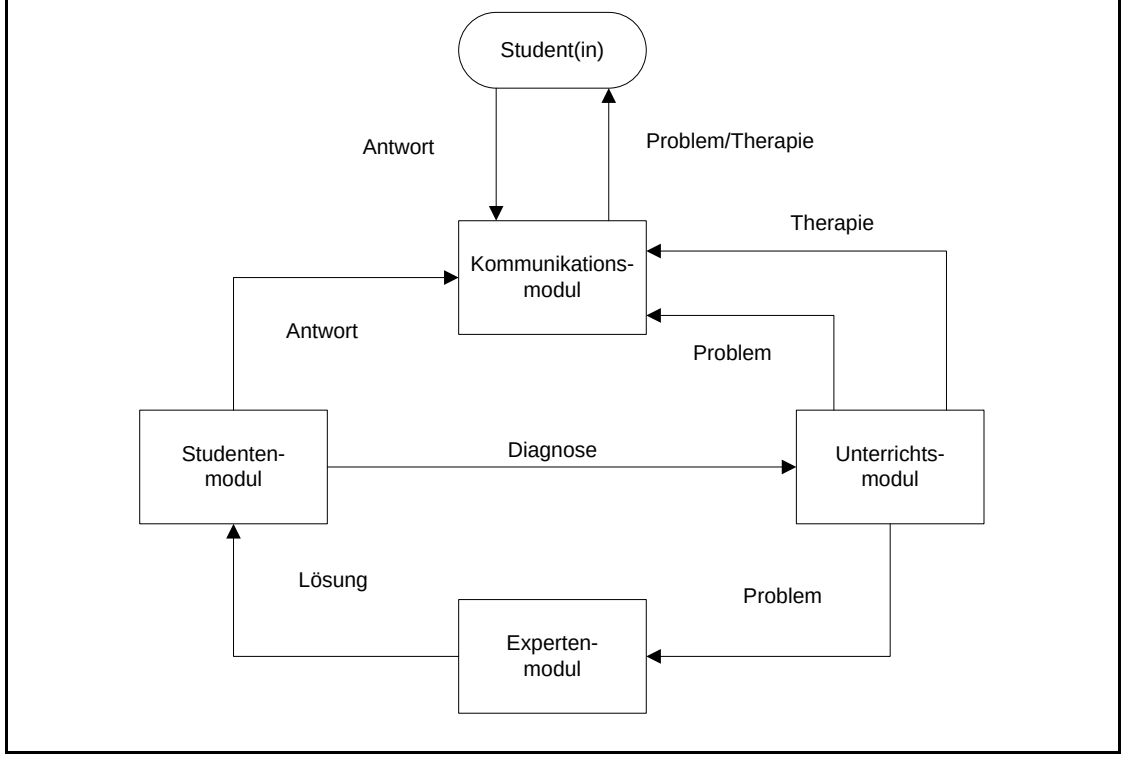
Heutige intelligente tutorielle Systeme (ITS) bestehen in der Regel aus vier Komponenten (Urban-Lurain, 1996):

Kommunikationsmodul

Das Kommunikationsmodul (siehe auch Abbildung 2-6) bildet die Benutzungsschnittstelle zwischen dem Studenten und dem System ab. Es stellt das Benutzungsinterface dar, und leitet die Ein- und

Ausgaben des Systems an die entsprechenden Module weiter. Das Kommunikationsmodul wählt die Kommunikationsform. Beispiele für die Kommunikationsform sind Textausgabe, Dialoge, Fenster und auch natürliche Sprache. Die Kommunikationsform hängt auch von den technischen Möglichkeiten des verwendeten Systems ab.

Abbildung 2-6: Struktureller Aufbau eines ITS nach Lusti (Quelle: (Lusti, 1992))



Um den Anforderungen der unterschiedlichen Module gerecht zu werden muss das Kommunikationsmodul die Benutzungsschnittstelle dynamisch aufbauen. Neben der Flexibilität empfiehlt sich auch ein Web-fähiges UI, um die Möglichkeiten des Internets voll ausschöpfen zu können.

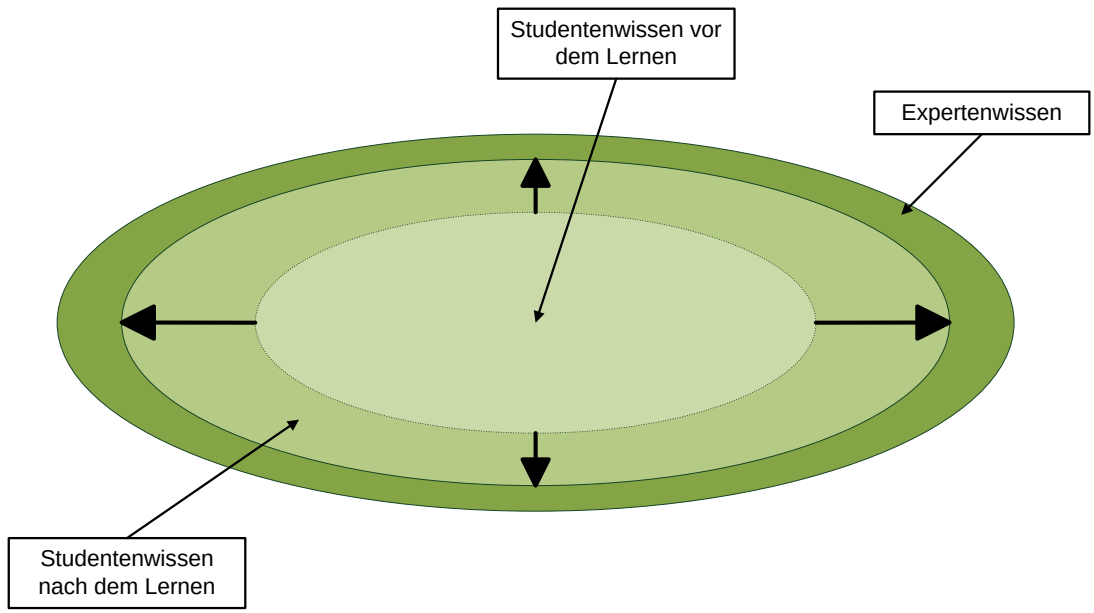
Studentenmodul

Das Studentenmodul (siehe auch Abbildung 2-6) zeichnet die Lösungsversuche eines Lernenden auf, und versucht daraus Rückschlüsse auf das Lernverhalten zu ziehen. Das Studentenmodul muss dazu auch das Profil des Studenten erfassen. Das Profil bein-

haltet die Vorkenntnisse des Studenten und weiß über Erfolg oder Misserfolge bisheriger Lernschritte Bescheid. Das Wissen über den Lernenden wird zum Teil vor und zum Teil während der Laufzeit erworben.

Die Diagnose der Fehler stellt die wichtigste Möglichkeit dar, Rückschlüsse auf den Wissensstand des Studenten zu ziehen. Der Wissensstand wird benötigt um festzustellen, ob die aktuelle Aufgabe gelöst wurde. Zur nächsten Aufgabe darf erst dann gewechselt werden, wenn die aktuelle Aufgabe ausreichend gelöst worden ist. Das Wissen über Art und Häufigkeit der vom Studenten gemachten Fehler, kann dazu verwendet werden um Aufgaben gezielt nach dem Schulungsbedarf auszuwählen.

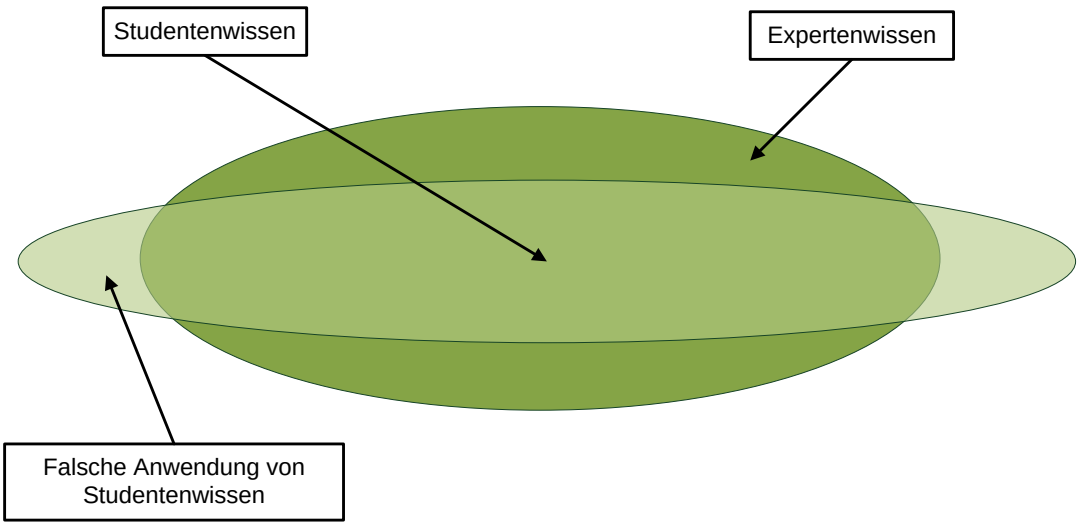
Abbildung 2-7: Überlagerungsmodell



Bei den Modellen der Fehlerdiagnose haben sich zwei verschiedene Klassen entwickelt. *Überlagerungsmodelle* (differential model) (Abbildung 2-7) gehen davon aus, dass der Fehler passiert weil der Student zwar über korrektes Wissen verfügt, dieses aber unvollständig ist. *Störungsmodelle* (bug model) (Abbildung 2-8) gehen

davon aus, dass der Student richtiges Wissen falsch anwendet und bei der Anwendung systematisch verändert (Martens, 2003).

Abbildung 2-8: Störungsmodell



Das Erkennen der Fehlerursachen gilt allerdings als schwierig und ist natürlich vom Stoffgebiet abhängig. Das Studentenmodul ist auch auf die Ausgaben des Expertenmoduls angewiesen um genügend Informationen über die Antworten des Lernenden zu erhalten (Lusti, 1992).

Expertenmodul

Das Expertenmodul (siehe auch Abbildung 2-6) bildet das Wissen eines Fachexperten ab. Dieses Wissen soll ein ITS befähigen, Aufgaben an den Lernenden zu stellen. Außerdem muss es in der Lage sein, die Antworten des Lernenden zu interpretieren. Dazu muss es auch Variationen oder Varianten einer Antwort richtig einordnen.

Das Expertenmodul muss auch das Feedback an den Lernenden entsprechend seiner Antwort verfassen. Jede Antwort an den Lernenden muss dementsprechend begründet werden. Je komplexer das Lehrgebiet ist, umso schwieriger ist auch die Entwicklung eines

Expertenmoduls dafür. Einige ITS bilden das Expertenwissen lediglich deklarativ ab, andere hingegen versuchen der Komplexität des Expertenwissens mit Methoden der künstlichen Intelligenz beizukommen, und auch eine prozedurale Abbildung des Expertenwissens ist möglich (Lusti, 1992).

Eine deklarative Wissensdarstellung beschreibt das Wissen in Form von Fakten. Das „was" steht dabei im Vordergrund. Eine deklarative Wissensdarstellung ist in der Regel für einen Lehrenden leichter lesbar als Programmcode. Auch die Anpassung an bestimmte Lehrinhalte ist meist einfacher zu gewährleisten. Deklarative Wissensbeschreibungen, wie z.B. Horn-Klauseln, können auch mit Methoden der künstlichen Intelligenz verarbeitet werden.

Die prozedurale Wissensdarstellung ist eine Abbildung des Wissens in Form von Programmcode. Das „wie" steht hier im Vordergrund. Die vorhandenen Datenstrukturen und Aktionen der verwendeten Programmiersprache müssen bei der Abbildung berücksichtigt werden. Die prozedurale Wissensabbildung ist für den Lehrenden aber schwerer lesbar als eine deklarative Beschreibung. Auch die Anpassung ist schwieriger, da in der Regel ein Programmierer benötigt wird.

Die deklarative Wissensdarstellung ist problembezogen, während die prozedurale Darstellung maschinenbezogen ist. Zwar ist die prozedurale Wissensdarstellung im Wesentlichen effizienter abzuarbeiten als die deklarative, dafür sind deklarative Wissensbasen aber in der Lage Fragen zum Wissen zu beantworten (Boersch, Heinsohn, & Socher, 2007).

Unterrichtsmodul

Das Unterrichtsmodul (siehe auch Abbildung 2-6) wählt Aufgaben aus und übergibt sie zur Präsentation an das Kommunikationsmodul. Dabei bedient es sich beim Studentenmodul. Die Einteilung

des Unterrichtsstoffes erfolgt aufgrund von Beurteilungen des Studenten durch das Studentenmodul. Die Unterrichtsreihenfolge wird durch die Lernsequenz vorgegeben. Erst wenn ein Student die Aufgabe gelöst hat wird die nächste Aufgabe freigeschaltet. Dazu wird der individuelle Lernfortschritt jedes einzelnen Studenten aufgezeichnet.

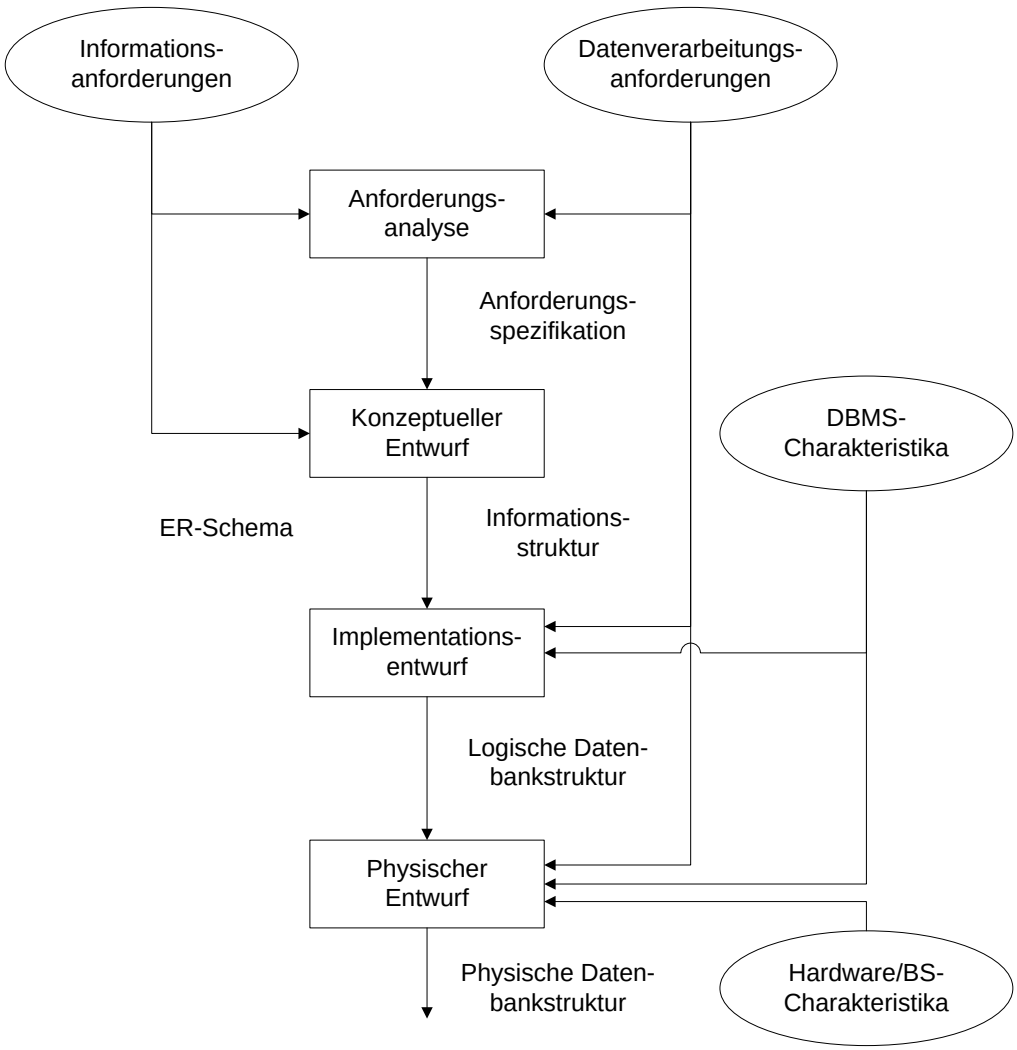
Die Ausgabe von Aufgaben und Fragen richtet sich nach der Antwort des Lernenden. Das heißt, es muss auf Fehler reagiert werden. So kann zum Beispiel auf bereits gelöste Aufgaben verwiesen werden oder Teile der Lösung angezeigt werden. Gegenbeispiele, welche den Widerspruch der Lösung aufzeigen, sind wegen der mit ihnen verbundenen Anschaulichkeit, besonders geeignet. Zudem sollte ein Unterrichtsmodul auch in der Lage sein auf Fragen des Lernenden zu antworten (Lusti, 1992).

Durch diese Modularisierung können auch Systeme für viele Benutzer gebaut werden. Die klare Aufgabentrennung der Module sorgt für einen übersichtlichen Aufbau. Diese logische Struktur kann auch softwaretechnisch in Moduln implementiert werden. Das Fachwissen wird im Expertenmodul „hinterlegt", so dass durch hinzufügen von neuem Expertenwissen im Expertenmodul, das System um neue Lehrgebiete erweitert werden kann.

2.3 Konzeptuelle Datenmodellierung

Beim Entwerfen eines Datenbank-Schemas wird meist mit dem konzeptuellen Modell begonnen. Diese Modellart dient dazu, datenbankunabhängig die Struktur und Zusammenhänge der Datensätze zu beschreiben (Rechenberg, 1994). Unzählige Lehrbücher zu diesem Thema sind bisher erschienen. Abbildung 2-9 zeigt die Phasen des Entwurfs einer Datenbank. Das konzeptuelle Modell ist auch die Ausgangsbasis für das logische und physische Modell.

Abbildung 2-9: Phasen des DB Entwurfs (Quelle: (Kemper & Eickler, 2006))



Für diese wichtige Tätigkeit beim Datenbankentwurf wurden auch unterschiedliche Modellierungstechniken, mit meist eigener Notation entwickelt. Einige davon sind z.B. die Entity-Relationship (ER)-Modellierung, die von Rumbaugh et.al 1991 entwickelte Object Modelling Technique (kurz OMT), als auch das von Jacobson entwickelte Object oriented Software Engineering (OOSE) (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1993).

Aus den Bestrebungen der Autoren von OOSE und OMT eine einheitliche Modellierungssprache zu entwickeln, entstand die Unified Modelling Language (UML). UML bietet verschiedene Diagramme,

wobei das Klassendiagramm für den konzeptuellen DB-Entwurf vorgesehen ist. Seit die Unified Modeling Language (UML) 1997 von der Object Management Group (OMG)[9] standardisiert worden ist, hat sie sich als de facto Standard für den konzeptuellen Datenbankentwurf entwickelt.

Auch am DKE Institut der Johannes Kepler Universität wird in der Lehre die UML als Notation für konzeptuelle Datenmodelle verwendet.

Textanalyse (Linguistic method)

Aufgaben werden als textuelle Beschreibung des Anwendungsfalls gegeben. Um von der textuellen Beschreibung einen Datenbankentwurf zu erstellen, ist ein Vorgehensmodell notwendig. Die Textanalyse ist ein bekanntes Vorgehensmodell. Aus dem Text mit der Beschreibung der realen Welt soll ein Datenbankschema abgeleitet werden. Dazu werden in einem ersten Schritt alle Nomen aus dem Text notiert. Diese Wörterliste dient als Basis für die Auswahl der Entitäten bzw. der Entitätstypen und damit der Klassen. Das Erkennen der Entitäten hängt dabei jedoch sehr stark von der Qualität des Textes ab.
Danach werden alle Synonyme aus der Liste gelöscht. Besondere Vorsicht ist bei Homonymen, das sind gleiche Wörter mit verschiedener Bedeutung, walten zu lassen. Es verbleit somit eine Liste mit Klassenkandidaten. Ein Nomen kann aber auch ein Attribut einer Klasse bezeichnen. Im folgenden Schritt werden daher aus der Kandidatenliste jene Begriffe, die Attribute einer Klasse bezeichnen, dieser Klasse zugeordnet. Die weiteren Attribute ergeben sich aus dem Text. Es werden alle Adjektive die in einer textuellen Beziehung zur Klasse stehen als Kandidaten für Attribute vermerkt.

[9] Siehe http://www.omg.org/

Auch hier müssen in einem weiteren Schritt alle nicht benötigten, sowie synonymen Attribute entfernt werden.

Aus den Verben können schließlich die Beziehungen der Klassen untereinander abgeleitet werden. Bestimmte Formen wie etwa „ist ein", deuten dabei auf Vererbungsbeziehungen hin. Aus Formen wie „hat ein" und „besitzt ein", können Assoziationen abgeleitet werden. Auch die Wertigkeiten der Beziehungen (Kardinalitäten) sollen sich aus dem Text ergeben.

Wichtig ist dabei noch zwischen allgemeinen und speziellen Begriffen zu unterscheiden. Nur Begriffe die einen allgemeinen Artefakt beschreiben eigenen sich als Entitätstyp. Begriffe die ein spezielles Artefakt (wie etwa eine bestimmte Person) bezeichnen sind als Entitätstyp ungeeignet (Overmyer, Lavoie, & Rambow, 2001).

Abbildung 2-10: Beispiel linguistische Textanalyse

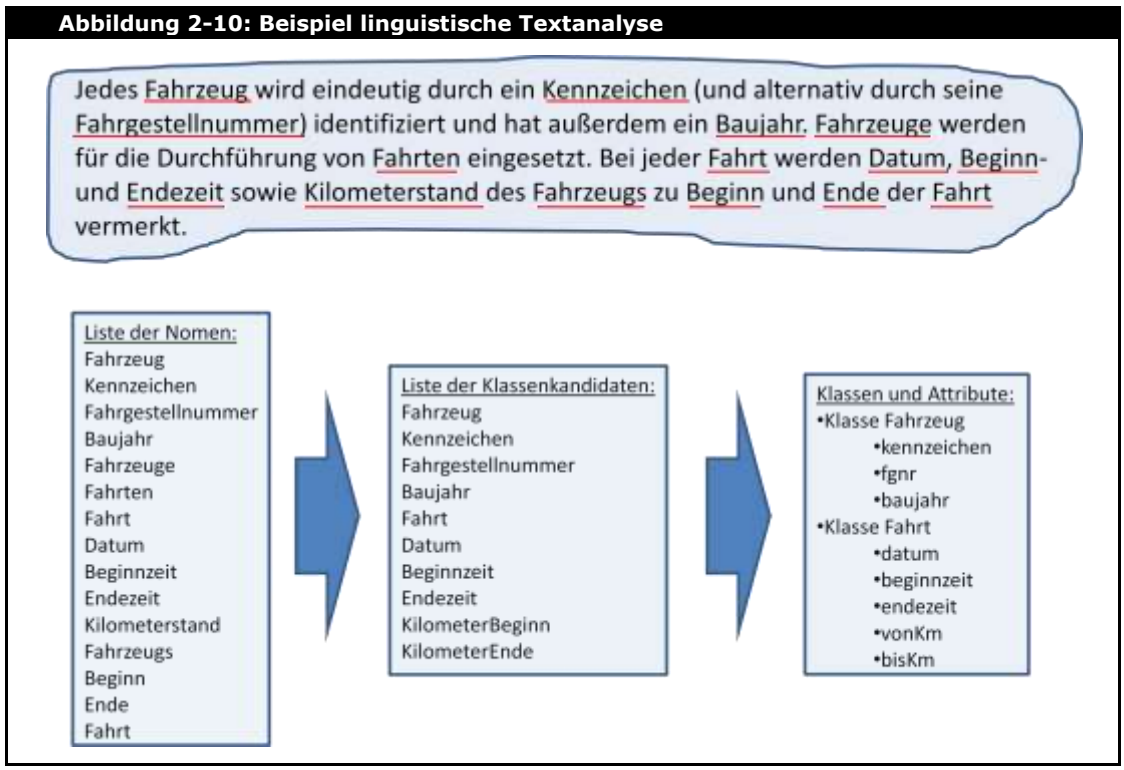


Abbildung 2-10 zeigt die Methodik anhand eines Textausschnitts des Übungsbeispiels. Es werden die Klassen „Fahrzeug" sowie „Fahrt" ermittelt. Durch den Satz „Fahrzeuge werden für die Durch-

führung von Fahrten eingesetzt", kann auch eine Beziehung der vorher ermittelten Klassen erkannt werden.

2.3.1 Entity Relationship Modeling

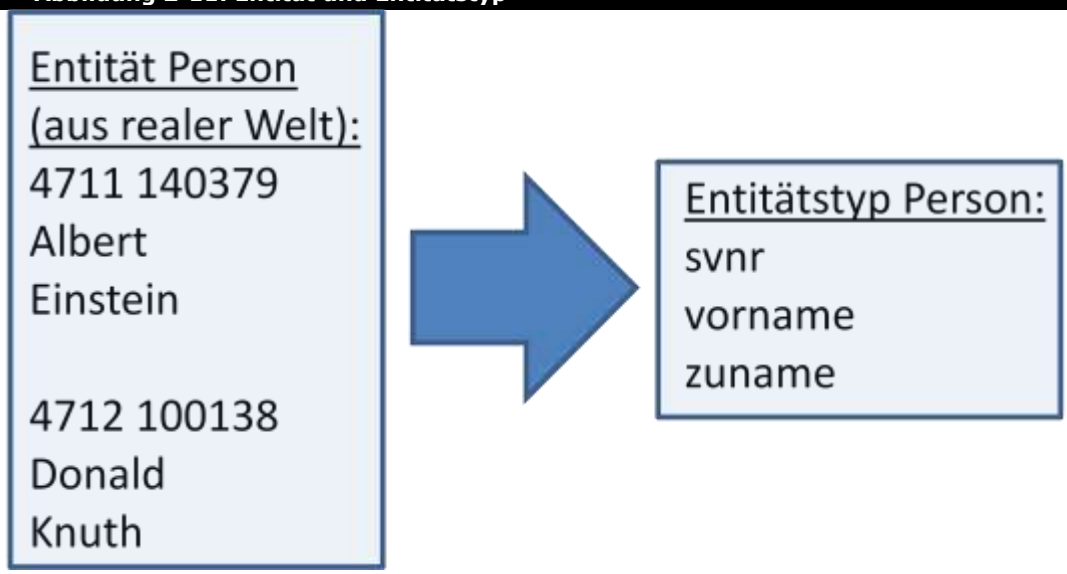
Das Entity-Relationship-Modell (ER-Modell) wurde entwickelt um die Struktur einer Datenbank zu visualisieren. Es ist somit eine Notation für den (konzeptuellen) Datenbankentwurf. Das ER-Diagramm dokumentiert die Entitäten und die Beziehungen der Entitäten zueinander. Die formale Basis für die ER-Modellierung basiert auf mathematischen Regeln. Es wird zwar heute vermehrt auf die UML-Notation zugegriffen, die Art und Weise wie die reale Welt modelliert wird hat sich aber nicht geändert, sie basiert nach wie vor auf den Konzepten der ER-Modellierung. Das ER-Modell wurde 1976 von Peter Chen vorgestellt (Chen P. P., 1976).
Das Konzept basiert auf der Überlegung, dass die reale Welt aus Objekten besteht, und diese Objekte Beziehungen zueinander haben. Es wurde entwickelt um das Design von Unternehmensdatenbanken zu erleichtern. Der semantische Aspekt liegt darin, dass die Bedeutung der Daten im Vordergrund steht. Gerade die Interpretation der Daten und ihrer Interaktionen ist auch für das konzeptuelle Design von großer Bedeutung (Silberschatz, Korth, & Sudarshan, 1996).

Das ER-Modell kennt 3 Basisnotationen und zwar das „entity set", das „relationship set" sowie „attribute". Im Folgenden wird auf diese 3 Basisnotationen und anschließend auf Erweiterungen dieser Basisnotation eingegangen.

Entität

Eine Entität (Entity) entspricht einem Objekt aus der realen Welt. Eine Entität ist also eine Beschreibung eines Teils der Realität. Die Entitäten sind voneinander unterscheidbar. Jede Entität hat darüber hinaus auch Eigenschaften. Eine Entität kann sowohl konkret (z.B. Buch) als auch abstrakt (z.B. Urlaub) sein. Allen Entitäten mit den gleichen Eigenschaften wird ein Entitätstyp zugewiesen.

Abbildung 2-11: Entität und Entitätstyp

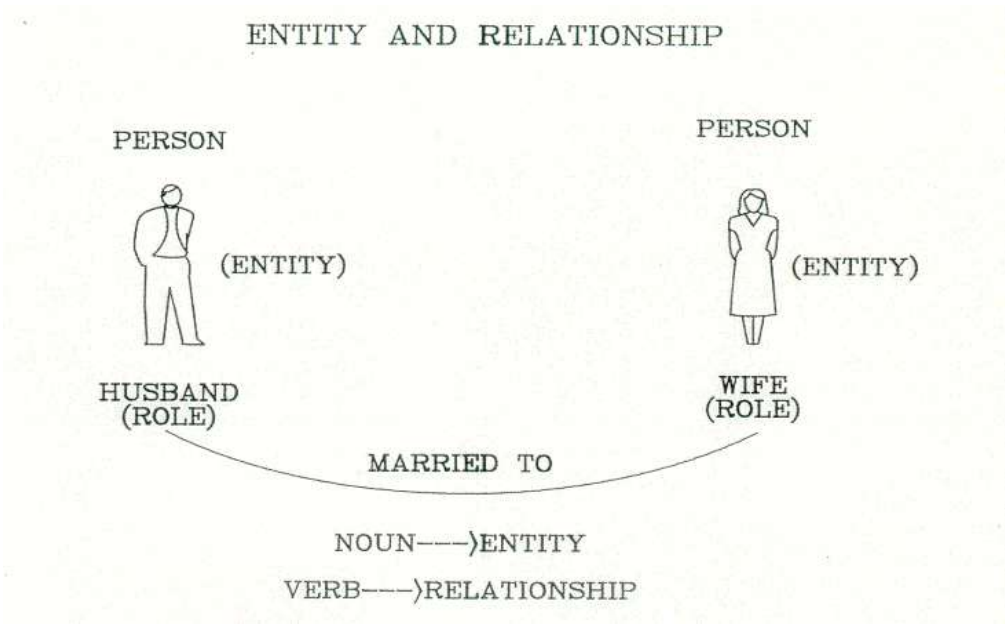


Eine Menge an Entitäten mit gleichem Typ wird als Entitätsmenge (entity set) bezeichnet. Alle Personen, die Prüfer für eine bestimmte Führerscheinklasse sind, könnten zum Beispiel in einer Entitätsmenge erfasst werden.

Entitäten werden über ihre Eigenschaften (Attribute) unterschieden. Attribute die eine Entität identifizieren werden als Schlüsselkandidat (candidate key) bezeichnet. Gibt es für eine Entität mehrere Schlüsselkandidaten, so muss einer als Primärschlüssel (primary key) ausgewählt werden (Chen P. P., 1976). Es gibt jedoch Entitätstypen, aus deren Eigenschaften sich kein eindeutiger Schlüssel bilden lässt. Man nennt diese Form schwache Entitätsty-

pen (weak entity set). Solche schwachen Entitätstypen sind von einem anderen (starken) Entitätstyp abhängig (Silberschatz, Korth, & Sudarshan, 1996).

Abbildung 2-12: Konzept des ER-Modells (Quelle: (Chen P. P., 2002))



Attribut

Attribute sind die beschreibenden Eigenschaften einer Entität. Jedem Attribut ist ein Wertebereiche (domain oder value set) zugeordnet. Konkrete Werte (value) müssen Elemente aus der zugewiesenen Wertemenge sein. Attribute können einfache Werte (simple) oder zusammengesetzte Werte (composite) beinhalten. Zusammengesetzte Werte können wiederum in Attribute mit einfachen Werten zerteilt werden. Neben Attributen mit nur einem Wert (single-value attribute), gibt es auch Attribute die über mehrere Werte (multivalued attribute) verfügen (Silberschatz, Korth, & Sudarshan, 1996).

Beziehung

Entitäten können Beziehungen (Relationships) eingehen. Eine Beziehung drückt aus, in welcher Art Entitäten miteinander interagieren. Gleichartige Beziehungen zwischen Entitätstypen werden Beziehungstypen genannt. Die Anzahl beteiligter Entitätstypen wird als Grad des Beziehungstyps bezeichnet. Normalerweise werden Beziehungen mit dem Grad 2 verwendet (also 2 Entitäten). Diese werden binär genannt. Es gibt aber auch Beziehungen mit nur einer Entität, als auch Beziehungen mit 3 (ternär) oder mehr beteiligten Entitäten.
Höherwertige Beziehungen (ab Grad 3) lassen sich auf binäre reduzieren, wenn ein eigener Entitätstyp eingeführt wird. Dieser neue Entitätstyp beschreibt die aufgelöste Beziehung. Auf diese Art kann man nur binäre Beziehungstypen modellieren ohne einen Verlust von Beziehungen zu erleiden (Chen P. P., 1976).
Neben dem Grad eines Beziehungstypen ist auch die Kardinalität (cardinality) von Interesse. Die Kardinalität einer Beziehung beschreibt die Anzahl der Entitäten die an der Beziehung teilnehmen können. Eine Person kann zum Beispiel mehrere Bücher lesen. So eine Beziehung wird 1:n (one-to-many) Beziehung genannt. Darüber hinaus sind noch 1:1 (one-to-one), n:1 (many-to-one) und m:n (many-to-many) Beziehungen möglich.

Obwohl mit diesen Basiselementen die meisten Datenbankaufgaben modelliert werden können, fehlten noch einige wichtige Konstrukte, so dass 1977 John Miles Smith und Diane C.P. Smith Erweiterungen zum ER-Modell vorstellten (Smith & Smith, 1977).

Spezialisierung und Generalisierung

Die Generalisierung bzw. Spezialisierung ist ein mächtiges Abstraktionsmittel. Dabei werden gemeinsame Eigenschaften von Entitätstypen geteilt. Das heißt, die gemeinsamen Eigenschaften von Entitätstypen bilden einen eigenen übergeordneten Entitätstyp. Die spezifischen, nicht verallgemeinbaren Eigenschaften bleiben bei den zugrundeliegenden Entitätstypen. Das Bilden einer übergeordneten Basisklasse mit gemeinsamen Eigenschaften, wird Generalisierung genannt. Erfolgt von einer Allgemeinen Klasse das Bilden von spezifischen Klassen so spricht man von Spezialisierung. Typischerweise wird durch diese Vorgehensweise eine hierarchische Struktur (meist in Form eines Baumes) aufgebaut (Smith & Smith, 1977).

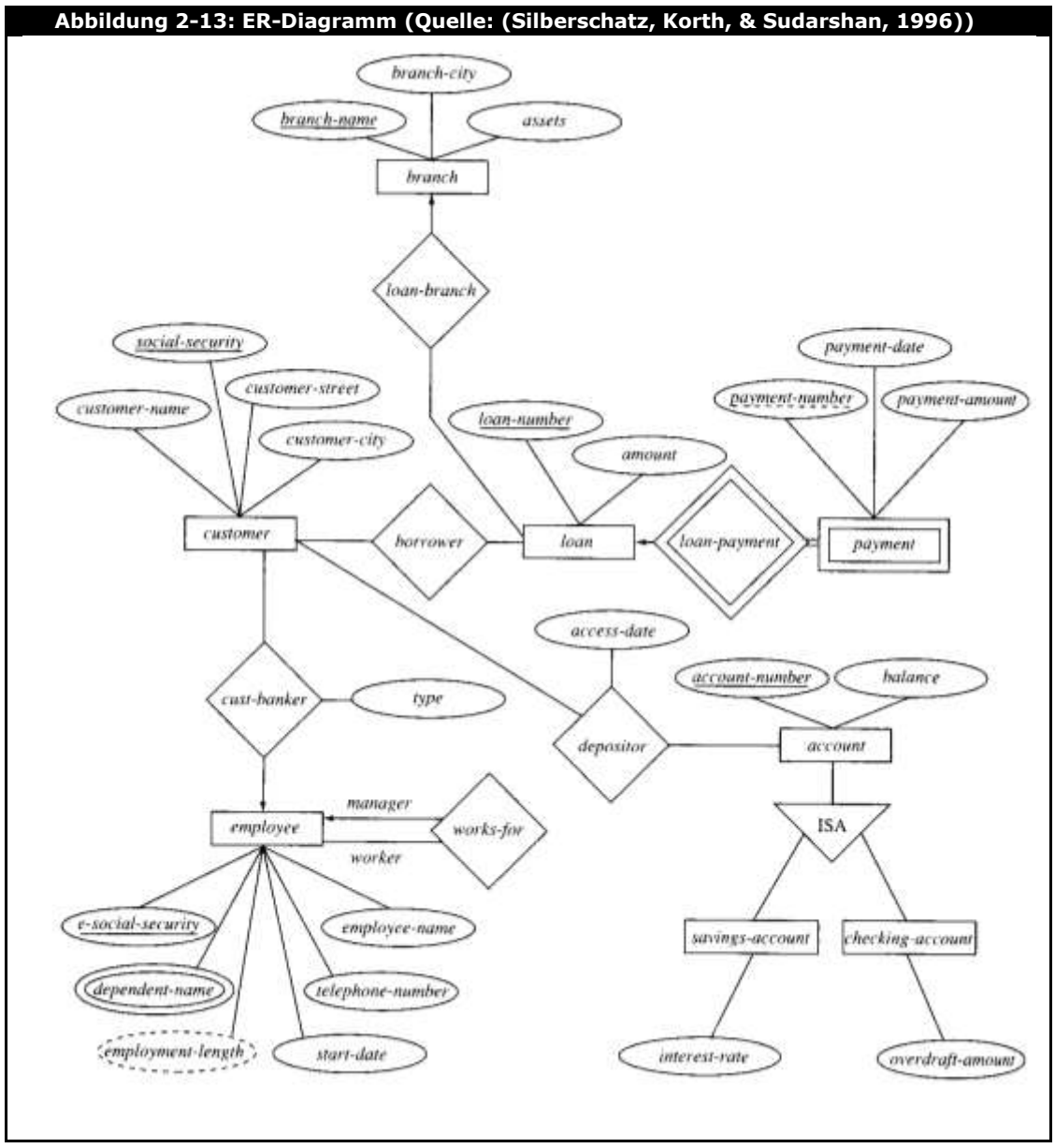
Aggregation

Bei der Aggregation werden eigenständige Entitätstypen zu einem Gesamt-Entitätstyp zusammengefügt. Eine Aggregation besteht also aus Komponenten. Im Unterschied zu einer Generalisierung bestehen hier keine gemeinsamen Eigenschaften, sondern es handelt sich um eine logische Gruppierung. Diesen Unterschied drückt auch die Bezeichnung „ist-Teil-von" (is-part-of) aus (Smith & Smith, 1977). Als Beispiel kann hier die Aggregation Lampe genannt werden. Komponenten dieser Aggregation sind Sockel, Schirm, Schalter und Verdrahtung. Eine Aggregation kann also als Aufgliederung eines Objektes in seine Teilobjekte gesehen werden (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1993).

ER-Diagramm

Für die Darstellung des ER-Modells gibt es sehr viele verschiedene Notationen. Bereits Peter Chen hat eine Notation eingeführt (Abbildung 2-13).

Abbildung 2-13: ER-Diagramm (Quelle: (Silberschatz, Korth, & Sudarshan, 1996))



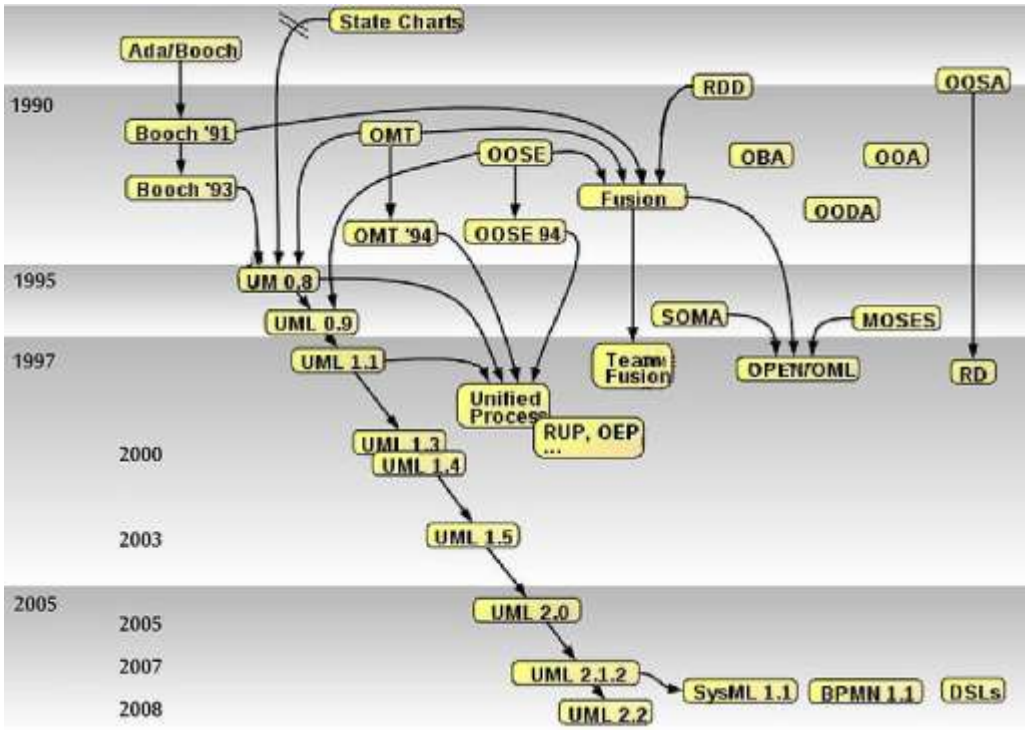
Die unterschiedlichen Notationen unterscheiden sich in Umfang und Darstellungsformen. Bekannte Notationen sind unter anderem die IDEF1X Notation, die der de-facto Standard bei US-amerikanischen

Behörden war[10]. Von einigen Modellierungswerkzeugen wird auch die „Krähenfuß"-Notation verwendet[11].

