TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Umsetzung einer e-Learning Applikation für SQL mithilfe eines regelbasierten Bewertungssystems

Ausgeführt am

Institut für Informationssysteme
Arbeitsbereich: Datenbanken und Artificial Intelligence

der Technischen Universität Wien

unter Anleitung von
Prof.Dr. Reinhard Pichler
und Dipl.Ing. Katrin Seyr als der verantwortlich mitwirkenden Mitarbeiterin

durch

Peter Gansterer
1040 Wien, Karolinengasse 29/6

Datum

Unterschrift

Danksagung

Mein Dank geht and Prof.Dr. Reinhard Pichler und Dipl.Ing. Katrin Seyr, die mir diese Arbeit ermöglicht haben.

Ich danke Dir, Margaretha, dafür, dass Du meine andauernde Abwesenheit im Privatleben so verständnisvoll erdulden konntest, und auch für die hilfreiche Unterstützung bei der Arbeit selbst.

Auch die Generosität von Rudi Bauer und meiner Kollegen ist mir sehr zugute gekommen, und ich konnte so des Öfteren den einen oder anderen Tag von der Arbeitszeit im Büro abzweigen. Ohne diese Hilfe hätte sich diese Arbeit wohl noch lange hingezogen.

Abstracts

Deutsch

Für die Lehrveranstaltung „Datenmodellierung“ wird ein e-Learning System entwickelt, das den vorhandenen Modus der Prüfungen und Übungen für das Lösen von SQL Aufgaben ersetzen soll. Es wird eine Webapplikation entwickelt, wo Studierende Aufgabenstellungen online bearbeiten können und direktes Feedback über die Richtigkeit der Lösung bekommen. Ein Ziel dabei ist die Entlastung der Lehrveranstaltungsbetreuer durch die automatische Bewertung der Lösungen.

Die beiden wesentlichen Komponenten bei der Umsetzung des Systems waren: ein System zur Bewertung der Aufgaben und ein möglichst effizientes Userinterface.

Zur Bewertung einer SQL Eingabe wird das Statement auf der entsprechenden Übungsdatenbank ausgeführt und die Ergebnistupel werden mit dem Soll-Ergebnis verglichen. Zusätzlich wird die Eingabe auch auf strukturelle bzw. semantische Korrektheit getestet.

Bei der Entwicklung des Userinterfaces wurde auf Übersichtlichkeit und effiziente Navigation geachtet aber auch die Aufbereitung des Userfeedbacks spielt dabei eine wesentliche Rolle.

Für die semantische Bewertung, für die Interpretation der Ergebnisdifferenzen und für die Aufbereitung des Userfeedbacks kommt ein einfaches, eigens entwickeltes Regelsystem zum Einsatz. Die vorgestellten Konzepte wurden in einer Prototyp-Implementierung erarbeitet und getestet.

English

An e-Learning system is developed for the course "Datenmodellierung", which will replace the existing mode of exams and tutorials in the domain where SQL exercises are to be solved. The system is a web-based application, where students can work on exercises online and receive immediate feedback about correctness of their solutions. One major objective is to make life easier for the supervisors of the course by supplying automatic assessment of students' solutions.

Two main components of the implementation were: a system to automatically assess exercise solutions and an efficient user interface.

For evaluation of an SQL input, the statement is executed at the corresponding course database and the resultset is compared to the expected resultset. In addition, the input is checked for structural and semantic correctness.

The user interface was designed to be clear and efficient in navigation. Also important was the question of how to present user feedback.

Semantic evaluation, interpretation of resultset differences and preparation of user feedback is done by a rule based system, which is developed especially for those tasks. Herein presented concepts were developed and tested by implementing a prototype system.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Jedes Semester besuchen 500 bis 700 Studierende die Lehrveranstaltung „Datenmodellierung“, in der sie unter anderem lernen sollen, die Datenbankabfragesprache SQL korrekt einzusetzen. Dabei sind für unterschiedlich anspruchsvolle Aufgaben „SELECT“ Statements als Lösung zu erarbeiten.

Übung und Prüfung der Lehrveranstaltung sind nach folgendem Prinzip organisiert: Es wird ein Thema vorgegeben, zu dem eine Übungsdatenbank gegeben wird. Ein Beispiel wäre das Thema „Sport“ mit einer Übungsdatenbank mit Tabellen für Sportler, Sportarten, Teams, Speisen, Mahlzeiten, etc. Als Angabe bekommt man die Beschreibung der Datenbank (Tabellen, Attribute, Constraints) und eine Reihe von Aufgabenstellungen, die als SQL Statements zu lösen sind.

Im praktischen Übungsteil der Lehrveranstaltung können die Aufgabenstellungen zuhause gelöst werden. Dabei steht derzeit auch die Möglichkeit zur Verfügung, die SQL Abfragen über eine HTML Schnittstelle abzusetzen, oder die Beispieldatenbank selbst bei sich am PC einzurichten. Die Ergebnisse werden im Rahmen eines Tutorengesprächs bewertet.

Die Prüfung, auf der anderen Seite, wird schriftlich ausgeführt, indem für konkrete Aufgabenstellungen Lösungen in Form von SQL Abfragen zu finden sind. Die Lösungen werden handschriftlich am Prüfungsbogen abgegeben.

Die Prüfung umfasst noch andere inhaltliche Teile der Lehrveranstaltung, aber die SQL Abfragen stellen für alle Beteiligten das größte Problem dar. Einerseits sind die Studierenden unzufrieden, weil sie Lösungen *handschriftlich* erarbeiten sollen – für Aufgabenstellungen die für eine solche Vorgehensweise sehr schlecht geeignet sind. Andererseits ist der Aufwand für die Leiter der Lehrveranstaltung sehr hoch, weil jedes Semester Unmengen von SQL Statements zu bewerten sind. Diese Statements sind noch dazu in vielen Fällen sehr komplex. Es kann meist nicht von einer einzigen gültigen Lösung ausgegangen werden. Jede Lösung jedes Studierenden könnte potenziell auch zum richtigen Ergenbnis führen.

e-Learning System

Nun ist ja SQL eine gut definierte technische Sprache, so dass es fast naheliegend scheint, eine Lösung zu erarbeiten, die den Prozess der Übung, Prüfung und Bewertung automatisiert. Das vorhandene System zum Absetzen von SQL Abfragen über eine HTML Schnittstelle stellt einen ersten Schritt in die Richtung dar, in die das neue e-Learning System gehen soll: Die Studierenden sollen die Möglichkeit bekommen, die Aufgabenstellungen vollständig online zu lösen. Dazu soll eine Applikation zum Einsatz kommen, die sowohl den Übungsteil der Lehrveranstaltung abdeckt, als auch den Prüfungsteil. Diese Applikation soll außerdem die Lösungen der Studierenden automatisch bewerten können, so dass damit auch der Aufwand für die Lehrveranstaltungsleiter verringert wird.

1.1 Aufgabenstellung

Das neue e-Learning System arbeitet nach demselben Prinzip, wie die derzeit gehaltenen Übungen und Prüfungen: Es wird ein Thema vorgegeben aufgrund dessen Aufgabenstellungen zu lösen sind.

Kurs und Übungsdatenbank

Das System soll auf der Grundlage arbeiten, dass die SQL Abfragen, die von den Studierenden als Lösung eingegeben werden, auf einer Übungsdatenbank ausgeführt werden und das Ergebnis mit einem Soll-Ergebnis verglichen wird. D.h. die Grundlage für die Aufgabenstellungen bildet eine Übungsdatenbank. Die Gesamtheit von *Übungsdatenbank*, *Beschreibung der Datenbank* und der *Aufgabenstellungen* soll hier im Sinne eines Lernsystems als *Kurs* bezeichnet werden. Ein Kurs hat ein bestimmtes Thema als Grundlage (wie zuvor beschrieben, z.B. das Thema Sport). Die Übungsdatenbank beinhaltet Tabellen und Beispieldaten zum Thema des Kurses.

Die Einschränkung auf „SELECT“ Abfragen macht es unproblematisch, die Abfragen direkt einer einzelnen Übungsdatenbank durchzuführen. Es können alle Studierenden gleichzeitig auf derselben Datenbank arbeiten, weil keine Berechtigungen für das Ändern von Daten vergeben werden.

Bewertung

Eine Aufgabe eines Kurses wird gelöst durch Eingabe einer SQL Abfrage in der Webschnittstelle. Die eingegebene Abfrage wird vom System ausgeführt auf der Übungsdatenbank und das dabei gelieferte Ergebnis wird verglichen mit einem vordefinierten erwarteten Ergebnis.

Zusätzlich soll die eingegebene Abfrage inhaltlich bewertet werden. Das wird bewerkstelligt, indem Eigenschaften der Abfrage verglichen werden mit Eigenschaften einer oder mehrerer Referenzabfragen (die als korrekte Lösungen vorgegeben sind). Damit wird einerseits für jedes Beispiel eine abgestufte Bewertung ermöglicht, andererseits kann damit überprüft werden, ob der Weg, der in der Lösung gewählt wurde, auch tatsächlich der in der Angabe geforderte Lösungsweg ist. Die Problematik des Lösungswegs soll in folgendem Beispiel veranschaulicht werden:

In der Angabe ist durch die Aufgabenstellung ein SQL Statement mit einem *nested select* gefordert, das aus einer Tabelle beispielsweise ein Maximum suchen soll, um dieses in der übergeordneten Abfrage zu verwenden. Wenn anstelle des *nested selects* einfach eine zuvor interaktiv ermittelte Konstante eingesetzt wird, so ist die Ergebnismenge dieselbe wie die erwartete. Dennoch kann diese Lösung nicht mit der vollen Punktezahl bewertet werden, da hier offensichtlich eine „Abkürzung“ verwendet wurde.

Übungs- und Prüfungsmodul

In Abbildung 1.1 ist ein Überblick der Konzepte des e-Learning Systems dargestellt.

Das System wird als Webapplikation umgesetzt, die einerseits die praktische Übung als auch ein Modul für die Prüfung beinhaltet. Die Übung kann von den Studierenden online am Browser durchgeführt und die Lösungen auf diesem Weg abgegeben werden. Für die Übung wird aber dennoch das Tutorengespräch als Bewertungsgrundlage herangezogen.

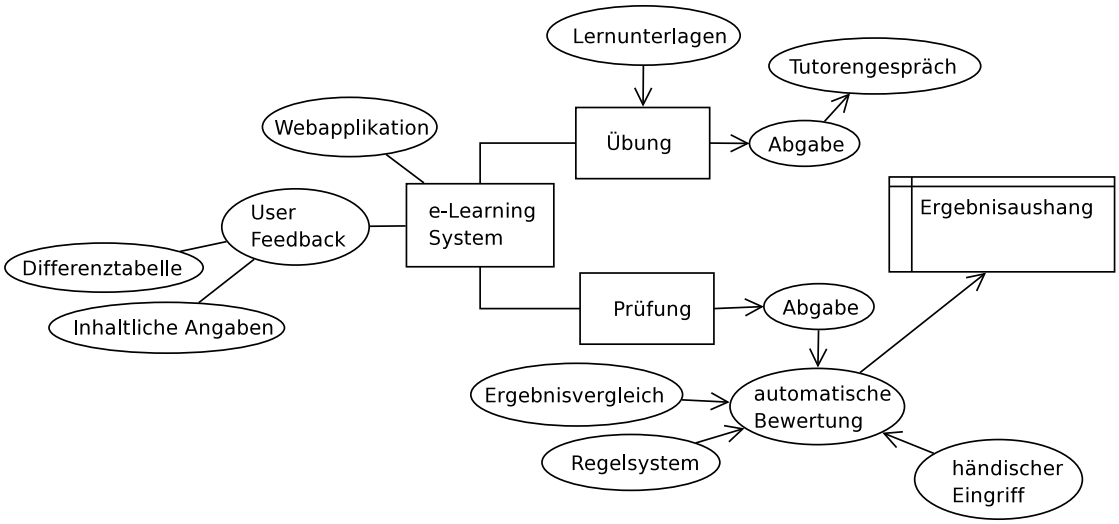


Abbildung 1.1: Konzeptioneller Systemüberblick

Die Prüfung wird in einem Prüfungsraum abgehalten - dort aber ebenfalls mit derselben Applikation und einem Browser. Die Prüfungen werden von den Studierenden abgegeben und dann automatisch bewertet. Die Lehrveranstaltungsleiter haben allerdings weiterhin die Möglichkeit, die automatische Bewertung in Einzelfällen mit einer eigenen Punktebewertung zu übergehen. Um das zu ermöglichen, wird das Ergebnis nicht sofort bekannt gegeben, sondern auf üblichem Wege, nach Abwicklung aller Prüfungen (eines Prüfungstermins) und Bearbeitung der Ergebnisse, ausgehängt.

Die interaktiv erarbeiteten Lösungen im Übugnsteil werden für die Bewertung im Tutorengespräch als Grundlage verwendet.

Im Sinne eines vollständigen e-Learning Systems werden auch die Unterlagen zur Verfügung gestellt, die dem Studierenden das Erlernen von SQL ermöglichen. Eine SQL Referenz oder andere Lehrveranstaltungsunterlagen können einfach online abgerufen werden.

Feedbacksystem

Ein umfangreiches Feedbacksystem soll dem Studierenden Aufschluss über die Probleme in seiner Lösung geben, so dass er diese jederzeit iterativ verbessern kann. Das Feedback besteht aus der Darstellung einer Tabelle, in der die Differenzen zum erwarteten Ergebnis hervorgehoben werden („Differenztabelle“) und einer Liste von Angaben über die inhaltliche Struktur der Eingabe, wie z.B. „In der Eingabe wird ein GROUP BY erwartet“.

1.2 Umsetzung

Im Rahmen dieser Arbeit wurde ein Prototyp für das System entwickelt, anhand dessen die wesentlichen Konzepte erprobt und verifiziert wurden. In diesem Prototyp wurde das *Übungsmodul* umgesetzt, das Userinterface mit den konzepten des User Feedbacks erarbeitet, sowie das Bewertungssystem implementiert und auf Verwendbarkeit getestet.

Das Modul zum Vergleich von SQL Resultsets wurde konzipiert und implementiert. Für das Bewertungssystem wurde die Vorgehensweise gewählt, ein regelbasiertes System zum Einsatz zu bringen, das die Flexibilität der Bewertung sicherstellen soll. Es wurde ein eigenes Regelsystem für die e-Learning Applikation entworfen und implementiert.

Nachdem diese Konzepte im Prototyp erarbeitet wurden, wird das vollständige System aufbauend auf diesem Prototyp umgesetzt.

1.3 Überblick

Die Kapitel 2 und 3 geben eine Beschreibung des umgesetzten Systems. Ausgehend von den Anforderungen über die gewählte Architektur bis hin zum Datenmodell werden die verwendeten Technologien, Konzepte und Strukturen beschrieben.

In Kapitel 4 und 5 wird auf die Aspekte des e-Learning eingegagen. Dabei wird in Kapitel 4 zuerst ein Überblick zum Thema e-Learning allgemein gegeben. Danach wird im Detail auf den Entwurf des Userinterfaces eingegangen, das dem Studierenden das Zurechtfinden im System leicht machen soll und ein interaktives Erlernen von SQL ermöglichen soll.

Die Kapitel 6, 7 und 8 beschreiben die technischen Konzepte des Bewertungssystems. Dabei wird zunächst in Kapitel 6 auf die Probleme und deren Lösungen beim Vergleich von Abfrageergebnissen eingegangen. In Kapitel 7 wird ein allgemeiner Überblick zum Thema „regelbasierte Systeme“ gegeben worauf schließlich eine detaillierte Beschreibung des umgesetzten Systems in Kapitel 8 folgt.

2 Anforderungen und Systemdesign

Dieses Kapitel beschreibt die gewählten Konzepte bei der Umsetzung des Systems. Zuerst werden die Anforderungen dargestellt, welche die Grundlage für alle Überlegungen bilden. Danach werden in den Abschnitten 2.2 und 2.3 einzelne Aspekte des Systemdesigns behandelt.

2.1 Anforderungen

In diesem Abschnitt werden die wesentlichen Anforderungen beschrieben, die zu den gewählten Systemdesign Ansätzen geführt haben. Hier werden nicht nur die Anforderungen an die Architektur und technische Umgebung beschrieben, sondern auch funktionale Anforderungen, auf die auch im Userinterface Design referenziert wird (siehe Kapitel 5).

2.1.1 Systemumgebung

Das System soll in Java entwickelt werden und auf einem Oracle Application Server zum Einsatz kommen. Es können dadurch die APIs der J2EE Umgebung verwendet werden. Wie bereits in der Einleitung deutlich gemacht, soll ein Webinterface die Schittstelle zum User darstellen. Es wird keine weiteren Teile geben, die außerhalb der Webapplikation zum Einsatz kommen müssten.

Für den Übungsteil werden die Studenten die Möglichkeit haben, den Stand ihrer Arbeit in eine Datei zu exportieren und später aus dieser Datei wieder ins System hochzuladen. Damit wird auch das Arbeiten offline ermöglicht: man kann die Datei mit den Angaben und Lösungen direkt editieren. Durch diese Vorgangsweise wird außerdem auf die Verwaltung von persistenten Usersessions für den Übungsteil verzichtet. Die Session wird quasi explizit durch das exportierte File repräsentiert.

Andererseits sollen die Sessions für Prüfungen auf jeden Fall persistent in der Datenbank abgelegt werden. Das soll organisatorische Probleme bei eventuellen technischen Komplikationen (wie z.B. Stromausfall) verringern.

Was nicht Teil des Systems ist, ist die Verwaltung von Studentenaccounts, Prüfungsanmeldung, etc. Diese Aufgaben werden von dem bereits produktiven Lehrveranstaltungssystem (*DBAI LVA Manager*) abgedeckt. Das neue e-Learning System muss allerdings mit Schnittstellen zu diesen Daten ausgestattet sein.

Auch nicht Teil des Systems wird die Bewertung des Übungsteils sein. Die Übungsaufgaben können abgegeben werden, werden auch automatisch bewertet, aber die endgültige Bewertung (und das Eintragen im LVA Manager) obliegt den Tutoren beim Tutorengespräch.

Die Definition und Verwaltung der Übungsdatenbanken (siehe Abschnitt 2.1.3), wird ebenfalls nicht vom System abgedeckt. Die Übungsdatenbanken werden von den Lehrveranstaltungsleitern definiert und mit Beispieldaten befüllt. Im e-Learning System muss lediglich die

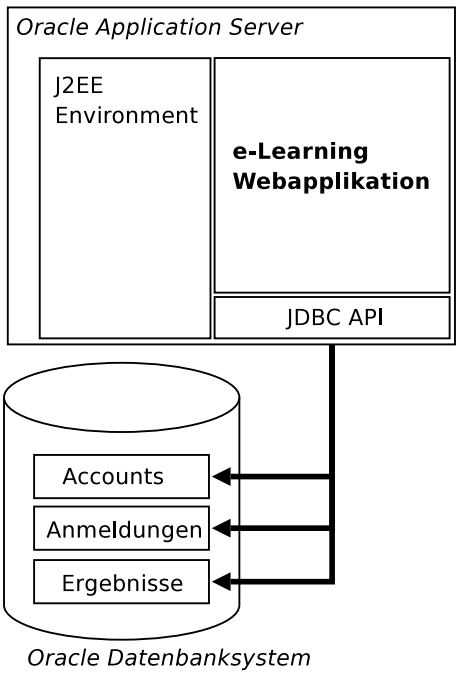


Abbildung 2.1: Schnittstellen und Umgebung des Systems

Information hinterlegt werden, wie auf diese Datenbank zugegriffen wird (JDBC Zugangsdaten).

2.1.2 Schnittstellen

Daraus ergeben sich auch die wesentlichen Schnittstellen, mit denen das System zu tun hat. In Abbildung 2.1 wird eine Übersicht der Schnittstellen schematisch dargestellt. Abgesehen von den APIs der J2EE Umgebung, muss das System noch mit der Datenbank des LVA Systems „kommunizieren“. Im LVA Manager wird insbesondere Folgendes verwaltet:

- Die Verwaltung der Studentenaccounts mit Username & Passwort. Das e-Learning System soll beim Login direkt auf diese Tabelle zugreifen.

- Die Verwaltung der Prüfungsanmeldungen. Beim Login für eine bestimmte Prüfung wird die Anmeldung überprüft.

- Die Ergebnisse der Prüfungen werden nach dem Ende eines Bewertungszyklus ins LVA System (geblockt) übernommen. Im Gegensatz zu den beiden anderen Zugriffen, wird diese Datenübernahme durch eine eigene „Übernahme Aktion“ angestoßen und geschieht nicht automatisch bei jeder abgelegten Prüfung.

Als Schnittstelle zum LVA Manager werden keine vorhandenen Java Libraries verwendet, sondern es wird direkt auf die Datenbank zugegriffen.

2.1.3 Datenstrukturen

Die Datenstrukturen, die dem System zugrunde liegen sind auf logischer Ebene relativ unkompliziert und werden hier im Überblick angeführt:

- Basis des e-Learning Systems bildet ein *Kurs*. Der Kurs ist die Sammlung von Aufgabenstellungen, die in einem Semester im Übungsteil bearbeitet, als auch der Aufgaben, die bei einer Prüfung gestellt werden. In einem Kurs wird eine Übungsdatenbank mit Datenbankschema und Beispieldaten definiert. Die Kursdefinition umfasst die Beschreibung des Datenbankschemas, ein EER Modell zur Veranschaulichung und die Definition der einzelnen Aufgaben (siehe unten). Jedes Semester wird ein neuer Kurs definiert. Die Studenten können aber zum Üben jederzeit auf vergangene Kurse zurückgreifen, dort allerdings keine Aufgaben zur Bewertung in der Übung abgeben.

- In jedem Kurs wird eine Reihe von *Aufgaben* definiert. Die Aufgaben sind die Übungseinheiten, in denen eine Aufgabenstellung vorgegeben wird, die der Student mittels SQL Eingabe zu lösen hat. Ein Kurs umfasst typischerweise ca. 30 Aufgaben. Eine Aufgabe besteht aus dem Text der Aufgabenstellung, einem Schwierigkeitsgrad und einer Reihe von Referenzlösungen (das sind SQL Statements, welche die Aufgaben korrekt lösen und als Grundlage für die automatische Bewertung verwendet werden). Die Aufgaben sind eindeutig einem Kurs zugeordnet und sind somit mit der entsprechenden Übungsdatenbank verbunden. Für eine Aufgabe ist definiert, ob sie bei der Prüfung zum Einsatz kommen soll oder nicht.

- Die *Prüfung* besteht im Grunde nur aus einem Termin, wann sie stattgefunden hat. Im Rahmen der Prüfung selbst wird jedem Teilnehmer per Zufallsauswahl eine Übungsdatenbank (und damit ein Kurs) zugewiesen. Die Kurse, die zur Auswahl für eine Prüfung stehen, können auch öffentlich im Übungsmodul zur Verfügung stehen, müssen aber nicht. Welche Kurse im Übungsmodul zur Verfügung stehen und welche für Prüfungen zur Auswahl stehen wird unabhängig gesteuert.

  Wenn ein Student bei einer Prüfung teilgenommen hat, werden die Aufgaben, die er gelöst hat als Zuordnung von Student zu Prüfung gespeichert. Auch die Aufgabenstellungen, die in einer Prüfung pro Student gestellt werden, werden per Zufall aus der Menge der Aufgaben des Kurses ausgewählt.

Weitere Details zu den Datenstrukturen sowie den Konzepten der Historisierung sind in Kapitel 3 zu finden.

2.1.4 Anforderungstabelle

Die weiteren Anforderungen (zusammen mit teilweise bereits erwähnten Anforderungen) werden in Tabelle 2.1 angeführt. Die Anforderungen werden kurz beschrieben, mit einem Schlagwort ausgezeichnet und einer der folgenden Kategorien zugeordnet:

- *Funktionale Anforderungen* (`FKT`) sind Anforderungen an die Funktionalität des Systems. Diese Anforderungen beschreiben im Großen und Ganzen den Umfang des Systems.

- Die *nichtfunktionalen Anforderungen* (`NF`) beinhalten technologische Anforderungen oder auch Anforderungen an das Verhalten des Systems, wie z.B. Antwortzeitverhalten.
- Die *Schnittstellenanforderungen* (`IF` für *Interface*) beinhalten Schnittstellen zu anderen Systemen mit denen zu arbeiten ist.
- *Userinterface Anforderungen* (`UI`) legen Details zum Userinterface fest. Das kann sich teilweise mit den funktionalen Anforderungen überdecken, weil große Teile der Funktionalität im Userinterface abgebildet werden.

Tabelle 2.1: Anforderungstabelle

Kat.	**Schlagwort**	**Beschreibung**
`FKT`	Inh. Bewertung	Eine einzelne Aufgabe (bzw. die entsprechende Eingabe des Benutzers) wird nicht nur mit 0 oder 1, sondern auch inhaltlich bewertet und es wird eine Punktezahl vergeben.
`FKT`	Nur Queries	Die Aufgabenstellungen bestehen allein aus SQL Abfragen, die in der Kursdatenbank auszuführen sind. Es werden keine Zusatzfragen oder ähnliches gestellt.
`FKT`	Sessionexport	Die Datei, die beim Exportieren der aktuellen Session gespeichert wird, soll so gestaltet sein, dass man diese auch offline bearbeiten und die Beispiele lösen kann. Das heisst insbesondere, dass die Angabentexte zu den Aufgaben mit exportiert werden müssen.
`FKT`	Zufallsauswahl	Der Kurs, bzw. die Übungsdatenbank für eine Prüfung wird pro Teilnehmer zufällig aus einer Menge von Kursen ausgewählt, die für Prüfungen freigegeben sind. Die Aufgaben der Prüfung werden ebenfalls per Zufall aus der Menge der Aufgaben des gewählten Kurses genommen. Dabei wird pro Schwierigkeitsgrad eine bestimmte Anzahl von Beispielen gewählt.
`FKT`	Pkt.Anpassung	Die Ergebnisse der automatischen Bewertung können im Nachhinein vom Lehrveranstaltungsleiter angepasst werden. Es können damit jederzeit Probleme im Bewertungssystem oder mit Spezialfällen in der Lösung händisch umgangen werden.
`FKT`	Nur SELECT	Die Aufgaben sind nur mit SELECT Statements zu lösen. Es werden keine Manipulationen an der Übungsdatenbank vorgenommen und auch keine Views definiert.
`FKT`	Zeitlimit	Im Prüfungsmodus wird dem Benutzer ein Zeitlimit für seine Prüfung vorgegeben. Dieses Zeitlimit wird im System berücksichtigt und auch für den User deutlich angezeigt.
`NF`	Kompatibilität	Die Oberfläche im Browser muss möglichst kompatibel für alle gängigen Browser sein. Das System soll offen zugängig sein. Deshalb kann keine Einschränkung auf einen bestimmten Browser gemacht werden.

Fortsetzung auf der folgenden Seite...

Tabelle 2.1 – Fortsetzung

Kat.	Schlagwort	Beschreibung
NF	Javascript	Javascript im Userinterface kann verwendet werden, sollte sich aber auf die nötigsten Funktionen zur Vereinfachung der Bedienung beschränken (um eventuelle Kompatibliltätsprobleme zu verringern).
NF	Antwortzeiten	Die Eingabe des Studenten zur Lösung einer Aufgabe wird immer „sofort“ bewertet und das entsprechende Feedback wird „ohne Verzögerung“ angezeigt. Das heisst, das Userinterface ist auf Interaktivität auszulegen und die Antwortzeiten sind gering zu halten.
NF	Oracle	Das System wird auf einem Oracle Application Server zum Einsatz kommen und Oracle Datenbanken werden, sowohl für das eigene Datenmodell als auch für die Übungsdatenbanken, verwendet.
NF	Java/J2EE	Das System ist in Java aufbauend auf das J2EE Environment zu implementiere.
NF	Öffentlich	Das Übungsmodul wird im Internet öffentlich zugängig sein. Die Übungen können von den Studenten von zuhause abgearbeitet werden.
IF	Accounts	Die Studentenaccounts werden direkt aus dem LVA System gelesen und nicht in einer eigenen Datenbank verwaltet.
IF	Ergebnisse	Die Prüfungsergebnisse werden geblockt in das LVA System zurückgespielt. Dafür wird eine eigene Aktion im Verwaltungsmodul vorgesehen.
IF	Anmeldung	Prüfungsanmeldungen werden im LVA System verwaltet und beim Login zu einer Prüfung überprüft.
UI	Sprache	Die Sprache des Systems ist ausschließlich Deutsch. Auf Mehrsprachigkeit muss keine Rücksicht genommen werden.
UI	Feedback	Dem Benutzer muss umfangreiches Feedback über die Richtigkeit oder die Fehler in seiner Eingabe geboten werden. Dabei wird - zusätzlich zu syntaktischen - auch auf inhaltliche (qualitative) Fehler in der Eingabe eingegangen.
UI	Logindaten	Der eingeloggte User muss deutlich sichtbar am Bildschirm erkennbar sein. Das soll eine Ausweiskontrolle bei der Prüfung unterstützen.
UI	Material	Aus dem System soll eine Liste von Zusatzmaterial erreichbar sein, wie z.B. das Lehrveranstaltungs-Skriptum oder eine SQL Referenz.
UI	Kursüberblick	Es soll dem Benutzer möglichst leicht gemacht werden, sich einen Überblick über die Kursdatenbank zu verschaffen (über das Relationenmodell und den Inhalt der Tabellen).

Fortsetzung auf der folgenden Seite...

Tabelle 2.1 – Fortsetzung

Kat.	Schlagwort	Beschreibung
UI	Übungsprüfung	Auch im Übungsmodul soll es möglich sein, einen Kurs im „Prüfungsmodus“ zu starten, damit man sich ein Bild über den Ablauf einer Prüfung machen kann, und sieht, worin die Unterschiede zur Übung bestehen.
UI	RS.Vergleich	Die Ergebnisse der Eingabe des Benutzers werden zur Bewertung mit einem Referenzergebnis verglichen. Die Differenzen dieses Vergleichs sollen für den Benutzer verständlich dargestellt werden (*Differenztabelle*).
UI	Angepasst	Die Oberfläche des Systems soll in der Struktur und farblich möglichst gut an das LVA System angepasst sein.

2.2 Architektur

In Abbildung 2.2 wird ein Überblick über den Aufbau der Applikation und der Systemumgebung dargestellt. Es ist zu sehen, dass die Applikation als J2EE Applikation auf einem Oracle Application Server umgesetzt wird. Es werden keine EJB Komponenten oder ähnliche Peristenzmechanismen verwendet, sondern es wird mit einer eigenen Datenbankschicht, die direkt auf JDBC aufsetzt, auf die einzelnen Datenbanken zugegriffen. Diese Entscheidung wurde vor allem deshalb getroffen, weil für die Abfragen auf der Übungsdatenbank und für den Vergleich von Resultsets ohnehin mit JDBC gearbeitet wurde. Die benötigten persistenten Datenzugriffsklassen sind nur relativ wenig Zusatzaufwand in der Implementierung.

Dadurch ergbit sich auch die Tatsache, dass obwohl das J2EE Environment des Oracle Application Servers zur Verfügung steht und im Grunde nur auf die APIs des Servlet Containers zurückgegriffen wird. Es kann damit die Applikation auch auf einem anderen Servlet Container wie z.B. Tomcat oder Jetty[1] zum Einsatz kommen.

Für die Umsetzung des HTML Userinterfaces wird auf STRUTS 2[2] und JSP gesetzt. Es wurde in den JSP Files weitgehend mit der STRUTS Userinterface Taglibrary gearbeitet sowie mit den STRUTS Control Tags. Es wurde davon Abstand genommen, direkt Java Code in den JSPs zu verwenden.

Das System wird in drei Module aufgeteilt, die unabhängig voneinander eingesetzt werden können, allerdings auf dieselben Java Packages aufbauen und auf dieselbe Datenbank zugreifen:

- *Übungsmodul:* Das Übungsmodul ist das via Internet öffentlich erreichbare Modul, das von den Studenten verwendet wird, um interaktiv SQL zu lernen und die Übungsbeispiele zu lösen. Die Studenten können im aktuellen Kurs Beispiele abgeben, die dann im

[1] Tomcat (http://tomcat.apache.org/) und Jetty (http://www.mortbay.org/jetty-6/) sind Opensource J2EE Servlet Container. Der Prototyp des e-Learning Systems wurde auf Tomcat entwickelt.

[2] STRUTS 2 ist ein Framework zur Unterstützung bei der Entwicklung von Webanwendungen. Siehe: http://struts.apache.org/

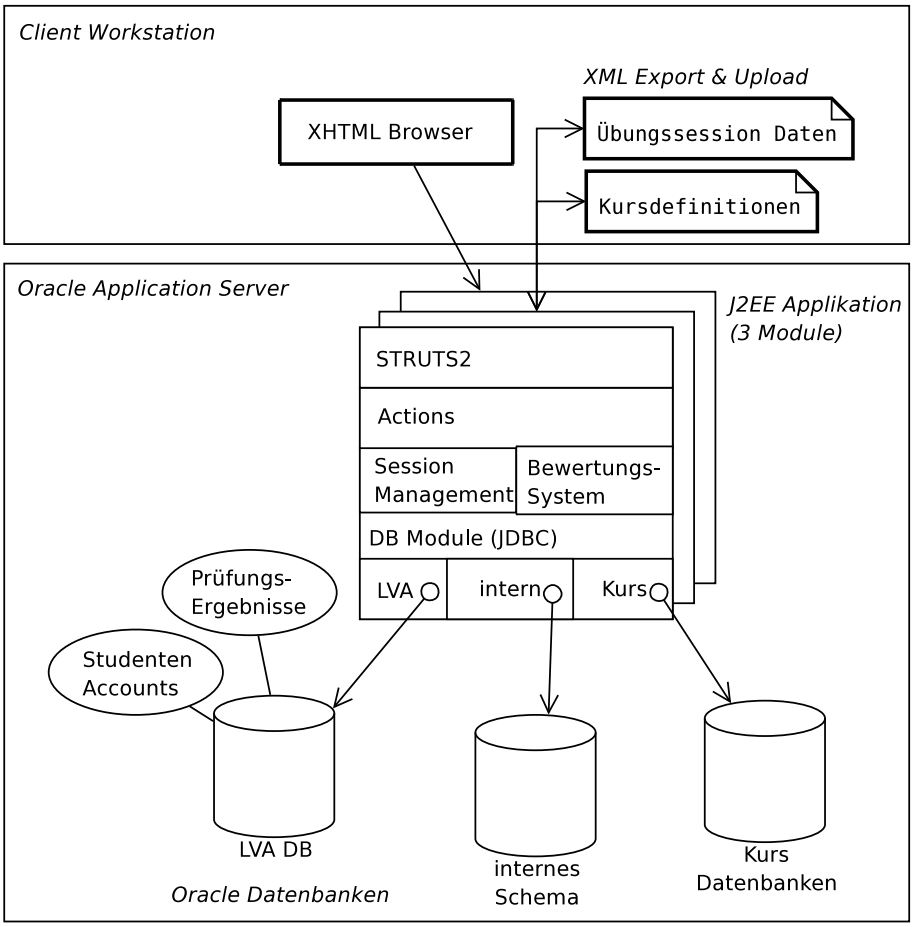


Abbildung 2.2: Architekturübersicht des e-Learning Systems

Tutorengespräch, das ebenfalls mit dem Übungsmodul abgewickelt wird, bewertet werden.

- *Prüfungsmodul:* Das Prüfungsmodul ist nicht öffentlich, sondern nur über ein eingeschränktes Netz erreichbar. Dieses Modul wird im Rahmen der beaufsichtigten Prüfung verwendet. Die Studenten haben ein vorgegebenes Zeitlimit um Aufgaben zu lösen, die am Ende der Prüfung abgegeben und bewertet werden. Prüfungen können nur in diesem Modul abgegeben werden.

- *Verwaltungsmodul:* Dieses Modul ist ebenfalls nur eingeschränkt erreichbar und kann nur von Administratoren (Lehrveranstaltungsleiter) benutzt werden. Es stellt die Funktionalität zur Verfügung, mit der die Prüfungsbewertungen geändert und die Prüfungsergebnisse ins LVA System übernommen werden können. Weiters bietet es auch die Möglichkeit, Kurse und dessen Aufgaben zu definieren.

Die Trennung der Module wurde vor allem deshalb vorgenommen, um die Sicherheit der Prüfungsdaten zu gewährleisten. Selbst wenn im öffentlichen Übungsmodul irgendwelche Sicherheitsprobleme auftreten sollten, sind damit noch nicht das Prüfungsmodul und die damit verbundenen Daten gefährdet.

Es gibt zwei unterschiedliche Bereiche im System, wo XML Files exportiert und auch upgeloadetet werden können: Die Definition von Kursen und Aufgaben (im Verwaltungsmodul) und die Session eines Studenten im Übungsmodul (das sind im Grunde die eingegebenen Lösungen). In beiden Fällen kann ein XML File durch eine Aktion im Userinterface über den Browser exportiert oder importiert werden. Dieser Ansatz verringert im ersten Fall den Aufwand in der Umsetzung der Kursdefinitionen und gestaltet gleichzeitig die Definition von Kursen effizienter, indem auf umständliche Webformulare verzichtet wurde. Im zweiten Fall wird damit die Notwendigkeit von persistenten Usersessions umgangen. Die Studenten müssen den aktuellen Stand exportieren und für sich selbst sichern, falls sie zu einem späteren Zeitpunkt weiterarbeiten wollen.

2.3 Applikationsstruktur

Die Applikation selbst ist intern in Java Packages strukturiert. Alle Packages werden in allen 3 Modulen verwendet. Die Unterscheidung der Funktionalität pro Modul wird dabei vor allem in den höherliegenden Schichten getroffen (Userinterface, Session und User Authentifizierungs Packages).

In Abbildung 2.3 wird ein Überblick über die Packagestruktur und die Abhängigkeiten des Systems dargestellt.

Am unteren Ende befinden sich einerseits die Datenbankzugriffs Packages und andererseits allgemein verwendete Utilitites. In der Mitte ist das *system* Package angesiedelt, das als Abstraktionsschicht (*Facade*) für alle darüberliegenden Packages dient. Alle Packages von Datenbank bis zum *system* Package werden in allen Modulen gleich verwendet. Die Packages darüber (bzw. die Verwendung der Funktionen in diesen) unterscheiden sich von Modul zu Modul.

Im Folgenden soll zu jedem Package kurz dessen Aufgabe beschrieben werden:

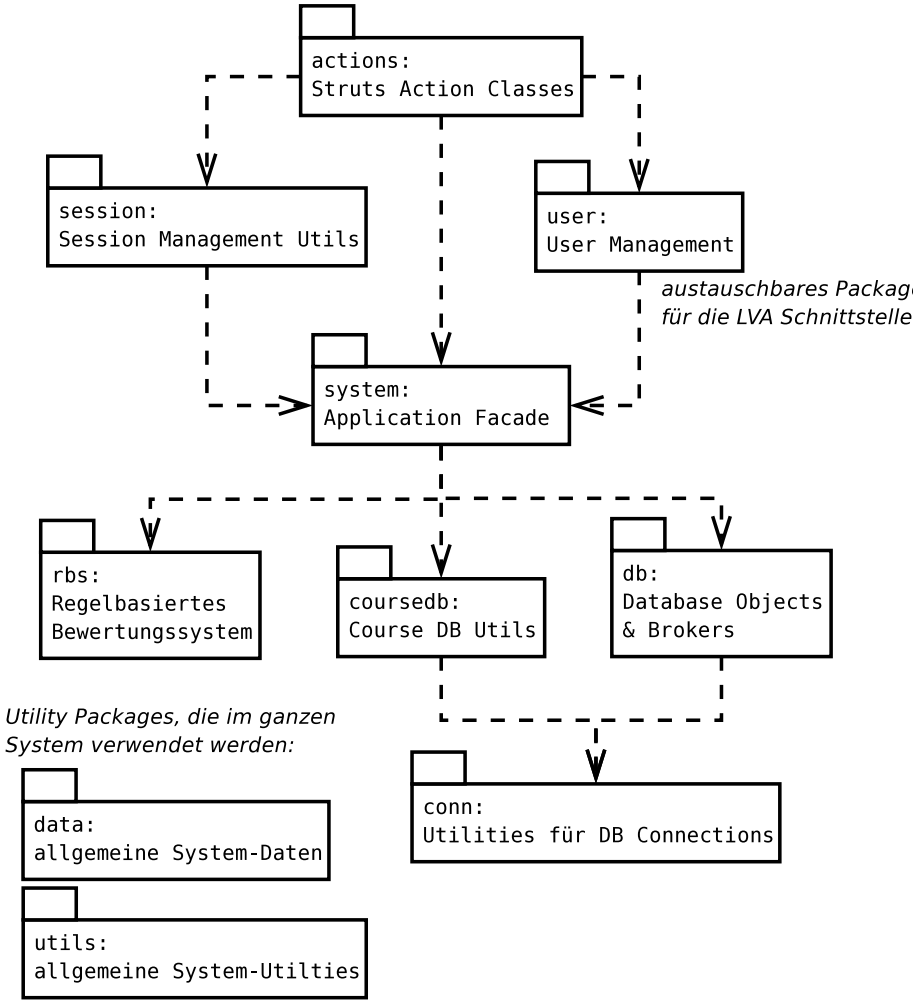


Abbildung 2.3: Interne Struktur der Java Packages des e-Learning Systems

- Das *utils* Package ist eine Sammlung von mehrfach verwendeten Utility Klassen. Im Grunde wird dieses Package von überall verwendet, weshalb auch die Dependencies nicht explizit eingezeichnet sind.