2.3.2 Die Unified Modeling Language (UML)

Von Grady Booch, James Rumbaugh und Ivar Jacobson wurde 1994 mit den Arbeiten an einer universellen Modellierungssprache begonnen, die schließlich im Juli 1997 bei der Object Management Group (OMG) zur Standardisierung eingereicht wurde.

Abbildung 2-14: Historie von UML (Quelle: (Wikipedia, 2008))



Die Unified Modeling Language (UML) Version 1.1 wurde am 14. November 1997 als neuer Standard angenommen. Die Weiterentwicklung der UML wurde von der OMG Revision Task Force über-

[10] FIPS 184 (Federal Information Processing Standards) inzwischen zurückgezogen. Siehe http://www.itl.nist.gov/fipspubs/withdraw.htm

[11] Die Kardinalitäten (Multiplizitäten) werden durch 0 (Null), | (Eins) bzw. dem ≺ Krähenfuß (beliebig viele) gekennzeichnet.

nommen (Booch, Rumbaugh, & Jacobson, 1999). Die Version 1.4.2 wurde im Jänner 2005 als ISO/IEC19501 veröffentlicht (OMG, 2005).

Da die UML als allgemeine Notation bzw. Sprache für alle Modellierungsaufgaben im objektorientierten Software Engineering konzipiert ist, verfügt sie über 13 verschiedene Modellarten. Mit dem Klassendiagramm bietet sie eine Diagrammart, mit der auch Datenbank-Schemata modelliert werden können. Die UML ist mittlerweile weit verbreitet und als Modellierungssprache allgemein anerkannt. Aktuell wird die UML in der Version 2 weiterentwickelt.

Die UML ist eine Sprache zum Modellieren, Dokumentieren und Spezifizieren. Sie besteht im Wesentlichen aus einer visuellen Notation, sowie Regeln zum Kombinieren der einzelnen Elemente der visuellen Notation (Booch, Rumbaugh, & Jacobson, 1999).
UML Modelle stellen keine Implementierung dar, und sind von Programmiersprachen unabhängig. Es können jedoch aus Codestücken von vielen objektorientierten Programmiersprachen UML Modelle generiert werden (Reverse Engineering), und aus UML-Modellen Codestücke (meist Skeletons) generiert werden (Forward Engineering). Es gibt eine Vielzahl von frei verfügbaren UML-Editoren, wie etwa das Java-basierte Argo-UML (Tigris, 2008).

Die UML besteht aus 3 so genannten „Building blocks":

Dinge

Bei den Dingen wird zwischen Struktur und Verhalten unterschieden. Zusätzlich gibt es noch Elemente für das Gruppieren, sowie für das Kommentieren. Dinge sind die grundlegenden objektorientierten Bestandteile der UML (Booch, Rumbaugh, & Jacobson, 1999).

Beziehungen

Beziehungen zwischen verschiedenen Elementen können unterschiedlicher Natur sein. Die UML sieht 4 verschiedene Typen von Beziehungen vor. Eine *Abhängigkeit* (dependency) beschreibt eine semantische Beziehung zwischen zwei Dingen. *Assoziationen* (association) beschreiben Mengen von Objektbeziehungen. *Generalisierungen* (generalization) beschreiben eine Beziehung zwischen allgemeinen und speziellen Objekten. Die *Realisierung* (realization) ist eine Beschreibung einer semantischen Beziehung zwischen Klassifizierungen (Booch, Rumbaugh, & Jacobson, 1999).

Diagrams

Diagramme zeigen einen Ausschnitt aus dem System und verbinden dabei Dinge und Beziehungen. Je nach Anwendungszweck gibt es verschiedene Diagramme. Es wird dabei in statische (structural diagrams), beziehungsweise dynamische (behavioral diagrams), und ab der Version 2.0 auch in Interaktions- Diagramme unterteilt (Booch, Rumbaugh, & Jacobson, 1999). Für das Entwerfen des konzeptuellen Schemas einer Datenbank ist das Klassendiagramm bestens geeignet. Im Folgenden wird das Klassendiagramm daher näher beschrieben.

UML Klassendiagramm

Klassendiagramme sind die hauptsächlich verwendeten Diagramme zum Modellieren von objektorientierten Systemen. Ein Klassendiagramm beinhaltet Klassen, Schnittstellen, Kollaborationen, Abhängigkeiten, Generalisierungs- und Assoziationsbeziehungen.

Eine Klasse beschreibt eine Gruppe von Objekten, die über gemeinsame Eigenschaften verfügen. Eine Klasse ist also eine abstrakte Beschreibung von Objekten. Jedes Objekt hat somit auch

eine Klasse. In einem Klassendiagramm werden die in einem System vorhandenen Klassen, sowie ihre Beziehung zueinander dargestellt (Rumbaugh, Blaha, Premerlani, Eddy, & Lorensen, 1993).

Abbildung 2-15: Wichtige Elemente des Klassendiagramms

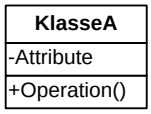
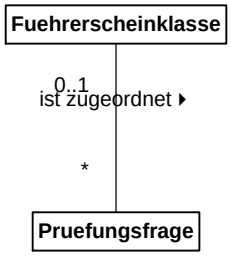
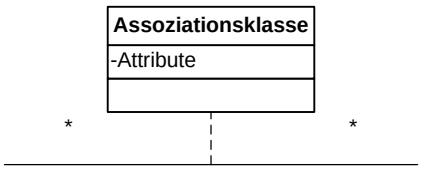
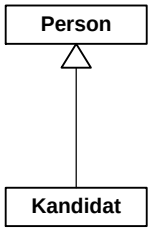
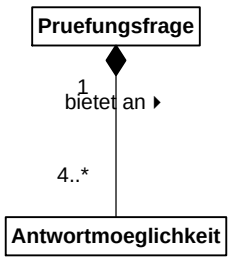
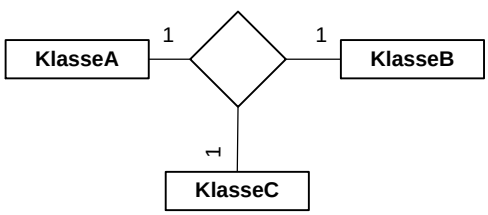
Symbol	Name	Englisch
	Klasse	Class
	Assoziation	Association
	Assoziationsklasse	Associationclass
	Vererbung/ Generalisierung	Generalization
	Aggregation /Komposition	Aggregation / Composition
	n-äre Beziehung	n-ary Relation

Abbildung 2-15 zeigt eine Auflistung der für den Datenbankentwurf wichtigsten Elemente eines Klassendiagramms. Nicht alle der Elemente werden für den Entwurf von Datenbank-Schemata benötigt.

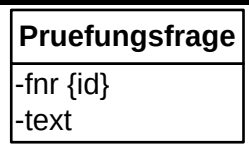
ER-Diagramme können auch in UML dargestellt werden. Während klassische E-R-Diagramme lediglich die Daten in den Mittelpunkt stellen, gehen Klassendiagramme einen Schritt weiter, indem sie auch das Modellieren von Verhaltensweisen zulassen. In der physischen Datenbank werden diese logischen Operationen im Allgemeinen in Trigger oder gespeicherte Prozeduren umgewandelt (Booch, Rumbaugh, & Jacobson, 1999).

Klasse

Eine Klasse wird als Rechteck mit drei Teilbereichen dargestellt. Im obersten Bereich befindet sich der Klassenname, der eindeutig sein muss. In der Mitte werden die Attribute der Klasse aufgelistet, und als letzter Bereich werden die Operationen (Methoden) der Klasse angeführt. Bis auf den Klassennamen sind alle Bereiche optional. Da die Operationen immer mit der Argumenteklammer „()" angeführt werden, können sie leicht von den Attributen unterschieden werden. Für den Entwurf von konzeptuellen Datenbank Schemata werden Operationen außer bei berechneten Attributen nicht benötigt. Sie werden deshalb in der Darstellung oft weggelassen. Jede Klasse hat beliebig viele Attribute. Attribute beschreiben die Eigenschaften der Klasse. Klassen sind abstrakte Beschreibung eines „Dinges" aus der abzubildenden Realität.
Abstrakte Klassen sind Klassen, von denen keine Instanz erzeugt werden darf. Diese Klassen dienen dazu, Schnittstellen für abgeleitete Klassen vorzugeben. Parametrisierte Klassen sind vor allem im objektorientierten Entwurf für Software Systeme interessant, im Datenbankentwurf werden sie jedoch nicht verwendet.

In UML besitzt jedes Objekt einen impliziten Identifier um es von anderen unterscheiden zu können. Dieser Identifier wird in der visuellen Notation nicht angegeben. Nun ist es aber so, dass gerade bei Datenbanken Objekte aufgrund von bestimmten Attribut-Ausprägungen identifiziert werden müssen. Aus diesem Grund kann an dieser Stelle ein Erweiterungsmechanismus der UML verwendet werden, um die Schlüssel für eine Klasse angeben zu können. Die Erweiterungsmechanismen (extensibility mechanisms) der UML erlauben eine kontrollierte Erweiterung der Sprache. Stereotypen (stereotype), Eigenschaftswerte (tagged values) und Einschränkungen (constraints) stehen dazu zur Verfügung (Booch, Rumbaugh, & Jacobson, 1999). Der eindeutige Schlüssel kann etwa durch einen Eigenschaftswert „id" eines Attributs angegeben werden.

Abbildung 2-16: Klasse mit Schlüssel



Eine Klasse kann über mehrere Schlüssel verfügen. Ein Schlüssel wird im Rahmen des Entwurfs als Primärschlüssel ausgewählt. Anstelle von „{id}" wird dann „{pk}" annotiert. Die Sichtbarkeit eines Attributs (public, protected, package, private) ist für den DB-Entwurf nicht von Bedeutung, da der Zugriff auf einzelne Attribute erst durch das Datenbank Management System geregelt wird. Auch die Datentypen eines Attributs sind für den konzeptuellen Entwurf nicht von Bedeutung, sie werden daher weggelassen.

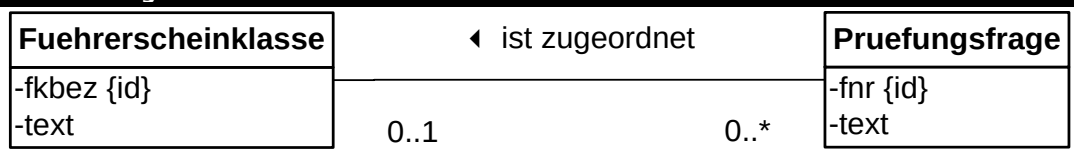
Assoziationen

Keine Klasse steht für sich alleine. In einem Klassendiagramm unterhalten Klassen Beziehungen zueinander. Diese Beziehungen

können unterschiedlicher Art sein. Eine binäre Assoziation (Abbildung 2-17) wird als Linie zwischen zwei Klassen dargestellt. Sind mehr als zwei Klassen an einer Beziehung beteiligt, so wird die höherwertige Assoziation durch eine Raute dargestellt. Von der Raute weg führen dann Linien zu den an der Assoziation beteiligten Klassen.

Assoziationen können an jedem Ende einen Rollennamen aufweisen. Dieser gibt an, in welchem Zusammenhang eine Klasse mit einer verbundenen Klasse steht. Auch die Beziehung selbst kann einen Namen haben. Um die Anzahl der erlaubten Verknüpfungen festzulegen werden an den Enden einer Assoziationslinie die Multiplizitäten aufgetragen.

Abbildung 2-17: Binäre Assoziation

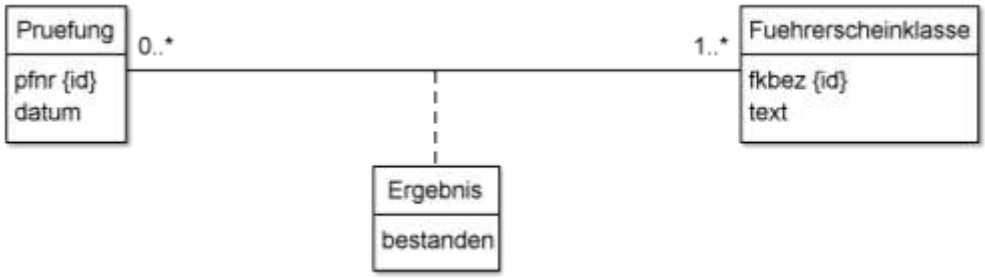


Die Multiplizitätsangabe ist eine Menge von Zahlen, die direkt angibt wie viele Verknüpfungen zulässig sind. Diese Angabe kann entweder ein Bereich (1..*) oder eine Aufzählung (1,3,5) sein. Für optionale Multiplizitäten wird der Wert „0" verwendet, und für beliebig viele das Symbol „*".

Assoziationsklasse

Wenn eine Assoziation selbst Eigenschaften (Attribute) hat, wird sie als Assoziationsklasse (Abbildung 2-18) modelliert. Eine Assoziationsklasse wird als Klasse dargestellt, die mit einer strichlierten Linie mit einer Assoziation verbunden ist. Die Assoziationsklasse kann selbst wieder Assoziationen haben.

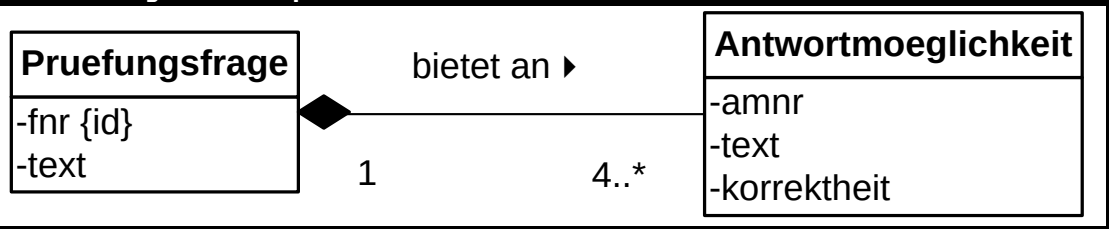
Abbildung 2-18: Assoziationsklasse



Aggregation/Komposition

Eine Aggregation ist eine besondere Form einer Beziehung zwischen Klassen. Sie drückt eine Teil-Ganzes Beziehung aus. Eine Aggregation wird als Linie mit einer Raute an dem Ende der Beziehung, die das „Ganze" repräsentiert, dargestellt. Immer dann wenn die „Teil"-Klasse ohne die „Ganzes"-Klasse nicht existieren kann, spricht man von einer Komposition. Bei einer Komposition (Abbildung 2-19) wird die Raute ausgefüllt.

Abbildung 2-19: Komposition

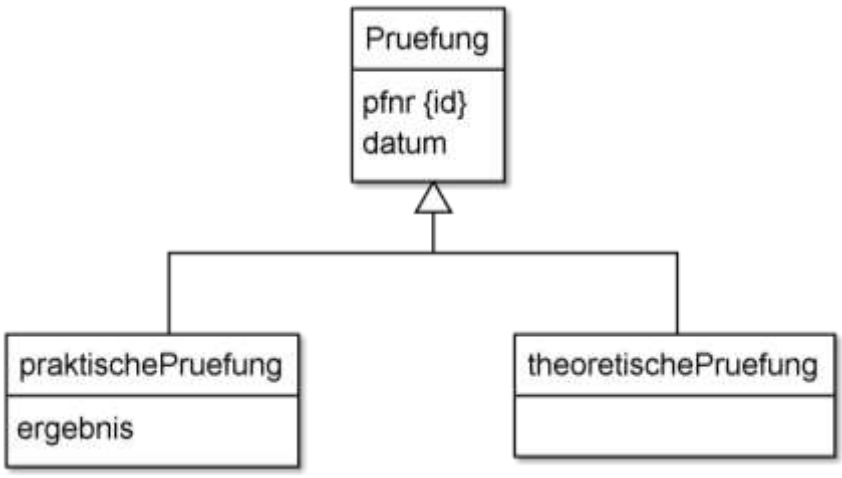


Generalisierung

Gibt es in einem System mehrere Klassen mit gleichen Eigenschaften, so können diese zu einer gemeinsamen Klasse generalisiert werden (Abbildung 2-20). Diese Superklasse enthält alle gemeinsamen Attribute der Klassen. Die Subklassen enthalten dann nur mehr die speziellen und ergänzenden Attribute. Die Subklassen können disjunkt oder überlappend sein. Eine Überlappung wird dabei mit {overlapping} angegeben. Eine Superklasse wird als abs-

trakt markiert, wenn es keine direkten Instanzen der Superklasse gibt. Das heißt eine Instanz einer Superklasse ist eine direkte Instanz einer Subklasse.

Abbildung 2-20: Generalisierung



3 Stand der Technik

Dieses Kapitel beschäftigt sich mit dem Stand der Technik im Bereich ITS für das Themengebiet „konzeptueller Datenbankentwurf". Dazu werden einige intelligente tutorielle Systeme, die sich mit dem Thema „konzeptueller Datenbankentwurf" beschäftigen, untersucht. Es werden die unterschiedlichen Ansätze zur Speicherung des Expertenwissens bzw. zur Beurteilung und Fehlererkennung besprochen. Die Anzahl an IST die den konzeptuellen DB-Entwurf unterstützen, ist noch sehr klein. Die bestehenden Systeme sind fast ausschließlich im Umfeld der akademischen Lehre angesiedelt. Es werden Kriterien erarbeitet, anhand derer die verschiedenen Systeme klassifiziert werden können. Diese Kriterien sind so gewählt, dass sie die Qualität des gesamten Systems, also nicht nur des Expertenmoduls erkennen lassen. Anhand dieser Kriterien wird auch das in Kapitel 5 vorgestellte Konzept zur Entwicklung des UML-Expertenmoduls für das E-Tutor System bewertet.

3.1 Ansätze zur Problemlösung

Bei den untersuchten Systemen haben sich zwei unterschiedliche Ansätze gezeigt. Es handelt sich dabei um die virtuelle Lernumgebung (Virtual Learning Environment), und um den sogenannten „Constraint based Modelling" Ansatz. Diese beiden Ansätze werden in den folgenden Absätzen näher beschrieben.

3.1.1 Virtual Learning Environment

In virtuellen Lernumgebungen (VLE) wird versucht, die Lernumgebung der realen Welt nachzubilden. Es wird versucht, die Rolle des Lehrenden bzw. Tutors durch ein Computerprogramm nachzubilden. Der Lernende kann aus einer Reihe von Aufgaben auswählen und diese bearbeiten. Die Bewertung der Lösungen erfolgt dabei

über intelligente Agenten. Diese geben auch Hilfestellungen und leiten so den Lernenden zum Erfolg (Gordon & Hall, 2000).
Eigene Zugänge für Lehrende ermöglichen die Wartung des Systems. Über Benutzungsschnittstellen stellt der Lehrende die Aufgaben und Lerninhalte zur Verfügung. Wegen der geforderten Orts- und Zeitunabhängigkeit von elektronischen Lernsystemen, benötigen VLE eine geeignete Infrastruktur. Das Internet stellt eine geeignete Infrastruktur zur Verfügung, weshalb Virtuelle Lernumgebungen als Web-Anwendungen entwickelt werden. Um am Lehrbetrieb teilzunehmen wird lediglich ein Web-Browser benötigt.
Ein weiteres Merkmal virtueller Lernumgebungen ist die Möglichkeit der gegenseitigen Hilfe von Studenten untereinander. Dazu wird über einen Chatbereich in der Benutzungsschnittstelle mit anderen Benutzern kommuniziert. Neben der Kommunikationsfunktion mittels Chat, werden auch Foren angeboten, in denen über Lösungsstrategien diskutiert werden kann. Diese Form von VLE wird auch Collaborative Learning Environment genannt. In manchen Systemen wird auch kollaboratives Arbeiten unterstützt. Hierbei müssen alle Mitglieder einer Übungsgruppe gemeinsam an der Lösung mitarbeiten (Constantino-González & Daniel Suthers, 2000).

3.1.2 Constraint-Based Approach

Bereits 1992 beschrieb Stellan Ohlsson (Zakharov, Mitrovic, & Ohlsson, 2005) den Constraint-based modeling Ansatz. Einige Constraint based Tutoring Systeme wurden an der Universität von Canterbury von der dort ansässigen "Intelligent Computer Tutoring Group" entwickelt[12]. Dabei wird versucht, Wissen in Form von „Constraints" abzulegen. Constraints bestehen aus drei Komponenten. Die erste Komponente beschreibt die sogenannte „Relevance Condition", also ob der Constraint im gegebenen Kontext anwend-

[12] siehe http://www.cosc.canterbury.ac.nz/tanja.mitrovic/ictg.html

bar bzw. anzuwenden ist. Die zweite Komponente (Satisfaction Condition) beschreibt die Regel zur Überprüfung des Kontextes, in Form einer Bedingung die erfüllt sein muss. Bei der letzten Komponente handelt es sich um die „Feedback Message", also eine Meldung an den Lernenden zur Unterstützung bei der Fehlerbehebung und Fehlererkennung (Antonija Mitrovic, Martin, & Suraweera, 2007).
Das Metamodell eines Constraints kann folgendermaßen definiert werden (Mitrovic, Martin, & Suraweera, 2007):

```
If <relevance condition> is true, then <satisfaction condition>
had better also be true, otherwise something has gone wrong.
```

3.2 Kriterien zum Vergleich von ITS

Es werden einige Kriterien beschrieben, die zur Beurteilung bzw. zum Vergleich von Intelligenten Tutoriellen Systemen herangezogen werden können. Diese Kriterien werden bei jedem der im Folgenden besprochenen Systeme evaluiert und übersichtlich dargestellt. Die Kriterien ergeben sich aus den Anforderungen an intelligente Tutorielle Systeme (Lusti, 1992), sowie Anforderungen für die Unterstützung beim konzeptuellen DB-Entwurf.

Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt

Hier geht es darum, ob das System eine Speicherung des User-Verhaltens durchführt. Der Student soll nur solche Aufgaben bekommen, die seiner erreichten Qualifikation entsprechen. Macht ein Student beispielsweise oft den gleichen Fehler, so sollen ihm spezielle Aufgaben zugeteilt werden um eben diesen Fehler auszumerzen.

Werden Fragen zum Lerninhalt beantwortet

Aus Studentensicht ist es wünschenswert, dass das System in der Lage ist, Fragen zur gestellten Aufgabe zu beantworten. Hier gibt es das Problem, dass natürlich-sprachliche Eingaben verarbeitet werden müssen. Dies ist noch nicht gelungen, jedoch gibt es Systeme, bei denen eine Kommunikation (Chat-Funktion) zwischen Team oder mit einem Coach stattfinden kann. Die Rollen „Team" bzw. „Coach" kommen bei sogenannten „Kollaborativen" Systemen vor. Der Lehrende nimmt dabei die Rolle des „Coaches" ein und kann während einer Sitzung Hilfestellung anbieten. Gleichzeitig kann der Lernende auch andere Lernende (das Team) um Rat fragen, bzw. werden seine Lösungsvorschläge diskutiert.

Wird jede Lösung erklärt

Jede Lösung (auch Teillösung) sollte so erklärt werden, dass dadurch das Wissen und Verständnis erweitert werden kann. Erklärung als Bestandteil des Feedbacks ist wesentlich für den Lernerfolg.

Erfolgt Erklärung benutzerbezogen

Das System soll in der Lage sein, bei Antworten bzw. Erklärungen an den Studenten, dessen Lernfortschritt zu berücksichtigen. Ein fortgeschrittener Student benötigt weniger an Erklärung als ein Anfänger. Dazu ist es notwendig, dass die Lernfortschritte zu jedem Benutzer individuell gespeichert werden.

Sind natürlich-sprachliche Eingaben und Ausgaben möglich

Kann mit dem System in natürlicher Sprache kommuniziert werden. Dies erfolgt in der Regel bei Systemen mit integrierter Kommunikationskomponente, wie etwa Foren oder Chat. Der Einsatz

natürlicher Sprache erfordert auch die Teilnahme von natürlichen Personen am System, und zwar auf Seite des Lernenden und auf Seite des Lehrenden. Eine maschinelle Verarbeitung natürlicher Sprache, besonders in Hinblick auf Beantwortung von Fragen, ist mit den derzeitigen Systemen nicht möglich. Das Forschungsgebiet der Computerlinguistik[13] befasst sich mit diesen Problemen. Dazu müssen die verschieden Herausforderungen wie etwa Semantik, syntaktische Mehrdeutigkeiten und Pragmatik gelöst werden.

Welcher Ansatz wird verfolgt (CBM vs. VLE)

Die Konzepte der derzeitigen Lösungen beschränken sich auf die beiden Ansätze Constraint Based Modeling sowie auf Virtual Learning Environment. Dieses Kriterium dient nur zur Dokumentation, eine Bewertung erfolgt nicht.

Online vs. Offline Lösung

Bei einer Online-Lösung genügt in der Regel ein Web-Browser um die Lerninhalte zu bearbeiten. Bei einer Offline-Lösung erfolgt eine Installation am lokalen System. Eine Online-Lösung kann einfacher verwaltet und zur Verfügung gestellt werden. Daher werden Online-Lösungen höher bewertet als Offline-Lösungen.

Verfügbarkeit (Offener vs. geschlossener Benutzerkreis)

Systeme sollen durch eine große Anzahl von Benutzern auf ihre Praxistauglichkeit getestet werden. Daraus entstehen in der Regel auch stabilere Systeme, da enthaltene Fehler früher bemerkt werden. Aus diesem Grund sind offene Systeme mit großem Benutzer-

[13] Für eine Übersicht über das Fachgebiet: http://www.uni-due.de/computerlinguistik/c_und_l.shtml

kreis höher zu bewerten, als Systeme mit geschlossenem, kleinem Benutzerkreis.

Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)

Das Feedback soll in mehreren Detaillierungsstufen abrufbar sein. Nicht immer soll sofort die vollständige Lösung angezeigt werden. Das System soll auch die Möglichkeit bieten, weitere Detaillierungsstufen abzurufen.

Grafik-Editor für Modellierung

Wird von den einzelnen Systemen der DB-Entwurf mit einer grafischen Notation, wie etwa UML oder ER unterstützt. Hier geht es speziell um die Frage ob ein Modellierungswerkzeug in das System integriert ist. Um einen Medienbruch zu vermeiden, ist es wünschenswert wenn die Systeme Modellierungswerkzeuge einbinden.

Werden fehlende Kenntnisse unterrichtet

Es wird überprüft, ob das System auch über ein Tutorial verfügt um unvollständiges Wissen zum Problembereich zu ergänzen. Der Student sollte die Möglichkeit haben sein Wissen aufzufrischen bzw. neues Wissen aufzunehmen.

Wird das Verständnis getestet

Um der Gefahr eines „blinden" Lernens vorzubeugen, sollen die Systeme auch überprüfen, ob der Student die Lösung verstanden hat. Speziell zur Evaluierung von Systemen ist dieses Kriterium von entscheidender Bedeutung. In der Regel kann das nur über eine Interaktion direkt mit dem Studenten (z.B. Vertiefungsfragen) erfolgen.

Sind alternative Lösungen möglich

Werden vom System auch alternative Lösungen erkannt und zugelassen? Bei einigen Systemen wird die Studentenlösung mit einer Musterlösung verglichen. Gerade bei der Erstellung eines konzeptuellen Datenmodells gibt es aber sehr oft die Möglichkeit von alternativen Lösungen. Wenn diese nicht erkannt werden, kann eine nicht korrekte Bewertung der Studentenlösung resultieren.

Wird eigene Benennung der Elemente unterstützt

Die einzelnen Objekte (Elemente) des konzeptuellen Datenmodells werden über ihre Benennung identifiziert, für einen Vergleich der Studentenlösung mit der Musterlösung müssen die Namen der Objekte bzw. Elemente übereinstimmen oder es muss ein Algorithmus die vergleichbaren Objekte finden. Werden Namen vorgegeben, so wird oft schon ein Teil der Lösung vorweggenommen. Der erste Schritt beim Entwerfen des konzeptuellen Datenmodells besteht im Finden der Entitäten aus dem Text der Aufgabe. Aus den Hauptwörtern werden die Namen der Entitäten gebildet. Werden diese Entitätsnamen bereits vorgegeben, so entfällt ein wichtiger Lernschritt für den Lernenden.

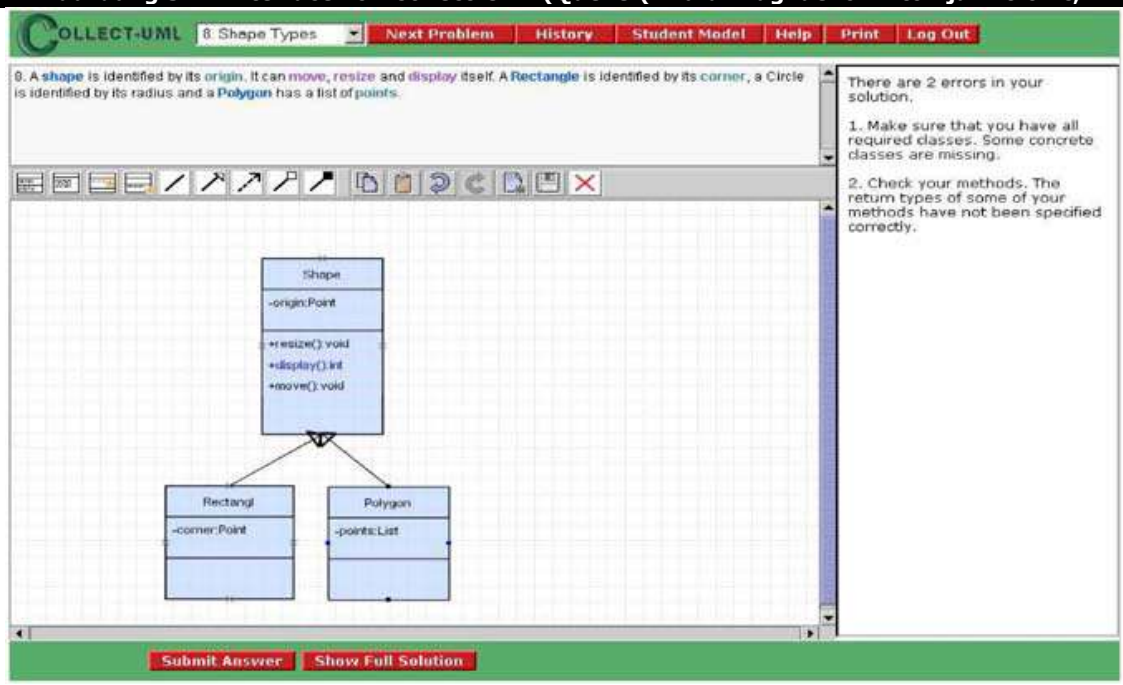
3.3 Bestehende ITS für den konzeptuellen Datenbankentwurf

In den folgenden Absätzen werden die bekanntesten ITS beschrieben. Die Evaluierung der Systeme erfolgte durch eine Dokumentenanalyse, da die Systeme nicht öffentlich zugänglich sind. Es handelt sich jedoch um Systeme die im akademischen Bereich angesiedelt sind, so dass die veröffentlichen Dokumente durchwegs wissenschaftlichen Charakter haben.

3.3.1 COLLECT-UML

COLLECT-UML wurde 2006 von Baghaei, Mitrovic und Irwin vorgestellt. Es handelt sich dabei um ein Tutoring System zum Erlernen der objektorientierten Modellierung mit der UML. Zwar wurde dieses System nicht primär für das Erlernen des konzeptuellen Datenbankentwurfs gebaut, aber es ist durch die Bedeutung des UML-Klassendiagramms für den Datenbankentwurf durchwegs von Interesse.

Abbildung 3-1: Interface von Collect-UML (Quelle:(Nilufar Baghaei & Antonija Mitrovic,



Es verwendet „Constraint based modeling" Technologien für das Expertenmodul. Die Constraints werden dabei in syntaktische und semantische unterteilt (N. Baghaei, A. Mitrovic, & Irwin, 2006).
COLLECT-UML verfügt über keine Problemlösungskomponente. Das System arbeitet mit einer Ideal-Lösung für jede Aufgabe, die mit der Studentenlösung verglichen wird. Der Vergleich wird über 88 semantische und 45 syntaktische Constraints gesteuert. Es werden keine alternativen Lösungen gespeichert, jedoch verfügt das Sys-

tem über Algorithmen, die alternative Lösungen erkennen sollen (Nilufar Baghaei & Antonija Mitrovic, 2005).

COLLECT-UML ist eine Web-Anwendung und nur für Studenten an der Universität Canterbury, New Zealand vorgesehen.

Kriterium		Begründung
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	😐	Bei Collect-UML werden die Vorkenntnisse eines Studenten nicht direkt berücksichtigt, es gibt jedoch ein Studentenmodul das eine Übersicht über die gelösten bzw. noch nicht gelösten Aufgaben speichert. Auch die fehlerhaften Lösungen werden vermerkt.
Werden Fragen zum Lerninhalt beantwortet	☹	Das System gibt nur einen Hinweis auf verletzte Constraints, es können keine Fragen an das System gestellt werden.
Wird jede Lösung erklärt	☹	Es gibt nur eine Erklärung zu den verletzten Constraints, nicht jedoch eine Erklärung der Lösung.
Erfolgt Erklärung benutzerbezogen	☹	Es werden bei Verletzung eines Constraints immer die gleichen Rückmeldungen pro Feedbackstufe ausgegeben.
Sind natürlich sprachliche Eingaben und Ausgaben möglich	☹	Nein
Welcher Ansatz wird verfolgt (CBM vs. VLE)	CBM	Das System verwendet den Constraint based modeling Ansatz. Es gibt keine Problemlösungskomponente. Das Problemwissen bzw. die Ideallösung sind durch Constraints abgebildet.
Online vs. Offline Lösung	☺	Collect-UML ist eine Web-basierte Online Lösung.
Verfügbarkeit (Offen vs. Geschlossener	😐	Collect-UML ist nur für Studenten an der Universität Canterbury in Neu Seeland verfügbar. Eine „Pilot Study wurde im

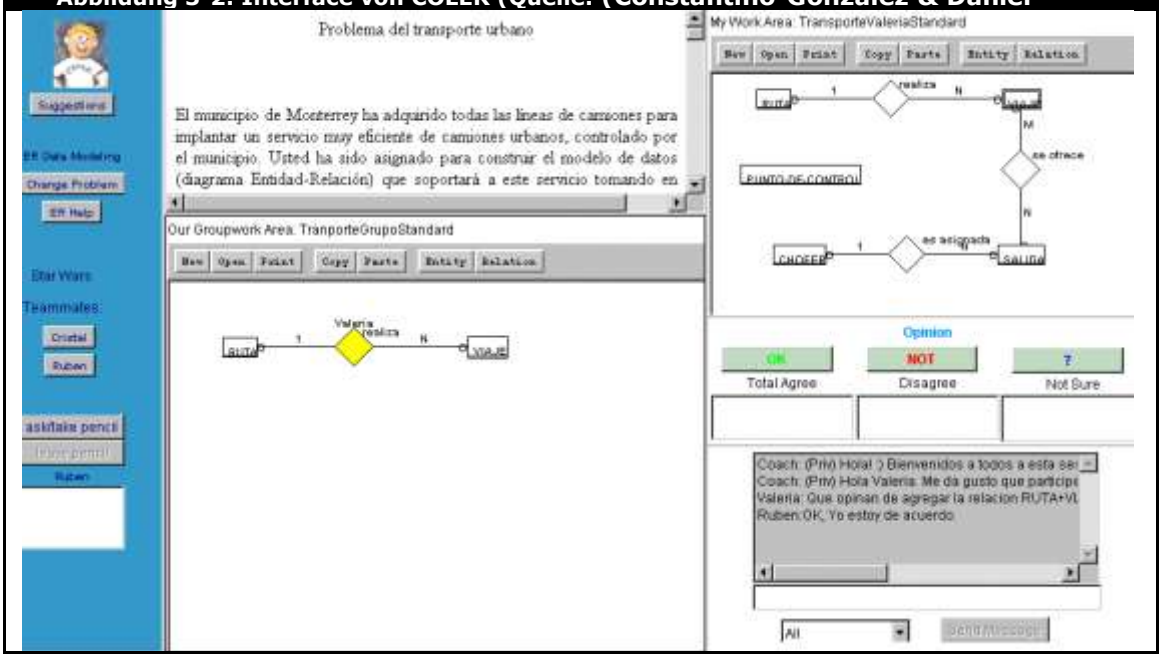
Benutzerkreis)		März 2005 durchgeführt.
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☺	Die Rückmeldungen können vom Studenten in 5 Stufen abgerufen werden. Diese reichen von Simple Feedback, wo nur ein Hinweis auf Korrektheit erfolgt bis hin zur vollständigen Lösung. Bei der Hinweisstufe „Hint" wird beim ersten verletzten Constraint begonnen, während bei „All Hints" eine Liste aller verletzten Constraints angezeigt wird.
Grafik-Editor für Modellierung	☺	Das System verfügt über einen Arbeitsbereich mit Grafik Editor. Alle benötigten Modellelemente können über eine Buttonleiste in den Arbeitsbereich übernommen werden. Somit kann ein komplettes Modell erstellt werden.
Werden fehlende Kenntnisse unterrichtet	☹	Es gibt kein eigenes Lernmodul, in dem fehlende Kenntnisse nachgeschlagen oder abgerufen werden können.
Wird das Verständnis getestet	☺	Es wird jede Verletzung eines Constraints vermerkt. Da in den Constraints das Wissen zur Modellierung vermerkt ist, kann über das Erkennen von bestimmten Fehlermustern auf Modellierungsfehler und somit auch auf Verständnis geprüft werden. Im Studentenmodul wird das gemessene Verständnis in Form einer Übersichtsgrafik angezeigt.
Sind alternative Lösungen möglich	☹	Nein. Die Constraints werden auf eine „Ideal Solution" angewendet. Diese idealen Lösungen beziehen sich jeweils auf das Problem und das Constraint. Derzeit werden 14 Probleme im System verwaltet. Im Text der Aufgabenstellung werden Tags eingefügt, die die Verbindung zwischen Constraint und Ideallösung herzustellen. Diese Tags sind für die Studenten nicht ersichtlich.
Wird eigene	😐	Die Benennung der Elemente muss aus

Benennung der Elemente unterstützt	dem Aufgabentext erfolgen. Dazu muss durch Markieren eine Verbindung des Modellelements mit dem Aufgabenkontext hergestellt werden.

3.3.2 COLER

COLER (**Co**llaborative **L**earning environment for **E**ntity **R**elationship modeling) wurde 2000 von Constantino-González und Suthers vorgestellt. Es handelt sich um eine virtuelle Lernumgebung für das Erlernen des konzeptuellen Datenbankentwurfs mit Hilfe der Entity Relationship Modellierung. Neben der virtuellen Lernumgebung verfügt dieses System auch über ein System zum kollaborativen Lernen. Die Aufgabe muss von einer Gruppe von Studenten gemeinsam gelöst werden.

Abbildung 3-2: Interface von COLER (Quelle: (Constantino-González & Daniel



Studenten beginnen mit der Erstellung eines Diagrammes zuerst allein, später in kleinen Gruppen um eine gemeinsame Lösung zu erarbeiten (Collaborative work). Die Lösung wird von einem Soft-

ware-Coach mit der Gruppenlösung verglichen (Constantino-Gonzalez, D. D Suthers, & de los Santos, 2003). Jede Änderung an der Gruppenlösung wird angezeigt, und der Student bekommt die Möglichkeit seine Bewertung zu dieser Änderung abzugeben (Zustimmung, keine Zustimmung, Unentschlossen).Gleichzeitig wird auch die Partizipation der einzelnen Studenten an der gemeinsamen Problemlösung gemessen. Fällt diese unter eine bestimmte Grenze, so wird der Student dazu aufgefordert sich mehr zu beteiligen. Auch das entgegengesetzte Verhalten, das heißt wenn sich ein Student zu sehr beteiligt, wird kontrolliert.
Weicht nun eine Studentenlösung von der Gruppenlösung ab, wird vom Software-Coach ein Feedback generiert. Das Feedback wird mit Hilfe eines Entscheidungsbaums generiert (Constantino-González & Daniel Suthers, 2000).