- Das *data* Package wird ähnlich verwendet, wie *utils*. Hier werden systemweite Datenstrukturen gesammelt. Das *data* Package unterscheidet sich aber insofern von *utils*, als dass die hier gesammelten Datenstrukturen nur systemspezifisch sind, währen die Klassen in *utils* auch unabhängig von diesem System verwendet werden könnten.

  Hier wird zum Beispiel das `Tabular` Interface zur Repräsentation von Tabellendaten, sowie eine Defaultimplementierung davon definiert. Außerdem ist hier die Sammelstelle für alle Datenobjektreferenzen, die als Identifier Objekte im gesamten System verwendet werden.

- Das *conn* Package beinhaltet in erster Linie die Implementierung eines Connection Pools. Dieser Connection Pool wird sowohl für die Datenobjekte im *db* Package, als auch für die Tabellendaten (Abfrageergebnisse) im *coursedb* Package verwendet.

- Im *db* Package wird die eigene Datenzugriffsschicht implementiert. Es wurde entschieden, keine Object/Relational Mapper oder andere Objektpersistenz-Mechanismen zu verwenden. In diesem Package ist die entsprechende eigene Implementierung der Zugriffsmechanismen zu finden. Mehr Informationen zum Thema Datenzugriff sind in Abschnitt 2.3.1 nachzulesen.

- Im *coursedb* Package werden die Methoden zum Zugriff auf die Übungsdatenbanken implementiert. Hier findet sich vor allem die Klasse `QueryExecutor`, die eine Query als Input bekommt, und ein befülltes *Tabular* Interface, oder entsprechende Fehlermeldungen, als Ergebnis liefert. Das *Tabular* Interface wird dann z.B. dem Bewertungssystem weitergegeben um die Ergebnisse auszuwerten.

- Im Package *rbs* befindet sich alles, was zum Bewertungssystem der Applikation zählt. Hier finden sich Klassen, die zum Vergleich von Resultsets (hier konkret Implementierungen von *Tabular*) verwendet werden, oder Klassen für die Analyse der eingegebenen Statements (`Extractor`). Vorallem aber ist hier das Regelsystem für die Bewertung implementiert.

  Die Themen des Resultsetvergleichs und des Bewertungssystems allgemein werden in den Kapiteln 6 und 8 abgehandelt. Dort werden zu einzelnen Aspekten des Bewertungssystems auch detaillierte Klassendiagramme vorgestellt.

- Das Package *system* fungiert als *Facade* für die darunterliegenden Packages. D.h. es soll hier die Komplexität der Packages *db, coursedb* und *rbs* einerseits koordiniert, andererseits von den darüberliegenden Packages versteckt werden. Die koordinierende Aufgabe besteht darin, die Daten aus den *db* und *coursedb* Packages an das *rbs* Package und die Ergebnisse aus dem Bewertungssystem nach oben weiterzureichen. Das *system* Package übernimmt damit die Koordinationsaufgaben, die nötig sind, um das Bewertungssystem und die Datenbankzugriffe voneinander unabhängig zu halten.

- Das *user* Package ist eine Sammlung von Klassen, die zum Zugriff auf die Studentenaccounts und die Prüfungsanmeldungen verwendet werden. Im Rahmen der Prototyp Implementierung wurde dieses Package mit eigenen Datenbanktabellen zum Test umgesetzt. Im Zuge der Migration auf das Oracle Zielsystem soll dieses Package mit direktem Zugriff auf die LVA Datenbank implementiert werden.

- Im Package *session* werden Utilities für die Verwaltung der Usersessions gesammelt. Erwähnenswert ist hier die Klasse `SessionData`, die sämtliche Daten einer Usersession beinhaltet. Es wurde davon Abstand genommen, unkontrolliert eine Menge von Objekten in der `HttpSession` zu speichern, weil das üblicherweise dazu führt, dass man schnell den Überblick verliert und damit auch die Kontrolle über den Speicherverbrauch durch Sessiondaten.

- Schließlich werden im Package *actions* Die STRUTS Action Klassen gesammelt. In diesen Klassen ist im Wesentlichen die Funktionalität des Userinterfaces implementiert. Jede JSP Page im Userinterface hat eine Action Klasse als Grundlage.

2.3.1 Datenzugriff

Wie schon gesagt, wurde für den Datenzugriff auf die internen Strukturen direkt auf JDBC aufgesetzt und keine weiteren Hilfsmittel, wie EJB oder Objekt/Relational Mapper verwendet. Abgesehen von der Überlegung, dass es sich um relativ einfache Strukturen handelt, und dass ohnehin beim Zugriff auf die Übungsdatenbanken (im Package *coursedb*) direkt mit JDBC gearbeitet werden muss, spielen dazu noch folgende Überlegungen eine Rolle:

- Im Gegensatz zu Object/Relational Mapping Tools hat man hier eine bessere Kontrolle über die Granularität beim Buffering. D.h. man hat die Möglichkeit, selbst zu definieren, auf welcher Ebene die Objekte im Speicher gehalten werden, anstatt die gemappten Objekte einzeln im Speicher zu halten. So wurde Beispielsweise für Kursdefinitionen, die zusammen mit den definierten Aufgaben und Referenzlösungen in mehreren Tabellen in der Datenbank gespeichert werden, der Ansatz gewählt, eine Kursdefinition mit allen Aufgaben immer gesamt aus der Datenbank zu lesen, im Speicher zu halten und erst dann wieder zu lesen, wenn sich etwas in der Datenbank geändert hat.

- Zusätzlich ist es in manchen Bereichen notwendig, höherliegende Strukturen im Speicher zu halten, wo das Buffering auf der Datenzugriffs-Schicht unnötig wird. Beispielsweise können für die Aufgaben die Ergebnisse der Referenzabfrage gespeichert und darauf aufbauende Analyseinformationen gespeichert werden. Das lässt sich am einfachsten mit einem *BusinessObject* implementieren, das auch flexible Zugriffsmöglichkeiten zur Datenbank hat.

- Nicht zuletzt ist der Ansatz bei der Verwendung von ORM Tools meist derjenige, zuerst die Java Klassen und Relationen zu definieren und diese Definitionen auf ein Datenbankschema zu mappen. Dieser Ansatz wurde hier nicht gewählt. Das Design des Systems ist von einem Datenmodell als Grundlage ausgegangen.

Die gewählte Vorgehensweise bei der Implementierung der Datenzugriffs-Schicht entspricht in Annäherung dem *Data Access Object* (DAO) Pattern aus den *Core J2EE Patterns* (siehe

[J2EEPat01])[3], wobei die *Data Access Objects* hier als *Broker* bezeichnet werden (beispielsweise werden `CourseDef` Objekte vom `CourseDefBroker` verwaltet).

Der Unterschied zum DAO Pattern besteht allerdings darin, dass *DataTransferObject* und *BusinessObject* nicht getrennt werden. Die Objekte, die von einem *Broker* verwaltet werden, implementieren direkt die benötigte zusätzliche Business Logic.

Das wurde so entschieden, weil auf dieser Ebene kaum zusätzliche Business Logic zu implementieren ist. Die meiste zusätzliche Funktionalität wird im Bewertungssystem untergebracht, das aber ohnehin unabhängig implementiert ist.

2.3.2 Berechtigungskonzept

Die Zugriffsberechtigung eines Users für eine bestimmte Aktion oder Seite hängt einerseits von der Rolle des Users und andererseits vom Modul in dem man sich befindet ab. In allen Modulen werden dieselben User Rollen definiert: `STUDENT`, `TUTOR` und `ADMIN`. Je nach Modul sind für diese drei Rollen unterschiedliche Aktionen erlaubt oder nicht.

Die Verwaltung der Zugriffsrechte wird im Package *user* vorgenommen, wo eine zentrale Klasse (`AccessManager`) befragt werden kann, ob eine bestimmte Aktion erlaubt ist oder nicht. Das Überprüfen der Berechtigung selbst muss in den einzelnen STRUTS Action Klassen gemacht werden. Um das nicht in jeder Action Klasse neu implementieren zu müssen, wird in diesem Bereich mit Vererbungshierarchien gearbeitet. Die Überprüfung der Zugriffsrechte wird dann in einer allgemeinen abstrakten Klasse durchgeführt, von der alle anderen Action Klassen abgeleitet sind.

[3] online nachzulesen unter http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

3 Datenmodell

In den folgenden beiden Abschnitten wird das Datenmodell des e-Learning Systems beschrieben. Im Abschnitt 3.1 wird kurz das ER-Modell präsentiert, das die Struktur auf abstrakter Ebene beschreibt, in Abschnitt 3.2 wird im Detail erläutert, wie die Struktur der Tabellen im Datenbanksystem realisiert wurde.

Die Ausführungen konzentrieren sich mehr auf die physische Umsetzung und die zugrundeliegenden Überlegungen. Das logische Modell wird in erster Linie zur Übersicht angeführt.

3.1 Logisches ER-Modell

Das logische ER Modell wird aufgeteilt in zwei Sichten: die eine Sicht beschreibt die Kursdefinitionen mit den Beispielen und Lösungen der Studenten (dargestellt in Abbildung 3.1), die andere Sicht beschreibt die Historisierung der Prüfungen mit den Angaben, Lösungen und ebenfalls gesicherten Regeln aus dem Bewertungssystem und der Bewertung selbst (dargestellt in Abbildung 3.2).

Mit dem Mechanismus der Historisierung wird sichergestellt, dass die Ergebnisse von vergangenen Prüfungen in Zukunft immer nachvollzogen werden können. Eine Prüfung mit allen Ergebnissen und den zugrundeliegenden Regeldaten wird in den Bereich der historisierten Daten (`ExamExercise, ExamSolution, ...`) gespeichert (d.h. konkret die Daten werden kopiert), sobald die vollständige Beurteilung einer Prüfung für den „Aushang“ bereitsteht.

Nachdem die Prüfungsergebnisse festgelegt wurden (und damit die Daten in den Historisierungstabellen gespeichert sind), kann nur noch der Punktestand eines Ergebnisses (z.B. im Rahmen einer Einsichtnahme) korrigiert werden. Solange das Ergebnis noch nicht festgelegt wurde, besteht die Möglichkeit, die Referenzstatements der einzelnen Beispiele zu erweitern oder überhaupt an den Regeln im Bewertungssystem Änderungen vorzunehmen.

Weitere Erläuterungen zum Historisierungskonzept sind auch in der Beschreibung der der Physischen Umsetzung (Abschnitt 3.2) nachzulesen.

Die beiden hier dargestellten Sichten (Abbildung 3.1 und 3.2) sind aus Gründen der Übersicht getrennt. Es handelt sich aber um konsolidierte Sichten, wobei sich die grau dargestellten Entities in Abbildung 3.2 auf Entities beziehen, die in der anderen Abbildung bereits definiert wurden.

3.2 Physische Umsetzung

3.2.1 Übersicht

Abbildung 3.3 zeigt einen Überblick der physischen Umsetzung des vollständigen Datenbankschemas. Für diese Darstellung wurde eine UML Notation gewählt, wobei eigene Konventionen verwendet wurden, um die nötige Funktionalität für ein Datenmodell abzubilden.

Die Notation wird im Folgenden kurz erläutert.

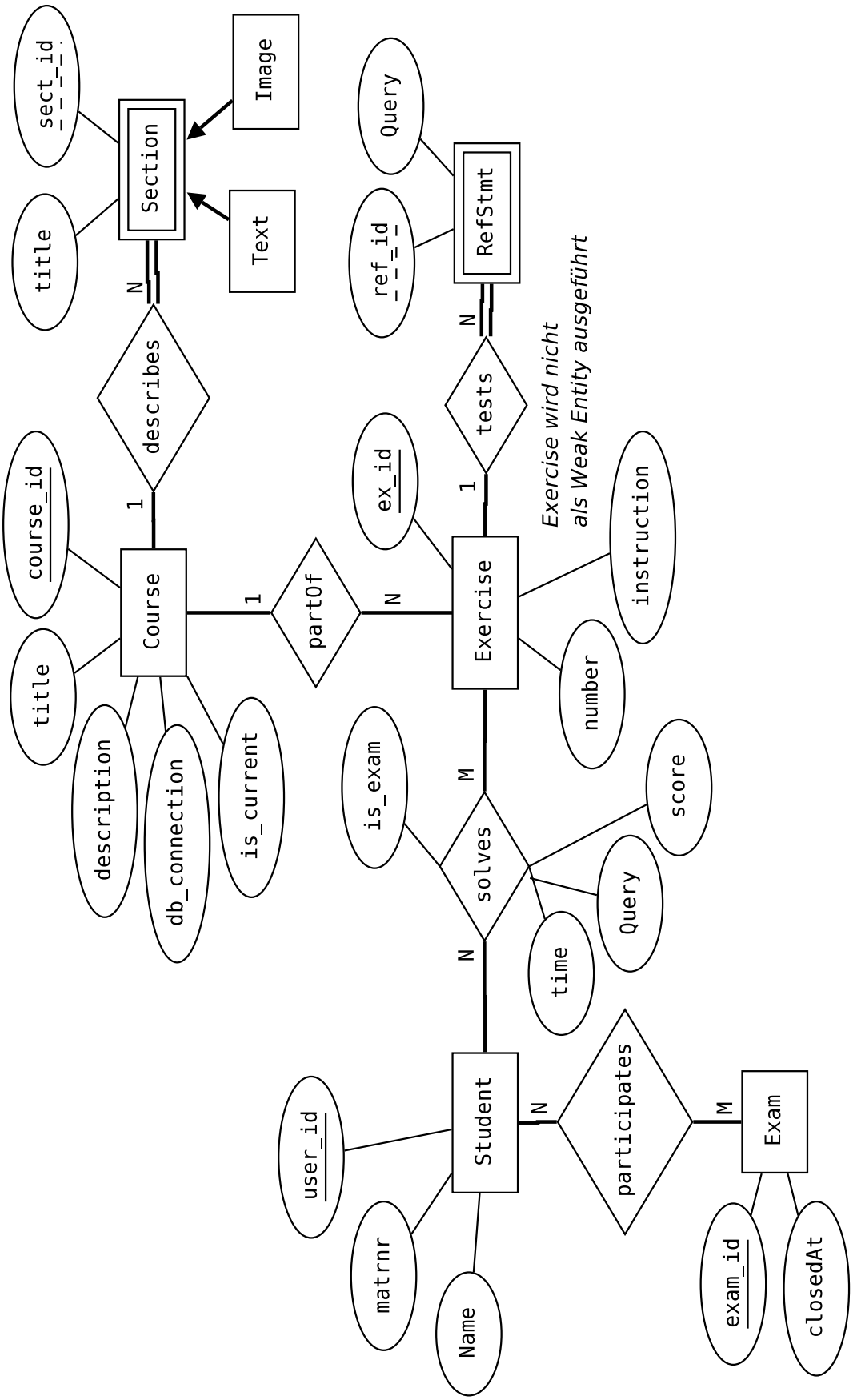


Abbildung 3.1: ER Diagramm: allgemeine Sicht

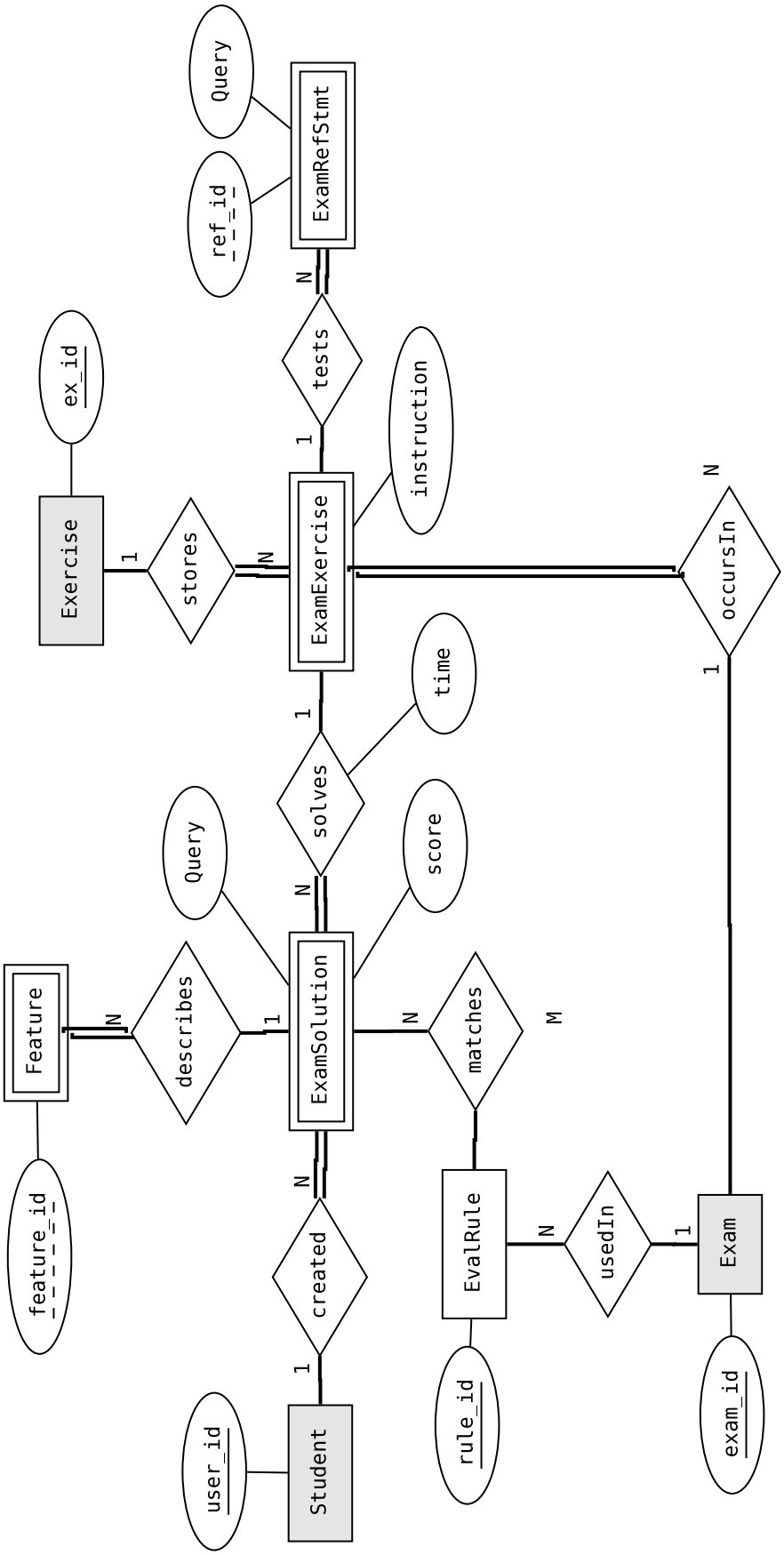


Abbildung 3.2: ER Diagramm: historisierte Sicht

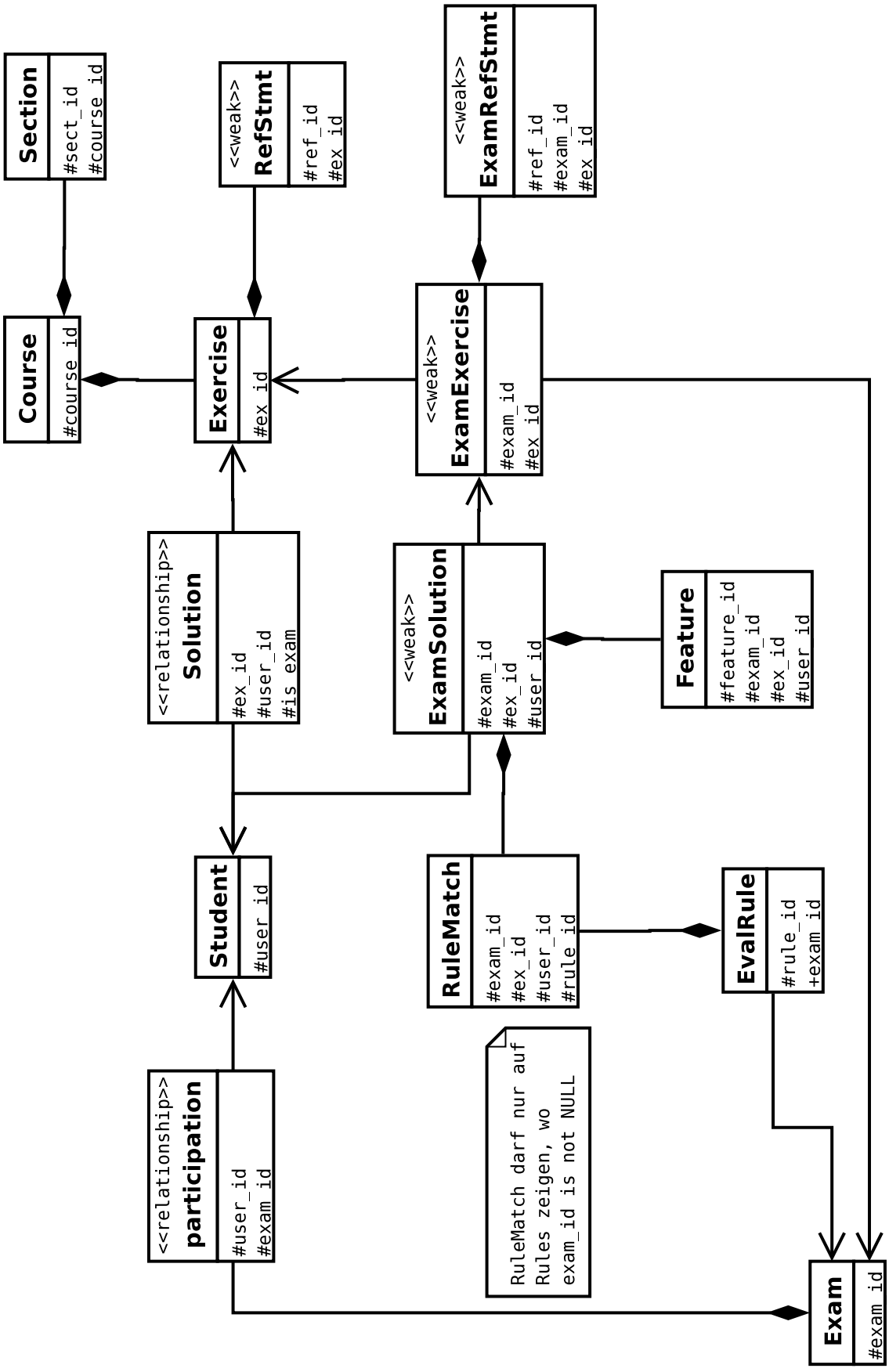
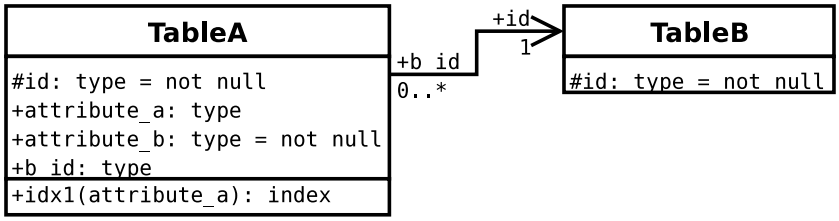