Kriterium		Begründung
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	☹	Es wird keine Information über die Lernfortschritte gespeichert.
Werden Fragen zum Lerninhalt beantwortet	😐	Fragen zum Lerninhalt werden durch das System nicht beantwortet. Es besteht jedoch die Möglichkeit der Beantwortung durch einen Coach.
Wird jede Lösung erklärt	😐	Es besteht die Möglichkeit zu diskutieren, und die Lösung eventuell durch einen Coach erklären zu lassen. Studenten haben überdies die Möglichkeit bei Unklarheiten nachzufragen.
Erfolgt Erklärung benutzerbezogen	😐	Erklärungen erfolgen nicht durch das System, sondern nur durch aktives Nachfragen an die Gruppe oder den Coach.
Sind natürlich	😐	Das System verarbeitet keine natürlichsprachliche Eingaben. Ausgaben werden

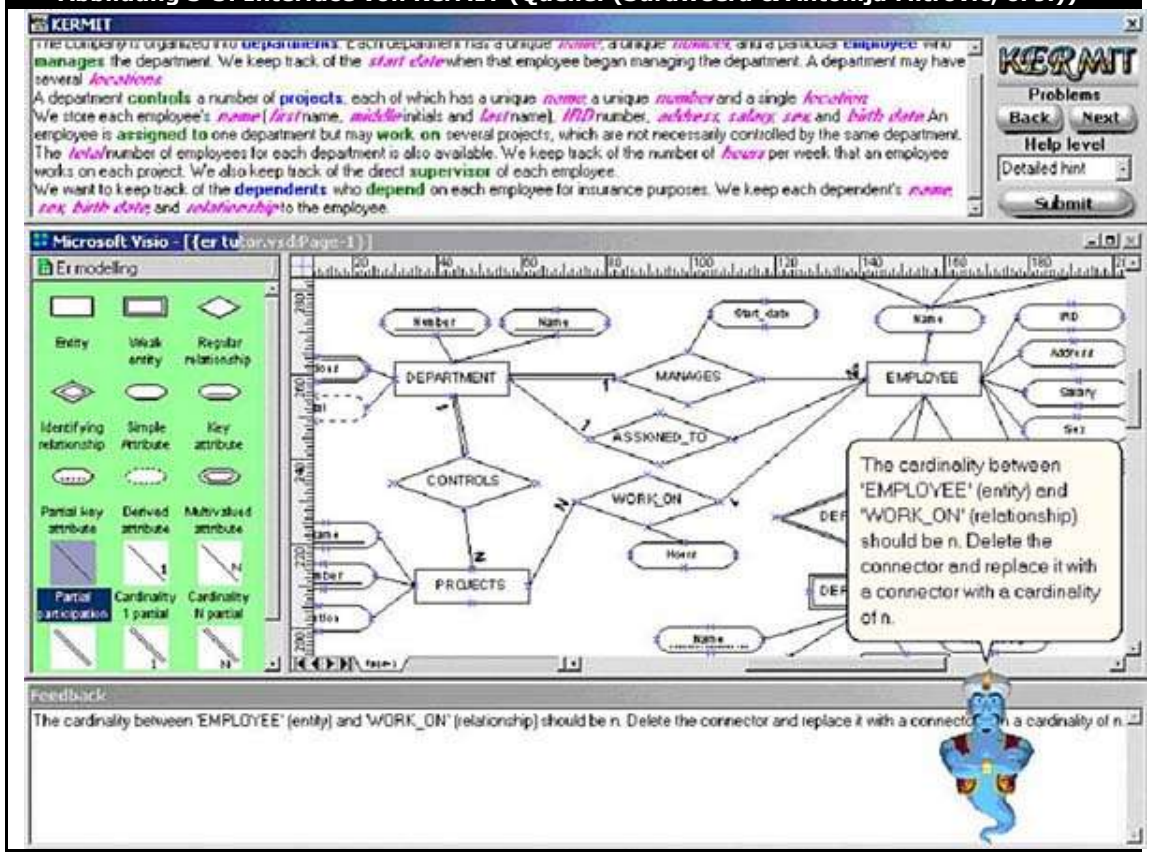
sprachliche Eingaben und Ausgaben möglich		nicht vom System erstellt. Es besteht jedoch die Möglichkeit der natürlichsprachlichen Kommunikation über das Chat-Modul
Welcher Ansatz wird verfolgt (CBM vs. VLE)	VLE	Das System ist als Coached Collaborative Learning Environment entwickelt worden. Neben der virtuellen Lernumgebung verfügt es auch noch über kollaborative Elemente um den Lernerfolg auch im Gruppenumfeld zu stärken.
Online vs. Offline Lösung	☺	COLER ist ein Web-basiertes, onlinefähiges System. Es basiert auf Java 1.1 Applets, für die verschiedenen Funktionalitäten des Systems, wie etwa Chat oder ER Modellierung. Die Module kommunizieren über einen Objekt Broker Mechanismus mit einer Datenbank, um Änderungen zu verfolgen. Damit kann sofort nach einer Änderung die entsprechenden Gruppe, bzw. der Coach informiert werden.
Verfügbarkeit (Offen vs. Geschlossener Benutzerkreis)	☹	COLER ist nicht öffentlich verfügbar. COLER wurde im Rahmen einer Forschungsarbeit für kollaborative Lernumgebungen entwickelt. Die daraus gewonnenen Erkenntnisse sollen in eine Weiterentwicklung einfließen.
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☹	Nein. Erklärungen können jedoch noch mit Erklärungen eines Coaches versehen werden.
Grafik-Editor für Modellierung	☺	Das System verfügt über einen Editor zur Erstellung von ER Diagrammen. Die Gruppenlösung wird in einem eigenen Grafikfenster angezeigt.
Werden fehlende Kenntnisse unterrichtet	☺	Studenten können über eine „Help" Schaltfläche, Hilfe zur ER Modellierung abrufen. Falls ein Coach im System an-

		gemeldet ist, können von diesem eingegebene Hinweise abgerufen werden.
Wird das Verständnis getestet	☺	Studenten erstellen zuerst ihr persönliches Modell, anschließend wird dieses mit einem Gruppenmodell verglichen. Jede Abweichung muss begründet werden. Damit wird der Student gezwungen seine Entscheidung zu überdenken. Allerdings besteht hier auch die Gefahr einer kollektiven falschen Modellierung, wenn etwa durch den Gruppendruck sich die Gruppenmitglieder zu einer falschen Lösung entscheiden.
Sind alternative Lösungen möglich	☺	Durch die freie Gestaltung der Lösung durch die Gruppe ist es möglich verschiedene Varianten bzw. alternative Lösungen zu erstellen. Die erstellte Lösung muss aber nicht zwangsläufig die beste sein.
Wird eigene Benennung der Elemente unterstützt	☺	Es gibt keine vorgegebenen Namen für die Modellelemente. Studenten in der Gruppe müssen sich gemeinsam auf die Benennung einigen.

3.3.3 Kermit

Kermit (Abbildung 3-3) wurde 2002 von Suraweera und Mitrovic vorgestellt. Kermit ist eine stand-alone Lösung. Das Programm verfügt, ebenso wie COLLECT-UML, über keine Problemlösungskomponente. Das Wissen wird in Form von 92 Constraints gespeichert.
Diese werden dazu verwendet das Studentenmodell zu bewerten. Als Arbeitsoberfläche wird Microsoft Visio® eingebettet. Sobald ein neues Element eingefügt wird, gibt das System im Feedback-Fenster entsprechende Meldungen aus. Somit kann der Student schrittweise sein Modell verbessern.

Abbildung 3-3: Interface von KerMIT (Quelle: (Suraweera & Antonija Mitrovic, o. J.))



Eine Evaluierung an der Universität von Canterbury 2001 zeigte, dass Kermit im Vergleich zu einer herkömmlichen Klassensituation effektiver ist. Die Ergebnisse der Studenten die Kermit verwendeten war signifikant besser als die Gruppe der Studenten die ein herkömmliches Modellierungstool verwendeten (Suraweera & Antonija Mitrovic, o. J.).

Kriterium		Begründung
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	☺	Durch das Studentenmodul wird vermerkt, welche Constraints während des Lernens besonders oft verletzt wurden. Das System versucht daraufhin nur solche Übungen auszuwählen die helfen diese Fehler auszubessern.
Werden Fragen zum Lerninhalt	☹	Es können keine Fragen an das System gestellt werden.

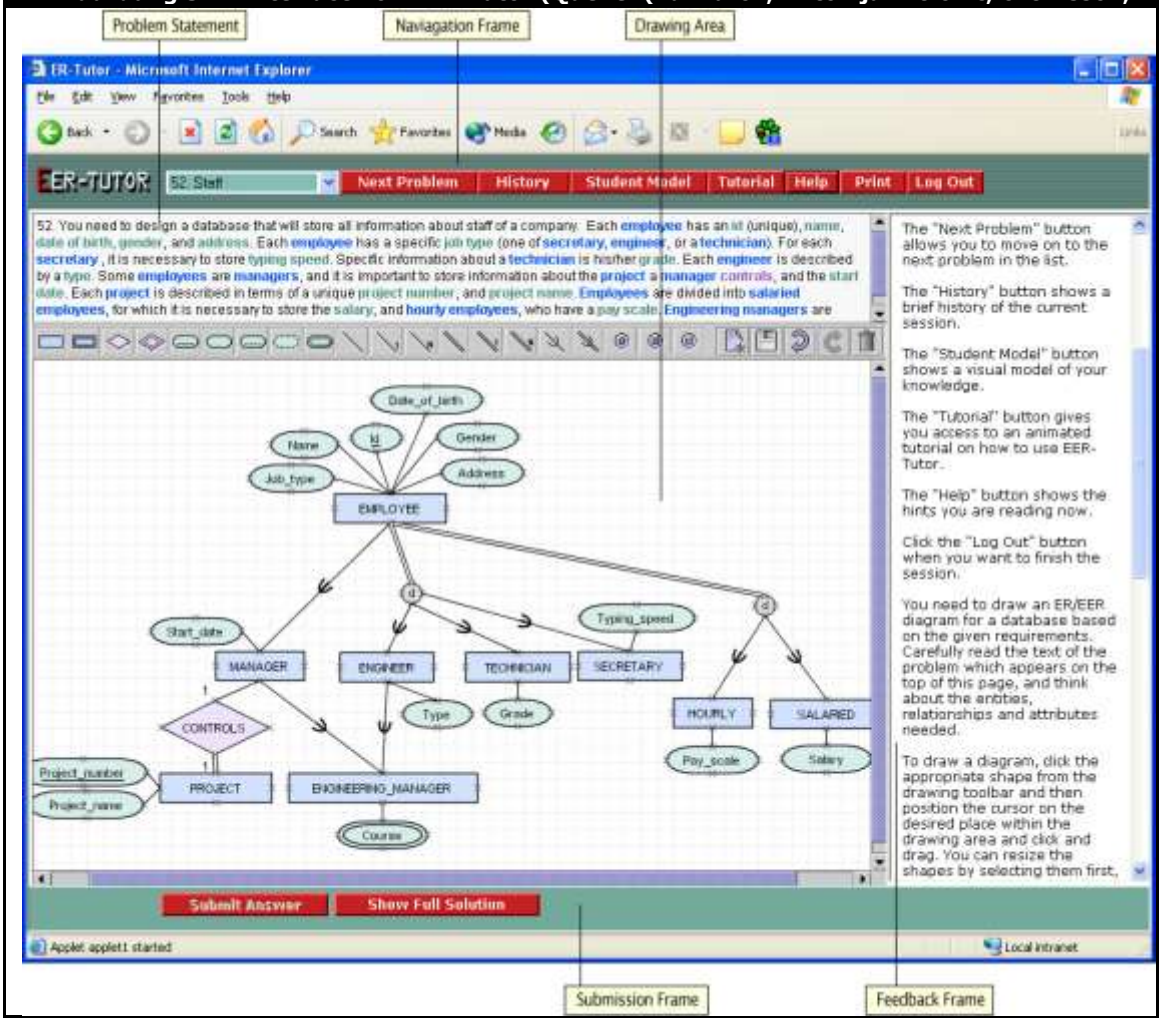
beantwortet		
Wird jede Lösung erklärt	😐	Es wird nach jedem Modellierungsschritt überprüft, ob Constraints verletzt worden sind. Dadurch wird der Student zur „richtigen" Lösung geführt. Jeder verletzte Constraint wird angezeigt und entsprechend der ausgewählten Feedbackstufe erklärt. Die Erklärung erfolgt jedoch allgemein und nicht fallspezifisch.
Erfolgt Erklärung benutzerbezogen	😐	Einerseits werden verletzte Constraints nur allgemein erklärt, andererseits wird nicht auf den Benutzer speziell eingegangen. Einzig die gewählte Feedbacktiefe wird berücksichtigt.
Sind natürlich sprachliche Eingaben und Ausgaben möglich	☹	Es können keine Anfragen an das System gestellt werden.
Welcher Ansatz wird verfolgt (CBM vs. VLE)	CBM	KERMIT verwendet den Constraint based Modeling Ansatz. Derzeit verfügt das System über 92 Constraints.
Online vs. Offline Lösung	☹	KERMIT ist eine Offline Lösung, die als Standalone Applikation entwickelt wurde.
Verfügbarkeit (Offen vs. Geschlossener Benutzerkreis)	😐	KERMIT ist nur für eine geschlossene Benutzergruppe an der Universität von Canterbury, Neu Seeland zugänglich.
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☺	Das System verfügt über mehrere frei wählbare Hinweisstufen. Diese reichen von einem allgemeinen Errorflag bis hin zur detaillierten Erklärung mit Benennung der Modellelemente.
Grafik-Editor für Modellierung	☺	In das System wurde Microsoft Visio ® eingebunden. Damit können ER Diagramme erstellt werden.

Werden fehlende Kenntnisse unterrichtet	☹	Es gibt kein Tutorial in dem die fehlenden Kenntnisse nachgeschlagen werden können. Auch Hilfsfunktionen werden nicht angeboten.
Wird das Verständnis getestet	☺	Durch das Speichern häufiger Fehler und auch durch die Auswahl der Aufgaben in Hinblick auf die gemachten Fehler wird das Verständnis bzw. der Lernfortschritt getestet.
Sind alternative Lösungen möglich	😐	Die Constraints prüfen jede Benutzereingabe gegen ein Idealmodell. Durch die Prüfung nach jeder Eingabe wird der Student an das Idealmodell herangeführt, ohne der Möglichkeit abweichende und ebenfalls richtige Lösungen zu erstellen. Das System ist jedoch in der Lage, äquivalente Lösungen zu erkennen und ist deswegen etwas flexibler als etwa COLLECT_UML.
Wird eigene Benennung der Elemente unterstützt	☺	Es muss eine Beziehung zum Text hergestellt werden. Durch Markieren des Textes in der Aufgabenstellung wird ein Bezug zwischen Modellelement und Kontext hergestellt.

3.3.4 EER-Tutor

EER-Tutor (Abbildung 3-4) ist eine Weiterentwicklung von Kermit. Obwohl die grundlegenden Funktionalitäten von Kermit kommen, wurde das System neu entwickelt um eine Web-fähige Benutzung zu ermöglichen (Zakharov, Mitrovic, & Ohlsson, 2005).

Abbildung 3-4: Interface von EER-Tutor (Quelle: (Zakharov, Antonija Mitrovic, & Ohlsson,



In der Übersicht der Kriterien werden deshalb nur jene Kriterien besprochen, die von KERMIT abweichen.

Kriterium		Begründung
Wird jede Lösung erklärt	☺	Die Menge der Constraints wurde um solche für Konzepte der ER Modellierung erweitert. Damit ist es möglich die einzelnen Lösungen besser zu erklären und auch speziell auf Modellierungsfehler einzugehen.
Online vs. Offline Lösung	☺	Das System ist als Web-basierte Online Lösung neu konzipiert worden. Dabei wurden die Konzepte von KERMIT übernommen und erweitert.

Grafik-Editor für Modellierung	☺	Das System verfügt jetzt über einen eigenen Grafikeditor zur Erstellung von ER Diagrammen. Die Modelldaten werden in einer XML-Datei gespeichert.

3.3.5 Vergleich der beschriebenen ITS

Abbildung 3-5 zeigt eine Tabelle mit den vorher besprochenen Kriterien für die Systeme.

Abbildung 3-5: Übersicht der Systeme

Kriterium	COLLECT-UML	COLER	KERMIT	EER-Tutor
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	😐	☹	☺	☺
Werden Fragen zum Lerninhalt beantwortet	☹	😐	☹	☹
Wird jede Lösung erklärt	☹	😐	😐	☺
Erfolgt Erklärung benutzerbezogen	☹	😐	😐	😐
Sind natürlich sprachliche Eingaben und Ausgaben möglich	☹	😐	☹	☹
Welcher Ansatz wird verfolgt (CBM vs. VLE)	CBM	VLE	CBM	CBM
Online vs. Offline Lösung	☺	☺	☹	☺
Verfügbarkeit (Offen vs. Geschlossener Benutzerkreis)	😐	☹	😐	😐
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☺	☹	☺	☺
Grafik-Editor für Modellierung	☺	☺	☺	☺
Werden fehlende Kenntnisse unterrichtet	☹	☺	☹	☹
Wird das Verständnis getestet	☺	☺	☺	☺
Sind alternative Lösungen möglich	☹	☺	😐	😐
Wird eigene Benennung der Elemente unterstützt	😐	☺	☺	☺

Symbol	Bedeutung
☺	Kriterium wird gut unterstützt, Unterstützung vorhanden und einwandfrei umgesetzt
😐	Unterstützung ist vorhanden, jedoch keine vollständige Umsetzung oder Mängel in der Umsetzung
☹	Kriterium wird nicht unterstützt

In diesem Kapitel wurden verschiedene Intelligente Tutorielle Systeme für den Bereich des konzeptuellen Datenbankentwurfs vorge-

stellt. Alle Systeme verfügen über einen integrierten Grafikeditor zur Erstellung von Modellen (ER bzw. UML). Die Systeme verfolgen zwei grundsätzlich verschiedene Ansätze. Beim ersten Ansatz (CBM) wird das „Wissen" in Form von Constraints abgelegt. Constraints sind Regeln die über die Komponenten Relevanz, Bedingung sowie Feedback verfügen. Die Studenten können frei modellieren, solange sie nicht eine Regel (Constraint) verletzen. Der zweite Ansatz ist die virtuelle Lernumgebung, hier wird das „Wissen" in Form intelligenter Agenten gespeichert. Diese vergleichen die Studentenlösung mit einer Musterlösung. Hier tritt ganz speziell das Problem auf, dass die Benennung der Objekte gleich sein muss um einen Vergleich durchführen zu können. Wenn die Studenten für die Benennung ihrer Objekte aus einer vorgegebenen Namensliste auswählen, wird oft schon ein wesentlicher Teil der Modellierung vorweggenommen. Deshalb versucht etwa Kermit mit einem Hervorheben im Text eine Verbindung der vom Studenten gewählten Namen mit dem Kontext zu ermöglichen.
Ein Sonderfall ist die kollaborative Lernumgebung COLER, hier wird die Studentenlösung mit einer Teamlösung verglichen. Diese Lösung birgt jedoch die Gefahr, dass eine falsche Lösung präferiert wird, weil die Mehrheit der Gruppe diese für richtig hält. Dieses System ist darauf angewiesen, dass ein Coach dafür sorgt falsche Ergebnisse zu vermeiden. Ein Coach bietet überdies die Möglichkeit der Beantwortung von Studentenfragen zum Lerninhalt. Allgemeine Probleme wie die freie Benennung der Objekte oder Modellierungsvarianten sind noch nicht vollständig gelöst.

Die Bewertung der untersuchten Systeme erfolgte nur aufgrund von Beschreibungen in wissenschaftlichen Dokumenten. Die Implementierungsdetails dieser Systeme sind dadurch jedoch nicht ersichtlich. Da diese Systeme in der Lehre der jeweiligen Universitäten eingesetzt werden, stehen sie auch nur einem eingeschränkten

Benutzerkreis zur Verfügung. Für diese Arbeit sind die Erkenntnisse deshalb auch nicht direkt anwendbar.

Beim CBM Ansatz nur das Prinzip beschrieben. Die konkreten Constraints werden nicht aufgezählt. Diese müssten aufwändig selbst erstellt werden, wobei der Aufwand als sehr hoch einzustufen ist. Weiters fehlt die Möglichkeit Modellierungsvarianten im Expertenwissen zu speichern. Die Vergabe von individuellen Punkteabzügen bei Fehlern ist mit dem CBM Ansatz ebenfalls nicht möglich.

Virtuelle Lernumgebungen arbeiten mit sogenannten Agenten, welche auch das Expertenwissen verwalten. Das einzige VLE System bei den betrachteten Systemen, COLER, benötigt einen Coach der auf eventuelle falsche Modellierungsvarianten eingeht. Wie beim CBM ist auch bei diesem Ansatz keine individuelle Punktevergabe bzw. Punkteabzug möglich. Für die Aufgabenstellungen im Rahmen dieser Arbeit müssen deshalb eigene Konzepte entwickelt werden.

3.4 Werkzeuge zur Erstellung von UML-Modellen

Zur Erstellung von Muster- und auch Studentenmodellen muss ein UML-Editor verwendet werden. Die Integration eines UML-Editors in das UML-Expertenmodul wird aufgrund des erwarteten hohen Aufwands derzeit nicht angestrebt. Es gibt einige frei verfügbare (Open Source bzw. Freeware) UML-Editoren. Um im UML-Expertenmodul verwendet werden zu können muss der UML-Editor zumindest die Erstellung von Klassendiagrammen unterstützen verfügen, sowie den Export dieser Modelle als XMI-Datei. Die im XMI-Format abgespeicherten Modelle sind die Eingabedaten für die Weiterverarbeitung durch das UML-Expertenmodul.

Abbildung 3-6: Liste frei verfügbarer UML-Editoren			
Tool	Hersteller	Lizenz	XMI
ArgoUML	University of California	Open-Source-Projekt	J
Dia	Alexander Larsson	GPL	N
EclipseUML	Omondo	Plugin für Eclipse	J (2.1)
Essmodel	SourceForge Projekt	Open-Source	J
Fujaba	Uni Paderborn	GPL	N
gModeler	G. Skinner		N
Objecteering/UML	Objecteering Software	Personal Edition kostenfrei	J
Together	Borland	Whiteboard Edition kostenfrei	J
UMLet	Martin Auer, Thomas Tschurtschenthaler	Open Source	N

Abbildung 3-6 zeigt eine Liste frei verfügbarer UML-Editoren mit Unterstützung für UML-Klassendiagramme. Die Unterstützung für XMI zeigte sich bei einigen Werkzeugen erst nach einer testweisen Installation bzw. eines Tests. Deshalb werden die wesentlichen Erfahrungen mit den Werkzeugen hier kurz dargestellt:

ArgoUML

ArgoUML bietet sowohl Unterstützung für UML-Klassendiagramme, als auch für den XMI-Export in der Version 1.3. Modelle können exportiert und auch importiert werden. Beim XMI-Import werden alle Modellelemente geladen. ArgoUML benötigt eine Java Laufzeitumgebung.

Dia

Dia verfügt über keine Möglichkeit die erstellten Modelle im Format XMI abzuspeichern. Dia ist in erster Linie ein Zeichenprogramm. Die Unterstützung für UML wurde erst später hinzugefügt. Diagramme können in einer Vielzahl gängiger Grafikformate abgespeichert werden.

EclipseUML

EclipseUML ist ein Eclipse-Plugin und benötigt deshalb Eclipse als Laufzeitumgebung. Dieses Werkzeug ist in erster Linie für das Forward Engineering gedacht. Bei jeder Änderung am Modell, werden auch die entsprechenden Änderungen in den korrespondierenden Java-Klassen nachgezogen. Die Diagramme werden direkt im Format XMI 2.1 abgespeichert. Ein Export bzw. eine Konvertierung ist daher nicht erforderlich.

Essmodel

Essmodel ist ein Open-Source Projekt, und bietet nur Reverse Engineering, als nur das Erstellen von UML Diagrammen aufgrund bestehender Klassen in zum Beispiel Java, eine direkte Erstellung von UML-Diagrammen ist nicht vorgesehen. Die Modelle können im Format XMI Version 1.0 exportiert werden.

Fujaba

Fujaba bietet derzeit noch keine Unterstützung fürden XMI Export an. Eine Unterstützung ist jedoch geplant. Fujaba dient in erster Linie zur Erstellung von Java Code aus den erstellten Modellen (Forward Engineering).

gModeler

gModeler ist eine Web-Anwendung. Es wurde mit FlashMX entwickelt, und bietet nur keine vollständige UML Unterstützung. Modelle können als XML Exportiert werden. Eine Unterstützung für XMI fehlt jedoch.

Objecteering/UML

Objecteering/UML ist ein kommerzielles Produkt, das auch in einer Personal Edition verfügbar ist. Die Personal Edition ist für nicht kommerzielle Verwendung frei verfügbar. Die Unterstützung für XMI ist jedoch nur bei der kommerziellen Version verfügbar.

Together

Together wird mittlerweile von der Firma Borland vertrieben. Die für nicht kommerzielle Verwendung gedachte freie Whiteboard Edition wird nicht mehr angeboten. Die Funktionalität konnte daher nicht überprüft werden.

UMLet

UMLet kann als Eclipse Plugin oder standalone betrieben werden. Es bietet eine rudimentäre Oberfläche zum Zeichnen von UML-Diagrammen. Eigenschaften von Modellelementen müssen in einem Textfeld nach einer vorgegebenen Syntax eingetragen werden. UMLet benötigt eine Java Laufzeitumgebung. Die Modelle werden in einem eigenen Format abgespeichert, XMI wird nicht unterstützt.

Zusammenfassend bietet sich also nur ArgoUML als Editor für die zu erstellenden Modelle an. Die restlichen Editoren bieten entweder keine Unterstützung für XMI, oder haben andere Einschränkungen. Damit wird auch nur die Unterstützung für XMI in der Version 1.3 benötigt.

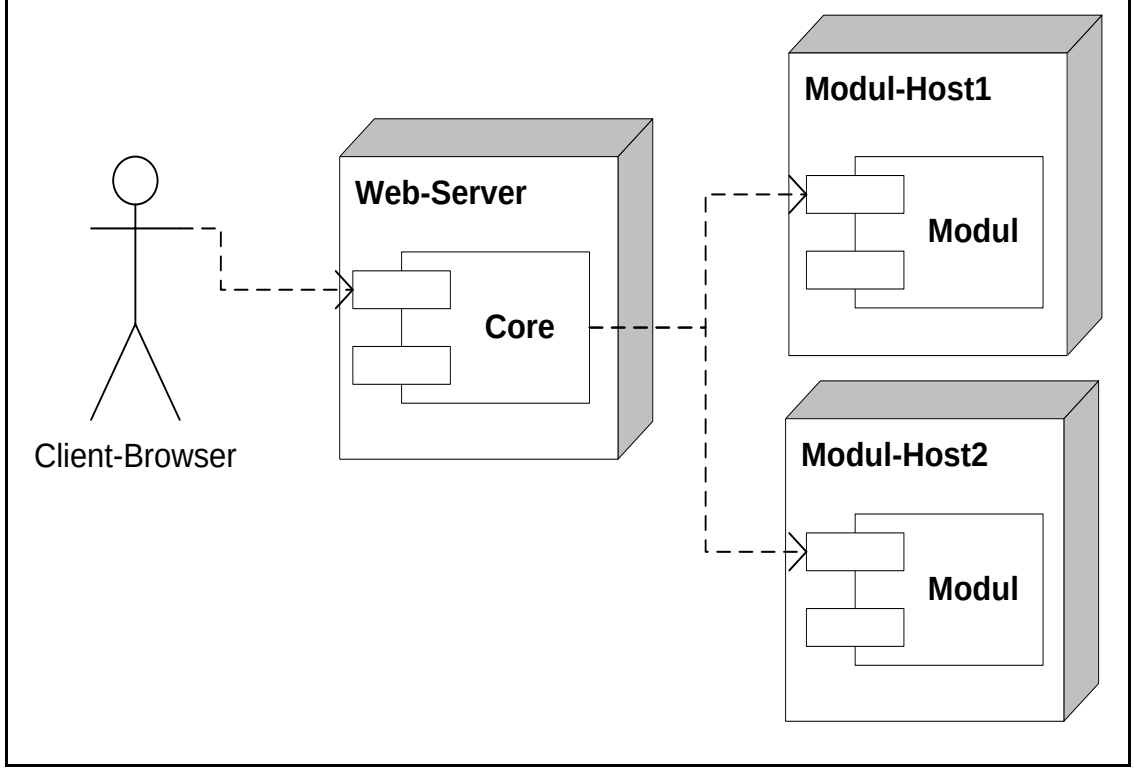
4 E-Tutor System

In diesem Kapitel wird die Architektur des bestehenden E-Tutor Systems beschrieben. Das E-Tutor System ist eine Web-Anwendung. Für die Einbindung weiterer Expertenmodule werden Schnittstellen zur Verfügung gestellt. Die für die Einbindung von Expertenmoduln notwendigen Schnittstellen sowie ihre Methoden werden beschrieben.

4.1 Architektur

Das am DKE Institut der Johannes Kepler Universität Linz betriebene System E-Tutor, besteht im Wesentlichen aus einem Kern (E-Tutor Core) und verschiedenen Expertenmoduln, die jeweils ein bestimmtes Fachgebiet abdecken (siehe Abbildung 4-1).

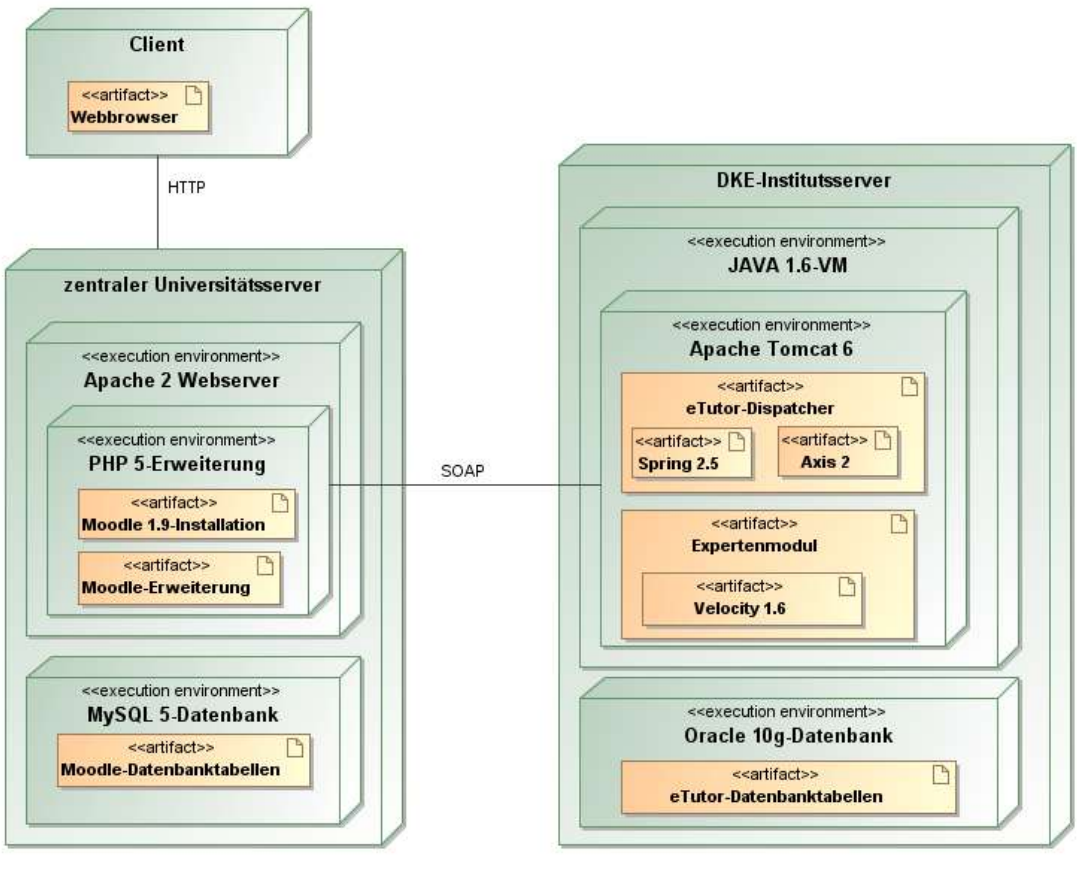
Abbildung 4-1: Architekturübersicht E-Tutor (Quelle: (Eichinger, Karlinger, & Nitzsche, 2006))



Die Expertenmodule wurden in eigenen Software-Moduln, die eine vorgegebene Schnittstelle implementieren, bereitgestellt (Eichinger, Karlinger, & Nitzsche, 2006).

Mittlerweile wurde das E-Tutor System in die Lernumgebung Moodle[14] integriert. Die Integration erfolgte im Rahmen einer Diplomarbeit am DKE-Institut (Fischer, 2010). Die Kernfunktionalität wird jetzt von der Komponente E-Tutor-Dispatcher übernommen. Sämtliche Funktionalität des E-Tutor Kerns wurde dahin verlagert. Für die Erstellung bzw. Generierung der Benutzungsoberfläche wird das Open-Source Werkzeug „Velocity"[15] verwendet. Damit können HTML-Seiten aus Java über Vorlagen (Velocity-Templates) generiert werden.

Abbildung 4-2: Architektur E-Tutor in Moodle integriert (Quelle: (Fischer, 2010))



[14] Siehe http://moodle.org/

[15] Siehe http://velocity.apache.org/

Expertenmodule wie das UML-Expertenmodul werden über die bereitgestellten Schnittstellen eingebunden. Jedes Modul muss die Schnittstellen „Evaluator" und „ModuleExerciseManager" implementieren. Für die Benutzungsoberfläche stehen die Interfaces „Analysis", „Editor", „Report", „PrintReportView" sowie „ShowEditorView" zur Verfügung (siehe Abbildung 4-2).

4.2 Schnittstellen

In den folgenden Kapiteln werden die Methoden der Interfaces „Evaluator" sowie „ModuleExerciseManager" beschrieben.

Anschließend wird auf die Art und Weise eingegangen in der Benutzungsinterfaces in das System integriert werden. Für die Rückgabewerte der einzelnen Methoden müssen Klassen erstellt werden, die das jeweilige Interface implementieren. Für jedes dieser Interfaces gibt es auch eine Standardimplementierung.

Diese Aufzählung erfolgt nur überblicksmäßig, für eine detaillierte Übersicht der E-Tutor Architektur und der verfügbaren Schnittstellen siehe (Eichinger, Karlinger, & Nitzsche, 2006) und (Fischer, 2010).

4.2.1 Evaluator

Dieses Interface dient zur Auswertung der Benutzereingaben. Dazu sind insgesamt 3 Methoden vorgesehen, die eine differenzierte Auswertung der abgegebenen Übung ermöglichen.

Methode „analyze"

Abbildung 4-3: Signatur der Methode analyze

```
public Analysis analyze(
       int exerciseID,
       int userID,
       Map passedAttributes,
       Map passedParameters) throws Exception;
```

Diese Methode wird vom E-Tutor-Dispatcher aufgerufen, nachdem der Benutzer die entsprechende Schaltfläche anklickt. Bei der Implementierung muss auch eine Implementierung des Rückgabeobjektes „Analysis" erstellt werden. Durch diese Methode wird eine Überprüfung der abgegebenen Lösung ermöglicht. Es erfolgt jedoch keine Bewertung und auch kein Feedback an den Studenten. Das Ergebnis der Analyse wird in einer Instanz des Typs Analysis zurückgegeben.

Methode „grade"

Abbildung 4-4: Signatur der Methode grade

```
public Grading grade(
       Analysis analysis,
       int taskID,
       Map passedAttributes,
       Map passedParameters) throws Exception;
```

Die Methode „grade" ist für das Bewerten der abgegebenen Lösung zuständig, dazu wird als Parameter das Analyse Objekt übergeben. Um das Analyse Object zu erhalten muss zuerst ein Aufruf der Methode „analyze" erfolgen. Das Ergebnis wird als Instanz der Klasse „Grading" zurückgeliefert.

Methode „report"

Abbildung 4-5: Signatur der Methode report

```
public Report report(
       Analysis analysis,
       Grading grading,
       Map passedAttributes,
       Map passedParameters) throws Exception;
```

Über diese Methode wird das Feedback an den Studenten erstellt. Je nach Hinweisstufe werden die Texte in das Rückgabeobjekt eingetragen. Es werden die Analyseergebnisse, und falls durchgeführt auch die Bewertungsergebnisse, als Parameter übergeben. Das Ergebnis wird als Instanz des Typs Report zurückgegeben.

4.2.2 ModuleExerciseManager

Dieses Interface dient der Verwaltung und Administration der Übungsaufgaben. Es darf nur von berechtigten Personen (Assistentenrolle) aufgerufen werden.

Methode „createExercise"

Abbildung 4-6: Signatur der Methode createExercise

```
public boolean createExercise(
int exerciseId,
Serializable exercise,
Map attributes,
Map parameters) throws Exception;
```

Diese Methode dient dazu, eine neue Aufgabe zu erstellen. Die Aufgabe selbst wird im Parameter „exercise" übergeben. Dieses Objekt muss die Serializable Schnittstelle erfüllen, damit es im E-Tutor Kern abgespeichert werden kann. Das Objekt muss alle notwendigen Informationen enthalten, damit die Übungsaufgabe erstellt werden kann.

Methode modifyExercise

Abbildung 4-7: Signatur der Methode modifyExercise

```
public boolean modifyExercise(
       int exerciseId,
       Serializable exercise,
       Map attributes,
       Map parameters) throws Exception;
```

Diese Methode dient dazu, eine bereits gespeicherte Übungsaufgabe mit neuem Inhalt abzuspeichern, also zu modifizieren.

Methode deleteExercise

Abbildung 4-8: Signatur der Methode deleteExercise

```
public boolean deleteExercise(int exerciseID) throws Exception;
```

Zum Löschen einer Übungsaufgabe muss nur die eindeutige „exerciseId" der zu löschenden Aufgabe übergeben werden.

Methode fetchExercise

Abbildung 4-9: Signatur der Methode fetchExercise und fetchExerciseInfo

```
public Serializable fetchExercise(int exerciseID) throws Exception;
public Serializable fetchExerciseInfo() throws Exception;
```

Die beiden Methoden werden benötigt um von einer Modulspezifischen View die Daten für die Übungsaufgabe abzurufen. Da es keine direkte Kommunikation zwischen der Administrationskomponente und der modulspezifischen View gibt, muss die Abfrage von Daten eben über diese Methode des Interfaces erfolgen.

Methode generateHtml

Abbildung 4-10: Signatur der Methode generateHtml

```
public String generateHtml(Serializable exercise, Locale locale);
```

Diese Methode dient in erster Linie dazu, den Angabetext für den Benutzer darzustellen. Ein Expertenmodul muss diese Methode jedoch nicht unterstützen. Das Objekt mit den Aufgabendaten wird als Parameter der Methode übergeben. Als zusätzlicher Parameter wird auch die locale übergeben. Dadurch ist es möglich, aufgrund der lokalen Spracheinstellungen des Benutzers die Angabe in der richtigen Sprache anzuzeigen. Dazu muss diese Funktionalität allerdings vom Modul bereitgestellt werden (Mehrsprachigkeit).

4.2.3 Views

Zur Darstellung der Benutzungsschnittstellen sind die Interfaces „ExerciseSettingView", „PrintReportView" sowie „ShowEditorView" vorgesehen. Diese generieren aus den Velocity Vorlagen (Abbildung 4-11) eine xhtml Zeichenkette, die direkt in die Webseite eingebettet werden kann.

Abbildung 4-11: Beispiel für Velocity Vorlage

```
<script src='%%RESOURCES%%/taskEditor.js'
type='text/javascript'></script>
<input type='hidden' id='command' name='%%IDPREFIX%%command'/>
<table>
    <tr>
        <td class='UMLTaskEditorTableData' align='left'>
             <input type='file' name='%%IDPREFIX%%uploadfile'
size='40' />
        </td>
    </tr>
    #if ( ${actions} )
        <tr>
            <td class='UMLTaskEditorTableData' colspan='2'>
                #if ( ${actions.contains( ${umlcons-
tants.ACTION_DIAGNOSE} )} )
                    <input class='UMLTaskEditorButton' type='submit'
value='${messageButtondiagnose}' name='diagnose_mode' on-
click="javascript:setCommand('${umlconstants.ACTION_DIAGNOSE}');"/>
    #end
    #if ( ${actions.contains( ${umlconstants.ACTION_SUBMIT} )} )
                    <input class='UMLTaskEditorButton' type='submit'
value='${messageButtonsubmit}' name='submit_mode' on-
click="javascript:setCommand('${umlconstants.ACTION_SUBMIT}');"/>
                #end
            </td>
        </tr>
    #end
</table>
<br />
```

4.2.4 Interface Editor

Das Interface Editor definiert 3 Methoden, welche für die Kommunikation zwischen Benutzungsschnittstelle und E-Tutor Kern benötigt werden. Über diese Methoden werden die Erstellung von Übungsaufgaben, sowie die Verarbeitung von Benutzereingaben, durch das entsprechende Expertenmodul des E-Tutor Systems aufgerufen.

Methode initPerformTask

Abbildung 4-12: Signatur der Methode initPerformTask

```
public HashMap<String, Object> initPerformTask(int exerciseId) throws Exception;
```

Mithilfe dieser Methode können vom Expertenmodul Initialisierungsparameter für die Übungsausarbeitung zurückgegeben werden. Diese Parameter werden vom E-Tutor Dispatcher in der Session des Benutzers abgelegt.

Methode preparePerformTask

Abbildung 4-13: Signatur der Methode preparePerformTask

```
public HashMap<String, Object> preparePerformTask(HashMap<String, Object> passedAttributes, HashMap<String, Object> passedParameters, Resource [] resources) throws Exception;
```

Diese Methode dient dazu, die Benutzereingaben während der Übungsausarbeitung zu verarbeiten, und die Ergebnisse in der Session des Benutzers abzulegen. Die getätigten Eingaben des Benutzers werden über die Parameter ‚passedAttributes' und ‚passedParameters', sowie ‚resources' übergeben.

Methode prepareAuthorTask

Abbildung 4-14: Signatur der Methode prepareAuthorTask

```
public HashMap<String, Object> prepareAuthorTask(HashMap<String, Object> passedAttributes, HashMap<String, Object> passedParameters, Resource [] resources, Serializable exerciseInfo) throws Exception;
```

PrepareAuthorTask dient dazu, die Benutzereingaben zur Erstellung einer neuen Aufgabe zu verarbeiten. Analog zur Methode ‚preparePerformTask' werden die Benutzereingaben als Parameter übergeben.

5 Konzeption des UML-Expertenmoduls

In diesem Kapitel geht es um ein Konzept zur Umsetzung des UML-Expertenmoduls. Aus der Problemstellung lassen sich drei Hauptprobleme ableiten. Zuerst muss eine Möglichkeit gefunden werden, das vom Studenten abgegebene Modell mit einem gültigen bzw. richtigen Modell zu vergleichen. Das zweite Problem besteht darin, eine Bewertung des Studentenmodells vorzunehmen. Als drittes Problem kann das Erzeugen von Feedback an den Studenten genannt werden. Damit wird auf Fehler eingegangen und dadurch das Lernen (durch Erkenntnis, Lernen durch Fehler) unterstützt.

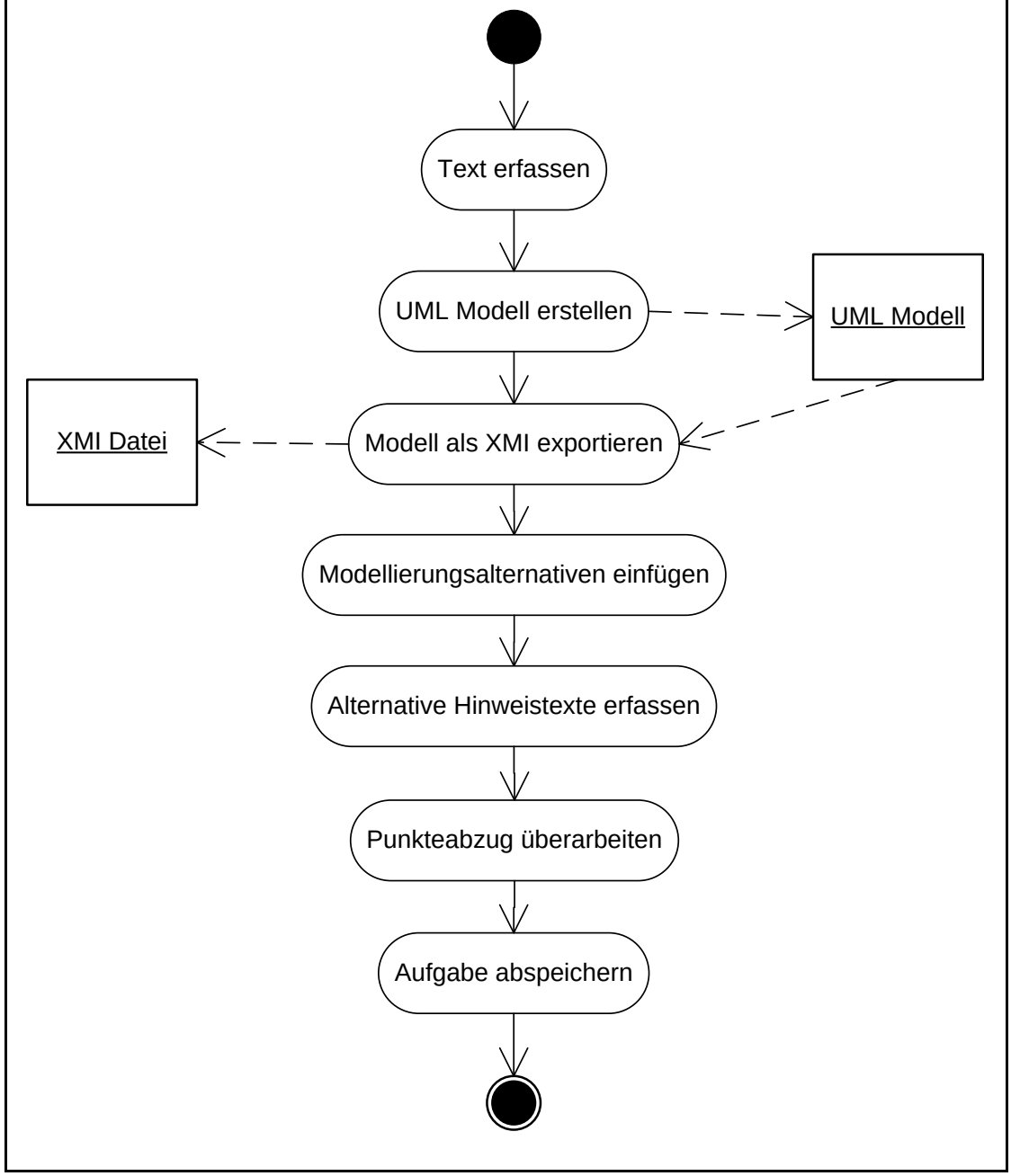
Es wird gezeigt, mit welchen Mitteln Modelle miteinander verglichen werden können. Fehler sollen erkannt werden und Hilfestellungen bzw. Erklärungen an den Lernenden zurückgegeben werden. Zuerst wird untersucht ob es möglich ist, Modellierungsvarianten automatisiert zu erkennen, bzw. aus welchen Gründen dies nicht möglich ist. Daran an schließt ein Konzept für die Bewertung des Studentenmodells, anhand der beim Vergleich der Modelle entdeckten Abweichungen. Für die Feedbackerstellung wird danach ebenfalls ein Konzept erarbeitet.

Zuerst wird der gedachte Ablauf bei der Ausarbeitung einer neuen Aufgabe und danach für den Vergleich und die Bewertung der abgegebenen Aufgabe vorgestellt. Abbildung 5-1 zeigt den groben Ablauf beim Erstellen einer neuen Aufgabe. Zuerst wird eine textuelle Beschreibung des Anwendungsfalles erstellt. Anschließend wird ein Modell erstellt und im editorunabhängigen Format XMI[16] (Version 1.3) abgespeichert. Eine detailliertere Beschreibung zu XMI findet sich im Anhang A. Im abschließenden Schritt müssen Varianten für die Modellierung, sowie dazugehörige Hinweistexte und Punkteabzüge erfasst werden. Das so angereicherte Modell

[16] XML Metadata Interchange, siehe nachfolgende Kapitel sowie Anhang A

wird danach als Aufgabe abgespeichert. Beim Speichern wird auch angegeben, bis zu welcher Detaillierungsstufe die Feedbacktexte für diese Aufgabe generiert werden sollen. Auch bei Collect-UML (Baghaei & Mitrovic, 2005) wird ein Mustermodell (das Idealmodell) verwendet, um Expertenwissen zu verwalten.

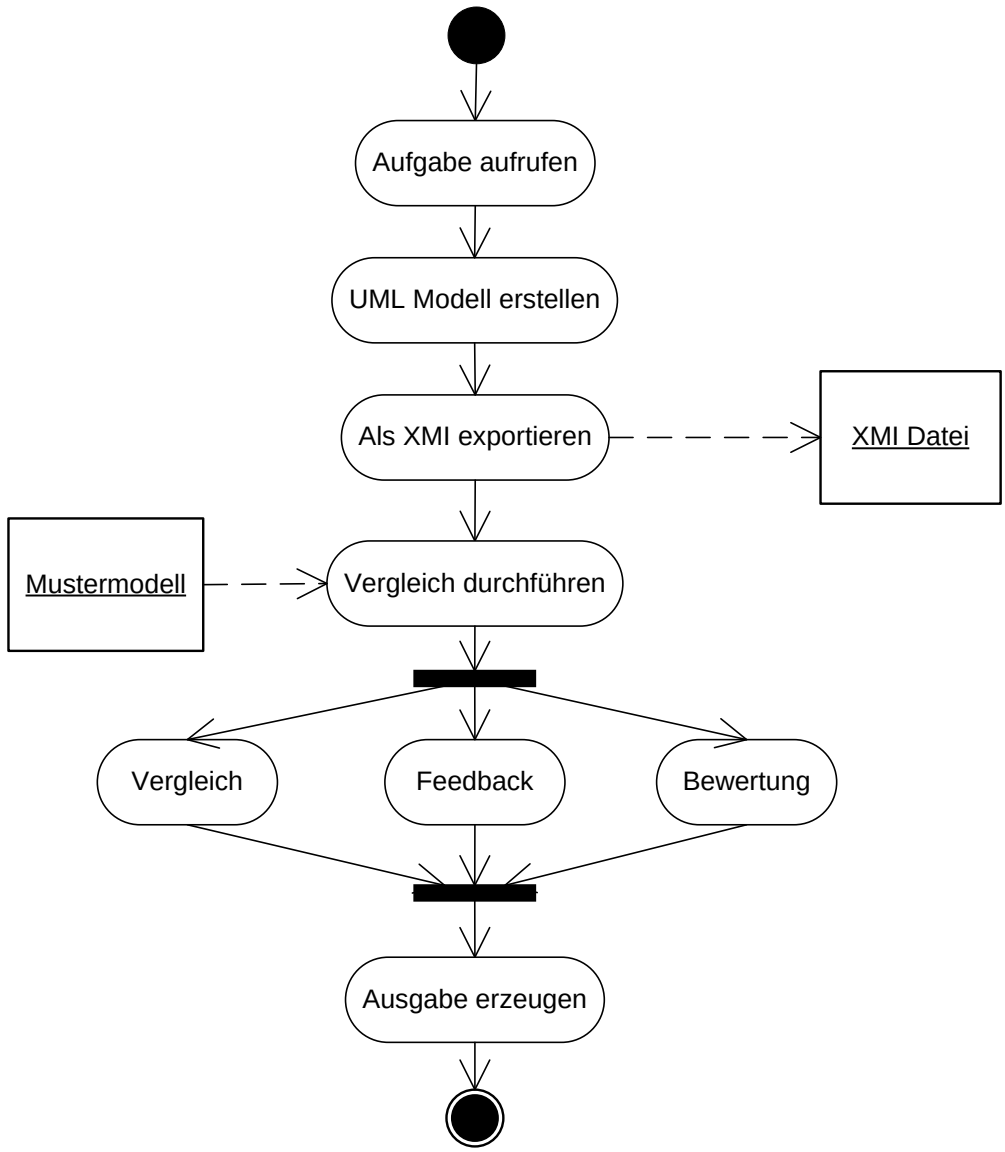
Abbildung 5-1: Ablauf beim Erstellen einer Aufgabe



Das Studentenmodell muss ebenfalls als XMI Datei zur Verfügung gestellt werden. Dabei kann entweder ein UML-Editor innerhalb des

Moduls verwendet werden, oder ein einfacher XMI-Datei Upload erfolgen (Abbildung 5-2). Beim Vergleich wird für jedes Artefakt bzw. Modellelement im Studentenmodell die Entsprechung im Mustermodell gesucht. Wird sie gefunden, so werden die allfälligen Zusatzinformationen, wie etwa Punkteabzug, Hinweistexte usw. gesammelt. Zum Schluss werden die erreichten Punkte und Hinweise an den Studenten zurückgegeben. Die Generierung der Feedbacktexte erfolgt bis zur maximalen, für diese Aufgabe vorgesehenen, Detaillierungsstufe.

Abbildung 5-2: Übung abgeben und bewerten



5.1 Speicherform des Expertenwissens

Das für die Aufgabe erforderliche Expertenwissen liegt zuerst in Form des Mustermodells vor. Dieses Modell kann im Austauschformat XMI gespeichert werden. Zwar wurde XMI von der OMG als Standard definiert, jedoch werden diese Standards von den einzelnen Herstellern von Modellierungswerkzeugen unterschiedlich interpretiert. Im Rahmen dieser Arbeit wird die XMI-Version 1.3 verwendet. Als Modellierungswerkzeug dient ArgoUML[17] in der Version 0.30.2. Das UML-Expertenmodul kann standardkonforme XMI-Modelle verarbeiten, jedoch wurden keine mit alternativen Werkzeuge erstellte Dateien getestet. Es empfiehlt sich also auf ArgoUML für die Erstellung von Muster bzw. Studentenmodell zurückzugreifen.
Dieses modellbezogene Wissen reicht jedoch nicht aus, um die Aufgaben eines Expertenmoduls, wie etwa Vergleich, Bewertung und Feedback-Erstellung zu ermöglichen. Die dafür benötigten Zusatzinformationen müssen ebenfalls abgespeichert werden. Um eine manuelle Bearbeitung durch den Aufgabenersteller zu erleichtern, soll eine einfache und übersichtliche Möglichkeit zur Speicherung des Expertenwissens gefunden werden.

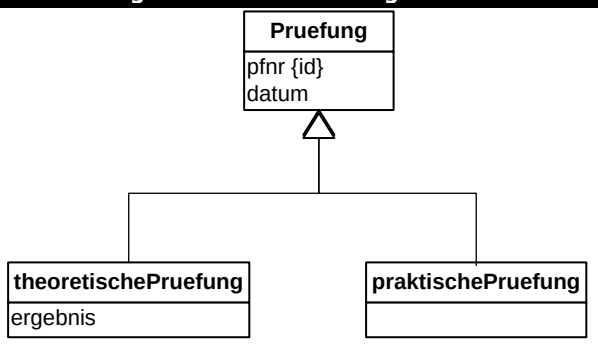
Konvertierung von XMI in XML

XMI wurde als Austauschformat für vollständige Modelle entwickelt. Dieses Austauschformat ist jedoch für eine maschinelle Verarbeitung gedacht, und enthält überdies viele Elemente die für ein konzeptuelles Datenbankschema nicht von Interesse sind. Der Lehrende muss allerdings in der Lage sein, das Mustermodell noch nachträglich zu bearbeiten. Deshalb bietet sich eine vereinfachte, übersichtlichere und vor allem leichter lesbare Version in XML an.

[17] Download von http://argouml.tigris.org/

Bei einem klassischen konzeptuellen DB-Modell interessieren vor allem die Objekttypen, also die Klassen und ihre Beziehungen zueinander. Von dieser Erkenntnis ausgehend wird die Klasse als das zentrale Element der komprimierten Darstellung gewählt. Die ansonsten im XMI-Format verzweigten Information diese Klasse betreffend werden nun zusammengezogen, und dieser Klasse zugeordnet. Das betrifft die Attribute genauso, wie die Vererbungs- und Beziehungselemente. Für jede Klasse wird ein Knoten „class" erzeugt. Als XML-Attribut „name" des Knotens wird der Klassenname eingetragen. Die Attribute der Klasse werden als Subknoten „attribute" innerhalb des Knotens „class" angelegt. Auch diese Knoten enthalten ein XML-Attribut „name". Sichtbarkeit und Datentyp der Attribute, sowie die Methoden sind für den konzeptuellen DB-Entwurf nicht von Bedeutung und werden daher weggelassen. Abbildung 5-3 zeigt eine Generalisierung an der drei Klassen beteiligt sind. In der XML-Struktur wird die Generalisierung als Knoten im Class Element abgelegt.

Abbildung 5-3: Generalisierung als Zusatzinfo zu Class

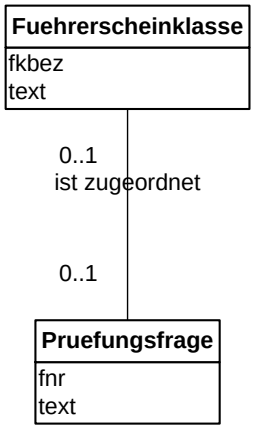



```
<class name='Pruefung'>
 <attribute name='pfnr'/>
 <attribute name='datum'/>
 <subclass
name='praktischePruefung'/>
 <subclass
name='theoretischePruefung'/>
</class>
<class name
='praktischePruefung'>
 <attribute name='ergebnis'/>
 <superclass name='Pruefung'/>
</class>
<class
name='theoretischePruefung'>
 <superclass name='Pruefung'/>
</class>
```

Die Generalisierungsklasse enthält den Knoten „subclass" und dieser Knoten enthält im XML-Attribut „name" den Namen der Spezialisierungsklasse. Im Beispiel von Abbildung 5-3 sind das also je ein Knoten „subclass" für die Klasse „praktischePruefung", sowie für die Klasse „theoretischePruefung". Bei den Spezialisierungsklassen

wird hingegen der Verweis auf die Generalisierungsklasse im Knoten „superclass" vermerkt. Auch hier wird im XML-Attribut „name" der Name der Generalisierungsklasse angegeben.

Abbildung 5-4: Eine Assoziation und ihre Entsprechung in der XML-Struktur




```
<class name='Fuehrerscheinklasse'>
 <attribute name='fkbez'/>
 <attribute name='text'/>
 <associationTo ClassTo='Pruefungsfrage'
   Type='none' lower='0' upper='-1'/>
</class>
<class name='Pruefungsfrage'>
 <attribute name='fnr'/>
 <attribute name='text'/>
 <associationTo ClassTo='Fuehrerscheinklasse'
   Type='none' lower='0' upper='1'/>
</class>
```

Für eine Assoziation wird ein Knoten „associationTo" bei der Klasse eingefügt. Dieser Knoten verfügt über XML-Attribute, welche Auskunft über die beteiligte Klasse („ClassTo"), den Typ der Assoziation („Type") sowie die Kardinalität („lower" bzw. „upper") geben. Es wird immer nur der „ausgehende" Teil der Assoziation bei der Klasse abgespeichert. Damit gibt es bei der verbundenen Klasse ebenfalls einen „AssociationTo" Eintrag.

Bei Kompositionen bzw. Aggregationen wird die besondere Form dieser Beziehung im XML-Attribut „Type" vermerkt. Gültige Werte und ihre Bedeutung sind in Abbildung 5-5 vermerkt.

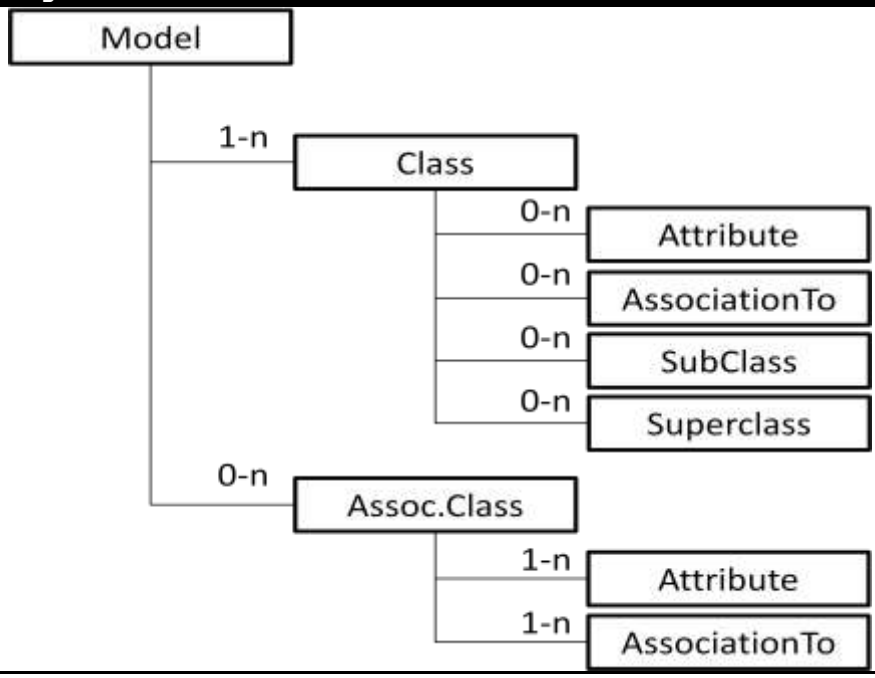
Abbildung 5-5: Gültige Werte des Beziehungstyps

Wert	Bedeutung
none	Assoziation
composite	Komposition
aggregate	Aggregation
n-ary	n-äre Assoziation

Neben binären Assoziationen werden auch n-äre Assoziationen abgebildet. Da diese Spezialform nicht direkt ablesbar ist, wird immer

dann, wenn eine Assoziation aus mehr als 2 Assoziationsenden besteht, für den Typ der Assoziation „n-ary" gesetzt.
Für die Erstellung der XML-Datei kann nun ein Schema angegeben werden. Abbildung 5-6 zeigt die Struktur bzw. das Schema der zu erzeugenden XML-Datei. Die notwendigen Informationen, die einen Knoten beschreiben, werden als XML Attribute hinzugefügt. Da das zugrundeliegende Format der XMI Datei ebenfalls XML ist, kann zur Transformation in das Zielformat entweder XSLT[18] oder direkte Programmierung mittels Parser erfolgen.

Abbildung 5-6: Das Schema der XML Datei



Da die erzeugte XML Struktur für die weitere Verarbeitung benötigt wird ist es sinnvoll, gleich eine Java Bibliothek unter Verwendung von beispielsweise SAX zu implementieren. Die so während des Parsens erzeugten Objekte können für die folgende Verarbeitung gleich weiterverwendet werden.

[18] Zur Beschreibung von XSLT siehe http://www.w3.org/TR/xslt

Abbildung 5-7: unterstützte Elemente der statischen Sicht der UML		
Level	Element	Unterstützt
Classifiers	*Actor*	*Nein*
Classifiers	*Class*	*Ja*
Classifiers	*Class-in-state*	*Nein*
Classifiers	*Classifier role*	*Nein*
Classifiers	*Component*	*Nein*
Classifiers	*Data type*	*Nein*
Classifiers	*Interface*	*Nein*
Classifiers	*Node*	*Nein*
Classifiers	*Signal*	*Nein*
Classifiers	*Subsystem*	*Nein*
Classifiers	*Use case*	*Nein*
Relationships	*Association*	*Ja*
Relationships	*Dependency*	*Nein*
Relationships	*Flow*	*Nein*
Relationships	*Generalization*	*Ja*
Relationships	*Realization*	*Nein*
Relationships	*Usage*	*Nein*
Constraints	*Constraint*	*Nein*
Instances	*Object diagram*	*Nein*

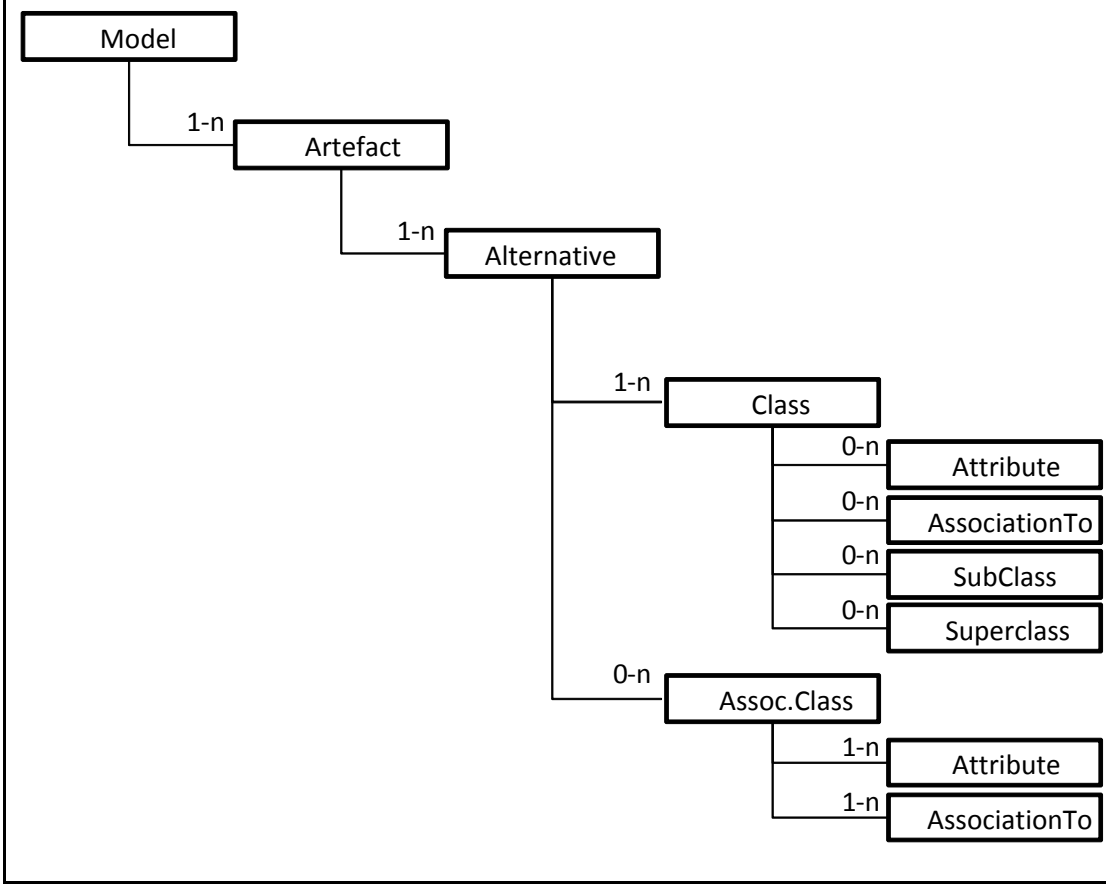
Abbildung 5-7 zeigt welche Elemente der statischen Sicht (static view) von UML im UML-Expertenmodul verarbeitet werden können. Es werden jedoch nicht alle Elemente in dieser Sicht auch für den konzeptuellen Datenbankentwurf benötigt. Auch werden bei manchen Elementen nicht alle Details benötigt. Bei der Klasse etwa sind die Methoden für den konzeptuellen Entwurf nicht notwendig, sie werden daher, genauso wie die Datentypen, nicht unterstützt.

Verwalten von Modellierungsvarianten

Durch die Umwandlung des Modells aus dem XMI-Format in ein selbst definiertes XML-Format, wird für jede Klasse bzw. Assoziationsklasse ein eigener Knoten innerhalb des Wurzelknotens „Model" erzeugt. Klassen werden als Hauptelement des Diagramms be-

trachtet und die Beziehungen zu anderen Klassen werden als ausgehende Verbindungen ebenfalls bei diesem Knoten gespeichert. Durch diese spezielle Speicherungsform ist es nun möglich auch andere alternative Varianten der Modellierung abzuspeichern. Dazu wird die Struktur des XML-Formats erweitert.

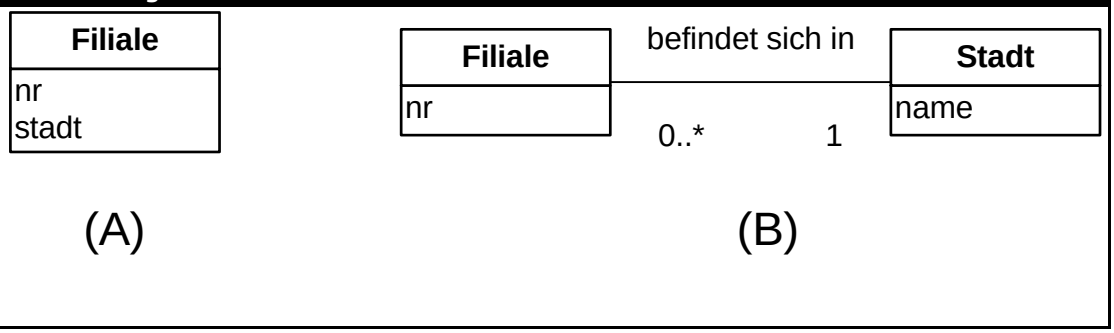
Abbildung 5-8: Erweiterung der Struktur für Mustermodell



Jede Klasse bzw. Assoziationsklasse aus dem Modell mit ihren Beziehungen zu anderen Klassen stellt ein Artefakt dar. Damit wird ein Ausgangspunkt für die Bildung von Varianten geschaffen. Innerhalb eines Artefakts können nun beliebig viele Varianten (jeweils mit eigenem Knoten „Variant") gespeichert werden. Durch die Klammerung im Knoten „Artefact" wird die Zugehörigkeit zu einem bestimmten abgebildeten Sachverhalt dargestellt.

Varianten unterscheiden sich je nach Anforderung in Effizienz und Komplexität. Manche Varianten sind zwar formal korrekt, können aber für den gegebenen Anwendungsfall nicht optimal sein. Dies muss bei der Bewertung bzw. dem Vergleich berücksichtigt werden. Abbildung 5-9 zeigt zwei unterschiedlich modellierte Varianten eines Sachverhalts.

Abbildung 5-9: Zwei Varianten für einen Sachverhalt



Vom Lehrenden können auch bewusst falsche Varianten eingetragen werden, um einen besseren Vergleichswert mit dem Studentenmodell auch bei Fehlern zu erreichen. Damit können bekannte Fehlerquellen als Varianten eingetragen werden, und erleichtern so das Auffinden von „erwarteten" Fehlern im Studentenmodell. Beim Constraint based modelling wird ein ähnlicher Ansatz verfolgt. Es werden auch solche Constraints abgespeichert, die bekannte Fehler erkennen sollen. Bei diesem „Lernen durch Fehler" muss der Student zuerst den Fehler erkennen, um danach den Fehler beheben bzw. vermeiden zu können (Mitrovic, Koedinger, & Martin, 2003).

Für den Lehrenden wird ein Werkzeug vorgesehen, mit Hilfe dessen das Mustermodell in der XML-Datei editiert werden kann. Die einzelnen Klassen, als Inhalt der Modellierungsvarianten können dabei innerhalb der Datei in andere Varianten verschoben werden. Auch neue Klassen können erzeugt und eingefügt werden. Somit kann eine Variante eines Sachverhalts zum Beispiel aus zwei Klassen bestehen, die miteinander verbunden sind, eine andere Variante die-

ses Sachverhalts beinhaltet nur eine Klasse mit allen Attributen aus den zwei Klassen der vorher beschriebenen Variante. Abbildung 5-10 zeigt die XML-Datei mit den zwei Modellierungsvarianten wie in Abbildung 5-9 dargestellt.

Abbildung 5-10: Ausschnitt aus der XML Datei mit den Varianten

```
<Artefact>
   <Alternative nr=”1”>
      <Class name=“Filiale”>
         <Attribute name=”Nr”/>
         <Attribute name=”Stadt”/>
      </Class>
   </Alternative >
   <Alternative nr=”2”>
      <Class name=”Filiale”>
         <Attribute name=”Nr”>
         <AssociationTo  className=”Stadt”  type=”association”  lower=”1”  up-
per=”1”/>
      </Class>
      <Class name=”Stadt”>
         <Attribute name=”name”/>
         <AssociationTo  className=”Filiale”  type=”association”  lower=”0”
upper=”-1”/>
      </Class>
   </Alternative >
</Artefact>
```

Aufwändiger wird es bei Klassen die Beziehungen zu anderen Klassen bzw. Assoziationsklassen eingehen, oder aber auch in einer Vererbungshierarchie stehen. Diese Beziehungen müssen bei der Bildung von Varianten berücksichtigt werden. Das bedeutet unter anderem auch, dass beim Erstellen einer Variante für einen Sachverhalt unter Umständen auch bei einem anderen verbundenen Sachverhalt eine neue Variante erzeugt werden muss.

Die Beziehungsinformation die bei einer Klasse gespeichert wird, beinhaltet nur die „ausgehende“ Richtung (zu welcher Klasse und mit welcher Kardinalität und welchem Typ). Das heißt es gibt immer auch einen „Gegeneintrag“ bei jener Klasse mit der eine Beziehung eingegangen wird. Ebenso verhält es sich mit Superklasse und Subklassen Elementen.

Master-Model-Editor

Das Arbeitsformat des Mustermodells ist XML, daher genügt für das Bearbeiten des Modells ein einfacher XML-Editor. Es werden unterschiedliche freie XML Editoren zum Download im Internet angeboten[19].

Vorteile eines XML-Editors sind unter anderem Syntax-Highlighting, Suchfunktionen bzw. Ersetzungsfunktionen und auch die Möglichkeit die Schriftgröße zu ändern. Neben dem Erstellen von Varianten, können auch kontextabhängige Punkteabzüge eingestellt werden. Die XML-Knoten für die Punkteabzüge werden beim Umwandeln des XMI-Formats in das XML-Format automatisch eingefügt.

Abbildung 5-11: Beispiel des XML-Editors auf einer Webseite

Beim Erstellen von neuen Varianten müssen diese jedoch manuell eingefügt werden. Der Wert der einzelnen Punkteabzüge wird beim Erzeugen mit unterschiedlichen Default-Werten pro Fehlerkategorie aus der Konfigurationseinstellung belegt. Während der Bearbeitung des Mustermodells können diese jedoch an die Situation angepasst und verändert werden.

---

[19] zB EditArea von C. Dolivet: http://www.cdolivet.com/index.php?page=editArea

5.2 Analyse des Studentenmodells

Mit dem Speichermodell für das Mustermodell, ist auch die Basis für einen Modellvergleich gegeben. Das Studentenmodell kann ebenfalls in das vorgestellte XML-Schema transformiert werden. Bei dieser Transformation müssen allerdings keine Zusatzinformationen verwaltet werden. Durch die im Mustermodell abgespeicherten Zusatz-Informationen können auch Modellierungsvarianten erkannt und berücksichtigt werden. Es können allerdings nur jene Modellierungsvarianten erkannt werden, die auch im Mustermodell vorhanden sind.

5.2.1 Modellierungsvarianten und Modellierungsfehler

Bei der Bewertung bzw. dem Vergleich des Studentenmodells sollen auch Varianten der Modellierung beachtet werden. In diesem Kapitel werden verschiedene Varianten der Modellierung eines Sachverhalts (meist Entscheidungsproblem) besprochen. Diese Entscheidungsprobleme sind auch Quelle für Fehler und daher für diese Arbeit von Interesse. Für eine automatisierte Bewertung ist es von Vorteil wenn die in Frage kommenden Modellierungsvarianten auch automatisch erkannt werden könnten. Deshalb wird für jede angeführte Variante bewertet ob eine automatische Erkennung möglich ist. Naturgemäß können nur die wichtigsten und bekanntesten Muster behandelt werden. Diese Aufzählung erhebt in keinster Weise den Anspruch auf Vollständigkeit.

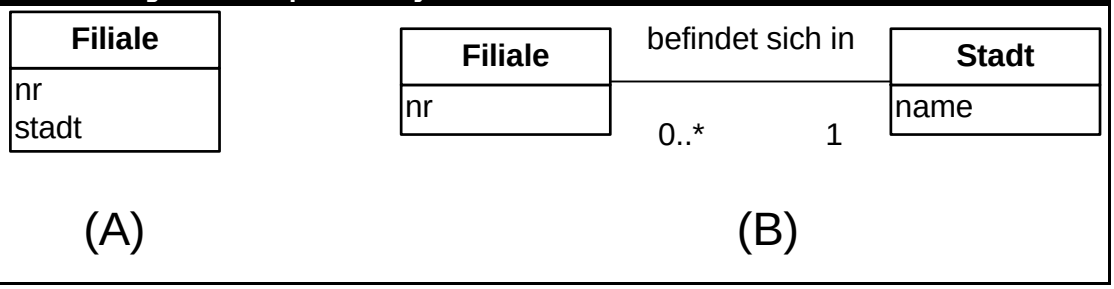
Objektklasse vs. Attribut

Falls eine Objektklasse nur aus einem Identifier und einem Attribut besteht kann sie auch als Attribut modelliert werden. Dies ist allerdings nur solange gültig als die Objektklasse nicht mehrwertig an Beziehungen zu anderen Klassen teilnimmt oder wenn außer dem Schlüssel noch beschreibende Attribute nötig sind. Auch wenn die

Objektklasse noch Beziehungen zu anderen Klassen eingeht, kann sie nicht als Attribut modelliert werden.

Eine spezielle Variante sind Objektklassen, die mit einer eins zu eins Beziehung miteinander assoziiert sind. Solche Klassen können zu einer Klasse zusammengefasst werden, indem alle Attribute übernommen werden. Die Assoziation entfällt dadurch natürlich.
Abbildung 5-12 zeigt ein einfaches Beispiel, aus den Schulungsunterlagen des DKE Instituts an der JKU, da aus dem Beispiel (siehe Kapitel 1.3) kein geeigneter Fall ableitbar ist. Die Variante (A) zeigt eine Klasse „Filiale" mit den Attributen „nr" und „stadt".

Abbildung 5-12: Beispiel für Objektklasse oder Attribut



Der Name der Stadt wird hier also als Attribut der Klasse „Filiale" modelliert. Dies ist richtig, falls es nur eine Filiale in einer Stadt gibt, und nur der Name der Stadt von Bedeutung ist. Die Variante (B) zeigt zwei Klassen „Filiale" und „Stadt" die miteinander assoziiert sind. Diese Variante ist notwendig, wenn die Klasse „Stadt" auch andere Beziehungen zu Klassen eingeht, oder zusätzliche Attribute zur Beschreibung notwendig sind. Falls sich Variante (B) im Mustermodell befindet, kann falls die vorher genannten Bedingungen erfüllt sind, die Variante (A) erzeugt werden. Welche der beiden Varianten für das zu erstellende Modell geeignet ist, hängt vom Kontext ab, deshalb kann es keine automatische Aussage zur Korrektheit dieser Varianten geben. Es wird darüber hinaus auch angenommen, dass das Mustermodell korrekt ist und alle erforderlichen Attribute modelliert worden sind.

Komposition vs. Assoziation

Bei der Frage ob eine Assoziation oder eine Komposition modelliert werden soll, muss bewertet werden, ob ein Objekt einer Objektklasse von einem anderen existenziell abhängig ist. Viele Fehler beim Modellieren von Kompositionen gehen auf eine Falschinterpretation der Existenzabhängigkeit zurück. Während zum Beispiel eine Fahrt ohne Fahrzeug keinen Sinn macht, ist eine Objektklasse Student ohne Inskription durchaus denkbar.

Die Existenzabhängigkeit kann nur über Kontextwissen erkannt werden. Eine Komposition wird auch oft als Teil-Ganzes-Beziehung bezeichnet. Ein wichtiges Merkmal der Komposition ist die Kardinalität von 1 auf der Seite des Ganzen. Existiert keine solche Beziehung (Existenzabhängigkeit, Teil-Ganzes), so ist eine Assoziation zu modellieren. Falls die Assoziation selbst über Eigenschaften verfügt, dann ist eine Assoziationsklasse notwendig.

Abbildung 5-13: Komposition vs. Assoziation

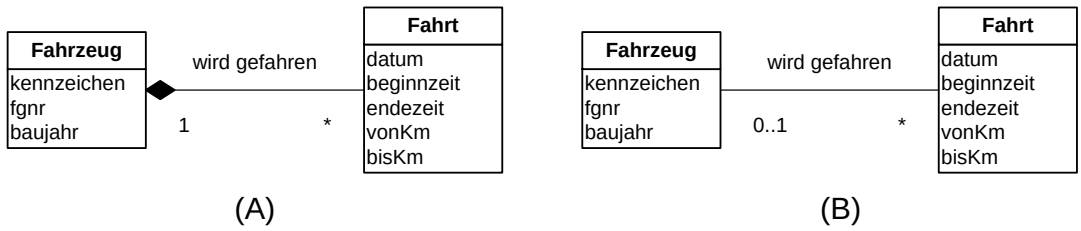


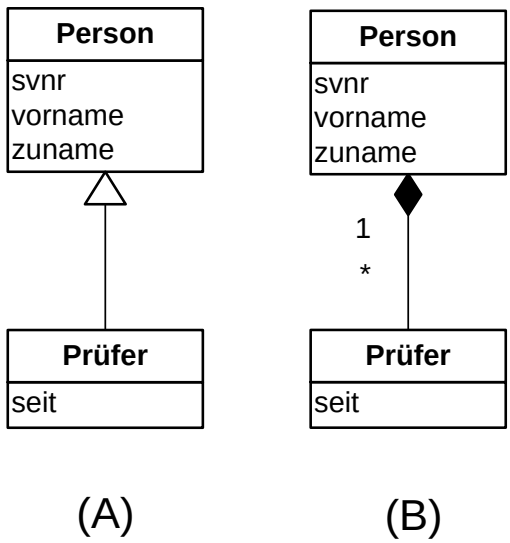
Abbildung 5-13 zeigt die korrekte Modellierung der Beziehung zwischen Fahrzeug und Fahrt als Komposition (A). Die Variante mit der Assoziation (B) ist im gegebenen Kontext der Aufgabe nicht korrekt, da die Fahrt vom Fahrzeug abhängig ist. Zudem ist auch die Kardinalität (0..1) falsch, da eine Fahrt mit genau einem Fahrzeug durchgeführt wird. Eine automatische Generierung von Varianten ist hier nicht notwendig, da sich Komposition im XML-

Schema des UML-Expertenmoduls nur durch das XML-Attribut „Type" unterscheiden. Eine fälschlicherweise als Assoziation modellierte Komposition, wird als „Beziehung mit falschem Typ" erkannt.

Komposition vs. Generalisierung

Oftmals werden auch Komposition und Generalisierung verwechselt. Wie bereits weiter oben beschrieben, ist eine Komposition eine Beziehung von Objektklassen auf der Ausprägungsebene. Das heißt, die konkreten Instanzen der jeweiligen Objektklassen (also die Objekte) haben eine Abhängigkeitsbeziehung.
Bei der Generalisierung ist diese Art der Beziehung auf der Schemaebene angesiedelt. Eine Generalisierung drückt meist eine „ist ein" (isA) Beziehung aus, also eine Einordnung in ein allgemeineres Objekt.