Abbildung 3.3: Übersichtsdiagramm: Physisches DB Schema

Notation

In folgender Abbildung wird eine einfache Beziehung zwischen den beiden Tabellen `TableA` und `TableB` dargestellt:



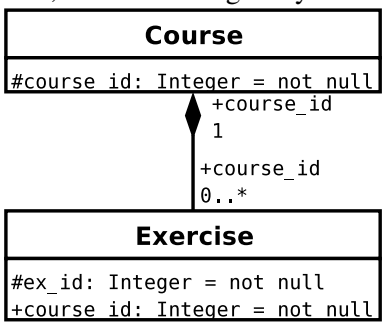
Die Stelligkeit der Beziehung wird zur UML Assoziation (Verbindungslinie) auf der Seite der Tabelle nach den UML Konventionen geschrieben: d.h. auf der Seite der jeweiligen Tabelle steht, wieviele Einträge dieser Tabelle in einer Beziehung vorkommen können. Die Rollen der Assoziation werden für die Bezeichnungen der Attribute (Foreign Keys und Primary Keys) in der Beziehung verwendet. Die Foreign Keys müssen in der Tabelle auch explizit angeführt werden.

Im vorliegenden Fall handelt es sich also um eine N:1 Beziehung zwischen `TableA` und `TableB`. `TableA` enthält einen Foreign Key (`b_id`), der auf Tabelle `TableB` referenziert. Der Pfeil in der Assoziation ist zur einfacheren Lesbarkeit eingezeichnet.

Die Attribute werden als UML Attribute mit `not null` als Wert dargestellt, wenn es sich um Pflichtattribute handelt. Protected Attributes (mit „#“ vor dem Namen) identifizieren den Primary Key der Tabelle.

Indizes auf einer Tabelle werden als UML Methoden dargestellt. Dabei bildet die Argumentliste die Liste der Attribute im Index.

Wenn eine Composition als Beziehung zwischen zwei Tabellen verwendet wird, dann bedeutet das, dass die Foreign Key Referenz mit einem `on delete cascade` versehen wird:



Im Übersichtsdiagramm (Abbildung 3.3) wird auf die Details verzichtet: es werden nur die Primary Keys dargestellt. In den Beziehungen werden keine Stelligkeiten (diese ergeben sich aus den Pfeilen) oder Attribute eingezeichnet.

Beschreibung

Die Darstellung der Übersicht in Abbildung 3.3 zeigt alle Tabellen mit den Primary Keys aber ohne Typen. Die Typen für die Primary Keys sind durchgehend Integer. Für die Tabellen wurden durchgehend englische Bezeichnungen gewählt.

Die Tabellen `Course, Section, Exercise` und `RefStmt` im rechten oberen Bereich der Übersicht beschreiben die Daten eines Kurses. Mit `Course` wird ein Kurs definiert

(mit Verbindungsparametern zur Übungsdatenbank, Titel und Beschreibung). `Section` beinhaltet Abschnitte für die Beschreibung des Relationenmodells der Übungsdatenbank bzw. Bilder als BLOBs. In `Exercise` und `RefStmt` sind die Übungsbeispiele mit den Referenz Queries zur Bewertung definiert.

In der Tabelle `Solution` werden die Lösungen der Studenten abgelegt. Lösungen kommen entweder aus der Übung oder aus einer Prüfung. Übungsdaten werden so behandelt, dass die Lösungen in der Tabelle `Solution` abgelegt werden und dort bleiben. Die endgültige Übungsbewertung findet im Tutorengespräch mit dem Studenten statt und wird im System nicht gespeichert.

Die Handhabung von Prüfungen ist etwas komplizierter. Die Daten von vergangenen Prüfungen sollen möglichst nachvollziehbar und permanent abgelegt werden. Das erfordert einen Historisierungsmechanismus im Zusammenhang mit den einzelnen Prüfungen.

Folgende Vorgehensweise ist in der gewählten Umsetzung abgebildet:

1. Prüfungen können jederzeit abgelegt werden. Im Datenmodell wird auch kein Eintrag für eine Prüfung im Voraus gemacht. Bei einer Prüfung wird ein Kurs zufällig ausgewählt und zu den ebenfalls zufällig gewählten Beispielen dieses Kurses werden die Lösungen des Prüfungsteilnehmers gespeichert. Das geschieht bei der Abgabe der Prüfung durch den Teilnehmer. Es werden die Einträge in der Tabelle `Solution` gemacht. Durch diese Einträge wird festgelegt, dass der Student zu einer Prüfung angetreten ist sowie, welche Beispiele er zu lösen hatte.

2. Nach den Abgaben der einzelnen Prüfungen wird mit einer Bearbeitungsphase der Prüfungen gerechnet. Dabei können die Regeln des Regelsystems angepasst oder die Referenz Queries der Beispiele erweitert oder Lösungen händisch mit anderer Punktezahl gewertet werden. Diese Anpassungen werden mit den „aktuellen“ Daten in den Tabellen `Solution, RefStmt` und `EvalRule` gemacht.

3. Nach Abschluss der Bearbeitungsphase (d.h. wenn die Benotung feststeht und „ausgehängt“ wird) wird die Prüfung erstmals in der Tabelle `Exam` gespeichert. Gleichzeitig werden alle relevanten Daten historisiert. Konkret bedeutet das: Die Beispielangaben werden in den Tabellen `ExamExercise` und `ExamRefStmt` und die Lösungen in der Tabelle `ExamSolution` (gemeinsam mit den `Features` aus dem Bewertungssystem und den zutreffenden Regeln [`RuleMatch`]) gesichert. Alle zu dem Zeitpunkt gültigen Regeln werden auch in die Tabelle `EvalRule` mit einer zusätzlichen Referenz auf `Exam` kopiert. Aktuell gültige Regeln haben keine solche Referenz in derselben Tabelle.

Dadurch ist im Grunde auch schon die gesamte Funktionalität festgelegt. Zusätzlich wird noch in der Tabelle `participation` redundant gespeichert, welcher Student an welcher Prüfung teilgenommen hat.

3.2.2 Detailaspekte

In den folgenden Abschnitten wird auf einzelne Details in der Umsetzung eingegangen um die Überlegungen dahinter zu erläutern und die getroffenen Entscheidungen einsichtiger zu machen.

```
<<postgres:typemap>>
Title:        varchar(140);
Text:         text;
ConfigString: varchar(200);
BLOB:         bytea;
Byte:         smallint;
Integer:      int;
Small:        smallint;
Time:         timestamp;
TypeCode:     char(3);
Username:     varchar(20);
Name:         varchar(100);
```

Abbildung 3.4: Type Map

Abstrakte Typen

Im Sinne der einfacheren Portierbarkeit werden keine physischen Datentypen, sondern eigens definierte Typen, die soweit möglich die logische Funktion eines Attributs beschreiben (z.B. `Title` oder `Username`) verwendet. Die Umsetzung für ein bestimmtes Datenbank System wird durch eine `TypeMap` festgelegt. In Abbildung 3.4 wird die `TypeMap` für `PostgreSQL` dargestellt.

Vereinfachungen von Primary Keys

Als Primary Keys werden im gesamten Datenbankschema nur künstlich definierte Integer Keys verwendet. In Tabellen wie `Course` oder `Solution` lässt sich das ohnehin nicht vermeiden, aber auch im Fall von `Student` wurde davon Abstand genommen, die Matrikelnummer als Primary Key zu verwenden. Im Fall von `Exam` wird z.B. auch nicht das Prüfungsdatum zur Identifikation herangezogen.

Hinter diesem Ansatz steckt die Überlegung der einfachen Erweiterbarkeit. Wäre eine Prüfung z.B. durch ein Prüfungsdatum identifiziert, dann könnte dieses im Moment vielleicht eindeutig sein. Aber wenn irgendwann die Notwendigkeit besteht, mehrere verschiedene Prüfungen am selben Datum abzuhalten, dann ist im vorliegenden Fall die Änderung einfacher (bzw. konkret wäre gar nichts zu tun). Im Sinne der Konsequenz wurde dieses Prinzip auch auf Studenten umgelegt, auch wenn es hier eher unwahrscheinlich ist, dass in absehbarer Zukunft die Matrikelnummer nicht eindeutig eine Person identifiziert.

Die Erhaltung der logischen Primary Keys (z.B. Matrikelnummer in `Student`) wird durch zusätzliche `unique indexes` sichergestellt.

Eine weitere Vereinfachung stellt das Weglassen der `course_id` ab der Tabelle `Exam` dar. Im Grunde könnte `Exam` als schwache Entity zu `Course` umgesetzt und damit die `course_id` als Primary Key in `Exam` mitgenommen werden. Das hätte aber zur Folge, dass die `course_id` in sämtlichen Tabellen als Teil des Primary Keys auftaucht, worauf einfach nur aus Gründen der Einfachheit verzichtet wurde (`ex_id` in `Exam` ist also global eindeutig).

Diese Vereinfachung ermöglicht zwar das Entstehen von Inkonsistenzen (`Exam/Course` Zuordnungen die im Widerspruch zu entsprechenden `ExamExercise/Exercise/Course` Zuordnungen stehen. Diese Probleme müssen aber ohnehin auch in der Applikation abgefangen werden.

Vereinfachte Generalisierung

In einigen Bereichen des Relationenmodells könnte man die Struktur auch sauberer gestalten, indem man Generalisierungen verwendet, wo eine Tabelle die gemeinsamen Attribute beinhaltet und zwei Tabellen mit den Unterschieden auf diese allgemeine Tabelle referenzieren.

Diese Sichtweise wäre mit den Tabellen `Exercise` und `ExamExercise`, `Solution` und `ExamSolution` (also den Historisierungstabellen) sowie `Section` und `EvalRule` möglich. In `Section` sind Text und Bilddaten mit einem Typattribut in einer Tabelle vereint. In `EvalRule` werden sowohl die aktuellen Regeln, als auch die historisierten Regeln zu einer Prüfung gemeinsam abgelegt.

Die getroffenen Vereinfachungen verfolgen allesamt folgende Ziele: weniger Tabellen und einfachere Abfragen. Auch wenn z.B. beim Historisieren mehr Daten kopiert werden müssen, ergibt sich aus der Erfahrung dadurch auch generell weniger komplexe Handhabung in der Applikation.

Application Constraints

Aus den oben beschriebenen Vereinfachungen wird teilweise die Möglichkeit geschaffen, inkonsistente Daten in der Datenbank abzulegen. Soweit möglich wird versucht, diese Inkonsistenzen mit Constraints im Datenbanksystem abzufangen.

Wo das nicht mehr möglich ist, muss sichergestellt sein, dass die Applikation die Daten korrekt verwaltet. Da das aber auf der anderen Seite ohnehin eine Anforderung an die Applikation ist (selbst die besten Konsistenzbedingungen im Relationenmodell muss die Applikation kennen und einhalten), ergibt sich dadurch keine zusätzliche Komplexität.

3.2.3 Detailtabellen

In den Abbildungen 3.5, 3.6 und 3.7 wird das physische Datenmodell vollständig dargestellt. Die Aufteilung in drei Diagramme dient der besseren Übersicht. Die Unterteilung wurde nach den Bereichen *Kursdefinition, Lösungen der Beispiele* und *Auswertung der Beispiele* vorgenommen.

Die Tabellen mit dem UML Stereotype «`placeholder`» beziehen sich auf Tabellen, die in einem der anderen Diagramme definiert wurden.

Die dargestellten Diagramme werden verwendet, um daraus direkt die DDL Statements zum Erzeugen des Datenbankschemas abzuleiten. Die DDL Statements werden aus diesen Diagrammen, die mit `dia`[1] erstellt wurden, mit `tedia2sql`[2] generiert.

[1] Dia ist ein Opensoure Diagram Tool für Linux. Siehe: http://www.gnome.org/projects/dia/
[2] Tedia2sql generiert DDL Statements aus Dia Diagrammen: http://tedia2sql.tigris.org/