Abbildung 5-14: richtige und falsche Modellierung der Beziehung Prüfer und Tier



Eine Objektklasse „Prüfer" zum Beispiel kann zur Objektklasse „Person" generalisiert werden, eine Komposition von der Objektklasse „Prüfer" zur Objektklasse „Person" ist jedoch falsch.

Welche der beiden Varianten korrekt ist, kann ohne Kontextwissen nicht automatisiert erkannt werden. Wenn davon ausgegangen werden kann, dass das Mustermodel die korrekte Variante enthält, so kann die jeweils andere Variante generiert werden.

Attribute vs. Generalisierung

Oftmals wird für den Wertebereich (Domain) eines Attributs eine Menge vorgegeben. Fälschlicherweise wird dies allerdings oft als Generalisierung modelliert. Der Einband eines Buches kann zum Beispiel ein Wert aus dem Bereich Leder, Textil oder Karton sein. Das qualifiziert das Attribut aber nicht zur abgeleiteten Objektklasse. Eine Generalisierung ist nur dann zu modellieren, wenn eine Teilmengenbildung möglich ist, und die Subtypen auch unterschiedliche Attribute benötigen. Eine Generalisierung ist auch dann möglich, wenn mit den Subtypen unterschiedliche Aktionen bzw. Objektklassen verbunden sind.

Abbildung 5-15: Verwendung von Generalisierung

Abbildung 5-15 zeigt die nicht optimale Verwendung der Generalisierung für den Einband des Buches. Rechts ist die bessere Variante dargestellt: ein Attribut für den Einbandtyp in der Objektklasse Buch. Durch die Einschränkung auf den definierten Wertebereich können Objekte leicht kategorisiert werden. Auch hier wird Kontextwissen benötigt, um die korrekte Variante ermitteln zu können. Aus der Variante mit den Einschränkungen könnte, die andere Variante mit der Generalisierung erzeugt werden. Der UML Erweite-

rungsmechanismus Einschränkung (Constraint) wird allerdings vom eingesetzten Werkzeug ArgoUML nicht unterstützt, und kann daher auch nicht erkannt werden.

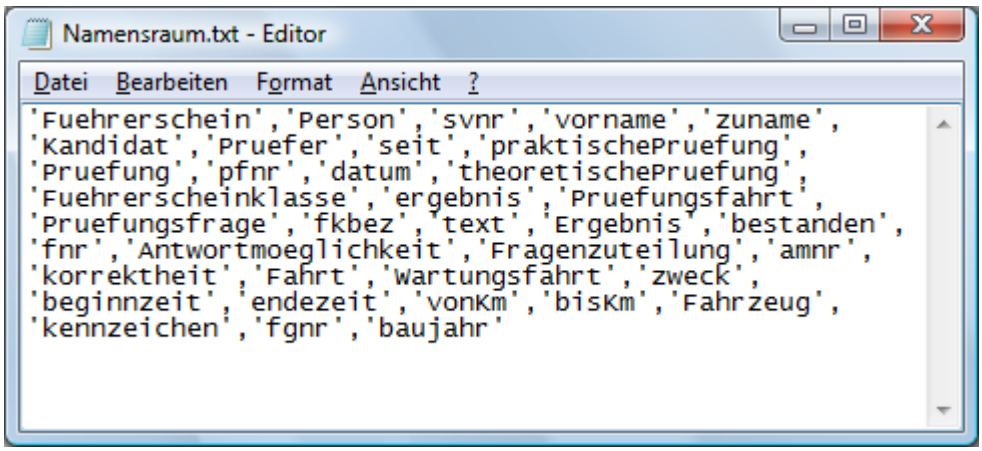
Muster vs. Studentenmodell

In diesem Abschnitt wurde versucht ein Konzept für das automatische Erkennen von Modellierungsvarianten zu erarbeiten. Es wurden vier bekannte Typen von Entscheidungsproblemen beschrieben. Eine Möglichkeit automatisch Varianten zu erzeugen ist bei den Typen „Objektklasse vs. Attribut" und „Komposition vs. Generalisierung" möglich. Diese Generierung ist jedoch immer nur einseitig und setzt voraus, dass die Variante von der aus die andere Variante generiert werden kann, im Mustermodell vorhanden ist. Weiters muss auch angenommen werden, dass die Variante im Mustermodell korrekt ist. Wegen dieser Einschränkungen wird auf das automatische Erzeugen von Varianten verzichtet. Überdies müssten diese auch nach einer automatischen Generierung manuell nachbearbeitet werden. Die Modellierungsvarianten müssen daher manuell zum Expertenwissen hinzugefügt werden. Mit der XML Repräsentation des vom Lehrenden erstellten Mustermodells ist jedoch ein gute Basis gegeben, von der aus auch bewusst die oben angeführten Fälle ergänzt werden können. Damit besteht die Möglichkeit ein Studentenmodell auch auf falsche Anwendung von Modellierungen zu prüfen. Bei Auftreten eines solchen Falles kann dann auch ein spezifisches Feedback zurückgegeben werden. Grundsätzlich ist jedoch das Mustermodell die Basis eines Elementvergleichs mit dem Studentenmodell.

5.2.2 Modellvergleich

Damit Mustermodell und Studentenmodell miteinander verglichen werden können, müssen die zu vergleichenden Modellelemente zuerst identifiziert werden.

Abbildung 5-16: Namensraum aus dem Mustermodell

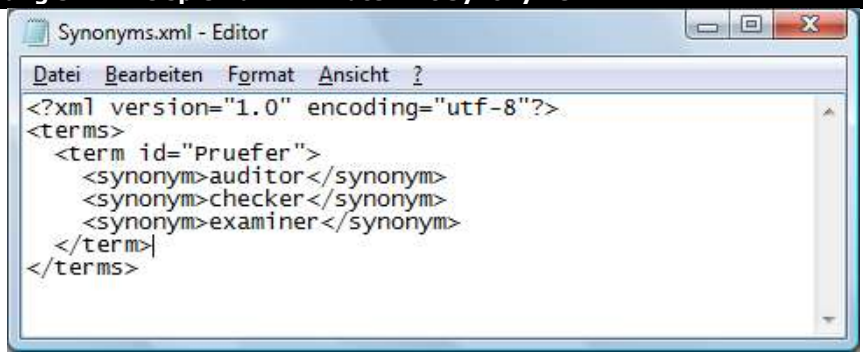


Viele Modellelemente lassen sich über ihren Namen identifizieren. Bei der Modellierung von Sachverhalten werden jedoch oft unterschiedliche Namen vergeben. Zwei Personen die denselben Sachverhalt abbilden, werden vermutlich die Elemente unterschiedlich benennen. Damit würden auch für Mustermodell und Studentenmodell für ein Modellelement unterschiedliche Namen verwendet werden. Um trotzdem einen Vergleich anhand des Namens zu ermöglichen, müssen die zu verwendenden Namen in einer Liste vorgegeben werden. Auch die Assoziationen müssen benannt werden. Der Namensraum kann aus dem Mustermodell direkt abgeleitet werden. Zusätzlich kann auch eine Liste mit gleichartigen Begriffen (Synonyme) verwaltet werden.

Diese Liste kann in der XML Datei für das Mustermodell mitgespeichert werden, damit ist diese mit dem Muster verfügbar und kann auch jederzeit mit neuen Synonymen ergänzt werden. Ergänzend dazu kann auch eine globale Synonym Liste erstellt werden, um bei

immer wieder vorkommenden Begriffen nicht immer die Synonyme neu erstellen zu müssen.

Abbildung 5-17: Beispiel für XML Datei mit Synonymen

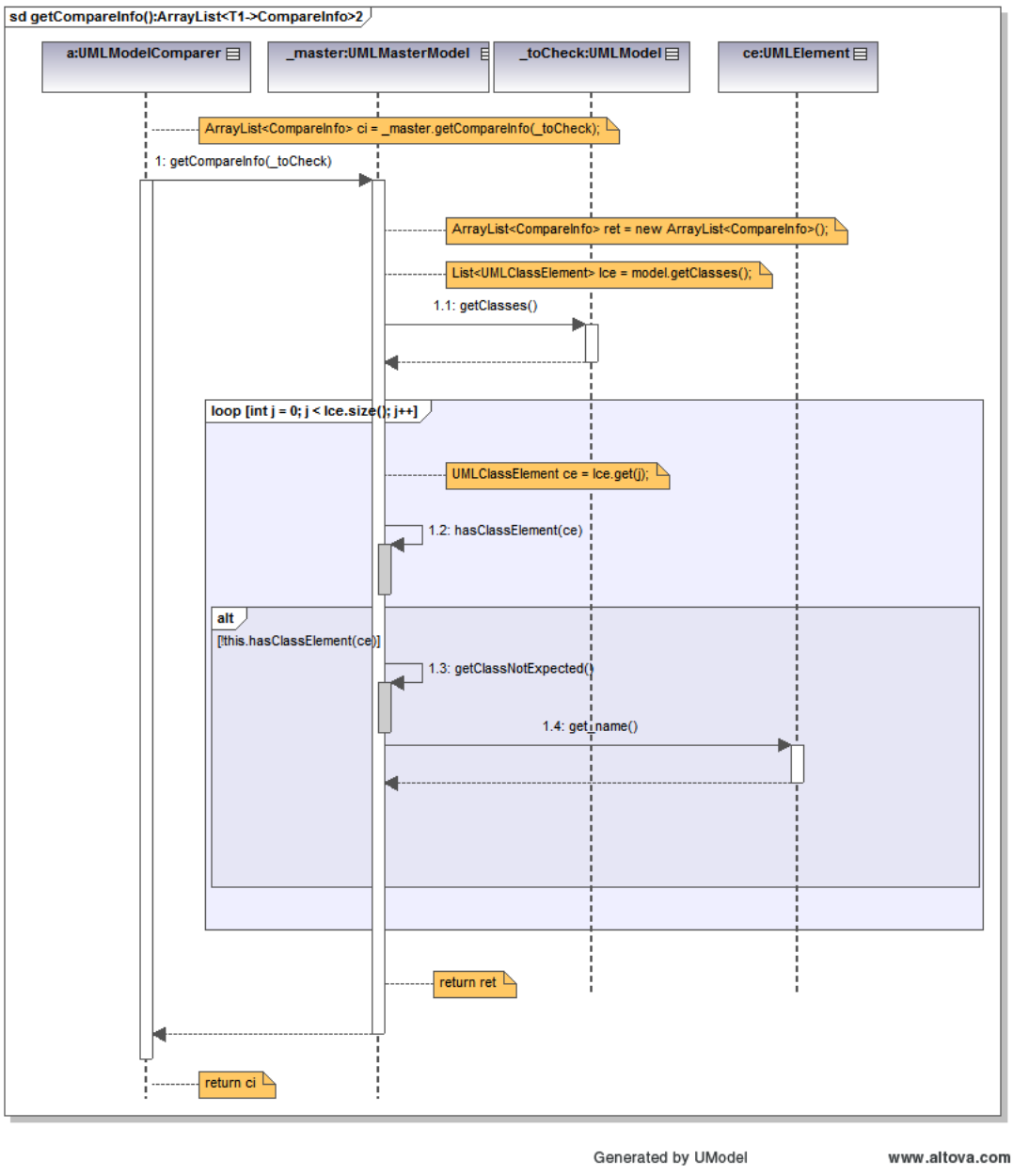


Diese globale Liste hat außerdem den Vorteil, dass sie ständig mit neuen Synonymen erweitert wird, und so im Laufe der Zeit viele ähnliche und unterschiedliche Begriffe zu einem Sachverhalt enthält.

Das Modul wird neben dem reinen Tutoring auch im Lehrbetrieb eingesetzt, deswegen erscheint es sinnvoll, zusätzlich zu den vorkommenden Begriffen, auch bewusst falsche Begriffe einzufügen. Damit soll ein „Vorgeben" der Lösung vermieden werden. Eine Soundex[20]-Funktion ist nicht zielführend, da hierbei lediglich auf ähnlich klingende Namen abgezielt wird. Durch das Umwandeln der XMI Datei in das vorgestellte XML Format, wird das Modell in einzelne Sachverhalte (Artefakt) unterteilt. Im Mustermodell gibt es jetzt unter Umständen mehrere Varianten pro Sachverhalt. Für den Vergleich wird für jeden Sachverhalt aus dem Mustermodell eine Entsprechung im abgegebenen Studentenmodell gesucht. Dabei werden alle Varianten eines Sachverhalts überprüft. Anschließend werden die Elemente einer Variante mit der Entsprechung im Studentenmodell verglichen.

[20] Siehe http://resources.rootsweb.ancestry.com/cgi-bin/soundexconverter

Abbildung 5-18: Sequenzdiagramm Vergleich



In einer zweiten Runde müssen noch die Elemente im Studentenmodell gesucht werden, die nicht im Mustermodell erscheinen. Diese überzähligen Elemente werden als ‚nicht erwartet' kategorisiert. Das oberste Element im Mustermodell ist, wie bereits besprochen, das Artefakt. Unterhalb des Artefakts sind im Mustermodell die Varianten zu finden. Und eine Variante beinhaltet nun Klassen bzw. Assoziationsklassen. Die Entsprechung einer im Mustermodell enthaltenen Klasse oder Assoziationsklasse wird im Studentenmodell

gesucht. Nachdem eine Übereinstimmung gefunden worden ist, können nun fehlende oder abweichende Eigenschaften dieses Elements festgestellt werden. Wenn im Mustermodell mehrere Varianten vorgesehen sind, müssen alle Varianten gegen das Studentenmodell geprüft werden. Es wird geprüft welche Variante die höchste Übereinstimmung mit dem Studentenmodell hat. Falls zwei Varianten den gleichen Grad der Übereinstimmung haben, wird auch die Anzahl der Fehler pro Variante als Auswahlkriterium herangezogen. Die Variante mit der geringsten Fehleranzahl wird als Übereinstimmung betrachtet.

5.3 Bewertung

Während des Vergleichs von Studentenmodell und Mustermodell werden sämtliche Abweichungen erfasst. Als Ergebnis des Vergleichs wird eine Liste mit allen Abweichungen zurückgegeben. Diese Liste ist die Basis für die Bewertung des Studentenmodells. Eine Abweichung dokumentiert einen Fehler, der vom Expertenmodul entsprechend bewertet wird. Für jedes Mustermodell muss vom Ersteller eine maximale Punkteanzahl vorgegeben werden.

Die Bewertung des abgegebenen Studentenmodells erfolgt über einen Punkteabzug für jeden entdeckten Fehler. Zunächst erscheint es sinnvoll die Fehlerquellen bzw. –möglichkeiten systematisch zu erfassen. Neben den klassischen Fehlern, wird auch zusätzlich noch die Möglichkeit geboten, für Varianten die zwar richtig sind, aber im Kontext der Aufgabenstellung nicht optimal sind, Strafpunkte zu vergeben. Bei der Verwendung von bewusst falschen Varianten, sollten bei dieser Variante die Punkteabzüge für Elemente dieser falschen Variante mit 0 belegt werden, und nur ein Gesamtpunkteabzug für die Variante vergeben werden. Dadurch wird ver-

hindert, dass Abweichungen von der falschen Variante, die durchaus korrekt sein können, als Fehler bewertet werden.

Ist ein erwartetes Element vorhanden, dann können Subelemente entweder fehlen oder falsch sein. Es können auch Elemente im Studentenmodell vorkommen die nicht erwartet wurden. Solche Elemente werden als ‚nicht erwartet' bewertet. Einige Elemente besitzen auch spezielle, beschreibende Attribute, diese können ebenfalls fehlen oder falsche Werte besitzen.

Abbildung 5-19: Fehlerklassen und ihre Verwendung

Element	notoptimal	missing	class_notexpected	attribute_notexpected	associationto_notexpected	subclass_notexpected	superclass_notexpected	wrongtype	wrongcardinality
Alternative	X		X						
Class / AssociationClass		X		X	X	X	X		
Attribute		X							
Subclass / Superclass		X							
AssociationTo		X						X	X

Es ergeben sich also mehrere Kategorien in die Unterschiede unterteilt werden können (siehe auch Abbildung 5-19):

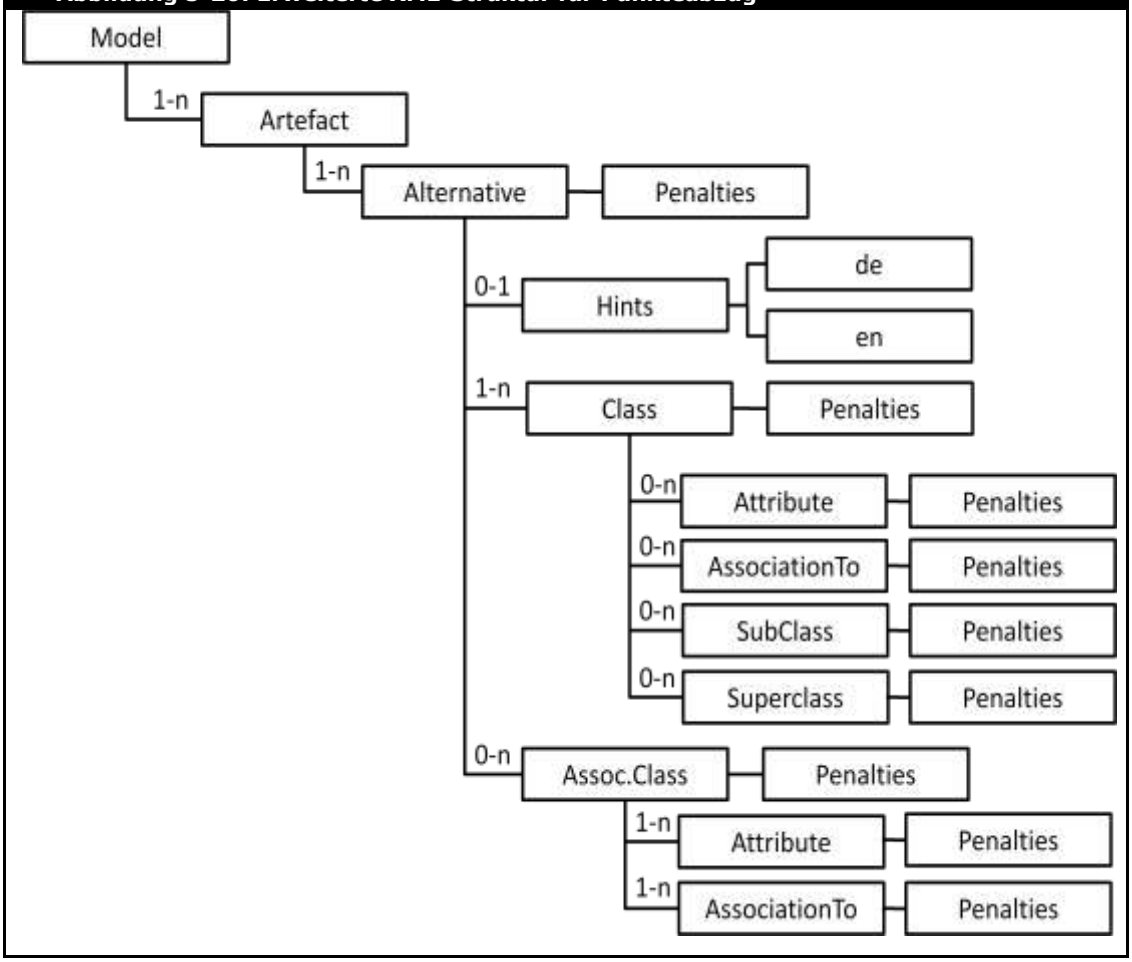
Nicht optimal: Im Mustermodell können Varianten mit Punkteabzügen gekennzeichnet werden, falls sie zwar korrekt sind, im gegebenen Kontext aber als nicht optimal betrachtet werden. Falls der Student eine solche Variante wählt, wird dieser Punkteabzug dem Strafkonto zugezählt.

fehlend: Ein Element fehlt, wenn es im Studentenmodell nicht gefunden werden kann. Da die Prüfung über das Attribut Name des Elements erfolgt, kann auch dann ein Element nicht gefunden werden, wenn der Name nicht aus dem Namensraum gewählt worden ist.

Nicht erwartet: Im Studentenmodell befinden sich Elemente, die im Mustermodell nicht vorhanden sind. Diese zusätzlichen Elemente werden als falsch gewertet.

falsch: Ein Element ist dann falsch, wenn die zusätzlichen beschreibenden Attribute über andere Werte verfügen als im Mustermodell vorgegeben ist (zB Typ oder Multiplizität).

Abbildung 5-20: Erweiterte XML-Struktur für Punkteabzug



Manche Fehlerkategorien werden nur bei bestimmten Elementen benötigt, das heißt es sind nicht alle Fehlerkategorien bei allen Elementen vorhanden. Die Fehlerkategorie „Nicht optimal" kommt zum Beispiel nur bei der „Alternative" zur Anwendung. Bei der Bewertung wird zuerst die Fehlerkategorie bestimmt und danach anhand des beteiligten Elements der Punkteabzug aus dem Mustermodell ausgelesen. Für jede Kombination aus Fehlerkategorie und Element wird auch ein Standard-Wert für den Punkteabzug verwaltet.

Um jedoch eine individuelle Gestaltung der zu vergebenden Punkte bzw. des Punkteabzugs zu ermöglichen, wird die Struktur des XML-Formats nochmals erweitert. Für jeden Elementknoten und auch für die „Variant"-Knoten wird ein Knoten „penalties" eingefügt (Abbildung 5-20). Dieser Knoten befindet sich innerhalb des jeweiligen Knotens für den die individuellen Punkteabzüge verwaltet werden sollen. Die Punkteabzüge werden als XML-Attribute verwaltet. In Abhängigkeit vom umschließenden Element sind unterschiedliche XML-Attribute vorgesehen.

Abbildung 5-21: XML-Attribute für Verwaltung von Strafpunkten

XML-Attribut	Knoten	Bedeutung
notoptimal	alternative	Die gewählte Modellierungsvariante ist nicht optimal bzw. falsch
class_notexpected	alternative	Eine modellierte Klasse ist im Mustermodell nicht vorhanden
missing	class, associationClass, associationTo, attribute, subclass, superclass	Ein Element aus dem Mustermodell ist im Studentenmodell nicht modelliert worden.
attribute_notexpected	class, associationClass	Ein Attribut einer Klasse im Studentenmodell ist im Mustermodell nicht vorhanden
Associationto_notexpected	class,	Eine Beziehung des Stu-

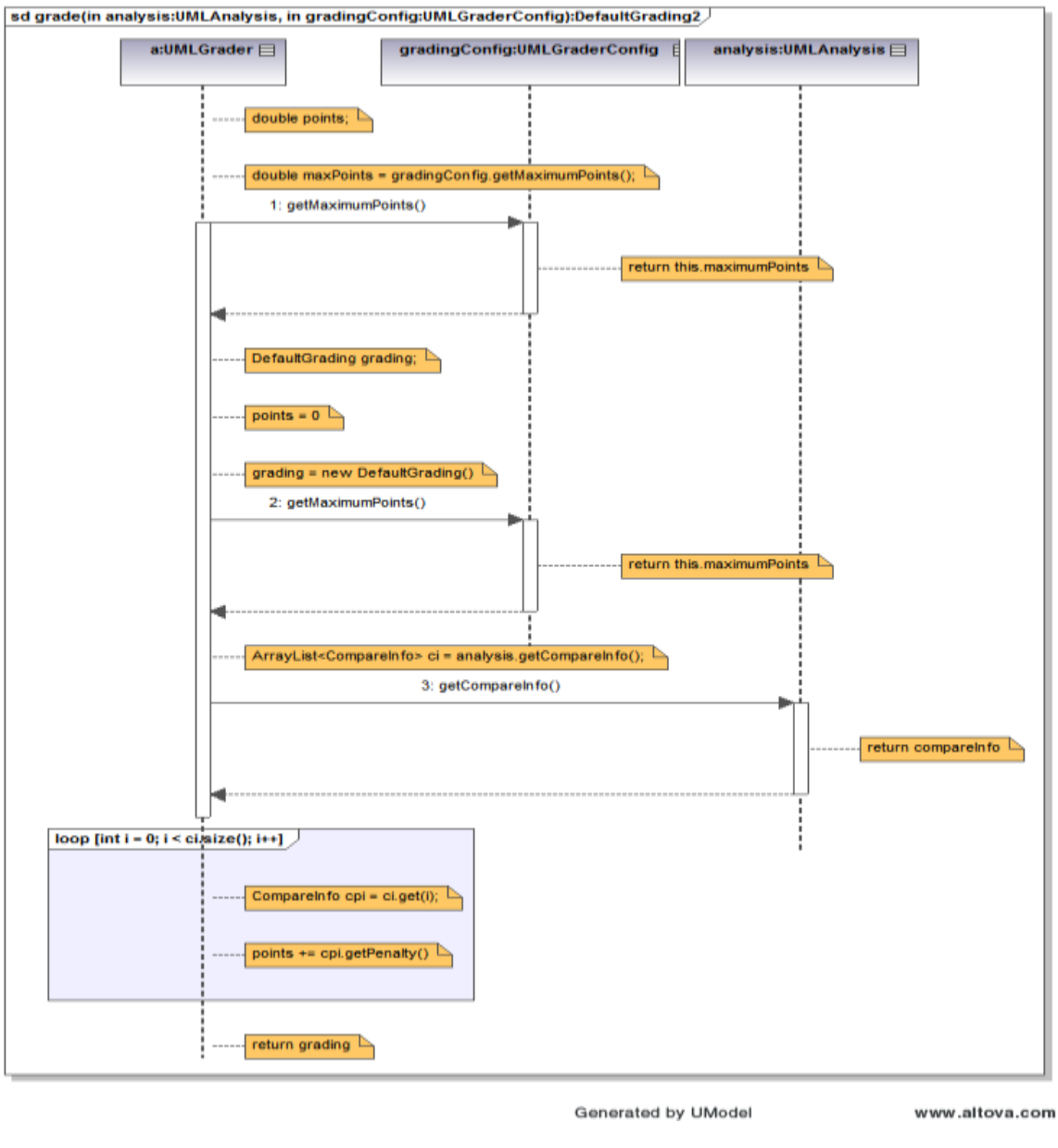
	associationClass	dentenmodells ist im Mustermodell nicht vorhanden
subclass_notexpected	class	Eine Generalisierungsbeziehung aus dem Studentenmodell ist im Mustermodell nicht vorhanden
superclass_notexpected	class	Eine Generalisierungsbeziehung aus dem Studentenmodell ist im Mustermodell nicht vorhanden
wrongCardinality	associationTo	Die Multiplizitätsangabe im Studentenmodell entspricht nicht der aus dem Mustermodell
wrongType	associationTo	Der Beziehungstyp im Studentenmodell entspricht nicht dem des Mustermodells

Für die gesamte Bewertung wird ein „Fehlerkonto" verwaltet. Für jeden Sachverhalt werden die Punkteabzüge aufsummiert, und zum Abschluss der gesamte Punkteabzug der Sachverhalte im Modell aufsummiert. Die Benotung erfolgt danach über die Anzahl von verbliebenen Punkten nach dem Punkteabzug. Je mehr Punkte abgezogen werden umso schlechter die Note. Die maximale Punkteanzahl muss bei der Erstellung der Aufgabe natürlich entsprechend der Größe des Modell und somit der Fehlermöglichkeiten vergeben werden.

Abbildung 5-22 zeigt das Sequenzdiagramm für die Durchführung der Bewertung. Durch den Vergleichslauf werden Informationen über die Fehler gesammelt. Dabei werden auch die in den Knoten gespeicherten Punkteabzüge mit ausgelesen und im Fehlerelement mitgespeichert. Es werden dabei die Punkteabzüge ausgelesen, die zum jeweiligen Fehler gehören und im Mustermodell hinterlegt sind. Durch die Kategorisierung der Fehler und die Kenntnis des betroffenen Elementes ist dies einfach möglich. Beim Berechnen des Gesamtergebnisses werden die Punkteabzüge aus den Fehlerobjekten addiert und von der maximalen Punkteanzahl abgezogen.

Die niedrigste Punkteanzahl ist nach unten mit 0 begrenzt, damit eine negative Punkteanzahl verhindert wird.

Abbildung 5-22: Sequenzdiagramm Bewertung



5.4 Feedback

Das E-Tutor System unterscheidet zwischen untersuchen und abgeben einer Aufgabe. Dafür werden auf der Web-Seite Schaltflächen angeboten. Beim Untersuchen soll das Studentenmodell ana-

lysiert und das Feedback angezeigt werden. Eine Bewertung des Studentenmodells soll erst bei der Abgabe erfolgen. Ein Studentenmodell kann während der zur Verfügung stehenden Zeit beliebig oft untersucht werden. Das Rückmelden von Fehlern bzw. das geben von Feedback ist in dieser Phase des Lernens besonders wichtig. Durch gezieltes Feedback soll der Lernende in die Lage versetzt werden seine Fehler zu erkennen und entsprechend der Erkenntnis bei wiederholtem Lösen der Aufgabe diese Fehler nicht mehr machen.

Um den Lerneffekt zu verstärken werden verschiedene Hinweisstufen angeboten. Während bei der höchsten Hinweisstufe sämtliche Information detailliert gegeben wird, ist bei niedrigeren Stufen nur eine Zusammenfassung der Fehler vorgesehen (Abbildung 5-23). Das Sammeln der Fehler wird während der Analyse durchgeführt. Aufgrund der Fehlerkategorie und des beteiligten Elements wird der Feedback-Text zusammengestellt. Die Hinweise werden, so wie die Punkteabzüge, im „Fehlerkonto" gesammelt.

Abbildung 5-23: Detailierungsgrad verschiedener Hinweisstufen

Hinweisstufe	Modus	Detailierungsgrad
HW-Stufe1	Abgabemodus	Keine Fehler- und Hinweistexte
HW-Stufe2	Lernmodus	Komprimierte Fehler- und Hinweistexte
HW-Stufe3	Lernmodus	Detaillierte Fehlertexte und Hinweise
HW-Stufe4	Lernmodus	Lösung

Durch die Kategorisierung der Fehler in Verbindung mit dem Element in dem der Fehler gefunden wurde, lässt sich für die Fehler- bzw. Hinweistexte ein System erstellen. Abbildung 5-24 zeigt die möglichen Texte. Diese sind so allgemein gehalten, dass die Texte

automatisch generiert werden können. Ähnlich wie bei den Punkteabzügen könnten auch spezielle Hinweistexte innerhalb des Mustermodells beim jeweiligen Knoten gespeichert werden. Dies erhöht jedoch den Aufwand beim Erstellen einer Aufgabe beträchtlich. Aus diesem Grund wird dieser Ansatz vorerst nur bei den Knoten „Variant" vorgesehen. Es kann aber in einer späteren Version ohne großen Aufwand auch für andere Knoten nachimplementiert werden, falls dies erforderlich ist. Durch die hohe Aussagekraft der Standardfehlermeldungen erscheint dies jedoch nicht erforderlich. Um die Hinweistexte mit Kontextinformationen verknüpfen zu können werden Platzhalter verwendet (zB $0). Die konkreten Kontextinformationen werden beim Analysieren eines Modells in den Fehlerobjekten gespeichert.

Abbildung 5-24: System der Rückmeldungen

#	Fehler	Element	Text HW-Stufe1	Text HW-Stufe 2	Text HW-Stufe 3
1	notoptimal	alternative	Modell beinhaltet nicht optimale Modellierung	Die Modellierungsvariante mit der Klasse/den Klassen $0 ist nicht optimal	Individueller Hinweistext
2	missing	class	Im Modell fehlen benötigte Elemente	Eine benötigte Klasse wurde nicht modelliert	Die Klasse $0 wurde nicht modelliert
3	missing	Associationclass	Im Modell fehlen benötigte Elemente	Eine benötigte Assoziationsklasse wurde nicht modelliert	Die Assoziationsklasse $0 wurde nicht modelliert
4	missing	Attribute	Im Modell fehlen benötigte Elemente	Benötigte Attribute einer Klasse wurden nicht modelliert	Das benötigte Attribut $0 der Klasse $1 wurde nicht modelliert
5	missing	Subclass	Im Modell fehlen benötigte Elemente	Im Modell fehlt eine Generalisierungsbeziehung	Die Generalisierungsbeziehung zwischen Basisklasse $0 und Kindklasse $1 fehlt
6	missing	Superclass	Im Modell fehlen benötigte Elemente	Im Modell fehlt eine Generalisierungsbeziehung	Die Generalisierungsbeziehung zwischen Basisklasse $0 und Kindklasse $1 fehlt
7	missing	AssociationTo	Im Modell fehlen benötigte Elemente	Im Modell fehlt eine Assoziation	Die Assoziation zwischen den Klassen $0 und $1 fehlt
8	class_notexpected	alternative	Im Modell wurde ein nicht benötigtes Element gefunden	Im Modell wurde eine nicht benötigte Klasse gefunden	Die Klasse $0 wurde nicht erwartet
9	attribute_notexpected	class	Im Modell wurde ein nicht benötigtes Element gefunden	Im Modell wurde ein nicht benötigtes Attribut gefunden	Das Attribut $0 der Klasse $1 wurde nicht erwartet

10	attribute_notexpected	Associationc-lass	Im Modell wurde ein nicht benötigtes Element gefun-den	Im Modell wurde ein nicht benötigtes Attri-but gefunden	Das Attribut $0 der Assozia-tionsklasse $1 wurde nicht erwartet
11	association-to_notexpected	class	Im Modell wurde ein nicht benötigtes Element gefun-den	Im Modell wurde eine nicht benötigte Assozi-ation gefunden	Die Assoziation zwischen den Klassen $0 und $1 wurde nicht erwartet
12	association-to_notexpected	Associationc-lass	Im Modell wurde ein nicht benötigtes Element gefun-den	Im Modell wurde eine nicht benötigte Assozi-ation gefunden	Die Assoziation zwischen der Assoziationsklasse $0 und der Klasse $1 wurde nicht erwartet
13	subclass_notexpected	class	Im Modell wurde ein nicht benötigtes Element gefun-den	Eine Spezialisierung wurde nicht erwartet	Eine Spezialisierung von der Basisklasse $0 zur Klasse $1 wurde nicht erwartet
14	superclass_notexpected	class	Im Modell wurde ein nicht benötigtes Element gefun-den	Eine Generalisierung wurde nicht erwartet	Eine Generalisierung von der Klasse $0 zur Basisklasse $1 wurde nicht erwartet
15	wrong_type	AssociationTo	Das Modell enthält Fehler	Eine Assoziation hat einen falschen Typ	Die Assoziation zwischen den Klassen $0 und $1 hat den falschen Typ
16	wrong_cardinality	AssociationTo	Das Modell enthält Fehler	Die Multiplizität einer Assoziation ist falsch	Die Multiplizität der Assozia-tion zwischen den Klassen $0 und $1 ist falsch

Für die Hinweisstufe 1 werden nur allgemeine Informationen ausgegeben, bei der Hinweisstufe 2 werden diese um die Elementinformation ergänzt. Die Hinweisstufe 3 erhält schließlich detaillierte Informationen über den Fehler. Damit können die Fehlertexte bzw. Hinweise einfach generiert werden. Die Hinweisstufe 4 stellt die Lösung dar, und ist nicht implementiert. Sie kann aber bei Bedarf durch das Anzeigen der ursprünglichen grafischen Ausarbeitung des Mustermodells leicht hinzugefügt werden. Durch die hohe Detailliertheit der Stufe 3 erscheint dies zum jetzigen Zeitpunkt jedoch nicht erforderlich.

Spezifische Hinweistexte bei Varianten

Zusätzlich zur automatischen Generierung von Feedback-Text, wird bei jedem Variantenknoten die Möglichkeit geboten einen alternativen und individuellen Feedback-Text zu erfassen. Gerade bei nicht korrekten Modellierungsvarianten ist ein entsprechender Hinweis

eine wertvolle Information für den Studenten. Dieses gezielte Feedback soll den Lernerfolg erhöhen, indem der Student nicht nur seinen Fehler erkennen soll sondern auch eine Information erhält warum diese Variante nicht korrekt ist. Dadurch soll eine Problemlösungskompetenz erreicht werden, so dass bei ähnlichen Modellierungsaufgaben dieser Fehler nicht mehr gemacht wird.

Abbildung 5-25: XML-Schema mit Hinweistexten für Varianten

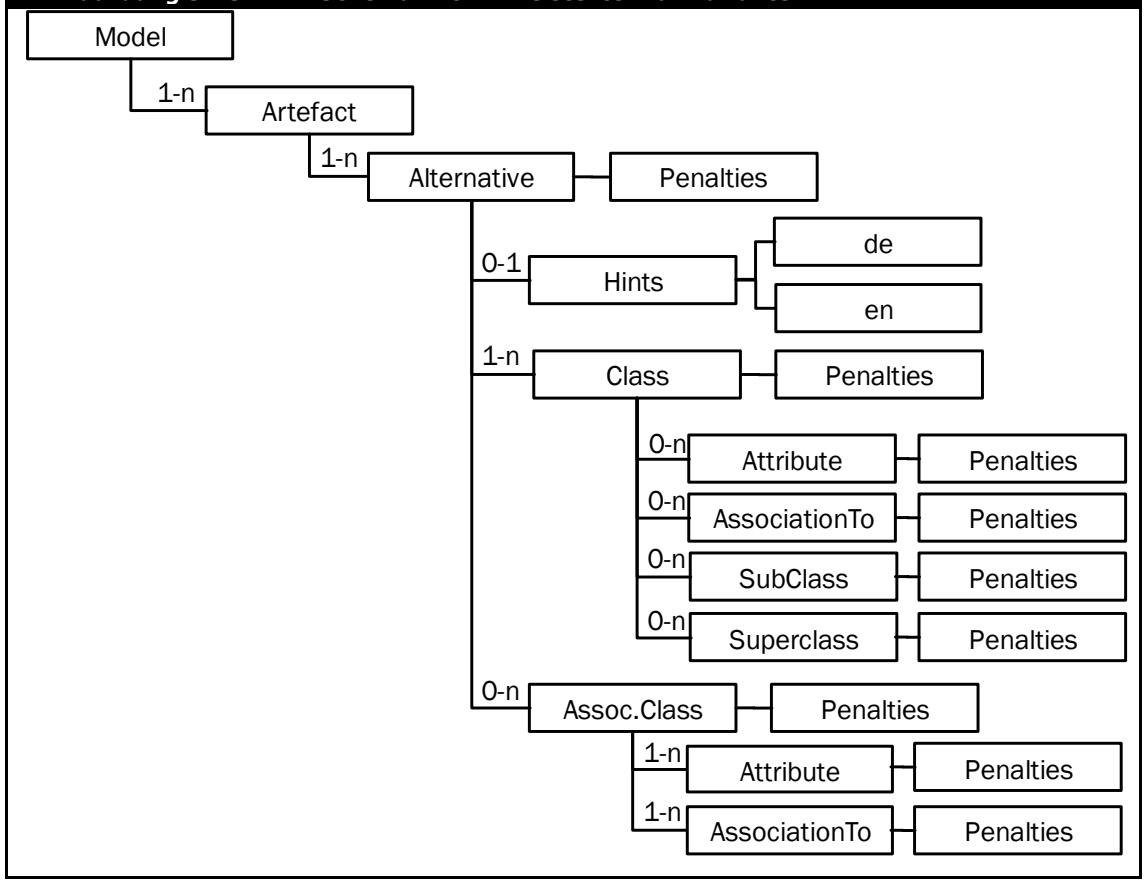
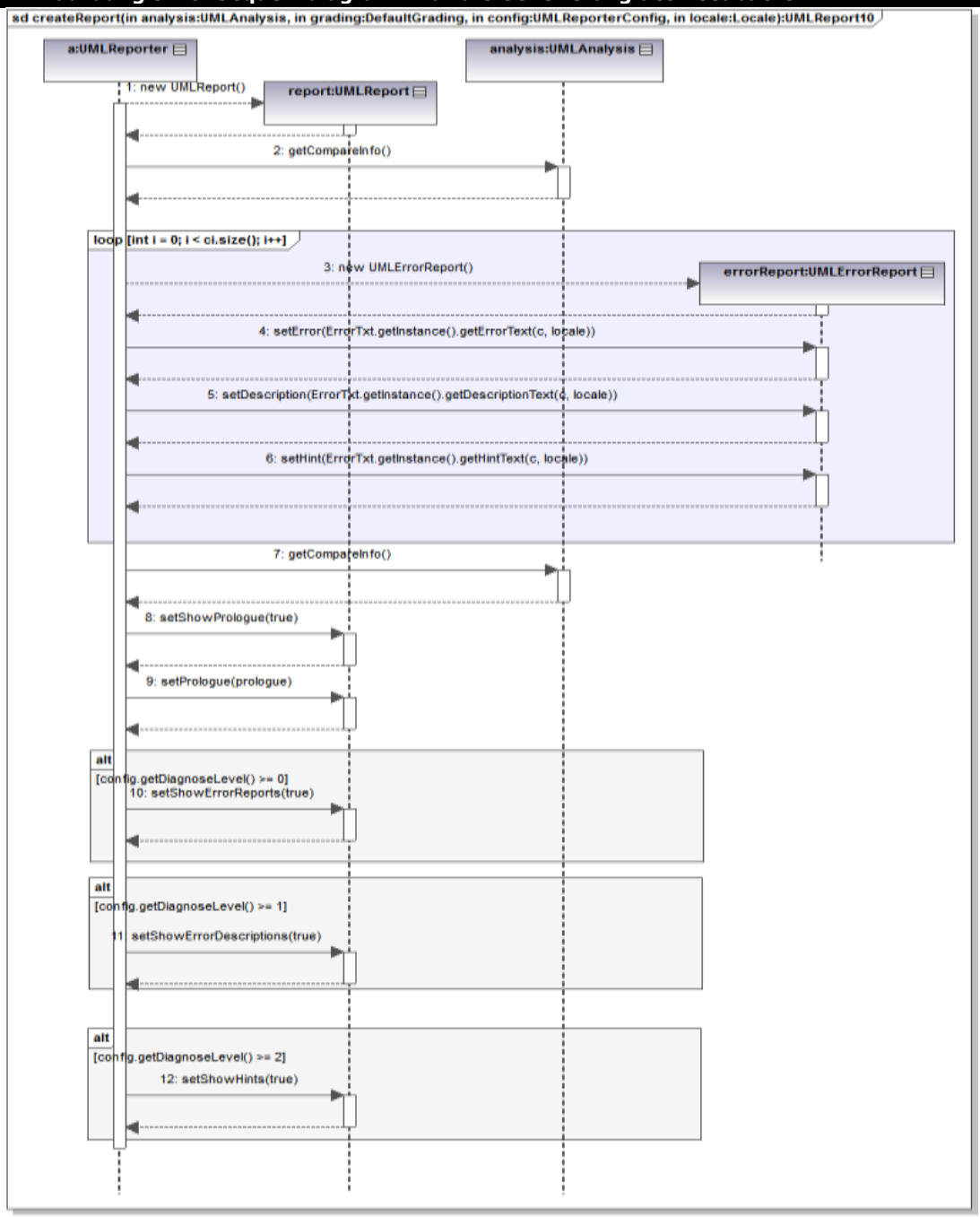


Abbildung 5-25 zeigt das XML-Schema zur Speicherung des Expertenwissens mit den eingefügten Knoten für die individuellen Hinweistexte. Diese Texte werden, falls sie erfasst wurden, zur Rückmeldung an den Studenten verwendet. Zur Unterscheidung der verwendeten Sprache müssen die Texte innerhalb eines Knotens mit der Sprachkennung (also etwa „de" für Deutsch) eingetragen werden.