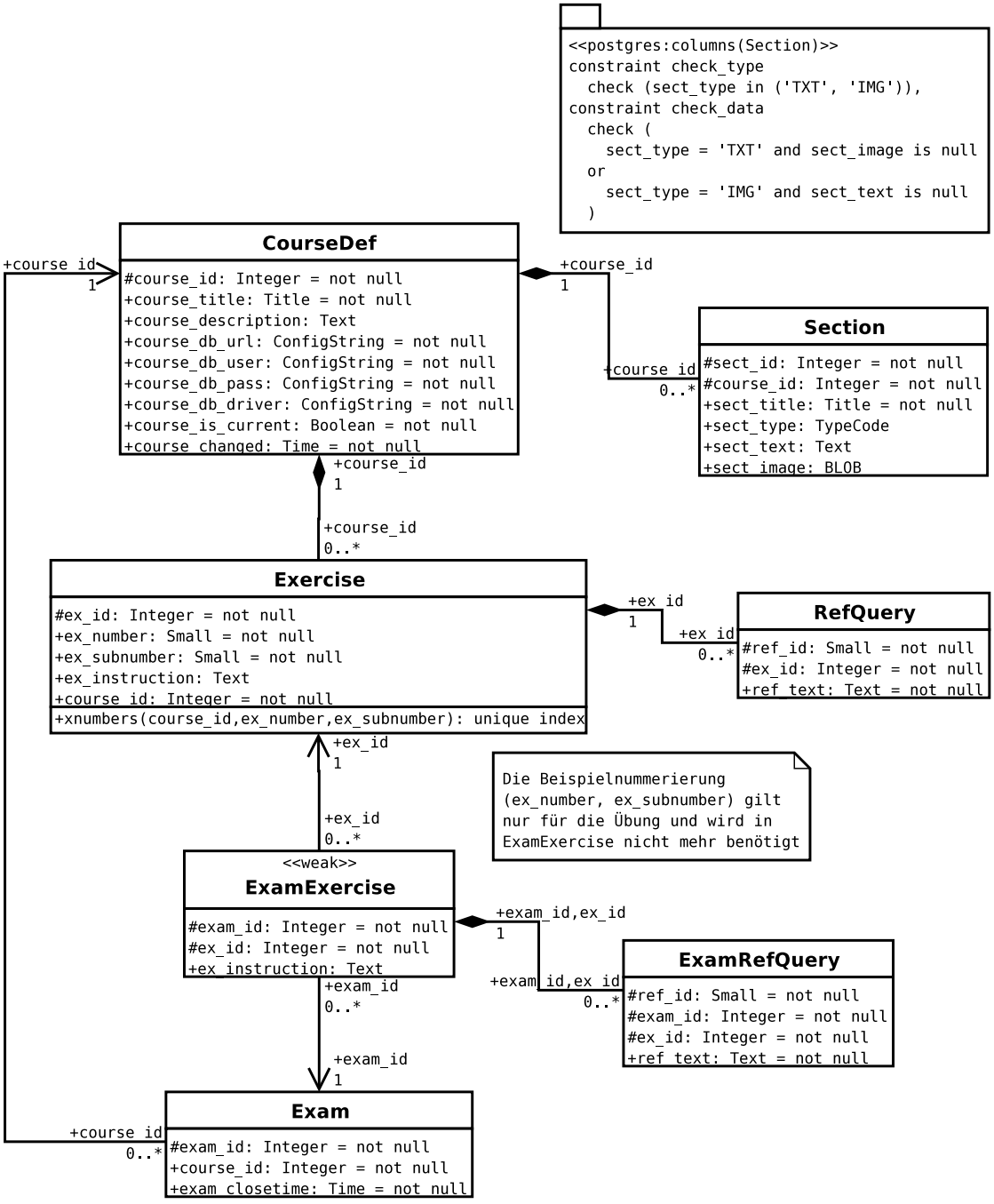


Abbildung 3.5: Physisches Datenmodell: Kursdefinition

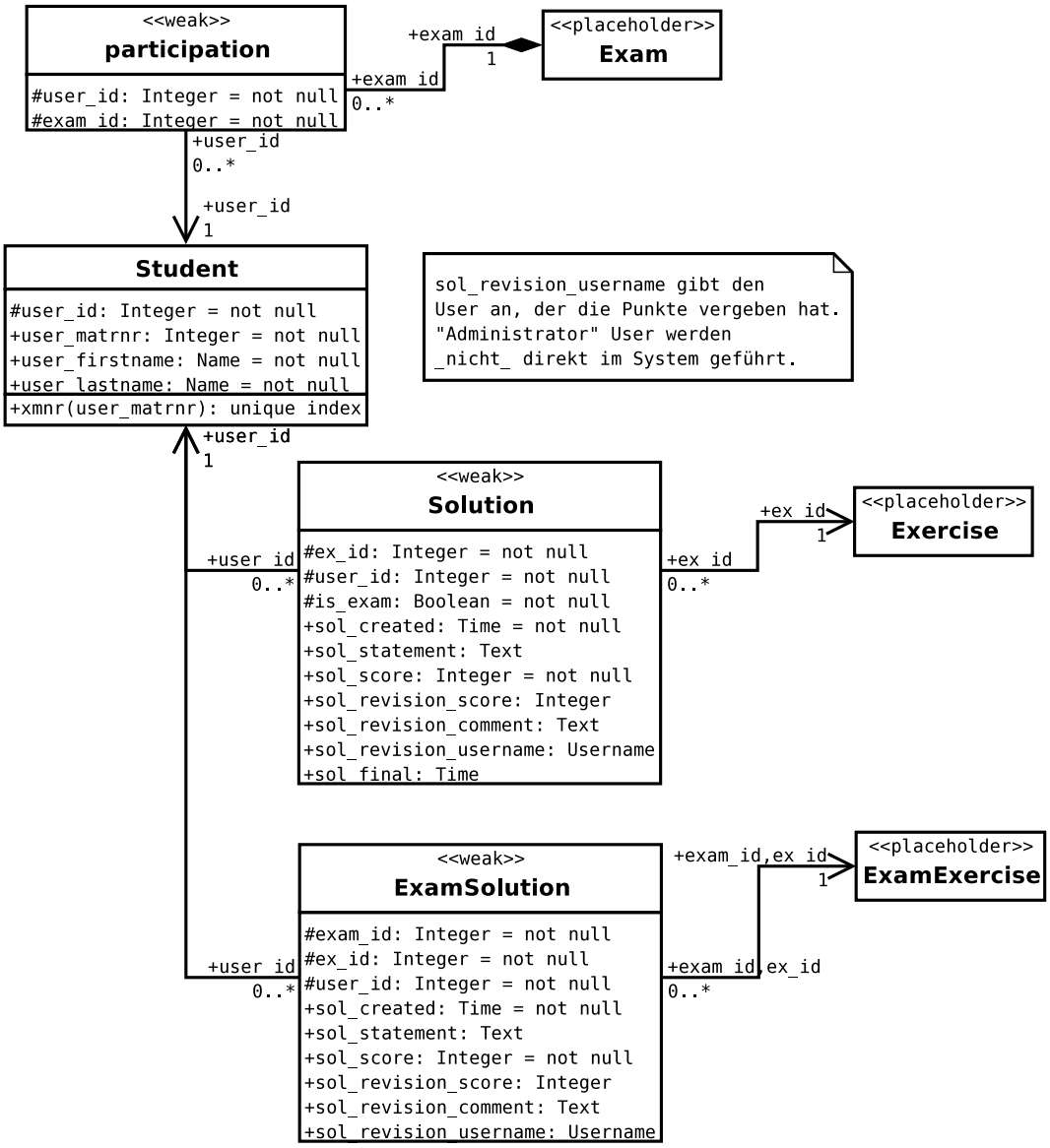


Abbildung 3.6: Physisches Datenmodell: Lösungen der Beispiele

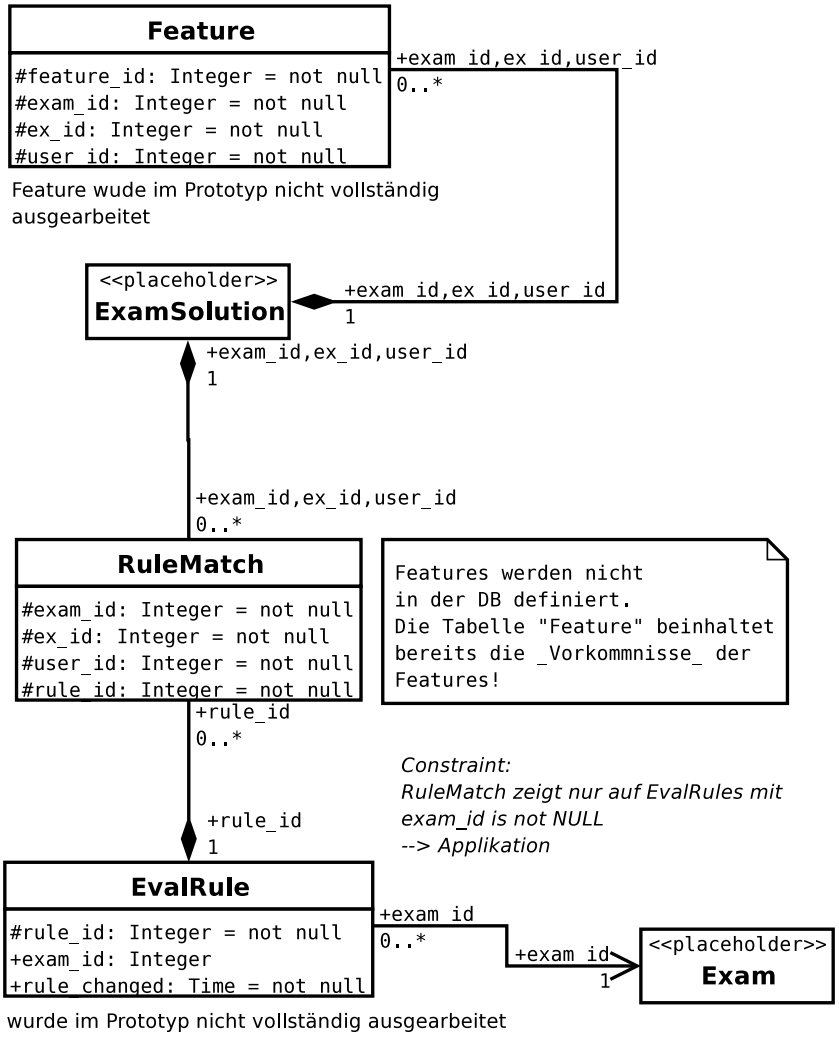


Abbildung 3.7: Physisches Datenmodell: Auswertung der Beispiele

4 e-Learning Überblick

Dieses Kapitel gibt einen kompakten Überblick über die heutigen Konzepte des e-Learning. Am Anfang wird auf die relevanten Lerntheorien eingegangen. Darauf folgt eine Einführung in e-Learning anhand der wesentlichen Begriffe. Die Lernobjektmodelle bilden eine Grundlage für die Wiederverwendbarkeit von einzelnen Lernobjekten. Es werden die unterschiedlichen Ansätze der KI in e-Learning betrachtet. Schließlich wird versucht, das hier beschriebene SQL e-Learning System in diesen Kontext einzuordnen.

4.1 Lerntheorien

Allen e-Learning Ansätzen zugrunde liegen lerntheoretische Konzepte die im Laufe des letzten Jahrhunderts entwickelt wurden. Die relevanten Theorien und deren Entwicklung sollen hier auch im Überblick erläutert werden. Ähnliche Übersichten sind vielmals in der Literatur angeführt, beispielsweise in [Baumgartner97].

- Den Anfang der Lerntheorien bilden Selbstbeobachtungsexperimente, die Anfang des 20. Jahrhunderts in Form von Gedächtnisexperimenten durchgeführt wurden. Aus diesen Experimente entstanden eine Reihe von Regeln und Gesetzen, wie beispielsweise die *Lernkurve* oder *Vergessenskurve*.

- Die Versuche von Ivan Petrovic Pavlov, aus denen die *klassische Konditionierung* hervorging, bildeten einen Ausgangspunkt für die Entwicklung des *Behaviorismus*. Die Theorie des Behaviorismus entstand aus Kritik an der Selbstbeobachtung und erforscht nun auch das Lernen von Verhaltensmustern im Gegensatz zur reinen Gedächtnisforschung. Als wichtigster Vertreter des Behaviorismus gilt B.F.Skinner.

  In Anlehnung an die Konditionierung wird im Behaviorismus ein Reiz-Reaktions Modell verwendet um den Lernvorgang zu beschreiben. An den im Gehirn ablaufenden spezifischen Prozessen ist der Behaviorismus nicht interessiert ([Baumgartner97]).

- Ganz im Gegensatz dazu stehen die kognitiven Lerntheorien (*Kognitivismus*), die seit den 1960er Jahren entstanden sind. In diesem Ansatz wird versucht, die inneren Prozesse im Gehirn zu erklären. Es werden theoretische Modelle für die Verarbeitungsprozesse beim Lernen aufgestellt. Der Prozess des menschlichen Denkens wird als eine Art der Informationsverarbeitung betrachtet.

  Die Art des Lernens im Kognitivismus ist das Problemlösen. Es geht darum, die passenden Methoden und Verfahren zu erlernen, die zur Problemlösung und damit zum Generieren der richtigen Antwort angewendet werden können ([Baumgartner97]). Im Behaviorismus hingegen versucht man allein mit den passenden Reizen die erwarteten Antworten zu „erzeugen“.

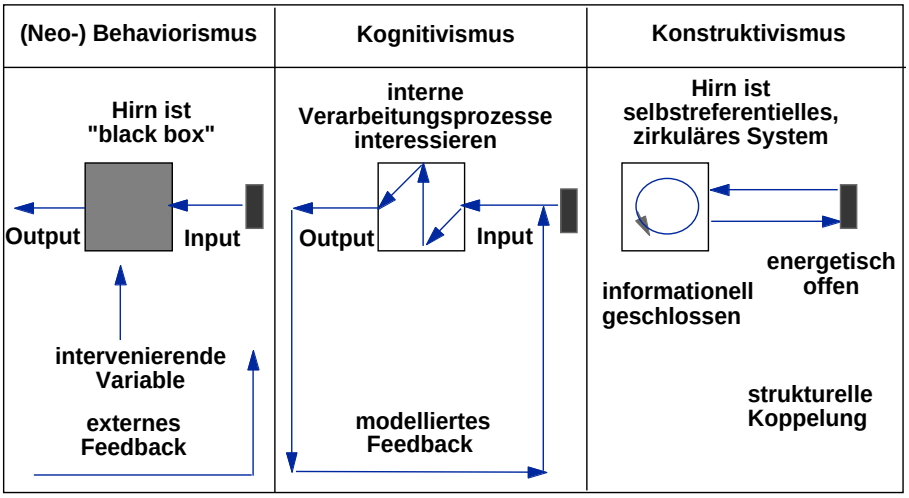


Abbildung 4.1: Drei Theorien des Lernens (schematisch) (aus [Baumgartner97])

- Sowohl Behaviorismus als auch Kognitivismus gehen davon aus, dass Probleme, deren Lösung zu erlernen ist, bereits vorgegeben sind. Im Ansatz des *Konstruktivismus* hingegen, soll der Lernende selbst Probleme generieren um anhand deren Lösung seine Erfahrung zu erweitern. Wissen wird eigenständig „konstruiert“ und mithilfe von Experimenten erweitert.

  Im Konstruktivismus wird nicht von einer gegebenen Außenwelt ausgegangen. Der Lernende erforscht seine Umgebung selbst und hat damit auch keine Instanz, die zu erlernende Prozesse vorgibt.

Die drei Lerntheorien *Behaviorismus, Kognitivismus* und *Konstruktivismus* und deren Konzepte werden in Abbildung 4.1 dargestellt.

Ausgehend von diesen drei Theorien, lassen sich drei unterschiedliche „Lehrmodelle“ definieren. In Abbildung 4.2 werden die wesentlichen Konzepte dieser drei Ansätze dargestellt. Der „Lehrende“ nimmt in den drei Modellen unterschiedliche Rollen ein:

- Im behavioristischen Ansatz fungiert der Lehrende als *Lehrer* im herkömmlichen Sinn. Er sorgt dafür, dass Wissen von ihm zum Lernenden transferiert wird.
- Im kognitivistischen Ansatz ist er *Tutor*. Der Lernende erarbeitet die Lösungen relativ eigenständig, aber der Tutor begleitet und unterstützt den Prozess.
- Im konstruktivistischen Modell schließlich nimmt der Lehrende die Rolle des *Coaches* ein. Der Coach hat nicht mehr den Anspruch, selbst die Lösungen zu wissen. Er unterstützt lediglich bei der Bewältigung komplexer Situationen.

4.1.1 Feedback

Der Rolle des Feedbacks im Bereich des Lernens soll noch etwas mehr Aufmerksamkeit geschenkt werden. In den beiden beschriebenen Modellen des Behaviorismus und Kognitivismus spielt externes Feedback eine wesentliche Rolle im Lernprozess (siehe auch Abbildung 4.1).

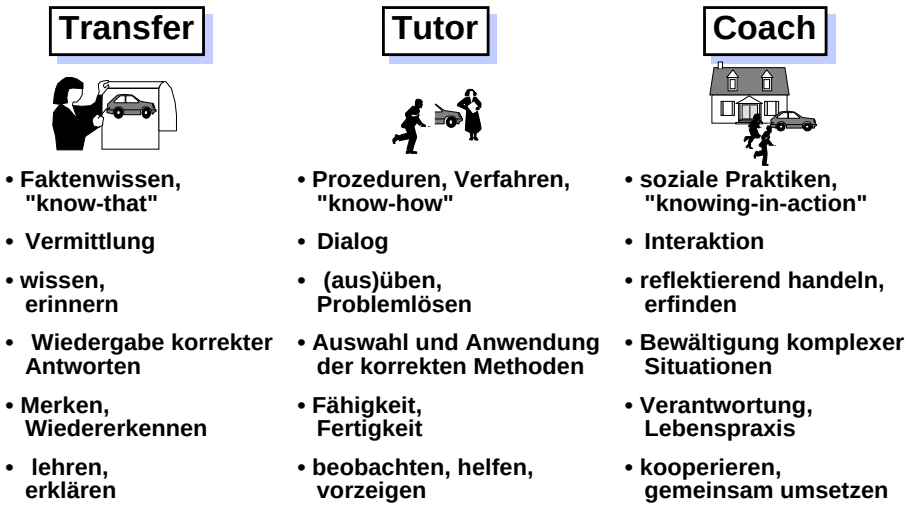


Abbildung 4.2: Drei Modelle des Lehrens (aus [Baumgartner97])

Steve Draper beschreibt in seinem Artikel ([Draper05]), dass externes Feedback nicht unbedingt zum Lernen benötigt wird. Dies ist auch konsistent mit dem Modell des Konstruktivismus. Er unterscheidet 5 Ebenen von Feedback:

1. Die unterste Ebene ist Feedback in Form einer einfachen *Benotung* oder *Richtig/Falsch* Bewertung der gebotenen Lösung.

2. Die nächste Ebene ist die Bereitstellung der *richtigen Antwort*. Der Unterschied zur Antwort des Lernenden wird damit ebenfalls angegeben.

3. Eine *„prozedurale Erklärung“* der richtigen Antwort wird in der nächsten Ebene gegeben. Die Lösung des Lernenden wird verbessert und die Lösungsschritte werden aufgezeigt.

4. Die *Begründung* für die richtige Antwort steht auf der nächsten Ebene. Es wird nicht nur die richtige Antwort gegeben und der Weg beschrieben, der zu dieser Antwort führt, sondern es wird auch argumentiert, warum dieser Weg gewählt wird.

5. Die letzte Ebene schließlich ist die *Erläuterung, warum die Lösung falsch ist*. Diese Art des Feedbacks wird gegeben, wenn der Lernende der Meinung ist, seine Lösung sei ebenfalls korrekt, wenn auch unterschiedlich von der erwarteten. Hier geht es darum, Fehlauffassungen zu erkennen und auszumerzen.

Bezug zum SQL e-Learning System

Das im Rahmen dieser Arbeit umgesetzte e-Learning System setzt auf Feedback für die Beurteilung der Eingabe eines Studenten. Die verwendeten Arten des Feedbacks können in Bezug auf die Ebenen von Draper betrachtet werden:

1. Feedback auf der untersten Ebene wird durch den Status der Aufgabe, der sofort angezeigt wird, gegeben.

2. Die korrekte Antwort wird dem Lernenden im System nicht gegeben.[1] Die Antwort ist quasi konstruktivistisch zu erarbeiten.

3. Die „prozedurale Erklärung“ wird dennoch teilweise gegeben, obwohl die korrekte Antwort nicht gezeigt wird. Zur SQL Eingabe des Studenten wird auf fehlende oder überflüssige Teile hingewiesen, damit wird zum Teil eine Lösungsvorschrift gegeben.

4. Die Begründung der richtigen Antwort wird nicht im interaktiven Teil des Lernsystems gegeben. Es wird davon ausgegangen, dass der Student jederzeit die Materialien abrufen kann, in denen die Begründungen (im konkreten Fall z.B. die SQL Syntax) nachzulesen sind.

5. Die letzte Ebene stellt eine Herausforderung dar. Es kann nämlich durchaus vorkommen, dass der Student eine Lösung findet, die nicht vom e-Learning System erwartet wurde, aber dennoch korrekt ist. Hier wurde der Ansatz gewählt, eine Warnung anzuzeigen, wenn dieser Fall möglich ist. Diese Fälle müssen durch einen Tutor geprüft werden.

4.2 Konzepte im e-Learning

Dieser Abschnitt soll einen kompakten Überblick über die Konzepte und Begriffe geben, die derzeit im Bereich e-Learning gängig sind. Ähnliche Übersichten sind in zahlreichen aktuellen Arbeiten zu finden, beispielsweise in [Knall05], [Bankwitz03] oder [Hugentobler04].

Lernsysteme im Allgemeinen können klassifiziert werden als *synchron/asynchron* oder als *online/offline*, um die beiden relevantesten Klassifizierungen zu nennen.

- Ein *synchrones* System (bzw. *synchrones Lernen*) erfordert, dass Lernender und Lehrender sich zur selben Zeit treffen. Das sagt aber noch nichts über die Standorte der beiden Beteiligten aus. Das Aufeinandertreffen zum synchronen Lernen kann beispielsweise auch in einem Chatroom passieren.

- *Asynchrones Lernen* hingegen verlässt sich auf asynchrone Kommunikationsmittel zwischen Lernendem und Lehrendem. Der Austausch von e-Mails zu Lernzwecken kann z.B. als asynchrones Lernen bezeichnet werden, weil Schreiben und Lesen der e-Mails voneinander entkoppelt sind.

- Bei *offline* Lernsystemen wird die Verbreitung von Lerninhalten durch CDs oder DVDs durchgeführt. Auch Lernprogramme, die im Internet zum Download angeboten werden, dann aber keinen Internetzugang mehr benötigen, können als *offline Lernsysteme* betrachtet werden.

- *online* Lernsysteme hingegen benötigen dauerhaften (oder „wiederkehrenden“) Zugang zu einem Verbindungsmedium - heute fast ausschließlich das Internet.

Lernsysteme, bzw. e-Learning Systeme speziell, nehmen die unterschiedlichsten Ausprägungen an. Es sind eine Reihe von Bezeichnungen für die verschiedenen Typen von Lernsystemen

[1] Natürlich besteht die Möglichkeit, die korrekte Antwort im Tutorengespräch zu erfahren, sollte man selbst nicht zum Ziel kommen.

gebräuchlich, die hier nicht in ihrer Fülle aufgezählt werden. Die gängigsten Typen von Systemen werden im Folgenden aufgelistet und kurz erläutert:

- *Distance Learning* (oder *Telelearning, Teleteaching, Teletutoring*) ist eine online Form des e-Learning und beschreibt im Grunde nur, dass Tutor und Student sich nicht am selben Ort befinden (müssen).
- *Computer Based Training (CBT)* ist ein gängiger Begriff, der typischerweise offline Lernsysteme beschreibt. Die Materialien oder Lernprogramme können vom Studenten auf den Computer transferiert werden (z.B. via CD/DVD oder durch Download von Software) oder es werden eigene Computer für das Training in Schulungsräumen o.Ä. verwendet. Der Lernende (Student) hat keinen Kontakt zum Lehrenden (Tutor). Das Wissen oder die Fähigkeiten werden zu beliebigen Zeiten selbst und mithilfe des Lernsystems erarbeitet.
- *Web Based Training (WBT)* (auch *Web Based Learning (WBL), Web Supported Learning (WSL)*) ist die Umlegung der CBT Konzepte auf das Internet. WBT Systeme haben damit die Möglichkeit, den Kontakt von Studenten und Tutoren oder Studenten und Studenten via Internet herzustellen. Außerdem können die Lerninhalte regelmäßig aktualisiert werden, was bei offline CBT Systemen nicht möglich ist.
- *Computer Supported Cooperative Learning (CSCL)* beschreibt Lernumgebungen, in denen die Teamarbeit, vor allem bei räumlicher Trennung der Lernenden, gefördert wird. Ein online CSCL System bietet Kommunikationsmöglichkeiten zwischen den Lernenden sowohl synchron (Chatrooms, etc.) als auch asynchron (e-Mail, Webforen, etc.). Offline CSCL Systeme erfordern, dass die Teilnehmer sich an einem Ort treffen (z.B. in CSCL Laboren).
- *Blended Learning* ist der Überbegriff für die Kombination moderner e-Learning Konzepte mit klassischen Lernmethoden (Vorlesungen, Übungseinheiten, etc.). Es werden typischerweise CSCL Konzepte verwendet, um beispielsweise die Lerneinheiten zu koordinieren oder verschiedene Zusatzinformationen oder Kurse zu den Lerneinheiten anzubieten.
- *e-Learning 2.0* ist ein Begriff, der erst seit kurzer Zeit in Gebrauch ist (erstmals in einem WWW Artikel von Stephen Downes 2005: [Downes05]). Das Prinzip von *e-Learning 2.0* besteht darin, dass die Lernenden (auch) Lerninhalte generieren. Die Inhalte entstehen damit in einer Community von Lernenden, genauso wie die Inhalte im *Web 2.0*.

4.2.1 Lernmanagement-Systeme

Als Lernmanagement-Systeme (LMS) bezeichnet man Systeme, die den Kern einer umfangreichen e-Learning Infrastruktur bilden. Ein LMS muss nicht zwingend das einfache Erstellen von Tests oder ähnlichem ermöglichen. Es muss auch im LMS kein Lerninhalt enthalten sein.

Handelt es sich um ein System in dem auch Lerninhalte verwaltet oder zusammengestellt werden, dann wird es auch oft als Lern-Contentmanagement-System bezeichnet (LCMS). Die Grenzen zwischen LMS und LCMS zerfließen aber sehr oft.

Im Gegensatz zu einer reinen Sammlung von Skripten oder Artikeln zum Download muss ein LMS, damit es als solches bezeichnet werden kann, zusätzliche Elemente besitzen, wie z.B.: Benutzerverwaltung, Kursverwaltung oder Kommunikationsmethoden.