Abbildung 5-26 zeigt ein Sequenzdiagramm zur Ermittlung der Anzeigetexte. Die Textvorlagen werden über eine statische Klasse bereitgestellt, und nach Sprache getrennt zurückgegeben. Die Platzhalter werden danach mit den Kontextinformationen aus den Fehlerobjekten ersetzt.

Abbildung 5-26: Sequenzdiagramm für die Generierung des Feedbacks



Beispielmodell und angepasstes Expertenwissen

Abbildung 5-27 zeigt ein Studentenmodell zur Lösung der Übungsaufgabe. Es wurden einige der verschiedenen Modellierungsvarianten bzw. Fehler in dieses Modell eingebaut. Damit diese Fehler vom Expertenmodul entdeckt werden können müssen diese auch im abgespeicherten Expertenwissen, also im Mustermodell, vorhanden sein. Die Beziehung zwischen den Klassen „Person" und „Pruefer" wurde zum Beispiel als Komposition modelliert. Da es sich hier aber bei der Klasse Pruefer eindeutig um eine Spezialisierung der Klasse Person handelt, ist diese Modellierungsvariante falsch.

Abbildung 5-27: Ein Studentenmodell mit Fehlern

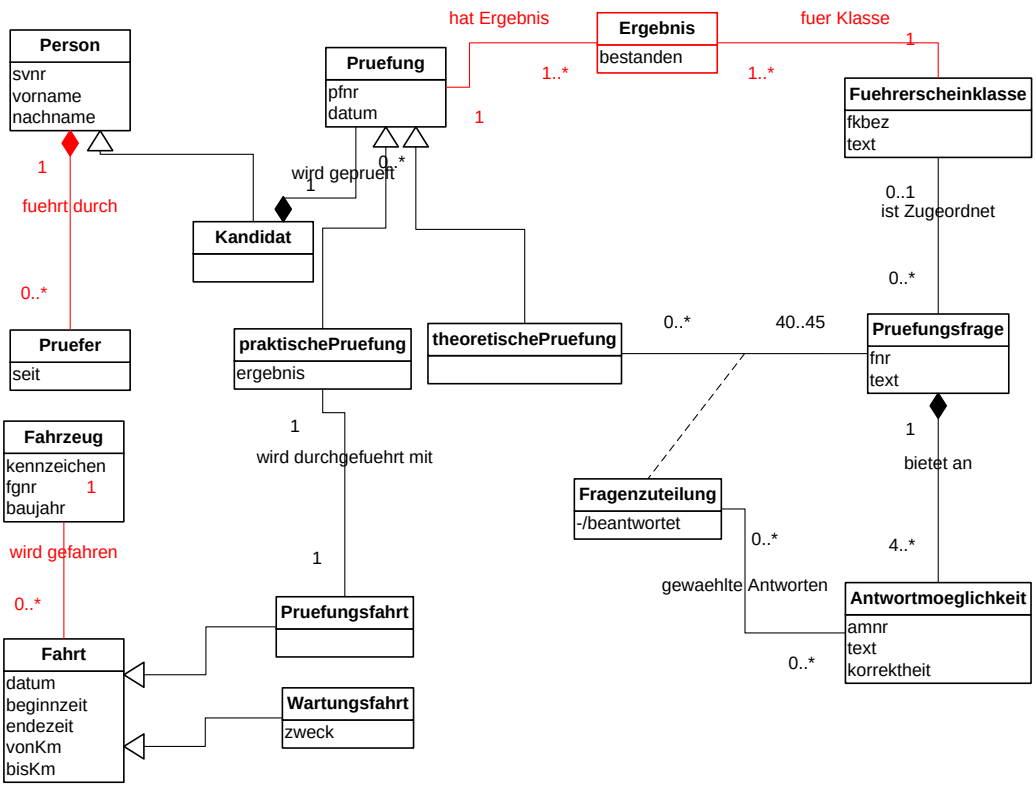


Abbildung 5-28 zeigt den relevanten Teil des Expertenwissens, mit der abgespeicherten Variante für die Klasse „Person". Bei der Variante mit der Nummer 2 ist die Beziehung zur Klasse „Pruefer" als Komposition (associationTo mit Type=composite) eingetragen. Zusätzlich wurde ein erklärender Hinweistext (Knoten hints) in den Sprachen Deutsch (de) und Englisch (en) hinzugefügt.

Abbildung 5-28: Teil des Expertenwissen mit Varianten

```
<?xml version="1.0" encoding="UTF-8"?>
<model exerciseId="0" name="Fuehrerschein">
  <artefact id="1">
    <alternative id="1">
      <penalties class_notexpected="5.0" notoptimal="0.0"/>
      <class name="Person">
        <penalties associationto_notexpected="2.5"
          attribute_notexpected="1.0" missing="5.0"
          subclass_notexpected="2.0" superclass_notexpected="2.0"/>
        <attribute name="svnr">
          <penalties missing="1.0"/>
        </attribute>
        <attribute name="vorname">
          <penalties missing="1.0"/>
        </attribute>
        <attribute name="zuname">
          <penalties missing="1.0"/>
        </attribute>
        <subClass name="Kandidat">
          <penalties missing="5.0"/>
        </subClass>
        <subClass name="Pruefer">
          <penalties missing="5.0"/>
        </subClass>
      </class>
    </alternative>
    <alternative id="2">
      <penalties class_notexpected="5.0" notoptimal="5.0"/>
      <hints>
 <de>Die Klasse 'Person' ist eine Generalisierung von 'Pruefer'. Daher
sollte die Beziehung als Generalisierung modelliert werden.</de>
 <en>Class 'Person' is a generalization of 'Pruefer'. Therefore rela-
tionship should be modeled as a generalization.</en>
      </hints>
      <class name="Person">
        <penalties associationto_notexpected="2.5"
          attribute_notexpected="1.0" missing="5.0"
          subclass_notexpected="2.0" superclass_notexpected="2.0"/>
        <attribute name="svnr">
          <penalties missing="1.0"/>
        </attribute>
        <attribute name="vorname">
          <penalties missing="1.0"/>
        </attribute>
        <attribute name="zuname">
          <penalties missing="1.0"/>
        </attribute>
        <associationTo lower="0" roleName="fuehrt durch" toC-
lass="Pruefer"
          type="composite" upper="-1">
          <penalties missing="2.5" wrongCardinality="1.0" wrong-
Type="1.5"/>
        </associationTo>
        <subClass name="Kandidat">
          <penalties missing="5.0"/>
        </subClass>
```

```
      </class>
    </alternative>
  </artefact>
```

5.5 Anwendung der Vergleichskriterien für ITS

Im Kapitel 3 - Stand der Technik wurden Kriterien zur Beurteilung von Tutoring Systemen erarbeitet. Diese Kriterien werden nun herangezogen um auch das UML-Expertenmodul (UML-Tutor) zu beurteilen. Es zeigt sich, dass einige der geforderten Kriterien erfüllt werden können. Die nicht erfüllten Punkte sind jene Punkte die, mit Ausnahme des grafischen Editors, auch von anderen Systemen nicht oder nur zum Teil erfüllt werden. Für die eigene Benennung der Elemente gibt es ebenfalls noch kein geeignetes Konzept.

Kriterium		Begründung
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	😐	Die Aufgaben werden von den Lehrveranstaltungsleitern für die Teilnehmer nach der Behandlung des Themas im Präsenzteil (Blended Learning) ausgegeben. Es gibt sonst keine Berücksichtigung von Vorkenntnissen.
Werden Fragen zum Lerninhalt beantwortet	😐	Es können keine Fragen an das System gestellt werden. Moodle ermöglicht allerdings Forumsdiskussionen.
Wird jede Lösung erklärt	☺	Für Fehler in Elementen wird ein Standard-Fehlertext generiert. Bei Varianten kann zusätzlich ein eigener individueller Hinweistext hinterlegt werden.
Erfolgt Erklärung benutzerbezogen	☹	Nein
Sind natürlich sprachliche Eingaben und Ausgaben möglich	☹	Es können keine Anfragen an das System gestellt werden.
Welcher Ansatz wird verfolgt (CBM vs. VLE)	VLE	Moodle ist eine Virtuelle Lernumgebung. Damit ist natürlich auch E-Tutor und das Expertenmodul diesem Konzept zuzurechnen.

Online vs. Offline Lösung	☺	Eingebettet in die Online Lernumgebung Moodle
Verfügbarkeit (Offen vs. Geschlossener Benutzerkreis)	☺	Für Studenten an der JKU Linz. Es gibt aber auch die Möglichkeit einzelne Lehreinheiten als Gast zu bearbeiten.
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☺	Ja. Es werden unterschiedlich detaillierte Feedbacktexte generiert. Durch die Angabe der höchsten angezeigten Feedbackstufe kann die Anzeige vom Lehrenden gesteuert werden.
Grafik-Editor für Modellierung	☹	Nein.
Werden fehlende Kenntnisse unterrichtet	😐	Nein. Durch den Einsatz im Blended Learning werden die Kenntnisse zuvor in den Präsenzteilen unterrichtet.
Wird das Verständnis getestet	☹	Nein. Das Verständnis kann durch die unterschiedlichen Hinweisstufen trainiert werden.
Sind alternative Lösungen möglich	☺	Ja. Modellierungsvarianten, welche im Mustermodell vorhanden sind, können bei der Analyse berücksichtigt werden.
Wird eigene Benennung der Elemente unterstützt	☹	Nein. Die Namen von Elementen müssen aus einer Liste von vorgegebenen Namen gewählt werden.

In Abbildung 5-29 wird nochmals die Tabelle aus dem Kapitel 3.6 gezeigt, jetzt jedoch mit den bewerteten Kriterien für UML-Expertenmodul (UML-Tutor).

Abbildung 5-29: Vergleich UML-Tutor mit bestehenden Systemen

Kriterium	COLLECT-UML	COLER	KERMIT	EER-Tutor	UML-Tutor
Werden Vorkenntnisse bei der Auswahl der Übungen berücksichtigt	😐	☹	☺	☺	😐

Werden Fragen zum Lerninhalt beantwortet	☹	😐	☹	☹	😐
Wird jede Lösung erklärt	☹	😐	😐	☺	☺
Erfolgt Erklärung benutzerbezogen	☹	😐	😐	😐	☹
Sind natürlich sprachliche Eingaben und Ausgaben möglich	☹	😐	☹	☹	☹
Welcher Ansatz wird verfolgt (CBM vs. VLE)	CBM	VLE	CBM	CBM	VLE
Online vs. Offline Lösung	☺	☺	☹	☺	☺
Verfügbarkeit (Offen vs. Geschlossener Benutzer-kreis)	😐	☹	😐	😐	☺
Gibt es eine mehrstufige Erklärung? (Hinweis bis vollständige Lösung)	☺	☹	☺	☺	☺
Grafik-Editor für Modellierung	☺	☺	☺	☺	☹
Werden fehlende Kenntnisse unterrichtet	☹	☺	☹	☹	😐
Wird das Verständnis getestet	☺	☺	☺	☺	☹
Sind alternative Lösungen möglich	☹	☺	😐	😐	☺
Wird eigene Benennung der Elemente unterstützt	😐	☺	☺	☺	☹

Der Vergleich zeigt, dass das UML-Expertenmodul (UML-Tutor) wesentliche Kriterien erfüllt. Daneben kann die Unterstützung für einige der derzeit fehlende Kriterien, wie etwa der Grafikeditor, in einer späteren Ausbauphase hinzugefügt werden. Für die eigene Benennung von Elementen ist derzeit noch kein Konzept umgesetzt. Natürlich sprachliche Eingabe bzw. Ausgaben sowie benutzerbezogene Erklärungen sind nicht vorgesehen.

6 Umsetzung des UML-Expertenmoduls

Dieses Kapitel gibt einen Überblick über die Umsetzung der Konzepte bei der Erstellung des Prototyps für das UML-Expertenmodul. Es wird die Entwicklungsumgebung, die Einbettung des Prototyps in das bestehende E-Tutor System, sowie der Aufbau und die Strukturierung des Java Source-Codes beschrieben. Abschließend werden noch die wichtigsten Algorithmen des UML-Expertenmoduls beschrieben. Die Installation und Konfiguration des UML-Expertenmoduls wird in Anhang B beschrieben.

6.1 Entwicklungsumgebung

Das E-Tutor System wurde in Java entwickelt, deshalb ist diese Programmiersprache, auch wegen der vorgegebenen Schnittstellen zwingend erforderlich. Während der Entwicklung empfiehlt es sich, eine lokale (Entwicklungs-)Version des E-Tutor Systems zu verwenden. Das E-Tutor System wurde in die Lernumgebung Moodle integriert, deshalb wird zusätzlich auch dieses System benötigt. Moodle benötigt eine relationale Datenbank, sowie einen Web-Server mit Unterstützung für PHP[21]. Das Paket XAMPP[22] bietet sich für diese Zwecke an, da es die notwendigen Programme beinhaltet, sowie über ein einfaches Installationsprogramm verfügt. Die benötigte Datenbankstruktur kann über ein SQL[23]-Skript erzeugt werden. Die eigentliche Applikation muss im htdocs-Verzeichnis des Web-Servers bereitgestellt werden.

---

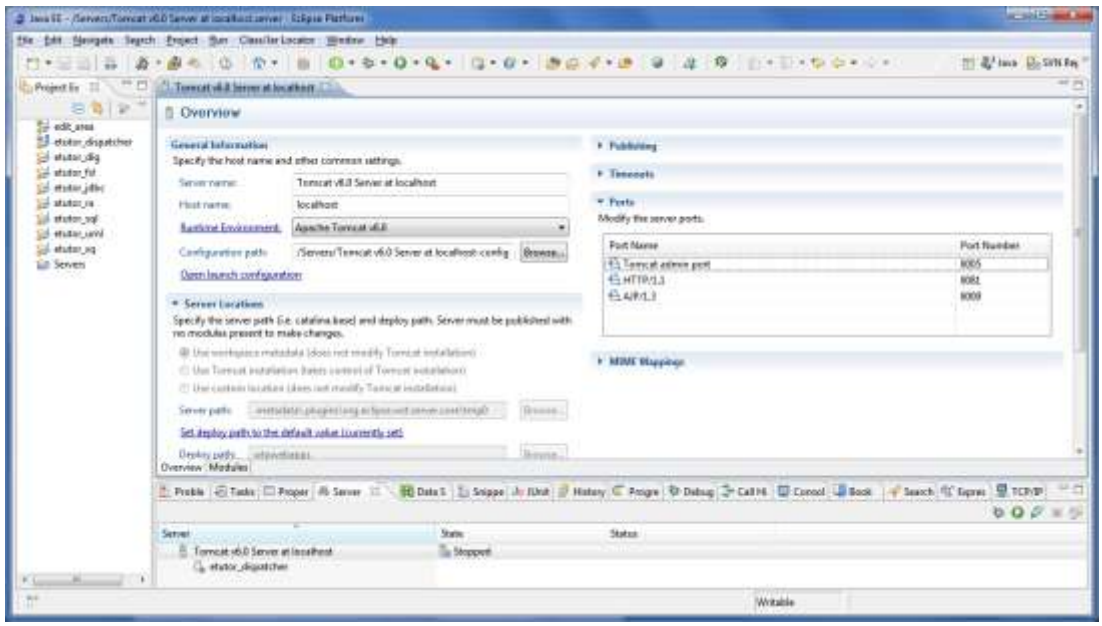
[21] PHP ist ein Akronym für Hypertext Preprocessor, eine Skript Sprache für Web-Entwicklung: siehe http://www.php.net/

[22] Siehe http://www.apachefriends.org/de/xampp.html

[23] SQL – Structured Query Language: normierte Datenbanksprache zur Abfrage, Manipulation und Definition von Daten in relationalen Datenbanken

Das E-Tutor System benötigt selbst einen eigenen Web-Server. Dazu kann die integrierte Tomcat[24] Umgebung der Entwicklungsumgebung Eclipse[25] verwendet werden. Der Source Code des bestehenden E-Tutor Systems inklusive der Projekt Dateien für Eclipse ist auf einem Server des DKE-Instituts verfügbar. Eine genaue Beschreibung der notwendigen Installations- und Konfigurationsschritte findet sich in (Fischer, 2010).

Abbildung 6-1: Entwicklungsumgebung Eclipse mit integriertem Webserver



6.2 Strukturierung des Java Source Codes

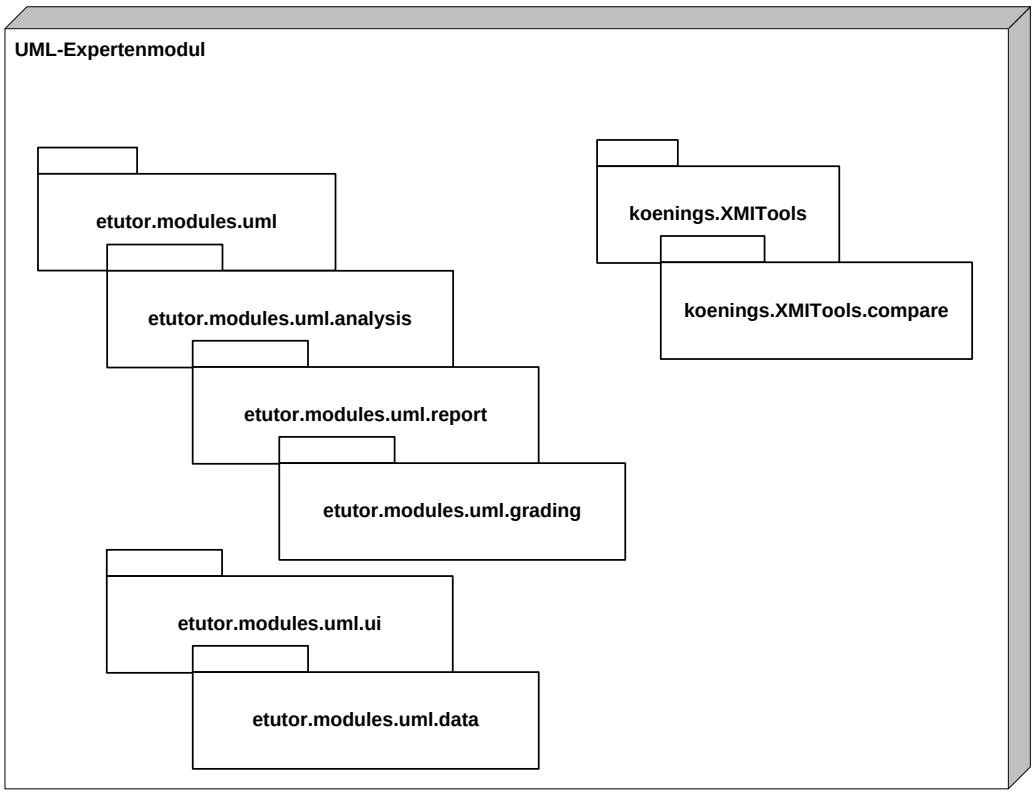
Bei der Strukturierung des Java Projektes wird zwischen Benutzungsoberfläche, Implementierung der E-Tutor-Schnittstellen und eigentlicher Funktionalität unterschieden. Die Generierung der Webseiten für die Benutzungsoberfläche, erfolgt im E-Tutor System

[24] Siehe http://tomcat.apache.org

[25] Siehe http://www.eclipse.org

durch das Tool Velocity[26]. Die Quellcode Dateien für Velocity (Velocity Templates) müssen dem Projekt „etutor_dispatcher" im Java-Package „resources.uml" hinzugefügt werden. Für die Implementierung der Schnittstellen zum E-Tutor System wird die Namensgebung der bereits vorhandenen Module übernommen. Die Klassen befinden sich deshalb im Java-Package „etutor.modules.uml".

Abbildung 6-2: Strukturierung des Java Projekts



Dieses Java-Package wird weiter unterteilt in die Packages „analysis", „grading" und „report" für die Implementierung der notwendigen Klassen des „Evaluator" Interfaces. Für die Klassen zur Implementierung der Benutzungsschnittstelle wird das Unterpaket „UI" angelegt. Die Klassen für die Persistierung der Mustermodelle wer-

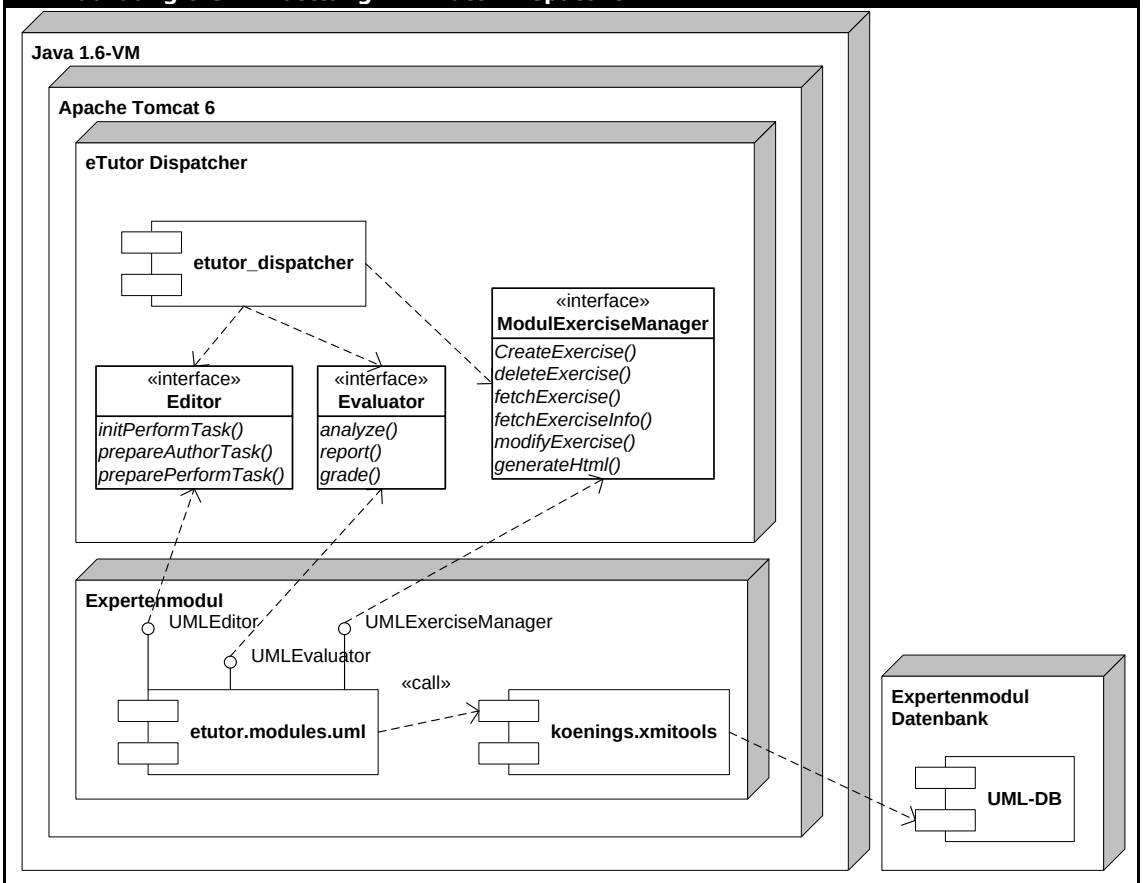
[26] Siehe http://velocity.apache.org

den im Unterpaket „Data" angelegt. Die für die eigentliche Verarbeitung der Modelle benötigten Klassen werden im Java-Package „koenings.XMITools" angelegt. Hier befinden sich alle Klassen, welche die benötigte Funktionalität zur Erstellung und Verwaltung der Muster- bzw. Studentenmodelle bereitstellt (Abbildung 6-2).

6.3 Einbettung des UML-Expertenmoduls in E-Tutor

Abbildung 6-3 zeigt die Einbettung des E-Tutor-UML-Moduls in das E-Tutor System. Im Wesentlichen müssen die vorgegebenen Java Interfaces implementiert werden.

Abbildung 6-3: Einbettung in E-Tutor Dispatcher



Die Namensgebung der Implementierungsklassen für die Schnittstellen des E-Tutor Systems folgt der Logik der bestehenden Modu-

le. Die Namen setzen sich aus dem Kürzel „UML" und dem Namen der Schnittstelle zusammen. Die Schnittstellen sind in Kapitel 4 beschrieben. Über diese Schnittstellen kommuniziert das UML-Expertenmodul mit dem E-Tutor System. Zur Speicherung des Expertenwissens (Mustermodell) ist eine Datenbank notwendig. Die Zugriffe auf diese Datenbank sind im Expertenmodul gekapselt. Die Verwaltung der Aufgaben wird über die Implementierung der Schnittstelle „ModulExerciseManager" erledigt. Für viele der in diesen Schnittstellen als Parameter übergebenen Objekte ist ebenfalls ein Java Interface vorhanden bzw. gibt es bereits eine Default-Implementierung im E-Tutor System. Die Installation, inklusive der notwendigen Einträge in Konfigurationsdateien wird in Anhang B beschrieben.

6.4 Algorithmen

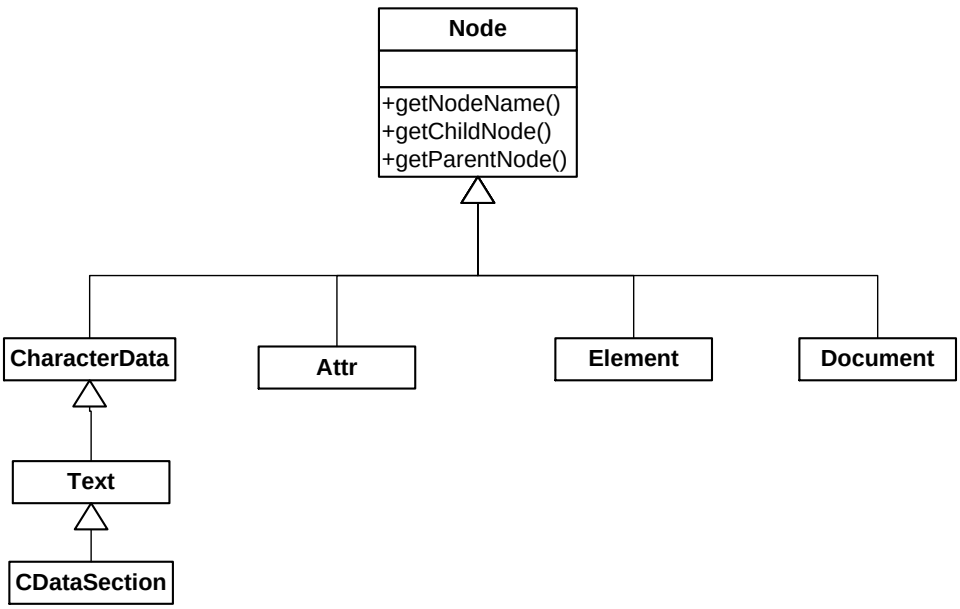
Nachfolgend werden die wichtigsten Algorithmen der Implementierung des UML-Expertenmoduls beschrieben. Die Beschreibung erfolgt aus einer technischen Sicht, so dass ein Software Entwickler die Implementierungsschritte nachvollziehen kann. Die Hauptaufgaben des UML-Expertenmoduls bestehen im Einlesen der Übergebenen XMI Dateien, der Verwaltung dieser Informationen, Funktionen zur Wartung der Mustermodelle (Expertenwissen) und schließlich der Analyse von Studenten- und Mustermodell inklusive Bewertung und Feedback-Erstellung.

6.4.1 Parsen der XMI Eingabedateien

Die zugrundeliegende Speicherstruktur von XMI ist XML. Dadurch kann für die Verarbeitung der XMI Dateien, auf bewährte Methoden für XML Dateien zurückgegriffen werden. Die bekanntesten Methoden zur Verarbeitung von XML sind das Document Object Model

(DOM[27]) und die Simple API for XML (SAX[28]). Für beide Methoden sind Implementierungen in der Form von Java Bibliotheken verfügbar. DOM ist vom World Wide Web Consortium Standardisiert worden. DOM bietet ein Objektmodel (Abbildung 6-4) zum Zugriff auf die Knoten in einem XML Dokument an. Bei Verwendung von DOM wird das gesamte XML Dokument in den Speicher geladen, was speziell bei größeren Dokumenten zu Problemen durch erhöhten Speicherbedarf führen kann.

Abbildung 6-4: Klassenmodell für DOM



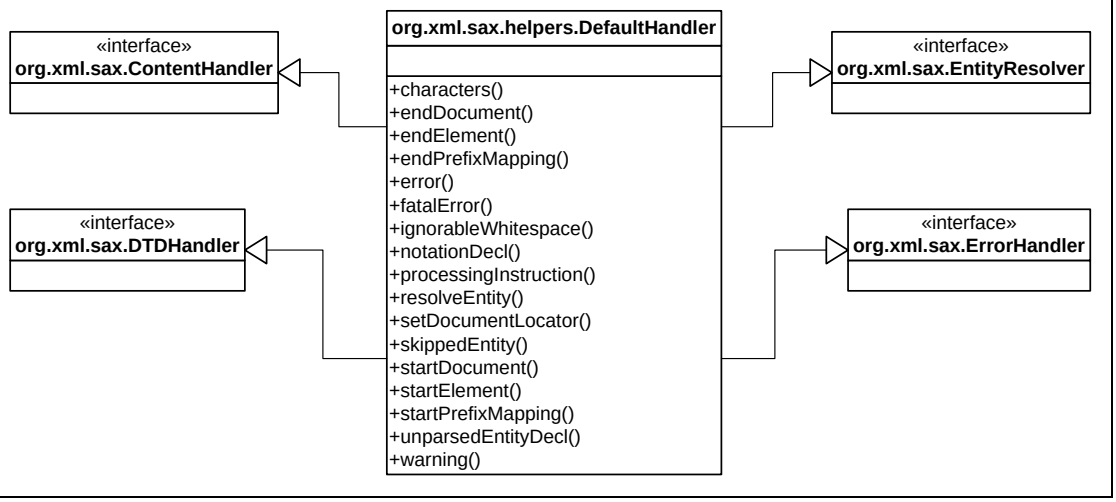
Bei SAX wird das Dokument beim Parsen von Knoten zu Knoten durchlaufen. Bei jedem Element wird eine Rückruffunktion (Callback Function) aufgerufen. Die Rückruffunktionen stellen einen generischen Rahmen für das knotenweise Lesen einer XML Datei zur Verfügung. Die SAX Bibliothek stellt mit der Klasse „DefaultHandler" (Abbildung 6-5) eine Basisimplementierung aller möglichen Rückruffunktionen bereit.

---

[27] Siehe http://www.w3.org/DOM

[28] Siehe http://sax.sourceforge.net/

Abbildung 6-5: Klasse DefaultHandler von SAX

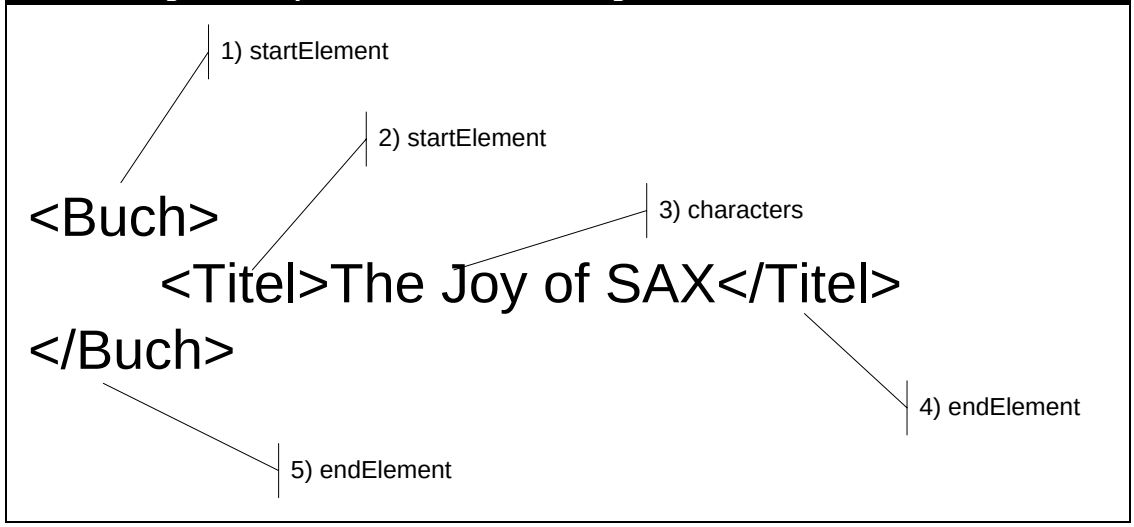


Diese Klasse wird als Basisklasse einer Vererbungshierarchie verwendet, und alle benötigten Methoden überschrieben. Dadurch kann die individuelle Verarbeitung der XML Datei erreicht werden.

Der Speicherbedarf von SAX ist wesentlich geringer als bei DOM, da hier nur jeweils ein Knoten in den Speicher geladen werden muss. Für das Parsen der XMI Dateien wird daher SAX verwendet. Bei der Ableitung von DefaultHandler müssen nur benötigte Funktionen überschrieben werden. Abbildung 6-6 zeigt ein Beispiel mit den verwendeten Methoden und bei welchen Knoten sie jeweils aufgerufen werden. Die XML Attribute der Knoten werden bei der „startElement" Methode als Aufrufparameter übergeben.

Abbildung 6-6 zeigt auch die Reihenfolge der Aufrufe von Verarbeitungsfunktionen durch SAX an einem einfachen XML Beispiel. Beim Parsen der XMI Struktur muss immer ein Verweis auf das Vaterelement des gerade zu verarbeitenden Elements gespeichert werden. Die Interpretation des Inhalts ist erst durch diese Kontextinformation möglich, da beispielsweise der XML Knoten „<XML:Class>" nicht nur zur Beschreibung von Klassen sondern auch zur Verwendung als Referenz auf Klassen verwendet wird.

Abbildung 6-6: Beispiel für verwendete Parsing Funktionen



6.4.2 Verwalten der eingelesenen Information

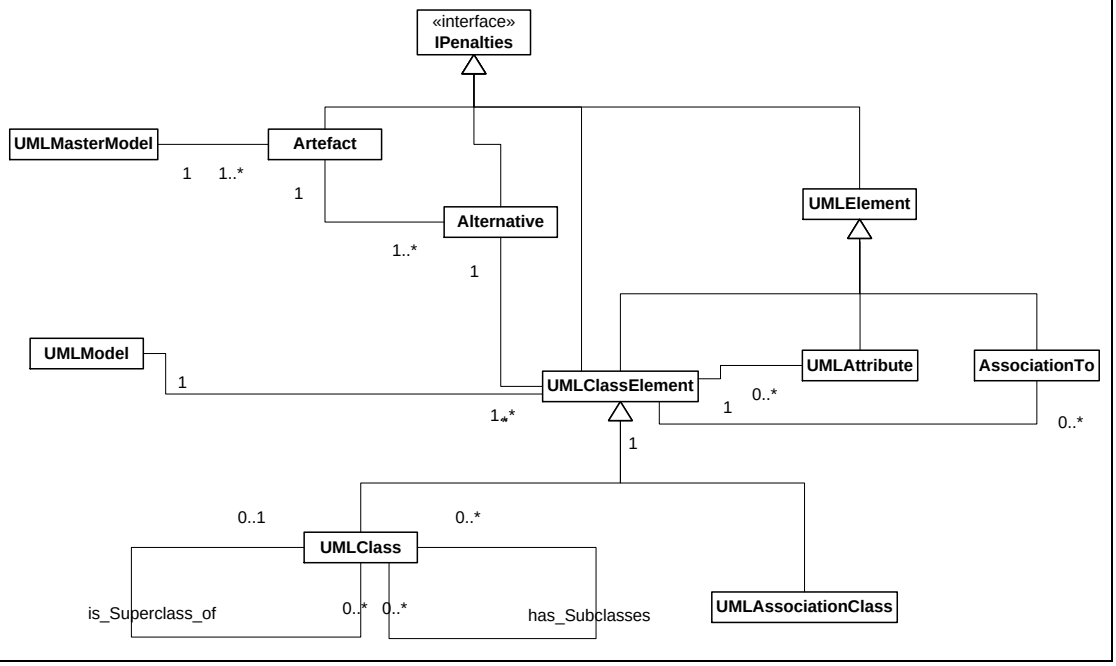
Das geparste XMI Dokument wird in eine interne Klassenstruktur eingelesen. Abbildung 6-7 zeigt die Klassenstruktur für die Verwaltung des Mustermodells. Aus dieser Struktur kann durch rekursives Durchlaufen der Listen sehr einfach eine XML Struktur erzeugt werden. Diese XML Struktur dient auch zur Serialisierung des Modells. Dadurch können erstellte Modelle einfach abgespeichert und wieder eingelesen werden.

Studentenmodelle werden ähnlich wie Mustermodelle verwaltet, jedoch werden hier die Java Klassen „Artefact" und „Alternative" nicht erzeugt. Die UML Klassen („UMLClassElement") werden hier direkt in der Klasse „UMLModel" verwaltet. Nachfolgend werden die Aufgaben dieser Klassen beschrieben.

IPenalties

Dieses Interface definiert die Methode zum Einlesen der Punkteabzüge aus dem XML String für das Mustermodell. Da alle beteiligten Klassen dieses Interface implementieren, kann zum Einlesen das objektorientierte Prinzip des Polymorphismus verwendet werden.

Abbildung 6-7: Klassenstruktur zur Verwaltung der Modelle



Artefact

Dient als „Klammer" für einen Sachverhalt. Es werden alle für diesen Sachverhalt gespeicherten Modellierungsvarianten in dieser Klasse verwaltet. Beim erstmaligen Einlesen eines Mustermodells im XMI-Format wird pro UML-Klasse ein „Artefact" angelegt.

Alternative

Dient als Klammer für alle UML Modellelemente, welche zu einer Modellierungsvariante gehören. In einer „Alternative" können auch mehrere UML-Klassen verwaltet werden. Beim erstmaligen Einlesen eines Mustermodells im XMI Format wird für jede UML-Klasse genau eine „Alternative" angelegt.

UMLClassElement

Basisklasse für UML Klassen und UML Assoziationsklassen. Hier werden die Attribute einer UML Klasse bzw. UML Assoziationsklasse, sowie die Beziehungen zu anderen UML-Klassen verwaltet.

UMLClass

Diese Java Klasse bildet die Eigenschaften einer UML-Klasse ab. Die zugehörigen Attribute und Beziehungen sind bereits in der Basisklasse abgebildet worden, und über die Vererbung verfügbar. Zusätzlich werden Generalisierungsbeziehungen zu anderen UML-Klassen verwaltet.

UMLAssociationClass

Dient zur Unterscheidung von UML-Klassen und UML Assoziationsklassen. Durch die Java Reflection kann dieser Unterschied aufgelöst werden. Die Attribute und Beziehungen werden wie bei der UMLClass bereits in der Basisklasse abgebildet.

UMLAttribute

Bildet ein Attribut einer UML-Klasse ab.

AssociationTo

Diese Java Klasse bildet die ausgehende Beziehung einer UML-Klasse zu einer anderen Klasse ab. Auch besondere Formen von Beziehungen wie Aggregation und Komposition, oder mehrwertige Beziehungen werden verwaltet.

6.4.3 Parsen der XMI-Datei des Mustermodells

Das Parsen der Eingabedatei im Format XMI, wird von der Klasse „XMIMasterLoader" des Java Packages „koenings.XMITools" erledigt. Dazu erbt die Klasse von „org.xml.sax.helpers.DefaultHandler" und überschreibt Methoden dieser Basisklasse. In der Implementierung von „XMIMasterLoader" werden die Ereignisse „startElement", „characters" sowie „endElement" verwendet. Die Ereignishandler werden jedes Mal wenn im Dokument ein XML-Element (z.B. <UML:Class>) bzw. ein XML-Endelement (z.B. </UML:Class>) gefunden wird aufgerufen.

Abbildung 6-8: Start des SAX Parsers

```
public void load(InputStream is) {
 try {
  //get an instance of sax parser from factory
  SAXParser saxParser = SAXParserFactory.newInstance().newSAXParser();
  //do the work
  saxParser.parse(is, this);
 } catch (Throwable t) {
  t.printStackTrace();
 }
}
```

Abbildung 6-8 zeigt das Erzeugen des SAX Parser Objects, über das von der verwendeten Java Bibliothek „org.xml.sax" bereitgestellte Factory Pattern (Gamma, Helm, Johnson, & Vlissides, 1995). Dem Aufruf der Methode „parse" werden anschließend das XMI Dokument in Form eines InputStreams sowie eine Referenz auf die eigene Klasse übergeben.

Die Verarbeitung der Datei erfolgt durch die überschriebenen Ereignishandler „startElement", „endElement" sowie „characters". Beim Einlesen des Knotens „<UML:Model>" wird eine Instanz der Klasse „UMLMasterModel" erzeugt und dem Klassenmember „_theModel" zugewiesen. Nachfolgend werden dann die gefundenen Klassen (<UML:Class>) dem MasterModel hinzugefügt. Für jeden Knoten „<UML:Class>" wird eine Instanz der Java Klasse „UMLClass" angelegt. Gleichzeitig wird auch je eine Instanz der Java Klassen „Artefact" und „Alternative" erzeugt. Das Java Objekt „UMLClass" wird in die Liste der verwalteten UML-Klassen beim Java Objekt „Alternative" eingetragen. Das Java Objekt „Alternative" wird in die Liste der Varianten beim Java Objekt „Artefact" eingetragen. Dadurch entsteht eine Baum-Struktur. Die gesamte Teilstruktur wird jetzt beim Java Objekt „UMLMasterModel" in die Liste der Artefakte eingetragen. Dadurch wird für jede UML-Klasse genau ein „Artefakt" mit einer „Alternative" angelegt. Abbildung 6-9 zeigt einen Ausschnitt aus der Methode „startElement" der Klasse „XMIMasterLoader".

Typischerweise müssen Kontextinformationen verfügbar sein, um den Knoten richtig zuordnen zu können. Um die komplexe Struktur von XMI aufzulösen werden die Attribute sowie die Beziehungen der entsprechenden UML-Klasse zugeordnet. Die Attribute können direkt über den Namen der UML-Klasse zugeordnet werden. Die Endpunkte einer Beziehung werden im XMI-Model nur als Referenz gespeichert. Für eine lesbare Darstellung im XML-Schema Format des UML-Expertenmoduls müssen diese Referenzen gegen Namen aufgelöst werden. Für jede ausgehende Beziehung einer UML-Klasse zu einer anderen UML-Klasse wird ein Java Objekt vom Typ „AssociationTo" angelegt und beim jeweiligen Java Objekt „UMLClass" gespeichert.

Abbildung 6-9: Ausschnitt aus der Methode startElement

```
} else if (eName.equals("UML:Class")) {  // compute classes
 if( _isPackage || _isAttribute)
  return;
 if( _isGeneralization )
 {
  String part = attrs.getValue("xmi.idref");
  UMLClassElement cl = _theModel.getClassForId(part);
  if( _isChild)
   _childClass = (UMLClass)cl;
  else
   _parentClass = (UMLClass)cl;
 }
 else if ( _isParticipant )
 {
  String part = attrs.getValue("xmi.idref");
  _curAssocEnd.setParticipantId(part);
 }
 else if (_curAssocEnd != null) {
  String part = attrs.getValue("xmi.idref");
  _curAssocEnd.setParticipantId(part);
 }  else {
  UMLClass cls = new UMLClass(attrs);

  Artefact a = new Artefact();
  _theModel.addArtefact(a);
  Alternative alt = new Alternative();
  a.addAlternative(alt);
  alt.addClassElement(cls);

  _currentClass = cls;
 }
} else if (eName.equals("UML:Attribute") && bez != null) {
```

Generalisierungsbeziehungen zwischen UML-Klassen werden ebenfalls in der Java Klasse „UMLClass" verwaltet. Es stehen die Klassenmember „SuperClass" für die Basisklasse sowie „subclasses" als Liste der abgeleiteten Klassen zur Verfügung. Auch hier müssen die Referenz-Ids aus der XMI-Datei gegen die KlassenNamen aufgelöst werden.

6.4.4 Wartung der Mustermodelle

Die Zusatzinformationen des Mustermodells wie beispielsweise Modellierungsvarianten und Punkteabzüge müssen manuell hinzugefügt werden. Deshalb muss eine Eingabemöglichkeit geschaffen werden, um die Mustermodelle verändern bzw. anpassen zu können. Die Serialisierungsform des Mustermodels ist XML, daher ist es naheliegend einen XML Editor zur Bearbeitung des Mustermodells zu verwenden. Wie im Kapitel 4 beschrieben werden die Eingabeseiten über das Werkzeug Velocity generiert. Diese Vorgehensweise ist durch E-Tutor Dispatcher zwingend vorgegeben. Vom UML-Expertenmodul müssen die Velocity Template Dateien zur Verfügung gestellt werden.

Abbildung 6-10: Velocity Template für Erstellung/Wartung von Aufgaben

```
<script language="javascript" type='text/javascript'
src='%%RESOURCES%%/create_exercise.js'></script>

<script language="javascript" type='text/javascript'
src='http://localhost/edit_area/edit_area_full.js'></script>
<script language="javascript" type="text/javascript">
editAreaLoader.init({
 id : "textarea_1"  // textarea id
 ,syntax: "xml"  // syntax to be used for highliting
 ,start_highlight: true  // to display with highlight mode on start-
up
 ,toolbar: "new_document, save, load, |, search, go_to_line,
fullscreen, |, undo, redo, |, select_font, |,
change_smooth_selection, highlight, reset_highlight"
 ,load_callback: "my_load"
 ,save_callback: "my_save"
 ,submit_callback: "my_submit"

});
// callback functions
```

```
  function my_save(id, content){
   alert("Saving the content of this edit has not been imple-
mented!\nCheck callback function for this.\n");
  }

  function my_load(id){
   alert("Loading from file has not been implemented!\nCheck call-
back function for this.\n");
   //editAreaLoader.setValue(id, "The content is loaded from the
load_callback function into EditArea");
  }

  function my_submit(id) {
   alert("submiting");
   document.command.value = editAreaLoader.getValue(id);
  }
</script>

<input type='hidden' id='command' name='command' />
<div class="exercise_wiz_title">${messageTitle}</div>

<div class="exercise_wiz_content">

<table cellspacing="0" cellpadding="5">
 <tr>

  <td width="85%" class='UMLTaskEditorTableData' align='left'>
            <input type='file' name='%%IDPREFIX%%uploadfile'
size='40' />
        </td>
 </tr>
</table>
<div>
<textarea id="textarea_1" name="content" id='content' cols="120"
rows="23">
#if(${xmlText}) ${xmlText} #else <leer> #end
</textarea>
</div></div>
```

Bei der Generierung der Web-Seiten durch Velocity wird HTML[29] Code erzeugt. Im HTML wird für die Darstellung bzw. Eingabe von größeren Textblöcken das HTML Element „textarea" verwendet. Um eine komfortablere Eingabemöglichkeit mit Syntax Highlighting[30]

[29] Hypertext Markup Language: Die Beschreibungssprache für Web-Seiten. Siehe auch http://www.w3.org/MarkUp

[30] Dabei werden Elemente entsprechend ihrer Bedeutung farblich hervorgehoben.

anbieten zu können, müssen JavaScript[31] Erweiterungen des HTML textarea Elements verwendet werden. Viele dieser Erweiterungen werden als freie Software zur Verfügung gestellt. Auf der Webseite „www.cdolivet.com" wird eine freie Implementierung eines Source Code Editors zum Download angeboten. Abbildung 6-10 zeigt die Einbettung dieses Editors in der Velocity Template Datei. Die Installation des Editors wird in Anhang B beschrieben.

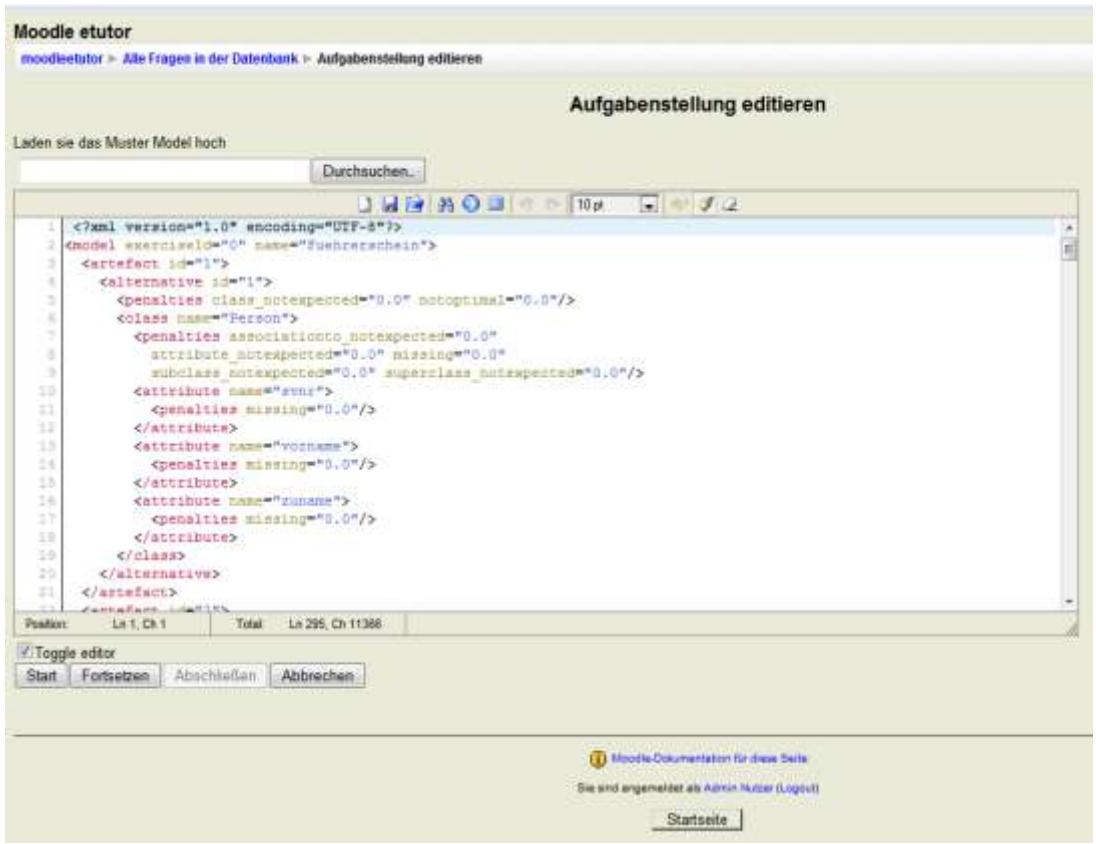
Durch den Aufruf der Methode „getXml" der Klasse „UMLMasterModel" wird die serialisierte Form des Modells, als String mit XML Formatierung zurückgegeben. Dieser String wird an den Editor zur Darstellung übergeben. Nach der Bearbeitung wird der Inhalt des Editors als String gelesen. Dieser String wird der Klasse „XMLMasterLoader" übergeben. Diese parst den XML String und erzeugt eine Instanz der Klasse „UMLMasterModel".

Für das Erstellen eines neuen Mustermodells muss zuerst über das HTML „input" Element die entsprechende XMI Datei hochgeladen werden. Dadurch wird die XMI Datei als Ressource verfügbar. Über das Objekt „InputStream" der Ressource kann der Dateiinhalt an die Klasse „UMLMasterLoader" übergeben werden. Diese parst den Inhalt und erzeugt die Klassenstruktur für das Mustermodell. Anschließend an das Parsen wird die serialisierte Form (XML) als Text im Editor-Fenster zur Weiterverarbeitung dargestellt. Abbildung 6-11 zeigt den Editor mit dem Mustermodell „Führerschein" unmittelbar nachdem die XMI-Datei eingelesen worden ist.

---

[31] Eine Skriptsprache, auch als ECMA-Script bezeichnet, die von den meisten Webbrowsern verarbeitet werden kann.
Siehe auch http://www.ecma-international.org/publications/standards/Ecma-262.htm

Abbildung 6-11: Erzeugte XML-Datei des Mastermodells zur Bearbeitung



Nach dem Editieren des Mustermodells muss dieses so persistiert werden, dass es als Basis für eine Analyse eines abgegebenen Studentenmodells geladen werden kann. Dazu wird die serialisierte Form des Mustermodells in einer Datenbanktabelle gespeichert.

Abbildung 6-12: Interface der Klasse DBManager

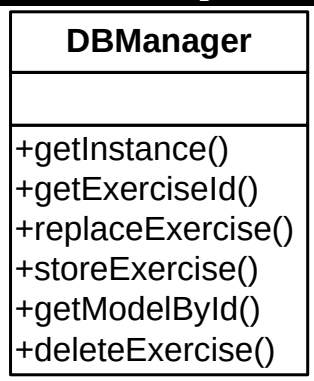


Abbildung 6-12 zeigt das Interface der Klasse „DBManager". Diese Klasse kapselt alle Funktionen zum Abspeichern und zum Zugriff

auf die Mustermodelle in der Datenbank. Die Methoden sind als statische Methoden implementiert, dadurch können sie jederzeit im Code aufgerufen werden, ohne vorher eine Instanz der Klasse „DBManager" erzeugen zu müssen.

Abbildung 6-13: statische Methode storeExercise der Klasse DBManager

```
/**
 * stores an exercise (a model) into database with the given id
 * @param exerciseId the exercise id
 * @param model the model
 * @throws SQLException
 */
public static void storeExercise(int exerciseId, String model)
  throws SQLException {
 CLOB clob = null;
 String sql = "";
 PreparedStatement pstmt = null;

 // sql statements and connection
 // Statement stmt;
 Connection conn;
 // Register the db driver
 java.sql.DriverManager
    .registerDriver(new oracle.jdbc.driver.OracleDriver());

 //sql statement
 sql = "insert into MasterModels (modelId, master_Model, creation-
Date) values(";
 sql += exerciseId + ", XmlType(?), sysdate)";

 //gets the connection
 conn = DriverManager.getConnection(UMLConstants.CONN_URL,
    UMLConstants.CONN_USER, UMLConstants.CONN_PWD);
 conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
 conn.setAutoCommit(true);
 // Get the statement Object
 pstmt = conn.prepareStatement(sql);

 // Get the CLOB object using the getCLOB method.
 clob = getCLOB(model, conn);
 // Bind this CLOB with the prepared Statement
 pstmt.setObject(1, clob);
 // Execute the Prepared Statement
 if (pstmt.executeUpdate() == 1) {
  System.out.println("Successfully inserted !");
 } else
  throw new SQLException("insert was not successful");
 if (pstmt != null)
  pstmt.close();

 if (conn != null)
  conn.close();
}
```

Da für bestehende Expertenmoduln bereits Oracle als Datenbank verwendet wird, kann auf eine bestehende Datenbank am DKE Institut zugegriffen werden. Die Verbindungsdaten sind als Konstanten in der Klasse „UMLConstants" abgelegt. Passend zur serialisierten Form des Mustermodells in XML wird der Datentyp „XmlType" des Datenbanksystems Oracle verwendet[32].

Die XML Struktur wird dabei in einem Large Object (LOB) Feld einer Datenbanktabelle gespeichert. Damit können Änderungen einfach durch Überschreiben eines Datenbankfeldes erfolgen. Die XML-Struktur wird dabei genauso abgespeichert wie in dem XML-String vorgegeben. Durch einfaches Auslesen des LOB Felds kann das Mustermodell wieder hergestellt werden. Dazu wird der ausgelesene String der Methode „load" der Klasse „XMLMasterLoader" übergeben. Diese Klasse ist für das Parsen eines abgespeicherten Mustermodells in der XML Form zuständig.
Abbildung 6-13 zeigt die statische Methode storeExercise der Klasse DBManager. Die eindeutige Identifikationsnummer („modelId") wird beim Aufruf bereits übergeben. Der Wert dieser Variable wird durch einen Aufruf der statischen Methode „getExerciseId" (aus der Klasse „DBManager") ermittelt. Diese Id wird später benötigt um das Mustermodel abzurufen. Die zum Abspeichern notwendige XML Struktur wird über die Serialisierungsmethode „getXml" der Klasse „UMLMasterModel" als String generiert. Nach dem Abspeichern wird die eindeutige ID der neuen Aufgabe zurückgegeben und im E-Tutor System als Aufgabennummer gespeichert.

6.4.5 Analysieren des Studentenmodells

Das Studentenmodell wird per File Upload (HTML-Input Type = "File") an das E-Tutor System und damit an das UML-Expertenmodul übergeben. Nach der Übertragung der Eingabeseite an den Server

[32] download.oracle.com/docs/cd/B10501_01/appdev.920/a96620/xdb04cre.htm

wird der gesamte Inhalt der Datei im Sessionattribut „submission" als String abgespeichert. Anschließend wird vom E-Tutor System die Methode „analyze" der abgeleiteten Klasse „UMLEvaluator" aufgerufen (Abbildung6-14).

Abbildung 6-14: Methode analyze der Klasse UMLEvaluator

```
/**
 * performs analysis of the submission
 * @param submission the user solution to the exercise
 * @param config configuration of the analysis
 * @return object UMLAnalysis contains information about mistakes and errors in user
model
 */
public UMLAnalysis analyze(Serializable submission, UMLAnalyzerConfig config) {
 String message;
 UMLAnalysis analysis;
 String submittedModel;

 //create container for analysis result
 analysis = new UMLAnalysis();
 analysis.setSubmission(submission);

 //submission is users solution to the exercise it must not be null
 if (submission == null) {
  message = new String();
  message = message.concat("Analysis stopped with errors. ");
  message = message.concat("Submission is empty. ");
  message = message.concat("This is an internal system error. ");
  message = message.concat("Please inform the system administrator.");

  this.logger.log(Level.SEVERE, message);
  analysis.setAnalysisException(new AnalysisException(message));
  return analysis;
 }

 //we expect a string
 if (submission instanceof String) {
  submittedModel = (String) submission;
 } else {
  message = new String();
  message = message.concat("Analysis stopped with errors. ");
  message = message.concat("Submission is not utilizable. ");
  message = message.concat("This is an internal system error. ");
  message = message.concat("Please inform the system administrator.");

  this.logger.log(Level.SEVERE, message);
  analysis.setAnalysisException(new AnalysisException(message));
  return analysis;
 }

 //config hold extra information  null is not allowed
 if (config == null) {
  message = new String();
  message = message.concat("Analysis stopped with errors. ");
  message = message.concat("No configuration found. ");
  message = message.concat("This is an internal system error. ");
  message = message.concat("Please inform the system administrator");

  this.logger.log(Level.SEVERE, message);
  analysis.setAnalysisException(new AnalysisException(message));
  return analysis;
 }

 //there is no special config setting, so config is always valid
 if (!this.isConfigurationValid(config)) {
  message = new String();
  message = message.concat("Analysis stopped with errors. ");
  message = message.concat("Invalid configuration found. ");
```

```
    message = message.concat("This is an internal system error. ");
    message = message.concat("Please inform the system administrator");

    this.logger.log(Level.SEVERE, message);
    analysis.setAnalysisException(new AnalysisException(message));
    return analysis;
  }

  // create instance of XMILoader
  XMILoader xl = new XMILoader();
  // create the model from XMI string
  xl.load(submittedModel);
  UMLModel model = xl.getModel();
  // we also need the master model for comparison
  UMLMasterModel master = getMasterModel(config);

  // create instance of model comparer class
  UMLModelComparer umc = new UMLModelComparer(master, model);
  // run comparison now
  ArrayList<CompareInfo> ret = umc.getCompareInfo();
  analysis.setCompareInfo(ret);
  return analysis;

}
```

Beim Parsen des Studentenmodells wird die gleiche Klassenstruktur wie beim Mustermodel erzeugt. Die beiden Hierarchiestufen „Artefact" und „Alternative" werden beim Studentenmodell allerdings nicht benötigt und deshalb auch nicht angelegt. Das Mustermodell wird über einen Aufruf von „getMasterModel" geladen. Diese Methode ruft über die Aufgabennummer die serialisierte Form des Mustermodells, also einen String mit XML Text, aus der Datenbank ab. Die Aufgabennummer wird vom E-Tutor System als Parameter „exerciseID" beim Aufruf der Methode „Analyze" übergeben. Aus diesem XML Text wird mit Hilfe der Klasse XMLMasterLoader eine Instanz der Klasse „UMLMasterModel" erzeugt.

Damit stehen nun sowohl das Mustermodell als auch das Studentenmodell als Java Objekte zur Verfügung. Der Analyselauf wird mit dem Aufruf der Methode „getCompareInfo" der Klasse „UMLModelComparer" gestartet. Beim Analyselauf werden alle Modellelemente aus dem Mustermodell durchlaufen. Die Methode „getCompareInfo" wird für alle Verwaltungsklassen der Studenten- und Mustermodelle (siehe Abbildung 6-7) implementiert. Dadurch kann die Methode rekursiv aufgerufen werden. Es wird immer nach folgendem Muster vorgegangen:

- Suche das aktuelle Element des Mustermodells im Studentenmodell
  - Falls das Element im Studentenmodell nicht gefunden wird
    - Fehler: Element nicht vorhanden (…Missing)
  - Falls das Element gefunden wird
    - Vergleiche die Eigenschaften des Elements
      - Erzeuge Info/Fehler bei jeder Abweichung
    - Rufe „getCompareInfo" für jedes Subelement auf

Wenn es im Mustermodell für ein Artefakt mehrere gespeicherte Varianten gibt, muss zuerst ermittelt werden, welche der Varianten am wahrscheinlichsten vom Studenten modelliert worden ist. Dazu wird ein „Match Faktor" ermittelt.

Abbildung 6-15: Berechnung des Match-Faktors

```
/**
 * get the factor of matching with the model
 *
 * @param model
 *            check this model
 * @return float value 0 - 1 percentage matching
 */
public float getMatchFactor(UMLModel model) {
 float ret = 0.0f;
 int total = this._classes.size() > 0 ? this._classes.size() : 1;
 for (int i = 0; i < this._classes.size(); i++) {
  ret += _classes.get(i).getMatchFactor(model);
 }
 return ret / total;
}
```

Der Matchfaktor zählt die gefundenen Elemente einer Modellierungsvariante und stellt sie der erwarteten Anzahl der Elemente gegenüber (siehe Abbildung 6-15). Wenn sich die Varianten nur geringfügig unterscheiden, kann es vorkommen dass die Varianten den gleichen Matchfaktor haben. Aus diesem Grund wird auch noch die Anzahl der Fehler als Kriterium herangezogen. Eine nicht modellierte Variante im Studentenmodell erhält demnach eine hohe

Anzahl an Fehlern bzw. Punkteabzügen. Für den Vergleich wird deshalb diejenige Variante angenommen, welche die wenigsten Fehler aufweist, falls die Varianten den gleichen Matchfaktor aufweisen (siehe Abbildung 6-16).

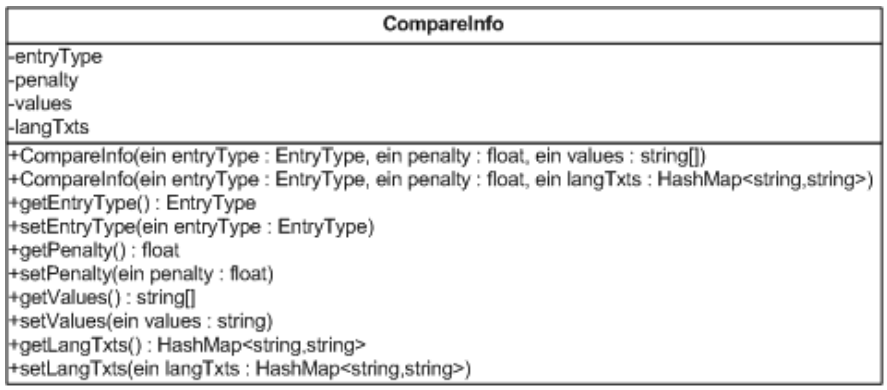
Abbildung 6-16: Anwendung des Matchfaktors

```
/**
 * compares the given model with this artifact
 * and returns a list of compare info objects
 * @param model the model to check
 * @return ArrayList of CompareInfo
 */
public ArrayList<CompareInfo> getCompareInfo(UMLModel model) {
 ArrayList<CompareInfo> ret = new ArrayList<CompareInfo>();
 int lastSize = 9999;
 float lastMatch = 0.0f;

 for (int i = 0; i < _alternatives.size(); i++) {
  ArrayList<CompareInfo> tmp =
_alternatives.get(i).getCompareInfo(model);
  lastSize = tmp.size();
  //check the match factor
  float match = _alternatives.get(i).getMatchFactor(model);
  if( match > lastMatch){
   ret = tmp;
   lastMatch = match;
  }
  else if (match == lastMatch) {
   //check for the alternative with less errors
   if( tmp.size() < lastSize ) {
    ret = tmp;
   }
  }
 }
 return ret;

}
```

Fehler bzw. Abweichungen werden in Objekten des Typs „CompareInfo" (Abbildung 6-17) gespeichert. Die Klasse „CompareInfo" verwaltet die Art des Fehlers, die dafür berechneten Punkteabzüge sowie ein Array mit Elementen vom Typ String. Für manche automatisch generierten Texte werden verschiedene Kontextinformationen, wie etwa Klassennamen benötigt. Diese Informationen werden als String-Array des jeweiligen „CompareInfo" Objektes gespeichert. Die individuellen Feedbacktexte für Modellierungsvarianten werden ebenfalls gespeichert. Die Art des Fehlers wird über ei-

nen Aufzählungstyp (siehe Abbildung 6-18) klassifiziert. Damit kann diese Information gemeinsam mit den Argumenten später für die Feedbackerstellung verwendet werden.

Abbildung 6-17: Klasse CompareInfo



Zusätzlich wird auch noch überprüft ob es Elemente im Studentenmodell gibt, die im Mustermodell nicht vorgesehen sind. Am Ende des Vergleichs wird die Liste mit allen erkannten Fehlern bzw. Abweichungen zurückgegeben.

Abbildung 6-18: Aufzählungstyp in der Klasse CompareInfo

```
public enum InfoEntryType {
 AlternativeNotOptimal,
 ClassMissing,
 AssociationclassMissing,
 AttributeMissing,
 SubclassMissing,
 SuperclassMissing,
 AssociationToMissing,
 ClassNotExpected,
 ClassAttributeNotExpected,
 AssociationclassAttributeNotExpected,
 ClassAssociationToNotExpected,
 AssociationclassAssociationToNotExpected,
 SubclassNotExpected,
 SuperclassNotExpected,
 AssociationToWrongType,
 AssociationToWrongCardinality,
 ClassMissingPK

}
```

Die gesamte Analyse wird in einem Objekt der Klasse Analysis gespeichert und an das E-Tutor System zurückgegeben.

6.4.6 Bewerten und Feedback-Erstellung

Die Bewertung der abgegebenen Übung erfolgt mit Hilfe der bei der Analyse erzeugten Daten. Im Mastermodel ist für jedes Modellelement ein eigener Subknoten „penalties" eingefügt, der die Punkteabzüge für dieses Modellelement pro Fehlerart vorgibt. Bei der Analyse wird der Fehler festgestellt und in das „CompareInfo" Objekt auch die Punkteabzüge aus dem Mustermodell für diesen Fehler übertragen. Die Bewertung erfordert nur mehr ein Summieren der einzelnen Punkteabzüge in der „CompareInfo" Liste. Die erreichten Gesamtpunkte errechnen sich aus der maximalen Punkteanzahl minus des Punkteabzugs (Abbildung 6-19). Zur Erzeugung des Feedbacks werden ebenfalls die gesammelten „CompareInfo" Objekte verwendet. Das System bzw. die Klassifizierung zur Generierung der Fehlertexte wurde im Kapitel 5 (Konzept) beschrieben.

Abbildung 6-19: Bewerten der abgegebenen Übung

```
public DefaultGrading grade(UMLAnalysis analysis,
              UMLGraderConfig gradingConfig) throws Exception {
 double points;
 double maxPoints = gradingConfig.getMaximumPoints();
 DefaultGrading grading;

 points = 0;
 grading = new DefaultGrading();
 grading.setMaxPoints(gradingConfig.getMaximumPoints());

 ArrayList<CompareInfo> ci = analysis.getCompareInfo();
 for( int i = 0; i < ci.size(); i++)
 {
  CompareInfo cpi = ci.get(i);
  points += cpi.getPenalty();
 }
 grading.setPoints(maxPoints-points);
 return grading;

}
```

Für jede Fehlerkategorie werden formatierte Texte mit Platzhaltern für Kontextinformationen benötigt. Diese Texte werden in einfachen Textdateien abgespeichert. Der Zugriff erfolgt über Java Res-

sourceBundles[33]. Für jede verwendete Sprache muss eine eigene Textdatei angelegt werden. Die Benennung der Dateien erfolgt nach einem festgelegten Schema aus Basisklasse und Kürzel für die Sprache. Beim Zugriff auf die Datei über RessourceBundles kann das Sprachkürzel als Parameter übergeben werden. Die Zugriffe auf Texte in den ResourceBundles Dateien werden in einer eigenen Klasse „ErrorTxt" gekapselt. Diese Klasse implementiert das „Singleton" Entwurfsmuster (Schmidt, Stal, Rohnert, & Buschmann, 2002). Dieses Entwurfsmuster stellt sicher, dass die Klasse nur ein einziges Mal als Objekt im Speicher vorhanden ist. Auf die Texte kann jederzeit, auch ohne eine neue Instanz der Klasse erstellen zu müssen, zugegriffen werden. Für die Bereitstellung der Texte wurden die drei Methoden „getErrorText", „getDescriptionText" und „getHintText" implementiert.

Abbildung 6-20: Ausschnitt aus der Singleton Klasse ErrorTxt

```
/**
 * gets the text from the resource bundle for a specific error, lev-
el and locale
 * @param ci the compareinfo struct containing error info
 * @param l the locale
 * @param i the level
 * @return the text
 */
private String getText(CompareInfo ci, Locale l, int i) {
 String fmt;
 ResourceBundle rb =
       PropertyResourceBundle.getBundle(ErrorTxt.class.getName(),
l);

 try {
  String key = ci.getEntryType().name() + "_H"+ String.valueOf(i+1);
  fmt = rb.getString(key);
 } catch (Exception ex)
 {
  fmt = "Could not get Text for Error class " +
ci.getEntryType().toString();
 }
 if(ci.getEntryType() == InfoEntryType.AlternativeInfoTxt)
  return String.format(fmt, (Ob-
ject[])ci.getLangTxt(l.getLanguage()));
 else
  return String.format(fmt, (Object[])ci.getValues());
}
```

[33] http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles

Jede dieser Methoden steht für eine Stufe der Feedbacktiefe. Die benötigte Feedbacktiefe und damit die Frage welche Texte benötigt werden, wird über den Parameter „diagnoseLevel" der Methode „report" mitgeteilt.

Den Methoden wird jeweils eine Instanz des „CompareInfo" Objekts (aus der Analyse) und ein Objekt vom Typ „Locale" für die Sprachinformation übergeben. Zurückgegeben wird der entsprechende, mit den Argumenten aus der „CompareInfo" formatierte, Text als String in der gewünschten Sprache. Diese aufbereiteten Texte werden in Instanzen der Klasse „UMLErrorReport" gespeichert. „UMLErrorReports" werden als Inhalt der Instanz vom Typ „UMLReport" an das E-Tutor System zur Anzeige zurückgegeben. Zur Generierung der Texte für die Anzeige auf der Ergebnisseite wird wiederum Velocity verwendet. Abbildung 6-21 zeigt das entsprechende Velocity Template. Welche Feedbackstufen angezeigt werden, kann über die Methoden „showError", „showDescription" und „showHints" des Objekts „report" gesteuert werden.

Abbildung 6-21: Velocity Template zur Anzeige der Hinweistexte

```
<script src='%%RESOURCES%%/report.js'
type='text/javascript'></script>
<table class='sql_report' cellspacing='20'>
    #if ( ${report.showPrologue()} )
        <tr>
            <td>
                <table class='section'>
                    <caption
class='section_caption'>${messageResult}</caption>
                    <tr>
                        <td class='section_ident'>
                        </td>
                        <td class='section_content'>
                            ${report.Prologue}
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
    #end
    #if ( ${report.showErrorReport()} && ${report.hasErrorReports()}
)
        <tr>
```

```
            <td>
                <table class='section'>
                    <caption
class='section_caption'>${messageErrorreport}</caption>
                    <tr>
                        <td class='section_ident'>
                        </td>
                        <td class='section_content'>
                            #foreach ( ${errorReport} in ${re-
port.iterErrorReports()} )
                                <table class='error_report'>
                                    <tr>
                                        <td
class='error_report_label'>
                                            ${messageError}:
                                        </td>
                                        <td
class='error_report_content'>
                                            ${errorReport.Error}
                                        </td>
                                    </tr>
                                    #if ( $re-
port.showErrorDescriptions() )
                                        <tr>
                                            <td
class='error_report_label'>
                                                ${messageDescrip-
tion}:
                                            </td>
                                            <td
class='error_report_content'>
                                                ${errorRe-
port.Description}
                                            </td>
                                        </tr>
                                    #end
                                    #if ( ${report.showHints()} )
                                        <tr>
                                            <td
class='error_report_label'>
                                                ${messageHint}:
                                            </td>
                                            <td
class='error_report_content'>
                                                ${errorReport.Hint}
                                            </td>
                                        </tr>
                                    #end
                                </table>
                            #end
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
    #end
</table>
```

Abbildung 6-22 zeigt das Ergebnis der Feedbackerstellung anhand des Beispiels aus Abbildung 5-27. Es werden alle drei Feedbackstufen angezeigt.

Abbildung 6-22: Ergebnis für das Studentenmodell aus Abbildung 5-27

ERGEBNIS

Das Modell ist nicht korrekt, es wurden Fehler gefunden.

FEHLERBERICHT

Fehler:	Eine gewählte Modellierungsvariante ist nicht optimal oder falsch (5.0 P)
Beschreibung:	Die Modellierungsvariante mit der Klasse/den Klassen Person ist nicht optimal
Hinweis:	Die Klasse 'Person' ist eine Generalisierung von 'Pruefer'. Daher sollte die Beziehung als Generalisierung modelliert werden.
Fehler:	Eine gewählte Modellierungsvariante ist nicht optimal oder falsch (5.0 P)
Beschreibung:	Die Modellierungsvariante mit der Klasse/den Klassen Pruefer ist nicht optimal
Hinweis:	Die Klasse 'Pruefer' ist ein Spezialisierung von 'Person'. Daher sollte die Beziehung als Generalisierung modelliert werden.
Fehler:	Eine gewählte Modellierungsvariante ist nicht optimal oder falsch (10.0 P)
Beschreibung:	Die Modellierungsvariante mit der Klasse/den Klassen Ergebnis ist nicht optimal
Hinweis:	'Ergebnis' sollte als Assoziationsklasse modelliert werden.
Fehler:	Das Modell enthält Fehler (1.5 P)
Beschreibung:	Eine Assoziation hat einen falschen Typ
Hinweis:	Die Assoziation zwischen den Klassen Fahrt und Fahrzeug hat den falschen Typ

Die aussagekräftigsten Texte sind jene, die bei den Varianten hinterlegt wurden. Damit kann für bestimmte wichtige Fehler gezieltes Feedback gegeben werden. Diese Texte können dadurch dass sie individuell erfasst werden, natürlich viel besser auf die gestellte Aufgabe eingehen bzw. Bezug nehmen als die automatisch erstellten Texte. Der höhere Aufwand bei der Erstellung des Mustermodells (Expertenwissen) wird durch einen vermutlich besseren Lernfortschritt kompensiert.

7 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Expertenmodul für das ITS E-Tutor des Instituts für Wirtschaftsinformatik – Data & Knowledge Engineering für den Bereich konzeptueller Datenbankentwurf mit der UML realisiert. Dieses Expertenmodul soll in der Lehre begleitend zum Präsenzunterricht eingesetzt werden. Besondere Herausforderungen wie die Berücksichtigung von Modellierungsvarianten, die individuelle Bewertung von Fehlern und das gezielte Feedback mussten gelöst werden.

Für den Vergleich und das Bewerten von Modellen, sowie das Erstellen von Feedback an die Studenten wurde ein Konzept erstellt. Wesentliche Punkte waren die Speicherung des Expertenwissens in Form eines Mustermodells, sowie die Möglichkeit, Zusatzinformation speichern zu können. Dazu wurde eine eigene Speicherform (XML-Schema) entwickelt, das aus einer XMI-Repräsentation eines UML-Klassendiagramms generiert werden kann. Die Zusatzinformation muss manuell hinzugefügt werden. Dazu genügt ein einfacher XML-Editor. Wesentliche Zusatzinformationen sind Modellierungsvarianten, und dazugehörige Feedbackmeldungen. Modellierungsvarianten werden als Sachverhalte geklammert, sodass im Mustermodell zwischen den Varianten unterschieden werden kann. Für jede Klasse wird ein Sachverhalt erzeugt, die Klassen können jedoch anschließend vom Lehrenden beliebig zu Sachverhalten kombiniert werden. Neben den unterschiedlichen Modellierungsvarianten wurde ein System von Fehlerkategorien entworfen, um Fehlertexte bzw. Feedbacktexte automatisch generieren zu können. Durch die eigene Speicherform des Mustermodells können vom Ersteller der Aufgabe auch individuelle Punkteabzüge vergeben werden. Die Vergabe von Punkteabzügen orientiert sich am der derzeitigen Ablauf der Übungen zur LVA „Datenmodellierung". Auch hier wird eine maximale Punkteanzahl pro Übung festgelegt. Für

jeden Modellierungsfehler werden dann Punkte abgezogen. Die verbleibenden Punkte ergeben die Note für die Übung.
Es wurde ein Prototyp in Java erstellt, der in das derzeitige unter Moodle eingebettete E-Tutor System als Expertenmodul eingebunden werden kann. Die für die Einbettung notwendigen Schnittstellen wurden implementiert. Die Anzeige der Texte kann mehrsprachig erfolgen. Derzeit werden die Sprachen Deutsch und Englisch unterstützt. Ein Vergleich von Kriterien mit anderen Systemen zeigt, dass wesentliche Punkte erfüllt worden sind, wenngleich einige der geforderten Kriterien nicht umgesetzt werden konnten.
Für Weiterentwicklungen des Konzepts stehen einige Ansatzpunkte zur Verfügung. Die Namensgebung bzw. der Objektvergleich basiert auf vorgegebenen Namen, d.h. Studenten dürfen nur Namen verwenden die sie aus einer Liste auswählen. Für dieses spezielle Problem (freie Namensgebung) gibt es derzeit noch kein Konzept. Für die Erzeugung von Modellierungsvarianten könnte mit Wizards die Arbeit erleichtert werden. Eine statistische Auswertung über die gemachten Fehler ist derzeit nicht implementiert. Dazu müssten die Fehler nach Fehlerkategorie gespeichert werden. Ein weiterer offener Punkt betrifft die Kennzeichnung des Primary Key(PK) einer Entität. Die Kennzeichnung des PK ist in der UML nicht definiert. Daher muss eine Möglichkeit gefunden werden, dies in der UML mittels Erweiterungsmechanismus zu erledigen, und aus der generierten XMI Datei auszulesen. Die derzeitige Implementierung sieht keinen grafischen Editor vor, um das Studentenmodell, bzw. auch das Mustermodell entwerfen zu können. Es gibt einige Softwareprodukte, die auch den Export der Modelle in XMI ermöglichen. Aus diesem Grund erscheint es derzeit nicht notwendig einen eigenen UML-Editor in das Expertenmodul einzubetten.

Anhang

A. XMI/MOF

Da UML als grafische Notation nicht direkt elektronisch verarbeitet werden kann, muss eine Konvertierung erfolgen. Die Object Management Group[34] (OMG), als Non Profit Organisation, ist bestrebt industrietaugliche Standards zu schaffen. In diesem Konsortium arbeiten auch Mitarbeiter nahezu aller großen Softwarefirmen mit. Viele der OMG Spezifikationen sind auch als ISO Norm übernommen worden (www.omg.org).