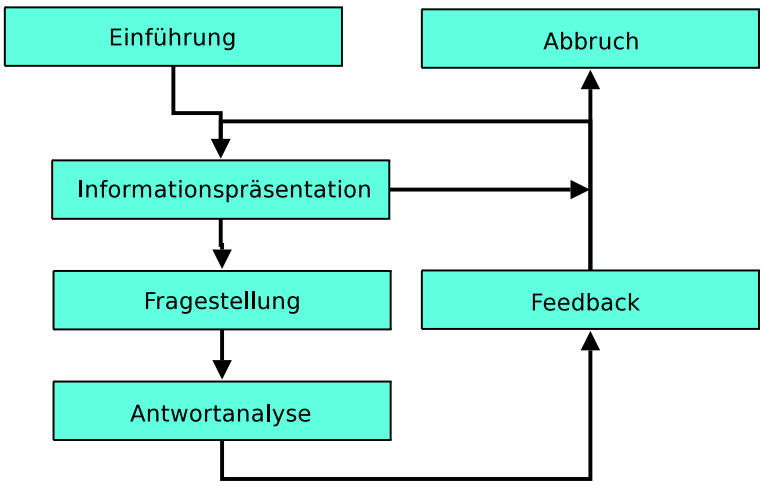


Abbildung 4.3: Prinzip eines Tutoring Systems (aus [Blumstengel98])

Beispiele für LCMS sind: Moodle[2] oder ILIAS [3]

4.2.2 Tutoring Systeme

Tutoring Systeme im Speziellen sind Systeme, die neues Wissen oder neue Fähigkeiten an den Lernenden übermitteln sollen. Tutoring Systeme übernehmen damit die Rolle eines Tutors.

> „Tutorielle Lernprogramme [...] übernehmen die Lernfunktion, bieten systematisch Informationen, stellen Aufgaben, analysieren Antworten und sind in der Lage, gezielte Rückmeldungen an den Lernenden zu geben.“ ([Knall05])

In Abbildung 4.3 wird der prinzipielle Aufbau eines Tutoring Systems dargestellt. Klassische Tutoring Systeme arbeiten nach einer starren Abfolge von Einheiten, die dem Lernenden dargeboten werden, oder die er zu lösen hat, bevor er zur nächsten Einheit weitergeht. Ein Ausbruch aus dieser starren Abfolge wird in *Intelligenten Tutoring Systemen* angestrebt. Siehe auch Abschnitt 4.4.

> „Reine Trainingssysteme können als spezielle Formen tutorieller Systeme angesehen werden. Sie beinhalten nur lineare Folgen von Tests mit entsprechendem Feedback, aber keine eigenständigen Komponenten zur Informationspräsentation.“ ([Blumstengel98])

Damit kann das hier entwickelte SQL e-Learning System in die Klasse der Tutoring Systeme eingeordnet werden.

[2] http://moodle.org/
[3] http://www.ilias.de/

4.3 Lernobjektmodelle

In [Heyer05] wird eine Bewertung von unterschiedlichen Lernobjektmodellen vorgenommen. Diese Bewertung liefert die Grundlage für die hier angeführte Zusammenfassung.

Der Begriff *learning object* (*Lernobjekt* wurde von Wayne Hodgins 1994 eingeführt und ist seit dem im Bereich der Inhaltsaufbereitung für e-Learning Systeme (siehe [Polsani03]) gebräuchlich. Unterschiedlichste Definitionen des Begriffs Lernobjekt sind in der Literatur zu finden. Die Definition von l'Allier beschreibt am deutlichsten den Lerninformationscharakter eines Lernobjekts:

> „[A Learning Object] is defined as the smallest independent structural experience that contains an objective, a learning activity and an assessment." ([L'Allier97])

Ein Lernobjekt ist die kleinste Einheit, aus dem sich Lerninhalte zusammensetzen. Das Zusammenstellen von Inhalten aus einzelnen Lernobjekten wird auch als *Content Aggregation* bezeichnet.

In verschiedenen Systemen werden unterschiedliche Ansätze, woraus ein Lernobjekt besteht, oder wie Lernobjekte zu größeren Einheiten zusammengeführt werden verfolgt.

- Im Modell von Duval und Hodgins ([Hodgins03]) entspricht ein Lernobjekt einem Dokument. Es werden 5 Stufen für die Aggregation von Lernobjekten definiert. Auf der untersten Stufe („raw data") herrscht die höchste Wiederverwendbarkeit von Lernobjekten, während auf der obersten Stufe („courses, books") höchster Zusammenhang des Inhalts besteht. Das Hauptaugenmerk wird auf die Wiederverwendbarkeit von Lernobjekten gelegt. Konzepte aus diesem Modell sind vermutlich in ARIADNE[4] eingeflossen (vergleiche [Heyer05]).

- Das SCORM (Sharable Content Object Reference Model) Modell von ADL [ADL06] stellt ein umfangreiches Standards-Framework für e-Learning Systeme dar. Das Aggregieren von Information ist eines der wichtigsten Konzepte im SCORM Modell. Es werden drei Aggregationsstufen definiert:

  1. *assets* (z.B. Text, Bilder) bilden die kleinste Einheit von Information
  2. *Sharable Content Objects (SCO)* setzen sich aus Assets zusammen. SCOs können ausführbare Einheiten beinhalten, die in einem SCORM Run-Time Environment lauffähig sind. SCOs sollten möglichst „kontextfrei" sein, damit sie in der nächsten Stufe in unterschiedlichen Kontexten verwendet werden können.
  3. *Content Aggregation* ist schließlich das Zusammenstellen von Lerninhalten auf Basis von assets und SCOs.

  Das SCORM Modell ist „pädagogisch neutral". Auf pädagogische Konzepte in den SCOs wird nicht eingegangen. Man geht davon aus, dass die Inhalte „erlernbar" sind.

  Abgesehen vom *Content Aggregation Model* definiert SCORM noch die Module *Run Time Environment* und *Sequencing & Navigation*. Siehe Abbildung 4.4.

[4] http://www.ariadne-eu.org/

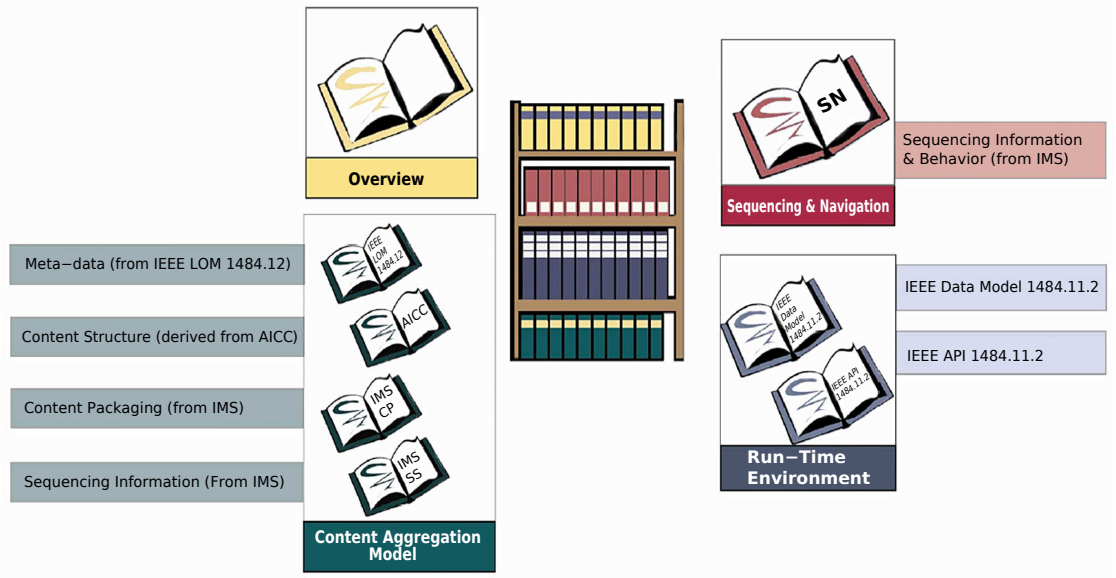


Abbildung 4.4: SCORM Bookshelf (aus [ADL06])

- Cisco Systems definiert ein pädagogisch ausgereiftes System für das Einsatzgebiet der beruflichen Weiterbildung ([Cisco99]). In diesem System wird zwischen *Reusable Information Object (RIO)* und *Reusable Learning Object (RLO)* unterschieden. Ein RLO setzt sich zusammen aus einem *Overview*, einem *Assessment* und 5-9 RIOs (siehe Abbildung 4.5).

- In NETg ([L'Allier97]) wird das Lernobjekt auch *Topic* genannt. Ein Topic besteht aus Lernziel, Lernaktivität und der Überprüfung (siehe Abbildung 4.6). Topics können zu *Units* und *Lessons* aggregiert werden. In NETg wird ein behavioristischer Ansatz verfolgt: das Lernziel beinhaltet die Anleitungen für eine Tätigkeit, die zu erlernen ist.

Die vorgestellten Modelle können in zwei Gruppen eingeteilt werden:

- Die Modelle von Duval & Hodgins und SCORM konzentrieren sich vorrangig auf technische Anforderungen und möglichst hohe Wiederverwendbarkeit von Lernobjekten.

- Cisco und NETg, auf der anderen Seite, beschreiben den Aufbau von Lernobjekten anhand pädagogischer Richtlinien.

4.4 e-Learning und Artificial Intelligence

In [Martens04] wird ein umfangreicher Überblick über den Einsatz von Artificial Intelligence im e-Learning geboten. Die folgende Zusammenfassung stützt sich zu einem wesentlichen Teil auf die Informationen in diesem Artikel.

Konzepte aus der KI wurden schon Anfang der 1970er Jahre mit e-Learning Konzepten verbunden. Die resultierenden Lernsysteme fallen in die Kategorie der *Intelligent Tutoring Systems*

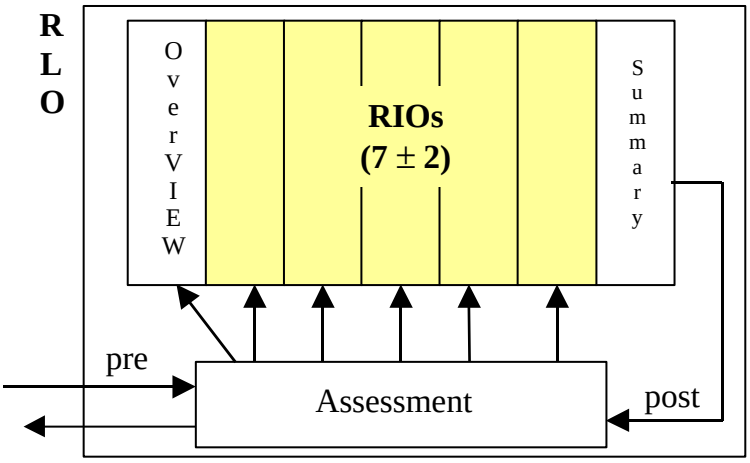


Abbildung 4.5: Aufbau eines RLO im Modell von Cisco Systems (aus [Cisco99])

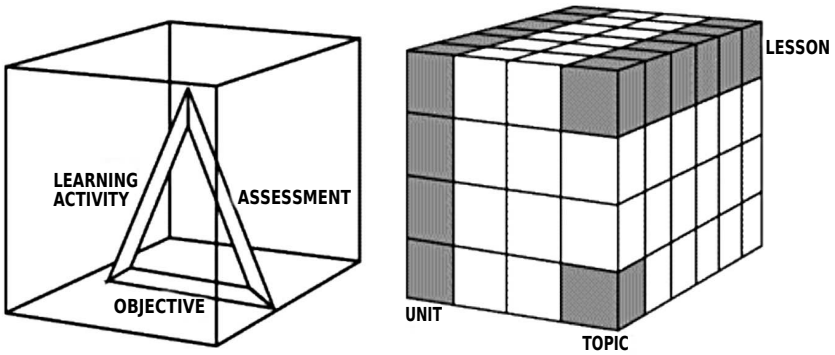


Abbildung 4.6: NETg Lernobjekt (Topic) und Aggregation (aus [L'Allier97])

(auch *Intelligente Lehr/Lernsysteme*). Als eine Erweiterung zu den klassischen Tutoring Systemen (siehe Abschnitt 4.2.2) bieten ITS beispielsweise eine Adaption des Lernprozesses an das Verhalten des Lernenden. Konkret werden drei Wissensgebiete unterschieden, die in ein ITS einfließen:

- Das *Domänenwissen* ist das Wissen über das zu erlernende Gebiet (Domäne).
- Das *Wissen über den Lernenden* (*Lernermodell*) beinhaltet Wissen über das Verhalten des Lerners.
- *Didaktisches Wissen* steuert das Verhalten des ITS.

Alle drei Gebiete können in Form von Expertensystemen (z.B. mit Produktionsregeln) im ITS manifestiert sein.

Lelouche ([Lelouche99]) unterscheidet zusätzlich zwischen ICAI (*Intelligent Computer Aided Instruction*) und ITS. Er bezeichnet Systeme, die nur Expertenwissen über die Anwendungsdomäne besitzen, als ICAI Systeme im Sinne einer Erweiterung zu einfachen CAI Systemen. Erst wenn auch Wissen über Lerner im System abgebildet wird, nennt er das System ein ITS (vergleiche [Martens04]).

Beispiele für ITS bzw. ICAI Systeme:

- Eines der ersten Beispiele für ein CAI System mit Artificial Intelligence war SCHOLAR ([Carbonell70]). SCHOLAR lehrt südamerikanische Geographie mithilfe von umfangreichem Fakten- und Regelwissen. Der Benutzer wird dabei mit einem Dialogsystem konfrontiert. Das System besitzt keinerlei Lernermodell.
- Anfang der 1970er Jahre stellten Brown et al. SOPHIE vor ([Brown74]). SOPHIE (SOPHIsticated Instructional Environment) ist ein System für die Diagnose von Fehlern in elektronischen Schaltkreisen. Der Student soll lernen, einen fehlerhaften Schaltkreis zu korrigieren. Mithilfe von Simulation wird die Funktion des Schaltkreises simuliert. Expertenwissen und Regelwissen des Systems bewerten die Lösung des Studenten mithilfe des Regelsystems.
- SYPROS ist ein System zum Lehren von Synchronisationsmechanismen in parallelen Prozessen ([Rotenhofer93]). In SYPROS werden Aufgaben automatisch generiert und die didaktischen Aspekte, die dem Lernprozess zugrunde liegen, in einem Regelsystem abgebildet.

Die folgende Auflistung gibt einen Überblick über weitere mögliche und auch realisierte Einsatzgebiete für KI in e-Learning:

- *Planung & Planerkenner* werden verwendet um beispielsweise Unterrichtspläne zu erstellen. Insbesondere Planerkenner können zur Bewertung von Lösungen verwendet werden, indem die Lösung mit einer „geplanten“ Lösung verglichen wird.
- Wie bereits besprochen werden *regelbasierte Produktionssysteme* verwendet, um Wissen im Anwendungsbereich zu implementieren, als auch um Wissen über den Lerner (Lernermodell) abzubilden.

- *Mining und Clustering* Methoden können verwendet werden, um in Lernmanagement Systemen das Verhalten oder die Interessengebiete von Lernenden zu analysieren und so User mit ähnlichen Mustern zusammenzuführen.

- Neuere Arbeiten versuchen vermehrt *Recommender Systems* für die Verwaltung und Bewertung der Lerninhalte zu verwenden. In [Tang03] wird ein Vorschlag für ein intelligentes Lernmanagement System mit Clustering Methoden und Content Filter erbracht. Der Ansatz soll dynamische Inhalte von den Lernenden selbst steuern und dadurch die Qualität verbessern.

- KI zur *Lerneradaption*. Das sind unterschiedliche Konzepte, die das Ziel verfolgen, das Lernsystem dem Lerner anzupassen. D.h. das Lernsystem reagiert auf die unterschiedlichen Verhaltensmuster des Lerners und passt sich entsprechend an. Es wird also das Lernermodell dynamisch angepasst. Vergleiche [Sauerstein07].

4.5 Positionierung

Das im Rahmen dieser Arbeit umgesetzte System kann als *Intelligent Tutoring System* bezeichnet werden. Es verfügt über Wissen zum Aufgabengebiet, weil die SQL Statements einfach durch Ausführen in einer Datenbank getestet werden können.

Das spezifizierte Regelwissen deckt zum Teil ebenfalls das Anwendungsgebiet SQL ab (die Analyse der eingegebenen SQL Statements), wenn man aber das Bewertungssystem betrachtet und das Feedbackkonzept, so wird deutlich, dass in diesem Fall die didaktische Komponente als Regelwissen hinterlegt ist.

Nach [Lelouche99] allerdings wäre das System als ICAI System zu bezeichnen, da hier in keiner Weise ein Lernermodell verwendet wird. Das System verhält sich immer gleich, egal welcher Student mit welchem Wissensstand die Aufgaben löst.

Das ganze System kann entweder als ein *Learning Object*, oder, wenn die Materialien[5] entsprechend aufgeteilt werden, als mehrere LOs, die sich jeweils mit unterschiedlichen Aspekten von SQL beschäftigen, gesehen werden.

Betrachtet man das System ohne informationsvermittelnder Materialien, sondern nur als SQL Abfrage Trainer, dann ist das System als reines Trainingsprogramm zu verstehen (WBT).

Da die reguläre Vorlesung weiterhin gehalten wird, und das e-Learning System nur einen Bereich des Inhalts abdeckt, kann die ganze Lehrveranstaltung als *Blended Learning* Veranstaltung bezeichnet werden.

Aufgrund des aus dem Output (Ergebnis der SQL Query) generierten Feedbacks für den User könnte man sagen, es wird das Lernmodell des *Kognitivismus* verwendet. Ansonsten hat es aber eine Tendenz zum *behavioristischen* Ansatz, weil der Student im Grunde nur die Lösung eingibt und diese am Ende bewertet wird. Betrachtet man die gesamte Lehrveranstaltung, dann fließen auch *konstruktivistische* Ansätze ein: der Student kann einzelne Aspekte von SQL selbständig aus den Unterlagen erarbeiten.

[5] Die online Ressourcen im System zum Lernen von SQL werden hier nicht vorgegeben. Diese Aufgabe obliegt dem Institut beim Einsatz des Systems.

5 Userinterface Design

In diesem Kapitel wird im Detail auf die Überlegungen und Ergebnisse des Userinterfacedesigns eingegangen.

Im ersten Abschnitt (5.1) wird das Umfeld abgesteckt und die prinzipellen Strukturen und Anforderungen beschrieben; in den darauf folgenden Abschnitten (5.2 und 5.3) werden detailliert die Überlegungen und Ergebnisse des Userinterfacedesigns einerseits aus Perspektive der grafischen Darstellung andererseits aus funktionaler Sicht erläutert.

Im letzten Abschnitt (5.4) wird ein Gesamtüberblick über die Applikationsoberfläche gegeben. Ausgehend von einem Navigationsdiagramm werden die einzelnen Seiten und ihre Funktionalität beschrieben.

5.1 Überblick

5.1.1 Kontext

Es wird hier vor allem auf das Design des Userinterfaces im Prototyp eingegangen. Da im Prototyp nur das Übungsmodul umgesetzt wurde, bezieht sich alles hier Gesagte auch vor allem auf den Bereich des Übungsmoduls des Systems.

Die meisten hier beschriebenen Konzepte treffen aber auch auf das Prüfungsmodul[1] zu. Auch mit dem Verwaltungsmodul gibt es große Überschneidungen, da in allen drei Modulen die Darstellung und Navigation der Aufgaben eine wesentliche Rolle spielen.

5.1.2 Seitenaufbau

Um die Bedienbarkeit und Übersicht der Applikation zu gewähren, setzt sich jede HTML Seite des Systems aus Teilbereichen zusammen, die immer gleich verwendet werden. In Abbildung 5.1 sind diese Bereiche dargestellt und im Folgenden sollen diese noch näher beschrieben werden:

- Im *Header* befinden sich Logos, der Name des Systems und die Login Information des Benutzers. Die Login Information wird deutlich lesbar dargestellt, um damit bei der Prüfung die Ausweiskontrolle zu erleichtern.
- Im Bereich des *Globalen Menüs* werden allgemeine Links untergebracht. Beispielsweise ein Link zur allgemeinen Hilfe, Übungsmaterial oder Logout. Hier ist auch der Link zur Übersicht der angebotenen Kurse zu finden.
- Der Bereich des *Lokalen Menüs* ist nur interessant, wenn ein Kurs vom Benutzer ausgewählt wurde. In diesem Bereich befindet sich das kursspezifische Menü. Hier kann der

[1] Das Prüfungsmodul ist im Grunde nur eine eingeschränkte Variante des Übungsmoduls. Außerdem können auch im Übungsmodul Prüfungen zum Test durchgespielt werden.

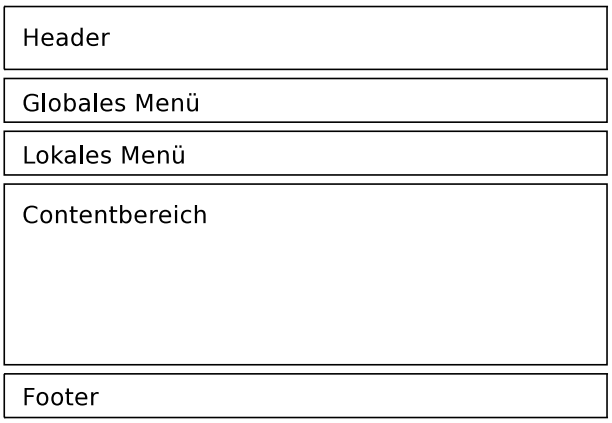


Abbildung 5.1: Aufbau einer HTML Seite des Systems

Benutzer z.B. zu den Aufgaben, zur Übersicht des Kurses oder zur Beschreibung der Kursdatenbank navigieren.

- Im *Contentbereich* wird der gesamte informative Inhalt untergebracht. Die Gestaltung in diesem Bereich untergliedert sich nicht weiter. Es wird nur versucht, die Kontroll-Elemente möglichst auf einer gedachten linken Spalte anzuordnen (siehe Abschnitt 5.2.1).

- Im *Footer* werden schließlich noch allgemeine Links, wie Kontakt & Impressum oder Copyright, untergebracht.

Dieser grundsätzliche Seitenaufbau ergibt sich aus der „Vorgabe“ durch das Lehrveranstaltungssystem, wo der blaue Balken, der hier als Bereich für das Globale Menü genutzt wird, ein wichtiger Bestandteil der Seite ist.

Ausgehend von diesen Vorgaben wurde versucht, den vorhandenen Platz auf der Seite möglichst gut zu nutzen. So stammt die Entscheidung, das lokale Menü horizontal unter dem blauen Balken anzuordnen aus der Überlegung, möglichst wenig Platz vom Content wegnehmen zu wollen.

5.1.3 Navigation

Aus Abbildung 5.2 ist die wesentliche Navigationsstruktur der Applikation (bzw. des betrachteten Übungsmoduls) zu entnehmen.

In der *Kursauswahl* wird eine Übersicht der vorhandenen Kurse mit Beschreibung dargestellt. Der Benutzer hat von hier aus die Möglichkeit, einen Kurs zur Bearbeitung auszuwählen. Durch die Auswahl eines Kurses wird im Grunde eine Kursdatenbank [2], sowie die Beschreibung der Datenbank und die Aufgabenstellungen ausgewählt.

Wenn ein Kurs ausgewählt ist, kann man sich über die drei Menüpunkte *Übersicht, Datenbank* und *Aufgaben* einen Überblick über den Kurs verschaffen. Unter *Übersicht* wird ein einleitender Überblick über die Kursdatenbank gegeben. Die Inhalte werden beschrieben und mit

[2] Eine Kursdatenbank ist eine Beispieldatenbank, auf der die Testabfragen ausgeführt werden.

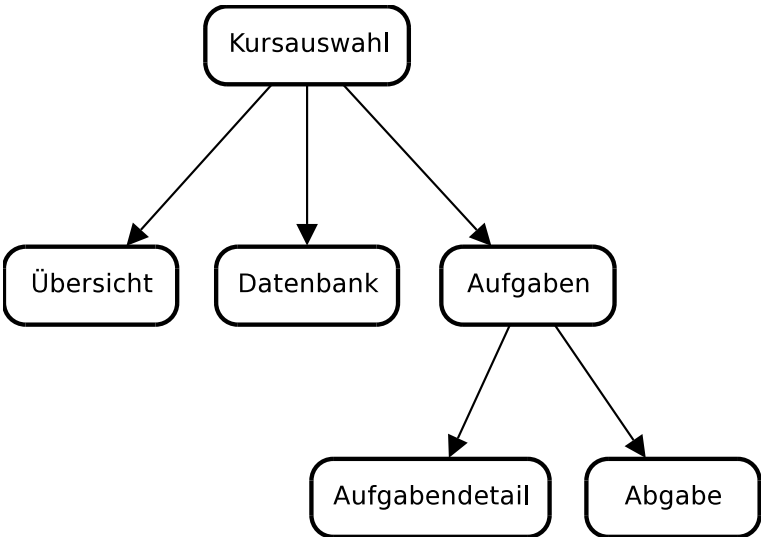


Abbildung 5.2: Navigationsfluss in der Applikation

einem ER Diagramm dargestellt. Unter *Datenbank* hat der Benutzer die Möglichkeit, sich interaktiv mit der Kursdatenbank auseinanderzusetzen. Es wird hier einerseits das Relationenmodell (dynamisch aus den vorhandenen Daten) dargestellt. Andererseits wird hier die Möglichkeit geboten beliebige Datenbankabfragen durchzuführen, oder sich einfach den Inhalt einer Tabelle anzeigen zu lassen. Unter *Aufgaben* schließlich werden alle Aufgaben im Überblick angezeigt. Von hier aus kommt man auf die *Aufgabendetail* Ansicht. Dort wird die Aufgabenstellung nochmal angezeigt, und man kann die Lösung bearbeiten.

Mit der Aktion *Abgabe* schließlich können sowohl im Übungsmodul als auch im Prüfungsmodul die Ergebnisse abgegeben werden. D.h. die Ergebnisse werden in die Datenbank übernommen und sind zur Bewertung freigegeben.

Eine ausführlichere Beschreibung der Navigation und der einzelnen Seiten kann im Abschnitt 5.4 nachgelesen werden.

5.1.4 Grafische Anforderungen

Anforderungen an das Erscheinungsbild der Applikation ergeben sich aus der Notwendigkeit, die Applikation in das vorhandene Lehrveranstaltungssystem zu integrieren. Dort werden bestimmte Stilelemente verwendet, an die sich die Applikation nach Möglichkeit halten sollte.

Diese Vorgaben (welche eher als „Richtlinien“ zu verstehen sind), sind im Wesentlichen folgendermaßen zusammenzufassen: Die grundlegende Farbgebung ist blau auf weißem Hintergrund; die Logos des Instituts und der Universität sind ident zu übernehmen; der vorhandene blaue Balken sollte weiterverwendet werden.

Zusätzlich wurde die Anforderung gegeben, eine möglichst breite Masse an Webbrowsern zu unterstützen, wenngleich in diesem Bereich auch keine konkreten Browserversionen gefordert sind. Als Minimum wird festgelegt, die aktuellen Versionen von Mozilla Firefox, Opera, Internet Explorer und Konqueror zu unterstützen. Dass die Oberfläche in allen Browsern genau gleich aussieht ist *keine* Forderung - die Verwendbarkeit muss gewährleistet sein.

5.2 Grafisches Design

In diesem Abschnitt werden die Aspekte des grafischen Designs des Userinterfaces näher beleuchtet.

Die Überlegungen, die zum vorliegenden grafischen Design des Systems geführt haben, stützen sich zum Großteil auf gestalterische Grundlagen, wie sie z.B. in [Khazaeli05] oder [Krug00] zu finden sind.

Aufbauend auf diese Grundlagen ist das Aussehen der Applikation nach mehreren Anläufen schließlich zu dem geworden, was es jetzt darstellt.

Eingeflossen in das Design sind aber auch technische Aspekte, wie die Frage nach der Browserkompatibilität. Ziel des Designs war es, eine Umsetzung zu finden, die keine aufwändigen Features benötigt, welche nur in den neuesten Versionen von speziellen Browsern zu finden sind. Dieser Gedanke geht Hand in Hand mit der Anforderung eines „schlichten“, unaufwändigen Designs.

Die vorliegende Applikationsoberfläche wurde mit folgenden Browsern getestet: *Mozilla 1.0, Firefox 2, MS Internet Explorer 5, 6 & 7, Konqueror 3.5, Opera 9*. Jeweils (soweit möglich) auf MS Windows und Linux.

Es wurde Rücksicht darauf genommen, dass die Applikation auch ohne Javascript bedienbar ist, auch wenn einige Benutzererleichterungen sich auf Javascript stützen.

Bei der Verwendung von Fonts wurde darauf geachtet, dass jedem Fall zumindest auch weit verbreitete Standardfonts verwendet werden können, falls die eingesetzten spezielleren Fonts auf dem System nicht installiert sind. Weitgehend wurde allerdings ohnehin auf spezielle Fonts verzichtet.

Um Accessibility Anforderungen zu einem gewissen Grad gerecht zu werden, werden Buttons zur Verfügung gestellt, welche die Fontgröße gemeinsam mit Icongrößen anpassen. Es war allerdings im Umfang des Projekts nicht möglich, die Richtlinien des WAI [3] im Detail zu berücksichtigen. Zum einen weil die dynamische Natur der Applikation das nicht immer erlaubt (oder es zumindest erschwert) und zum anderen, weil dadurch oft die Kompatibilität mit älteren Webbrowsern gebrochen werden müsste.

Das Logo der Applikation (auch zu sehen in Abbildung 5.3 links oben) verwendet als primäre Farbe das Orange, das in der Applikation als Farbe zum Hervorheben verwendet wird (mehr dazu später). Es wird auf 3D Effekte verzichtet, was passend zum schlichten Stil der gesamten Applikation sein soll. Im Logo werden Symbole aus der Thematik verwendet: eine Tabelle verbunden mit einem „Q“ (für „Query“).

5.2.1 Oberflächenstil und Erscheinungsbild

Im Folgenden sollen die wesentlichen Stilelemente des Designs erläutert werden, sowie die Intentionen, die zu diesen geführt haben. Die hier beschriebenen Elemente sind in einem Beispielscreenshot in Abbildung 5.3 zu sehen. Auf Einzeldarstellungen wird deshalb verzichtet.

Das grundlegende Design der Applikation stützt sich auf einen „flachen“ Stil. D.h. es werden keinerlei 3D Effekte (Schatten oder Emboss-Effekte) verwendet. Das hat nicht zuletzt auch seinen Ursprung in der Anforderung, mit einem möglichst breiten Spektrum an Browsern kompatibel zu bleiben.

[3] Web Accessiblity Initiative des W3C - http://www.w3.org/WAI/

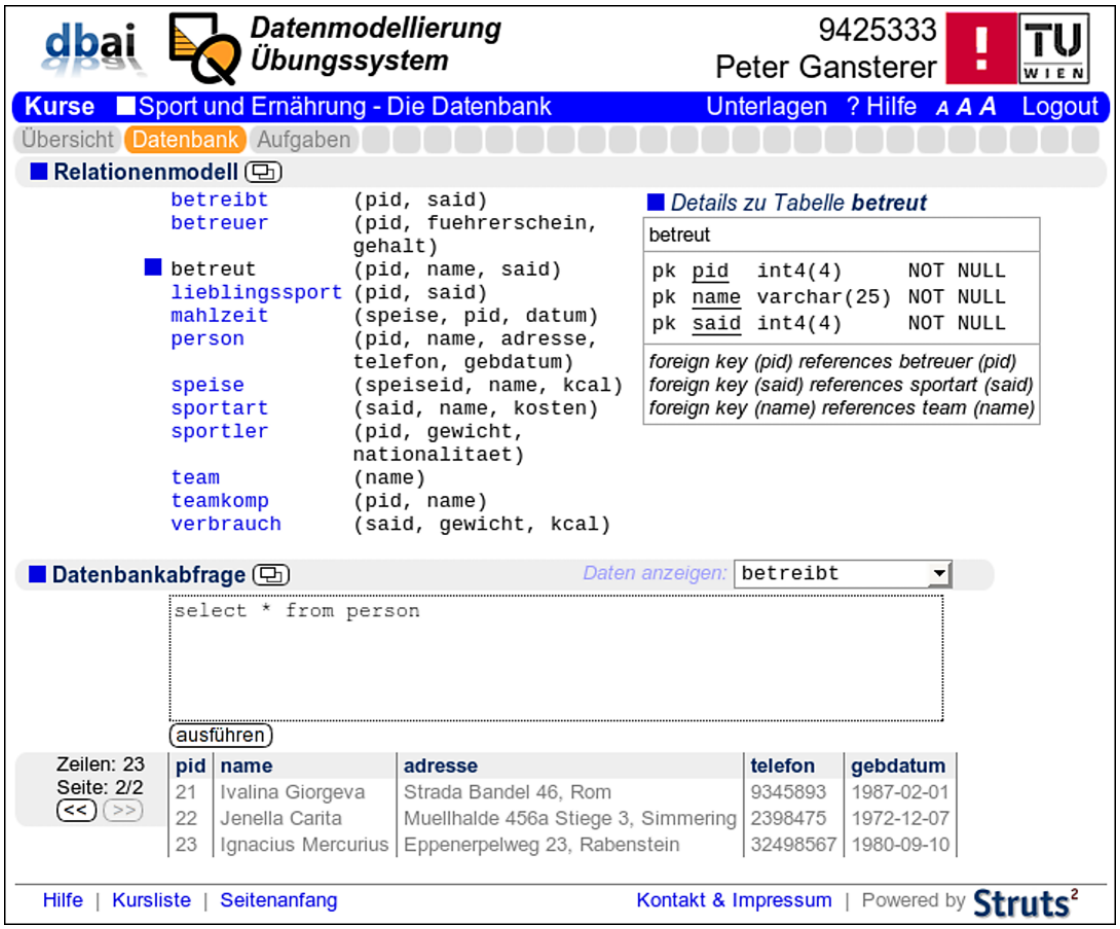


Abbildung 5.3: Beispielseite aus der Applikation mit wesentlichen Elementen

Wiederkehrende Elemente

Das Erscheinungsbild der Applikation soll durch die Verwendung von wiederkehrenden Elementen konsistent bleiben:

- *Abgerundete flächige Hinterlegungen:* Es wird an unterschiedlichen Stellen (z.B. Überschriften oder „Informationsboxen“) der Inhalt mit einer grauen Fläche hinterlegt, um ihn von der Umgebung abzugrenzen. Diesen Flächen werden immer die Ecken abgerundet.

  Dieselben gerundeten Ecken werden bei Icons und Buttons verwendet.

- *Quadrate zur Hervorhebung von Elementen:* Im Gegensatz zu den gerundeten Ecken stehen kleine blaue Quadrate, die zur Hervorhebung von Überschriften oder ausgewählten Elementen dienen. Die Quadrate in Fonthöhe werden dem Text vorangestellt, wodurch die Aufmerksamkeit auf diesen Text gelenkt werden soll.

- *Einheitliches Erscheinungsbild von Buttons:* Die Buttons im Contentbereich sind einheitlich als weiße Rahmen (mit den bereits erwähnten abgerundeten Ecken) um den Text oder das Icon gestaltet. Gemeinsam mit dem Hover-Effekt beim Aktivieren mit der Maus sollen so die Bedienelemente einheitlich erkennbar sein.

Die Buttons im Menü unterscheiden sich von den Buttons im Contentbereich. Dort wird eine abgerundete graue Fläche verwendet, um einen Menüeintrag erkennbar zu machen. So unterscheiden sich optisch die Bedienelemente aus dem „globalen Menü“ (weiße Links auf dem blauen Balken) von denen aus dem „lokalen Menü“ (grau hinterlegte Buttons) und diese wiederum von den Bedienelementen im Contentbereich (umrahmte weiße Buttons).

Die logische Hierarchie der Navigation wird so auch grafisch unterstützt. Dabei wird durch die Abrundungen, die Farbgebung und das flächige Design ein durchgehender Stil beibehalten.

Farben

Die gewählten Farben der Applikation waren durch das vorhandene Lehrveranstaltungssystem, mit dem diese neue Applikation integriert werden soll, vorgegeben. Dort wird blau als Hauptfarbe verwendet, die vor allem in dem dominanten blauen Menübalken zu sehen ist. In der Applikation wurde zusätzlich orange als Signalfarbe eingesetzt, um Elemente hervorzuheben (verwendet im Logo, bei aktivierten Menüeinträgen oder im Hover-Effekt für Bedienelemente).

Die Oberfläche erscheint also blau auf einem „offenen“ weißen Hintergrund (d.h. kein Rahmen um den Applikationsbereich). Zusätzlich werden graue Hintergründe verwendet, um Bereiche zu gruppieren oder hervorzuheben.

Mit diesen Farben kommt die Applikation im Normalfall aus. Ausnahmen sind Fehlerfälle im Feedback von Usereingaben, wo zusätzlich Rot als Signalfarbe (als Textfarbe oder zum Einfärben von Tabellenhinterlegungen) verwendet wird.

Fonts

Es wird ein Sans Serif Font für Text und Bedienelemente verwendet. Für Daten (d.h. für Ergebnisse von Datenbankabfragen oder die Darstellung von Metadaten) sowie für Usereingaben wird ein Monospace Font verwendet. Dabei wird zwischen Daten und Eingabe unterschieden.

Es werden zwar spezielle Fonts als erste Wahl verwendet (konkret: `Liberation Mono` und `Liberation Sans`), es wird aber in allen Fällen als Fallback mindestens ein Font angegeben, der allgemein als „sicherer“ Font für Webseiten angesehen wird [4]. Das sind konkret: `Arial` und `Helvetica` als Standardfonts und `Courier New` bzw. `Courier` als monospaced Fonts.

Anordnung

Die Anordnung der Elemente im Contentbereich folgt dem Gedanken, die Bedienelemente einheitlich in einer gedachten linken Spalte anzuordnen.

Ursprünglich bestand die Idee, auf der linken Seite des Inhalts eine sichtbare linke Spalte einzuführen, in der Überschriften, Bedienelemente und Hinweise zusammengefasst werden. Dieser radikale Ansatz wurde aber abgeschwächt, so dass nur die Tendenz bleibt, Bedienelemente links anzuordnen und den Inhalt konsequent ausgerichtet einzurücken.

In vielen Fällen hätte der ursprüngliche Ansatz der linken Spalte zu unlogischen Anordnungen geführt, und wäre zu ungewohnt für den Benutzer gewesen. Das hat sich auch teilweise aus Usability Tests ergeben (siehe Abschnitt 5.3.5).

Es bleiben die Vorschriften: Zwischenüberschriften beginnen ganz links, der Inhalt wird einheitlich eingerückt. Bedienelemente werden zuerst links angeordnet. Erst wenn das zu „unnatürlich“ erscheint, werden die Elemente an eine andere Stelle versetzt.

5.2.2 Vorausgehende Ansätze

Das vorliegende Ergebnis des grafischen Designs soll hier noch mit anderen, weniger erfolgreichen Lösungsansätzen, für einzelne Teilbereiche in Kontrast gesetzt werden. Verschiedene Überlegungen, die nicht aus dem Ergebnis ersichtlich sind, sollen so noch aufgezeigt und eine Idee der Entwicklungsgeschichte vermittelt werden.

Die vorgestellten Lösungsansätze und Überlegungen wurden zu unterschiedlichen Graden ausgearbeitet, aber meistens bereits vor dem Stadium einer fertigen Lösung verworfen.

Globales „Workflow Menü“

Im Laufe der Entwicklung bestand die Idee, ein globales Menü einzuführen, das den Benutzer durch die Vorgabe von Arbeitsschritten leiten sollte. Zusätzlich sollte dieses möglichst platzsparend im Kopf der Applikation angeordnet sein sollte.

Das Workflow-Menü sollte folgende Arbeitsschritte beinhalten: *Kurs wählen, Aufgaben Übersicht, Aufgaben bearbeiten* und *Abgeben*.

Siehe Abbildung 5.4, wo der erste Entwurf dieser Idee dargestellt wird.

Dieser Ansatz litt allerdings unter verschiedenen Problemen:

- Die künstlich erfundenen Arbeitsschritte wirken unlogisch, da es sich um eine Linearisierung und Gleichsetzung von Aufgaben und Aktionen handelt, die in einer logischen Hierarchie unterschiedlich angeordnet sind, oder auch gar nicht unbedingt in dieser Reihenfolge abgearbeitet werden müssen.

[4] siehe z.B.: http://web.mit.edu/jmorzins/www/fonts.html

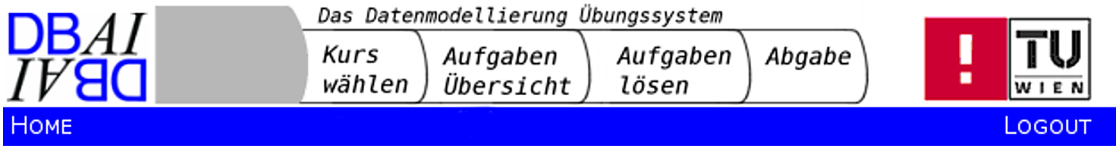


Abbildung 5.4: Idee eines „Workflow Menüs“

Abbildung 5.5: Versuche für das lokale Menü als „Tabs“

- Gestalterisch erwies sich die Idee, den freien Platz im Header damit zu nutzen als schwierig, da es durch die Trennung mit dem Inhalt (über den blauen Balken, der ja vorgegeben war) als kaum machbar erwies, einen optischen Zusammenhang zwischen dem ausgewählten Arbeitsschritt und dem dargestellten Inhalt herzustellen.