Natürlich ist die OMG auch bestrebt den Austausch der Daten zwischen verschiedenen Werkzeugen zu ermöglichen. Deshalb wurden verschieden Mechanismen dazu entwickelt. Als Grundlage dient die Spezifikation zur Meta Object Facility und darauf aufbauend der Spezifikation XML Metadata Interchange. Diese beiden Spezifikationen werden in den beiden nachfolgenden Kapiteln näher beschrieben. Obwohl es mittlerweile Spezifikationen für die UML 2.0 gibt, wird hier auf die Spezifikationen die mit der Version 1.4 der UML kompatibel sind eingegangen, da die Mehrzahl der verfügbaren Werkzeuge diese UML Version unterstützen. Für den Bereich des konzeptuellen Entwurfs ist diese Version ebenfalls ausreichend.

Meta Object Facility (MOF)

Die Meta Object Facility (MOF) beschreibt eine Architektur mit 4 übereinanderliegenden Schichten. In der untersten Schicht M0 ist die reale Welt (konkrete Objekte) angesiedelt. Die Schicht M1 darüber bildet diese Welt als Modell ab. Schicht M2 wiederum be-

[34] Siehe www.omg.org

schreibt die Modellierungssprache und wird als Meta-Model bezeichnet. Typisches Beispiel für eine Modellierungssprache ist natürlich die UML. Die letzte Schicht M3 beschreibt nun das MetaModell und wird Meta-Meta-Modell oder Meta²-Modell genannt. Das Meta²-Modell ist die eigentliche MOF Schicht (OMG, 2005).

Anhang Abb. A-1: Schichtenmodell der MOF (Quelle: (OMG, 2005))

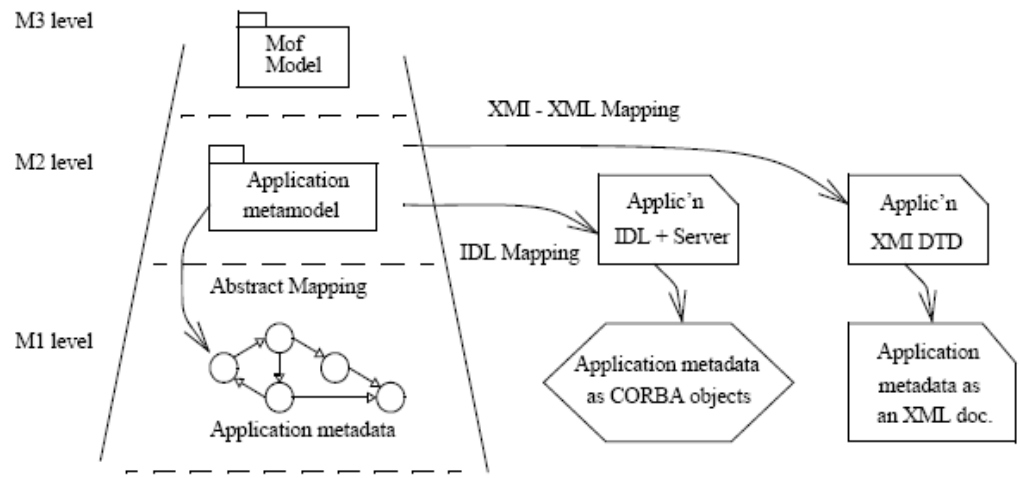
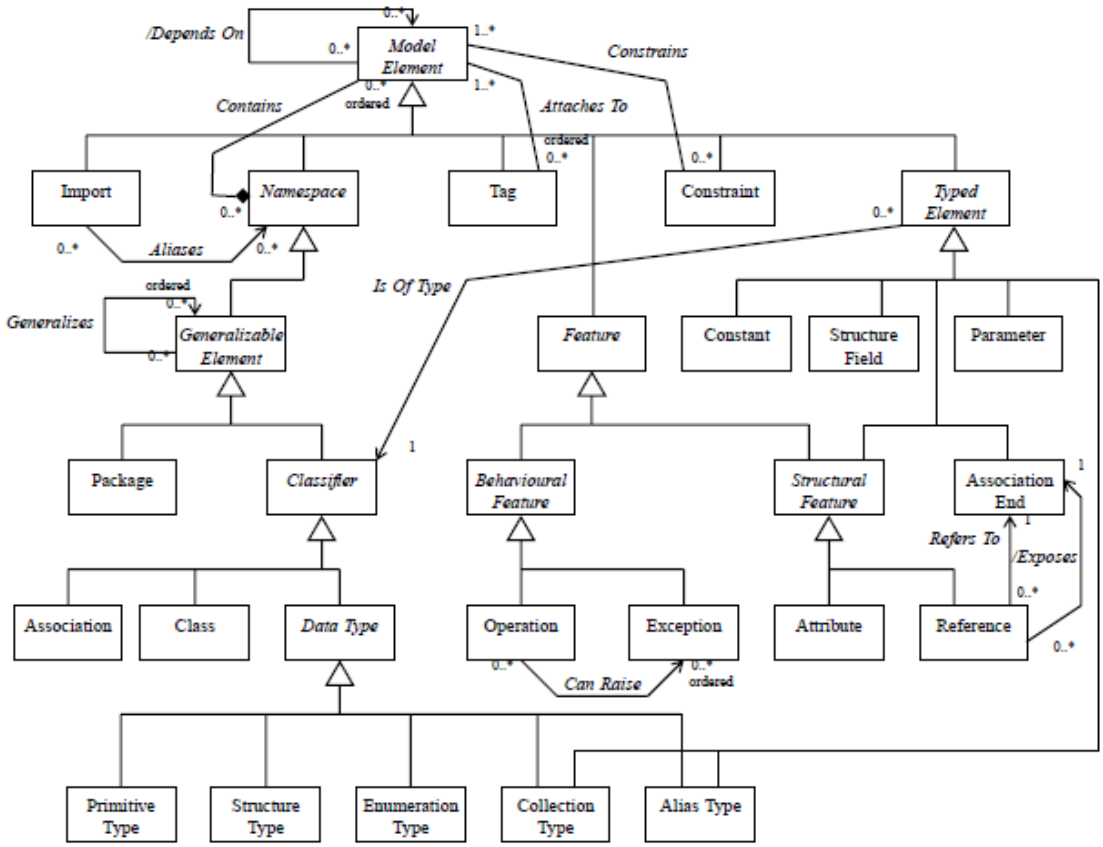


Abbildung A-1 zeigt den Strukturellen Aufbau der MOF, und beispielhafte Verwendungen einzelner Schichten.

Das Meta²-Modell ermöglicht plattformunabhängig Modelle auszutauschen. Dabei wird jedoch nur die Metainformation für das semantische Modell nicht aber die grafische Information beschrieben. Für den Austausch von grafischen Daten wurde von der OMG die Spezifikation zum Diagramm Interchange entwickelt (OMG, 2006). Damit wird es möglich, neben den semantischen Informationen, auch die grafische Repräsentation auszutauschen.

Anhang Abb. A-2: Die M3-Schicht (Quelle: (OMG, 2005))



XML Metadata Interchange (XMI)

Hauptaufgabe von XML Metadata Interchange (XMI) ist es, die drei Technologien extensible Markup Language (XML), UML und MOF zu verbinden. Die XMI bedient sich der XML um die Meta-Information strukturiert abzuspeichern. Die MOF-Klassen werden also in XML Tags abgebildet. Durch die Verwendung von XML können natürlich alle Vorzüge dieser Technologie ebenfalls verwendet werden. So ist es möglich, die Struktur gegen ein XML-Schema oder eine DTD zu prüfen, oder auch XQuery und XPath zu verwenden. Die XML Files können auch mit Extensible Stylesheet Language Transformation (XSLT) weiterverarbeitet werden (OMG, 2005).

Da das UML-Metamodell als MOF-Metamodell definiert ist, können somit auch UML-Modelle über XMI ausgetauscht werden.

Bisher freigegebene Spezifikationen für XMI von der OMG:

- **Version 1.0** wurde Anfang 1999 beschlossen. Die Standardisierung wurde vor allem von großen Unternehmen betrieben um einen Standard für objektorientierte Daten und deren Austausch zu bekommen. Dabei ist vor allem an UML, als die objektorientierte Modellierungssprache gedacht worden.
- Bereits im November wurde die **Version 1.1** freigegeben. Die neue Version beschäftigt sich mit den Anpassungen für die MOF 1.3 sowie den Einsatz von Namespaces.
- Anfang 2002 war eine neuerliche Anpassung an MOF 1.4 erforderlich, was zur **Version 1.2** des XMI führte.
- Im Mai 2003 wurde die Version 1.3 herausgegeben, die zwar inhaltlich gleich wie die Version 1.2 ist, aber erstmals Regeln für XML Schema enthält.
- Ebenfalls im März 2003 wurde die **Version 2.0** freigegeben. Diese unterstützt das Reverse-Engineering vom XML-Schema bzw. DTD, sowie vom XML-Dokument.
- Im Dezember 2005 wurde die **Version 2.1** freigegeben die die Neuerung zur MOF 2.0 berücksichtigt.

Anhang Abb. A-3: Vergleich der XMI Versionen

XMI	MOF/UML	DTD/XSD	Element Name	Elementverlinkung
1.0	MOF 1.3 UML 1.3	DTD	Foundation.Core.Class	xmi:id
1.1	MOF 1.3 UML 1.3	DTD	UML:Class	xmi.idref
1.2	MOF 1.4 UML 1.4	DTD	UML:Class	xmi.idref
1.3	MOF 1.4 UML 1.4	XSD	UML:Class	xmi.idref
2.0	MOF 1.4 UML 2.0	XSD	UML:Class	xmi.idref
2.1	MOF 2.0 UML 2.0, UML 2.1	XSD	UML:Class	xmi.idref

XMI Repräsentation eines UML-Klassendiagramms

Die Unterstützung für XMI in diversen UML-Werkzeugen beschränkt sich meist auf die Version 1.2 bzw. 1.3. Deshalb wird das Mapping

eines UML-Klassendiagramms in XMI anhand der XMI Version 1.2 (DTD[35]) bzw. 1.3 (XSD[36]) beschrieben.

Jedes XMI Dokument hat einen Root-Knoten „XMI". Dieser beinhaltet als Attribut die Versionsnummer des erzeugten XMI Dokuments. Innerhalb dieses Knotens befinden sich dann der „XMI.header" Knoten sowie der „XMI.content" Knoten. Der „XMI.header" Knoten beschreibt das Model bzw. Metamodel (hier UML), im „XMI.content" sind die tatsächlichen Elemente aus dem UML-Model zu finden.

Klassen werden im Knoten „UML:Class" abgebildet. Parameter wie der Klassenname, Sichtbarkeit und ähnliches werden als Attribute des Knotens abgebildet. Innerhalb der XMI-Struktur wird die Klasse über das Attribut „xmi.id" des XML-Knotens identifiziert.

Attribute einer Klasse werden im Knoten „UML:Attribute" abgebildet. Auch hier werden Parameter wie Name in den XML-Attributen des Knotens gespeichert. Die Zuordnung eines Attributs zu seiner Klasse erfolgt in der XMI-Datei durch die Struktur. Die „UML:Attribute"-Knoten werden innerhalb des „UML-Class" Knotens angelegt.

Assoziationen zwischen zwei oder mehreren Klassen werden im Knoten „UML:Association" gespeichert. Innerhalb des Knotens werden die Verbindungen im Knoten „UML:Association.Connection" zusammengefasst. Jede Connection besteht aus mindestens zwei „UML:AssocoationEnd" Knoten. Innerhalb dieser Knoten werden die Multiplizitäten („UML:AssociationEnd.multiplicity") sowie Referenzen zu den beteiligten Klassen („UML:AssociationEnd.participant") gespeichert. Der Typ der Assoziation wird im XML-Attribut „aggregation" des Knotens „UML:AssociationEnd" gespeichert. Assoziationsklassen („UML:AssociationClass") werden ähnlich wie Assoziati-

[35] Siehe http://www.w3.org/TR/html4/sgml/dtd.html

[36] Siehe http://www.w3.org/XML/Schema

onen gespeichert, jedoch werden zusätzlich noch die Attribute wie bei Klassen innerhalb des Knotens abgebildet.

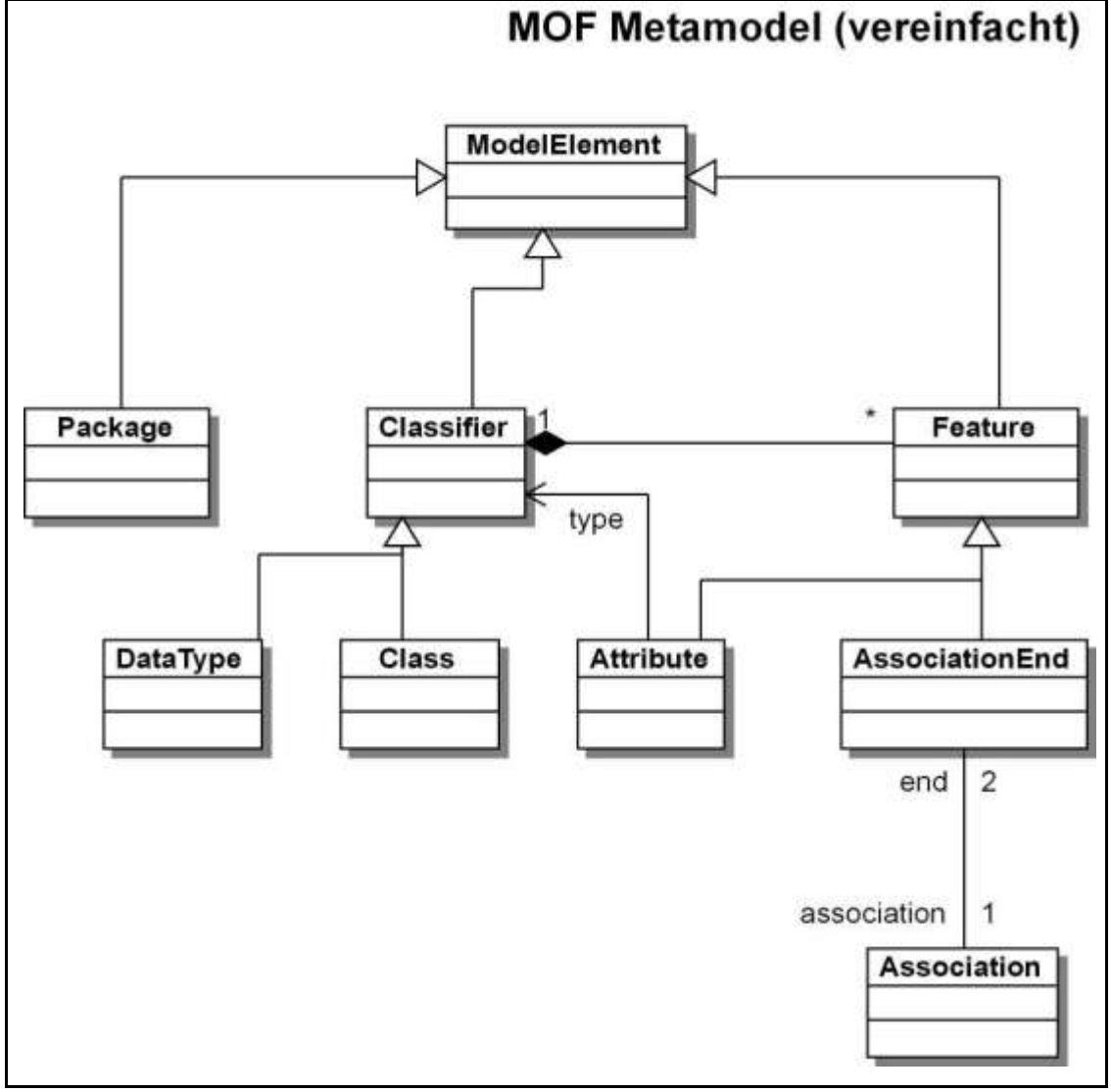
Anhang Abb. A-4: Klasse als XMI Knoten

```
<UML:Class xmi.id = '-64--88-2-2-6f1acc7:11ba79f8b49:-8000:0000000000000813'
  name = 'Pruefer' visibility = 'public' isSpecification = 'false' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
  <UML:GeneralizableElement.generalization>
     <UML:Generalization xmi.idref = '-64--88-2-2-6f1acc7:11ba79f8b49:-
8000:0000000000000822'/>
  </UML:GeneralizableElement.generalization>
  <UML:Classifier.feature>
    <UML:Attribute xmi.id = '-64--88-2-2-6f1acc7:11ba79f8b49:-8000:0000000000000815'
name = 'seit' visibility = 'public' isSpecification = 'false'  ownerScope =
'instance' changeability = 'changeable'
     targetScope = 'instance'>
      <UML:StructuralFeature.multiplicity>
       <UML:Multiplicity xmi.id = '-64--88-2-2-6f1acc7:11ba79f8b49:-
8000:000000000000081D'>
        <UML:Multiplicity.range>
         <UML:MultiplicityRange xmi.id = '-64--88-2-2-6f1acc7:11ba79f8b49:-
8000:000000000000081C' lower = '1' upper = '1'/>
        </UML:Multiplicity.range>
       </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:StructuralFeature.type>
       <UML:Class xmi.idref = '-64--88-2-2-6f1acc7:11ba79f8b49:-8000:
 000000000000081B'/>
      </UML:StructuralFeature.type>
     </UML:Attribute>
   </UML:Classifier.feature>
</UML:Class>
```

Generalisierungen werden im Knoten „UML:Generalization" gespeichert. Für Vaterklasse („UML:Generalization.parent") und Kindklasse („UML:Generalization.child") werden jeweils Referenzen zum Knoten „UML:Class" der beteiligten Klassen gespeichert.

Anhang Abb. A-4: zeigt einen Ausschnitt einer XMI-Datei mit dem Knoten „UML:Class" für die Klasse Prüfer aus dem Übungsbeispiel. Zur leichteren Lesbarkeit wurden die XML-Knoten blau und die Attributwerte rot gekennzeichnet.

Anhang Abb. A-5: Ausschnitt aus dem Metamodel der UML



Wie man an dieser Darstellung gut sehen kann werden lediglich die Metainformationen über die Struktur des Modells, nicht aber die Lageinformation der Elemente abgespeichert. Ohne diese grafische Information müssen sich Werkzeuge, die das Diagramm über ein XMI File importieren, selbst um die Visualisierung kümmern.

Die Object Managment Group hat deshalb im April 2006 die Spezifikation Diagram Interchange 1.0 herausgegeben. In dieser Spezifikation wird geregelt wie die grafische Lageinformation, also wo sich die einzelnen Elemente in der grafischen Darstellung des Models befinden, in XMI exportiert wird (vgl. (OMG, 2006)). Leider

unterstützen die verbreiteten Werkzeuge diese Spezifikation noch nicht. Abbildung A-5 zeigt das Meta-Modell der UML. Dieses Meta-Modell wird durch MOF beschrieben und diese Beschreibung wird im XMI Format verwendet, um die Modellinformation abzuspeichern.

B. Installation des Expertenmoduls

Neue E-Tutor Expertenmodule müssen dem E-Tutor Dispatcher bekannt gemacht werden. Dazu müssen eine Reihe von Konfigurationsdateien bearbeitet werden. Das vom E-Tutor Dispatcher verwendete Framework benötigt eine Konfigurationsdatei mit dem Namen „applicationContext.xml".

Anhang Abb. B-1: Ausschnitt aus applicationContext.xml

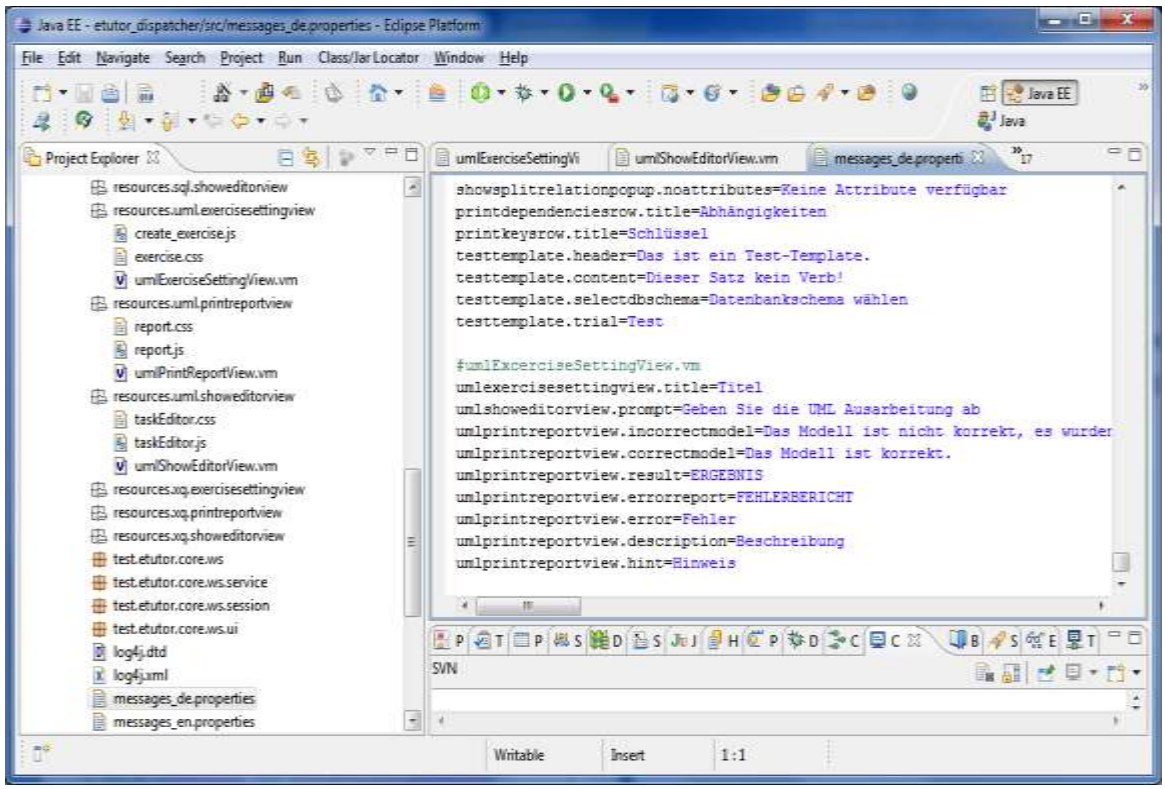
```
<!-- uml -->
 <bean id="umlVersion" class="java.lang.String">
  <constructor-arg value="3" />
 </bean>
 <bean id="umlImplementedFeedbacklevel" class="java.lang.String">
  <constructor-arg value="3" />
 </bean>

 <bean id="umlEvaluator" class="eTutor.modules.uml.UMLEvaluator" />
 <bean id="umlPrintReportView" class="etutor.modules.uml.ui.UMLPrintReportView"
>
  <property name="velocityEngine">
   <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
   <list>
    <value>/resources/uml/printreportview/report.js</value>
   </list>
  </property>
 </bean>
 <bean id="umlShowEditorView" class="etutor.modules.uml.ui.UMLShowEditorView" >
  <property name="velocityEngine">
   <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
   <list>
    <value>/resources/uml/showeditorview/taskEditor.js</value>
   </list>
  </property>
  <property name="cssFilenames">
   <list>
    <value>/resources/uml/showeditorview/exercise.css</value>
   </list>
  </property>
 </bean>
 <bean id="umlExerciseSettingView"
class="etutor.modules.uml.ui.UMLExerciseSettingView" >
  <property name="velocityEngine">
   <ref bean="velocityEngine"/>
  </property>
  <property name="filenames">
   <list>
    <value>/resources/uml/exercisesettingview/create_exercise.js</value>
    <val-
ue>/resources/uml/exercisesettingview/edit_area/edit_area_full.js</value>
   </list>
  </property>
  <property name="cssFilenames">
   <list>
    <value>/resources/uml/exercisesettingview/exercise.css</value>
   </list>
```

```
  </property>
 </bean>
 <bean id="umlEditor" class="etutor.modules.uml.ui.UMLEditor" />
 <bean id="umlExerciseManager" class="etutor.modules.uml.UMLExerciseManager" />

 <!-- uml -->
```

Abbildung B-1 zeigt den relevanten Ausschnitt aus der Konfigurationsdatei applicationContext.xml. Hier werden die Klassen und die entsprechenden Dateien für die Views eingetragen. Auch Versionsnummer und maximaler Feedbacklevel sind hier einzutragen.

Die Template Dateien für Velocity sowie die dazu benötigten Java-Script Dateien werden im Verzeichnis resources/uml des Projekts eTutor_dispatchers (E-Tutor Kern) bereitgestellt.

Anhang Abb. B-2: globale sprachabhängige Message Datei



Die Dateien müssen auf jeden Fall im angegebenen Verzeichnis vorhanden sein, da ansonsten die Anwendung nicht aufgerufen werden kann. Prüfungen der Eingabedaten müssen lokal erfolgen,

deshalb werden diese über die javascript-Engine des Webbrowsers durchgeführt. Die dazu benötigten javascript Funktionen sind in eigenen *.js Dateien ausgelagert. Formatierungen werden in eigenen Stylesheet Dateien mitgeliefert.

Die globalen Messagetexte („messages_de.properties" und „messages_en.properties") des E-Tutor Kerns müssen ebenfalls angepasst werden. Diese Dateien sind als Property Dateien im Projekt etutor_dispatcher zu finden (Abbildung B-2). Für die Sprachkennung wird, wie beim Ressource-Bundle, ein 2-stelliges Kürzel im Dateinamen verwendet. Die Dateien werden in das „Classes" Verzeichnis des Web-Server kopiert. Das Expertenmodul selbst wird als Java JAR-Datei im Web-Server bereitgestellt.

Installation von EditArea

Nach dem Download von http://sourceforge.net/projects/editarea/files/EditArea/EditArea%200.8.2/editarea_0_8_2.zip/download steht der Code von EditArea in einer komprimierten Archivdatei zur Verfügung. Das Softwarearchiv muss an eine beliebige Stelle des Web-Servers (zB direkt unter htdocs bei Apache) kopiert werden. Zur Verwendung in Webseiten muss auf die JavaScript Datei „edit_area_full.js" verwiesen werden.

Anhang Abb. B-3: Verwendung von EditArea in einer Webseite

```
<html>
<head>
<title>EditArea Test</title>
<script language="javascript" type="text/javascript"
src="/editarea/edit_area/edit_area_full.js"></script>
<script language="javascript" type="text/javascript">
editAreaLoader.init({
        id : "textarea_1"               // textarea id
        ,syntax: "xml"                  // syntax to be uses for
highgliting
        ,start_highlight: true          // to display with highlight
mode on start-up
});
</script>
</head>
```

```
<body>
<form method="post">
        <textarea id="textarea_1" name="content" cols="80"
rows="15"/>
</form>
</body>
</html>
```

Der Pfad zur JavaScript Datei mit dem Code für EditArea wird im Attribut „src" des Tags „<script>" angegeben. Der Verweis ist allerdings nur dann notwendig wenn „EditArea" auch tatsächlich bei dieser Web-Seite verwendet wird.

Durch den Aufruf der Funktion „editAreaLoader.init" wird ein eindeutiger Identifier für das Element vergeben, sowie andere Einstellungen vorgenommen. Eine Auflistung aller möglichen Einstellungen findet sich unter http://www.cdolivet.com/editarea/editarea/-docs/configuration.html. Zuletzt muss EditArea innerhalb des Tags „<form>" auf der Webseite aktiviert werden. Dazu muss ein HTML Element „<textarea>" verwendet werden. Im Attribut „id" dieses Elements muss der bei der Initialisierung verwendete Identifier eingetragen werden.

Konfiguration der Verbindung zur Datenbank

Die Verbindungsdaten zur Mustermodell Datenbank werden in der Klasse UMLConstants als Konstanten abgelegt (siehe Abb. B-4).

Anhang Abb. B-4: Verbindungsdaten als Konstanten

```
/**
 * class holds extra information in static members
 * @author Bernhard
 *
 */
public class UMLConstants {

 public static final String CONN_PWD = "uml";

 public static final String CONN_USER = "etutor_uml";

 public static final String CONN_URL =
"jdbc:oracle:thin:@localhost:1500:etutor";
```

C. Verzeichnisstruktur der beiliegenden CD

- **Dokumente**
  - DA
    In diesem Verzeichnis befindet sich die Diplomarbeit im Format PDF und als Microsoft Word Datei
  - Literatur
    Öffentliche Dokumente aus der Literaturliste, welche im PDF oder anderen elektronischen Formaten vorliegen.
  - Misc
    Hier befinden sich Bilder, Grafiken und Präsentationen zur Diplomarbeit
- **etutor_uml**
  Das gesamte Java Projekt inklusive der Eclipse 3.4 Projektdateien
  - src
    Kompletter Sourcecode des Prototypen für das Expertenmodul
  - lib
    Alle zur Kompilierung notwendigen Bibliotheken.
- **etutor_dispatcher**
  - src
    Die geänderten Dateien „messages_de.properties" sowie „messages_en.properties". Im Unterverzeichnis „resources/uml/" befinden sich die notwendigen Velocity Templates und Javascript bzw. Stylesheet Dateien.
- **Demo**
  - Mustermodell
    Hier liegen Demodateien für den UML-Editor Argo-UML, sowie die daraus erzeugten XMI-Dateien. Die generierte und erweiterte (mit Varianten) XML Datei ist ebenfalls hier zu finden.
  - Studentenmodell
    Test- und Demomodelle sowohl im Format Argo-UML als auch im exportierten XMI-Format.

Literaturverzeichnis

Aguirre-Urreta, M. I., & Marakas, G. M. (2008). Comparing conceptual modeling techniques: a critical review of the EER vs. OO empirical literature. *SIGMIS Database , 39* (2), S. 9-32.

Ambler, S. W. (2005). *The Elements of UML 2.0 Style.* New York: Cambridge University Press.

Angermeier, W. F. (1978). *Psychologie des Lernens II.* Hagen: Fernuniversität Hagen.

Arnold, P. (2005). *Einsatz digitaler Medien in der Hochschullehre aus lerntheoretischer Sicht.* Abgerufen am 22. 12 2008 von http://www.e-teaching.org/didaktik/theorie/lerntheorie/arnold.pdf

Baghaei, N., & Mitrovic, A. (2005). COLLECT-UML: Supporting Individual and Collaborative Learning of UML Class Diagrams in a Constraint-Based Intelligent Tutoring System. *KES 2005, LNAI 3684.* Berlin. Heidelberg: Springer-Verlag.

Baghaei, N., Mitrovic, A., & Irwin, W. (2006). Problem-Solving Support in a Constraint-based Tutor for UML Class Diagramms. *Technology, Instruction, Cognition and Learning Journal , 4* (1-2).

Balzert, H. (1999). *Lehrbuch der Objektmodellierung : Analyse und Entwurf.* Heidelberg ; Berlin: Spektrum, Akad. Verl.

Batra, M., & Marakas, G. M. (1995). Conceptual Data Modeling in Theory and Practice. *European Journal of Information Systems , 4* (3), S. 185-194.

Baumgartner, P., & Payr, S. (1994). *Lernen mit Software.* Innsbruck: Österreichischer Studien Verlag.

BF Skinner: Operant Conditioning. (kein Datum). Abgerufen am 22. 12 2008 von Simply Psychology: http://www.simplypsychology.pwp.blueyonder.co.uk/operant-conditioning.html

Boersch, I., Heinsohn, J., & Socher, R. (2007). *Wissensverarbeitung - Eine Einführung in die künstliche*

Intelligenz für Informatiker und Ingenieure (2. Auflage Ausg.). München: Elsevier.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *Das UML-Benutzerhandbuch* (2. Auflage Ausg.). (B. Kahlbrandt, & D. Reder, Übers.) Bonn: Addison-Wesley-Longmann.

Boudouries, M. A. (1998). Constructivism and Education - A shopping guide. *International Conference on Teaching of Mathematics.* Samos.

Brass, S. (2003). *Part 6: UML Class Diagrams.* Abgerufen am 29. 12 2008 von Datenbanken II: http://users.informatik.uni-halle.de/~brass/dd04/c6_umlcl.pdf

Brown, J. S., & Sleeman, D. (1982). *Intelligent Tutoring Systems.* London: Academic Press Ltd.

Burns, H. L., & Capps, C. G. (1988). Foundation of Intelligent Tutoring Systems - An Indroduction. In M. G. Polson, & J. J. Richardson, *Foundation of Intelligent Tutoring Systems.* Hillsdahle Hove and London: Lawrence Erlbaum Associates Publishers.

Chen, P. P. (2002). Entity-Relationship Modeling--Historical Events, Future Trends, and Lessons Learned. In M. Broy, & E. Denert, *Software Pioneers: Contributions to Software Engineering* (S. 100-114). Berlin: Springer-Verlag.

Chen, P. P. (1976). The entity-relationship model---toward a unified view of data. *ACM Trans. Database Syst , 1* (1).

Codd, E. F. (1990). *The relational model for database management* (Version 2 Ausg.). Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

Constantino-Gonzales, M. D., & Suthers, D. (2000). A Coached Colaborative Learning Environment for Entity-Relationship Modeling. *Intelligent Tutoring Systems* , S. 324-333.

Constantino-Gonzales, M. D., Suthers, D. D., & de los Santos, J. G. (2003). Coaching web-based collaborative learning based on problem solution differences and participation. *International Journal of Artifical Intelligence in Education* (13(2)), S. 263-299.

Dey, D., Storey, V. C., & Barron, T. M. (1999). Improving Database Design through the Analysis of Relationships. (ACM,

Hrsg.) *ACM Transactions on Database Systems (TODS)* (4), S. 453 - 486.

DKE. (2008). *Hompage des Instituts für Wirtschaftsinformatik - Data & Knowledge Engineering*. Abgerufen am 12. 11 2008 von http://www.dke.jku.at

Eichinger, C., Karlinger, M., & Nitzsche, G. (2006). *eTutor Modularchitektur.* Linz.

Einführung: Kognitivismus. (kein Datum). Abgerufen am 21. 12 2008 von http://www.edit.uni-essen.de/lp/kognitiv/kognitiv.htm

Fischer, C. (2010). *Integration des intelligenten tutoriellen Systems eTutor in die Lernumgebung Moodle.* Linz: Johannes Kepler Universität.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.* Bonn: Addison-Wesley-Longman.

Gopinath, S. (2006). *Real-Time UML to XMI Conversion.* Master Thesis, Stockholm.

Gordon, A., & Hall, L. (2000). Actively Supporting Collaboration in Virtual Learning Environments. *Proceedings of the Thirteenth International Florida Artifical Intelligence Research Society Conference*, (S. 44).

Hartmann, S., & Link, S. (2007). English sentence structures and EER modeling. *APCCM '07: Proceedings of the fourth Asia-Pacific conference on Conceptual modelling* (S. 27-35). Ballarat, Australia: Australian Computer Society, Inc.

Hawkins, H. H., Young, S. K., & Hubert, K. C. (2001). Conceptual Database Modeling for Understanding and Developing Information Management Applications. *RadioGraphics , 21* (3).

Heinrich, L. J., & Roithmayr, F. (1998). *Wirtschaftsinformatik - Lexikon* (6., vollständig überarbeitete und erweiterte Auflage Ausg.). München Wien: R. Oldenbourg Verlag.

Herrnstein, R. J. (1970). On the Law of Effect. *Journal of the Experimental Analysis of Behaviour*, *Number 2.*

Hilgard, E. R., & Bower, G. H. (1973). *Theorien des Lernens I und II.* Stuttgart: Ernst Klett Verlag.

Hofer, A. (2005). *E-Exercises in Data & Knowledge Engineering: Konzeption und Entwicklung eines intelligenten tutoriellen Systems.* Linz: JKU Linz.

Huang, S., Gohel, V., & Hsu, S. (2007). Towards interoperability of UML tools for exchanging high-fidelity diagrams. *SIGDOC '07: Proceedings of the 25th annual ACM international conference on Design of communication* (S. 134-141). El Paso, Texas, USA: ACM.

Kemper, A., & Eickler, A. (2006). *Datenbanksysteme - Eine Einführung.* Oldenbourg: Oldenbourg Wissenschaftsverlag.

Lusti, M. (1992). *Intelligente Tutorielle Systeme - Einführung in wissensbasierte Lernsysteme* (Bd. 15.4). München Wien: R. Oldenbourg Verlag.

Martens, A. (2003). *Ein Tutoring Prozess Modell für fallbasierte Intelligente Tutoring Systeme.* Dissertation, Fakultät für Ingenieurwissenschaften der Universität Rostock, Rostock.

Mitrovic, A., Koedinger, K. R., & Martin, B. (2003). A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. Berlin Heidelberg: Springer Verlag.

Mitrovic, A., Martin, B., & Suraweera, P. (2007). Intelligent Tutors for All: The Constraint-Based Approach. *IEEE Intelligent Systems* (22(4)), S. 38-45.

Neubert, S., Reich, K., & Voß, R. (2001). Lernen als konstruktiver Prozess. *Die Wissenschaft und ihr Wissen , Bd. 1*.

OMG. (April 2006). *Diagram Interchange Version 1.0.* Abgerufen am 28. 12 2008 von http://www.omg.org/docs/formal/06-04-04.pdf

OMG. (2004). *Human-Usable Textual Notation (HUTN) Specification.* Object Management Group, Inc.

OMG. (Juli 2005). *Meta Object Facility (MOF) Specification Version 1.4.1.* Abgerufen am 08. 01 2009 von http://www.omg.org/docs/formal/05-05-05.pdf

OMG. (2005). *Unified Modeling Language Specification.* Specification, Object Management Group.

OMG. (Juli 2005). *XML Metadata Interchange Specification Version 2.0.1.* Abgerufen am 08. 01 2009 von http://www.omg.org/docs/formal/05-05-06.pdf

Overmyer, S. P., Lavoie, B., & Rambow, O. (2001). Conceptual Modeling through Linguistic Analysis Using LIDA. *Proceedings of the 23rd International Conference on Software Engineering* , (S. 401-410). Toronto, Ontario, Canada .

Piaget, J. (1975). *Nachahmung, Spiel und Traum.* Klett Verlag.

Rechenberg, P. (1994). *Was ist Informatik? Eine allgemeinverständliche Einführung* (2. bearbeitete und erweiterte Auflage Ausg.). München Wien: Carl Hanser Verlag.

Reich, K. (1996). *Systemisch-konstruktivistische Didaktik.* Abgerufen am 21. 12 2008 von http://www.uni-koeln.de/hf/konstrukt/reich_works/aufsatze/reich_18.pdf

Rottach, T., & Groß, S. (2002). *XML kompakt: die wichtigsten Standards.* Heidelberg Berlin: Spektrum Akademischer Verlag.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1993). *Objektorientiertes Modellieren und Entwerfen.* (D. Märtin, Übers.) München Wien: Carl Hanser Verlag.

Schmidt, D., Stal, M., Rohnert, H., & Buschmann, F. (2002). *Patter orientierte Softwarearchitektur - Muster für nebenläufige und vernetzte Objekte.* Heidelberg: dpunkt.verlag GmbH.

Schrefl, M., Preuner, G., & Thalhammer, T. (2006). *Konzeptueller Entwurf mit der Unified Modeling Language (UML).* Linz: Institut für Wirtschaftsinformatik - Data & Knowledge Engineering.

Siebert, H. (1998). *Konstruktivismus : Konsequenzen für Bildungsmanagement und Seminargestaltung.* Abgerufen am 21. 12 2008 von http://www.die-bonn.de/esprid/dokumente/doc-1998/siebert98_01.pdf

Silberschatz, A., Korth, H. F., & Sudarshan, S. (1996). *Database System Concepts* (Third Edition Ausg.). Boston: McGraw-Hill.

Simons, A. J., & Graham, I. (1999). 30 Things that go wrong in object modelling with UML 1.3, chapter 17. In H. Kilov, B.

Rumpe, & I. Simmonds, *Behavioral Specifications of Businesses and Systems* (S. 237-257). Kluwer Academic Publishers.

Skinner, B. F. (1950). Are Theories of Learning necessary? *Psychological Review , 57*, S. 193-216.

Smith, J. M., & Smith, D. C. (1977). Database Abstractions: Aggregation and Gerneralization. *ACM Transactions on Database Systems* (2), S. 105–133.

Suraweera, P., & Mitrovic, A. (2002). KERMIT: A Constraint-Based Tutor for Database Modeling. *Intelligent Tutoring Systems: 6th International Conference, ITS 2002.* Biarritz, France.

Tigris. (2008). *ArgoUML*. Abgerufen am 28. 12 2008 von http://argouml.tigris.org/

Urban-Lurain, M. (1996). *Intelligent Tutoring Systems.* Abgerufen am 22. 12 2008 von http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm

Villar Angulo, L., & Alegre de la Rosa, O. (2008). *Supporting computer-mediated learning: A case study in online staff development and classroom learning environment assessment.* Springer Netherlands.

Waba, S. (2005). *Didaktische Konzepte.* Abgerufen am 22. 12 2008 von http://ilias.vhs21.ac.at/2bw/deutsch/sgl/diskussion/Lebenslang_Lernen_Didaktische_Konzepte.doc

Wikipedia. (2008). *Wikipedia*. Abgerufen am 27. 12 2008 von Unified Modeling Language: http://en.wikipedia.org/wiki/File:OO-historie.jpg

Zakharov, K., Mitrovic, A., & Ohlsson, S. (2005). Feedback Micro-engineering in EER-Tutor. *Proceedings of the 2005 conference on Artifical Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology*, (S. 718-725).

Zöller-Greer, P. (2007). *Künstliche Intelligenz - Grundlagen und Anwendungen.* Wächtersbach: Composia Verlag.