Schließlich wurde die Idee verworfen und wich einer klaren hierarchischen Gliederung.

Anderes Grunddesign

Vor der Festlegung auf ein flaches, einfaches Design wurde mit verschiedenen anderen Stilen experimentiert: 3D Buttons oder Hervorhebung von Navigationsbereichen durch 3D Effekte. Mit dem Gedanken an eine Prüfungsarbeit wurde der Ansatz einer „Papiermetapher“ versucht, wobei die Arbeitsfläche als ein Blatt Papier dargestellt werden sollte.

Bei diesen Ansätzen hat es sich aber meist als problematisch erwiesen, den Stil des vorhandenen Lehrveranstaltungssystems, mit den vorgegebenen Logos, Farben und dem blauen Balken, nicht zu brechen. Es hätte zuviel gestalterischen Aufwand für das geplante Projekt dargestellt, die effektvolleren Stile mit den vorgegebenen Elementen abzustimmen. Abgesehen davon war es bei diesen Versuchen allgemein schwieriger, das Design mit einer Reihe von Webbrowsern kompatibel zu halten.

Lokales Menü

Für das lokale Menü wurden auch unterschiedliche Ansätze versucht, bevor sich das jetzige Ergebnis manifestiert hat. So wurde z.B. versucht, das lokale Menü als „Tabs“ auszuarbeiten (siehe Abbildung 5.5). Dieser Ansatz wurde aber verworfen, da sich grafisch keine Konsistenz herstellen ließ mit den anderen Elementen.

Die Navigationsleiste für die Aufgabennavigation (die Aufgaben-Icons) wurden in verschiedenen Varianten ausgearbeitet. Weiters wurden unterschiedliche Arten angedacht, ein einzelnes

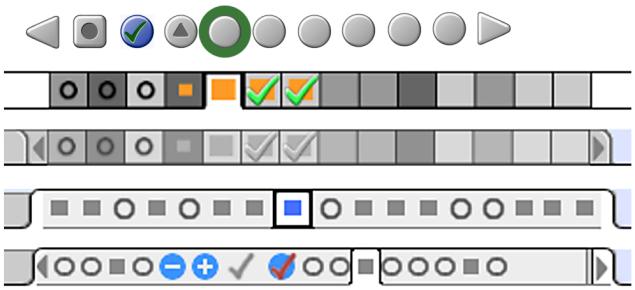


Abbildung 5.6: Versuche für Umsetzung der Aufgabennavigation

Icon hervorzuheben (um die ausgewählte Aufgabe anzuzeigen). Siehe Abbildung 5.6 für eine Übersicht verschiedener (gescheiterter) Ansätze.

Allen Versuchen gemein ist die fehlende Konsistenz der Oberfläche, oder der zu hohe Aufwand, die Konsistenz, unter Berücksichtigung der vorgegebenen Elemente, herzustellen. Auch die Browserkompatibilität hätte bei verschiedenen Ansätzen voraussichtlich leiden müssen.

Somit ergibt sich das Userinterface als einfaches flächiges Design, das versucht die grafische Konsistenz in sich, und zum Lehrveranstaltungssystem, zu gewähren.

5.3 Funktionales Design

In den folgenden Abschnitten werden einzelne Aspekte der Funktionsweise der Applikation beschrieben. Im Gegensatz zum vorigen Abschnitt geht es hier weniger um die gestalterische Umsetzung, sondern um die Funktionsweise und Usability des Systems. Es werden die Konzepte vorgestellt, die dem User Überblick verschaffen und die Bedienung vereinfachen sollen.

5.3.1 Navigationskonzept

Die Navigation des Systems gestaltet sich als funktionale Hierarchie, die sich (wie bereits in Abschnitt 5.2 erwähnt) in unterschiedlicher Verwendung grafischer Mittel, als auch in der Anordnung der Navigationselemente von oben nach unten zeigt: an oberster Stelle ist das globale Menü angesiedelt, darunter findet der Benutzer das lokale Menü, das sich auf einen ausgewählten Kurs bezieht und darunter, im Contentbereich, finden sich Bedienelemente, die sich nur auf die aktuelle Seite beziehen.

In Abbildung 5.7 wird die hierarchische Gliederung der Navigation verdeutlicht. Man sieht nochmal (wie im Abschnitt 5.1 bereits beschrieben), wie sich die Applikationsseite in *Header, globales Menü, lokales Menü, Contentbereich* und *Footer* gliedert.

Eine Ausnahme der hierarchischen Navigationsstruktur könnte der `Abgeben` Button darstellen. Dieser ist zu finden unter dem Menüpunkt `Aufgaben`, wo der Gesamtüberblick über die Aufgaben und die eingegebenen Lösungen dargestellt wird. Logisch gesehen ist das auch der passende Platz für den Button. Die Abgabe der Prüfung ist allerdings so wesentlich, dass dieser Button vielleicht an einen prominenteren Platz rücken sollte, wie z.B. in die lokale Menüleiste.

Ob diese mögliche Änderung umgesetzt wird, wird sich noch in späteren Usability Tests entscheiden.

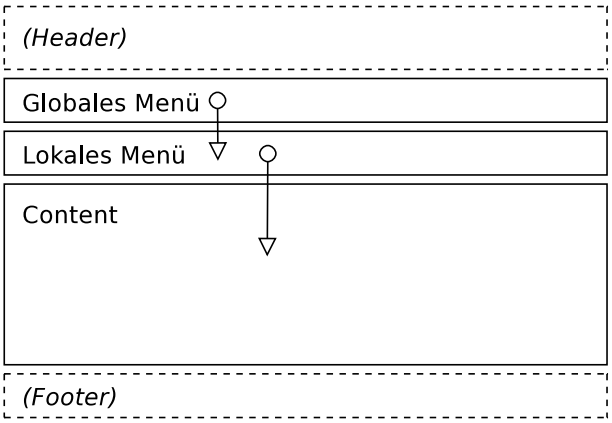


Abbildung 5.7: Hierarchische Gliederung der Navigation

5.3.2 Aufgaben bearbeiten

Solange der Benutzer an der Lösung einer Aufgabe arbeitet, wird nur eine einzelne Seite benötigt, nämlich die Detailseite dieser Aufgabe. Der Benutzer wird aber zwischendurch auf die Beschreibung der Datenbank oder auf die Darstellung des Relationenmodells wechseln müssen, was auch der Grund ist, warum diese in einem eigenen Fester geöffnet werden können (siehe unten). Ansonsten findet er alles, was er zum Bearbeiten einer Aufgabe braucht, auf der Detailseite der Aufgabe: das sind Angabe, Eingabefeld, Feedbackmeldungen und Ergebnistabelle. Ein Beispiel für eine solche Seite ist in Abbildung 5.8 dargestellt.

Die beiden Komponenten Feedback und Ergebnistabelle werden in den folgenden beiden Abschnitten kurz beschrieben.

Feedback

Um dem Benutzer mitzuteilen, ob seine Eingabe in Ordnung ist oder nicht, ist der Bereich des Feedbacks vorgesehen. Man kann sich das so vorstellen, dass hier die Fehlermeldung der SQL Abfrage ausgegeben wird. Dies ist aber nur ein kleiner Teil dieses Feedbacksystems.

Der Benutzer bekommt hier Feedback aus einer qualitativen Bewertung seiner Eingabe. So wird hier beispielsweise bestätigt, dass alle Tabellen in der Eingabe korrekt vorkommen (oder eben nicht), oder aufgezeigt, dass ein erwartetes `HAVING` oder `GROUP BY` fehlt.

Die einzelnen Meldungen, die im Feedbackbereich angezeigt werden, sind entweder positiv oder negativ. Es wird jede Meldung mit einem Icon versehen, das diese Eigenschaft verdeutlicht. Falls die Meldungen einem Ergebnis aus dem Tabellenvergleich entsprechen (siehe unten), also z.B. zuviele Spalten im Ergebnis vorhanden sind, wird die Meldung auch ident zu den entsprechenden Zellen in der Ergebnistabelle eingefärbt.

Die Inhalte der Feedbackkomponente werden im Bewertungssystem ermittelt, das in einem eigenen Dokument beschrieben wird. Dort kann mehr über die Art und die Inhalte des Userfeedbacks nachgelesen werden.

Die Auswertung einer Benutzereingabe liefert einen Status der Aufgabe. Die Aufgabe kann demnach z.B. korrekt gelöst, die Eingabe syntaktisch nicht in Ordnung, oder das Ergebnis mit

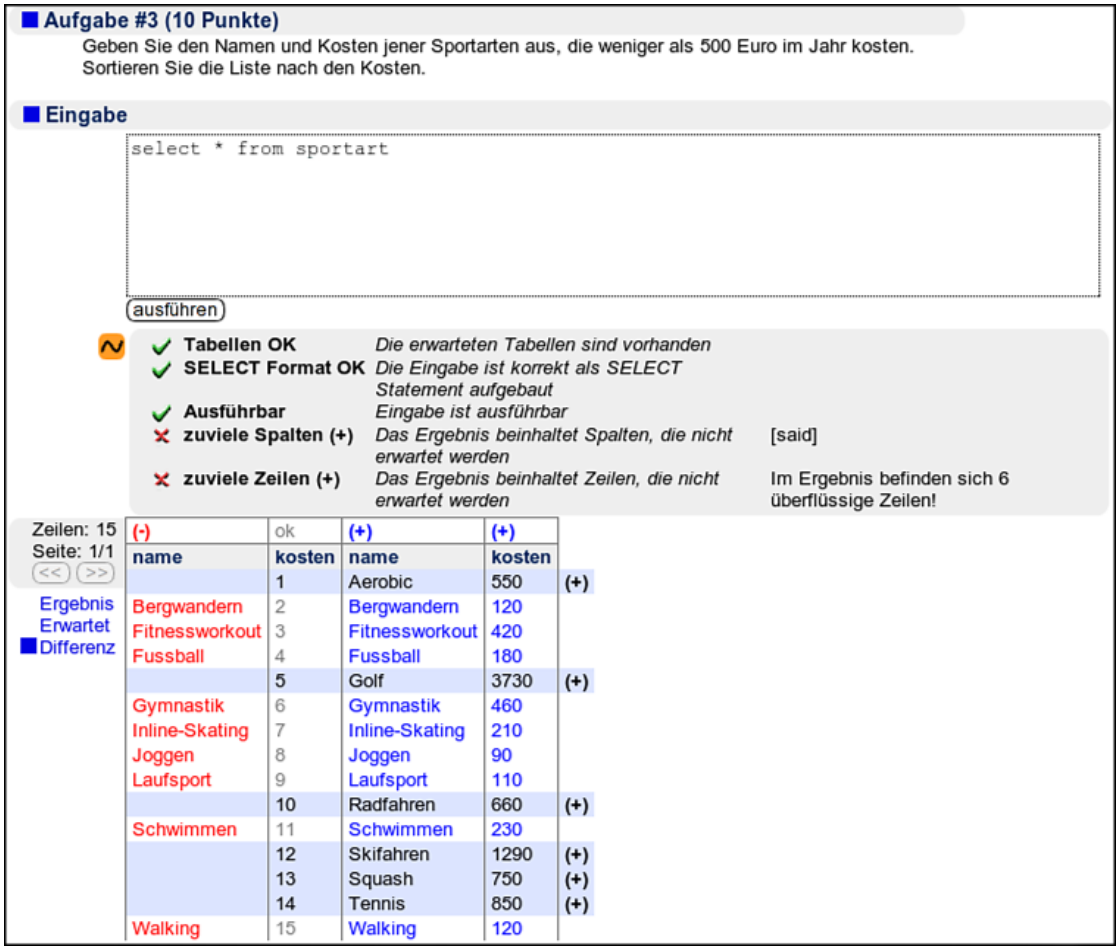


Abbildung 5.8: Screenshot einer Aufgabendetailseite mit Aufgabenstellung, Eingabe, Feedback und Ergebnistabelle

dem erwarteten nicht übereinstimmend sein. Jeder mögliche Status einer Aufgabe wird durch ein Statusicon repräsentiert, das einerseits in der Navigationsleiste der Aufgaben aufscheint (siehe Abschnitt 5.3.3) und andererseits auch hier im Feedbackbereich angezeigt wird.

Ergebnistabelle

Hier wird das Ergebnis der Abfrage als Tabelle, sofern die Abfrage syntaktisch korrekt ist und ein Ergebnis liefert, dargestellt.

Der Benutzer hat die Wahl zwischen 3 verschiedenen Tabellen, die hier angezeigt werden können:

- Die Tabelle der Ergebnisse der eigenen Eingabe (Menüpunkt `Ergebnis`). Es wird als Tabelle das Ergebnis angezeigt, das die Abfrage der Benutzereingabe liefert.
- Die Tabelle des erwarteten Ergebnisses (Menüpunkt `Erwartet`). Es wird eine Tabelle angezeigt, welche die Daten beinhaltet, die als korrektes Ergebnis erwartet werden.
- Eine Differenztabelle (Menüpunkt `Differenz`). In dieser Option wird eine Mischung aus der erwarteten und der Ergebnistabelle angezeigt, wobei alle Spalten und Zeilen aus beiden Tabellen aufscheinen. Übereinstimmende Spalten und übereinstimmende Zeilen werden nur einmal angezeigt. Die abweichenden Zeilen und Spalten werden farblich sowie mit einer Markierung am Rand der Tabelle gekennzeichnet. So werden beispielsweise Zeilen, die im Ergebnis des Benutzers vorkommen, aber in der Referenztabelle nicht erwartet werden, rot hinterlegt. Spalten, die im Ergebnis aufscheinen, aber keine Entsprechung im Referenzergebnis finden, werden mit roter Farbe dargestellt.

  Durch diese Darstellung kann der Benutzer sich schnell ein Bild davon machen, was an seiner Eingabe noch falsch sein könnte. Die abweichenden Zeilen und Spalten werden auch im Feedbacksystem eigens angeführt (siehe oben).

Aus technischen Gründen wird das Ergebnis der Abfrage auf eine konfigurierbare Anzahl maximaler Zeilen beschränkt. Wenn diese Zeilenbeschränkung eintritt (d.h. wenn der Benutzer tatsächlich mehr als z.B. 1000 Zeilen mit seiner Eingabe selektiert), wird der Benutzer darauf hingewiesen. Diese Beschränkung geht von der Annahme aus, dass die Aufgaben im Regelfall nicht mehr als eine Größenordnung von 100 Ergebniszeilen verlangen.

5.3.3 Aufgabennavigation

Als „Aufgabennavigation“ wird der Bereich im lokalen Menü bezeichnet, in dem die Statusicons der einzelnen Aufgaben für einen Kurs nebeneinander aufgelistet werden. Siehe Abbildung 5.9.

Der erste Ansatz zur Umsetzung der einzelnen Aufgaben wäre eine Überblicksseite für die Aufgaben, von der aus man auf die Detailseite für eine Aufgabe gelangen kann, gewesen. Vor- und Zurückbuttons auf der Detailseite hätten das wechseln zwischen den Aufgaben vereinfacht.

Nach weiteren Überlegungen hat sich aber bald ergeben, dass man dadurch entweder immer wieder auf die Überblicksseite zurückspringt, oder sehr schnell die Übersicht verliert. Daraus wurde die Idee geboren, eine Variante des Fokus & Kontext Konzepts aus der Visualisierung hier umzusetzen.

Abbildung 5.9: Icons in der „Aufgabennavigation“

In der Aufgabennavigation wird jede Aufgabe als einzelner Icon dargestellt. Der Status einer Aufgabe bestimmt den Inhalt des Icons. So kann man einen guten Überblick behalten, wieviele Aufgaben man bereits gelöst, bearbeitet etc., hat. Gleichzeitig wird die Aufgabe hervorgehoben, die im Moment bearbeitet wird. Dadurch sieht man auf einen Blick, wo man sich gerade befindet.

Was in den Icons dargestellt wird, ist der Status der einzelnen Aufgaben. Das kann konkret Folgendes sein:

1. *Keine Eingabe.* Hier wurde noch keine Eingabe gemacht. Die Aufgabe wurde noch nicht bewertet.

2. *Syntax Error.* Die Eingabe des Benutzers führt zu einem Syntaxfehler in der Datenbank. Es wird bei dieser Aufgabe keine Ergebnistabelle angezeigt.

3. *Ergebnisse nicht vergleichbar.* Dieser Status tritt ein, wenn die Eingabe des Benutzers zwar ausführbar ist, aber das Ergebnis so stark vom erwarteten Ergebnis abweicht, dass es keine übereinstimmenden Spalten gibt. In diesem Fall kann keine Differenztabelle angezeigt werden. Weitere Auswertungen (z.B. Ermittlung der fehlenden oder überflüssigen Spalten) werden ebenfalls nicht durchgeführt.

4. *Unterschiedliche Ergebnisse.* Dieser Zustand ist der normale Zustand, wenn der Benutzer eine gültige Abfrage eingegeben hat, aber das selektierte Ergebnis vom erwarteten Ergebnis abweicht. Das Bewertungssystem wird in diesem Fall nicht die ganze Punktezahl vergeben.

5. *Korrekt aber nicht erwartet.* Dieser Zustand ist ein Sonderfall, der aber durchaus oft auftreten kann. Er trifft genau dann zu, wenn der Ergebnisvergleich liefert, dass die beiden Ergebnisse exakt übereinstimmen (d.h. die Abfrage des Benutzers liefert dasselbe Ergebnis wie die Referenzabfragen), aber die qualitative Bewertung der Benutzereingabe feststellt, dass die Eingabe nicht in Ordnung ist.

   Dieser Fall kann zwei Ursachen haben: entweder der Benutzer hat eine Variante gefunden, die in den Referenzabfragen nicht berücksichtigt wurde, die aber völlig korrekt ist und auch so zu bewerten ist, oder der Benutzer „schummelt“, indem er für Teile der Abfrage Konstanten einsetzt, die er zuvor ermittelt oder einfach geraten hat.

   Der erste Fall muss als korrekt bewertet werden, der zweite Fall nicht. Leider sind die beiden Fälle vom System nicht (einfach) automatisch unterscheidbar, weshalb hier nur ein einziger Status vergeben wird.

6. *Das Ergebnis ist korrekt.* Dieser erstrebenswerte Zustand besagt, dass die Benutzereingabe dasselbe Ergebnis liefert, wie die Referenzabfragen und die Eingabe qualitativ einer der Referenzabfragen entspricht. In diesem Fall wird garantiert die volle Punktezahl vergeben.

Zusätzlich zum Status einer Aufgabe können (in einer Erweiterung) die Icons in der Aufgabennavigation auch noch mit unterschiedlicher Helligkeit versehen werden. Die Helligkeit würde dem Schwierigkeitsgrad der Aufgabe entsprechen (bzw. der Höhe der erreichbaren Punkte dieser Aufgabe).

Diese Option ist derzeit nicht umgesetzt, kann aber im endgültigen System noch zum Einsatz kommen.

Aufgabenübersicht

Trotz der Darstellung der einzelnen Aufgaben als Icons in der Aufgabennavigation bleibt die Aufgabenübersicht als eigene Seite erhalten (lokales Menü: `Aufgaben`). Dort wird zusätzlich Funktionalität in Form von Buttons untergebracht, welche die Gesamtheit der Aufgaben betreffen: *Aufgaben drucken, Upload & Download von Lösungen, Abgeben der Aufgaben*.

Der Benutzer bekommt, dadurch dass auf dieser Seite alle eingegebenen Lösungen des Benutzers gesammelt angezeigt werden, einen besseren Gesamtüberblick. Die Statusicons der Aufgaben werden auch hier nochmal dargestellt.

Diese Seite stellt zusammenfassend alles dar, was der Benutzer im Rahmen eines Kurses (oder einer Prüfung) bearbeitet hat.

5.3.4 Bedienungshilfen

Die im Folgenden beschriebenen Funktionen sind Hilfen für den Benutzer, welche die Bedienung des Systems vereinfachen sollen. Das System wäre auch ohne diese Hilfen vollständig verwendbar.

Javascript Helpers

Eine Reihe von kleinen Javascript Scripts sollen die Bedienung in den Bereichen, in denen das System am meisten verwendet wird (Überblick verschaffen über die Datenbank und Aufgaben lösen), erleichtern.

Für den ersten Fall - Überblick verschaffen (erreichbar im lokalen Menü über `Datenbank`) - wird eine Liste aller Tabellen zur Verfügung gestellt, aus der der Benutzer auswählen kann. Aus der gewählten Tabelle wird der Inhalt selektiert und angezeigt. Auf derselben Seite können auch beliebige Datenbankabfragen gemacht werden. Die Anzeige des Tabelleninhalts schreibt eine einfache `select * from <table>` Abfrage in den Eingabebereich und zeigt die Ergebnistabelle an.

In der Aufgabendetailseite sorgt ein kleiner Helfer dafür, dass bei Anzeige der Tabelle das Eingabefeld für die Abfrage sofort im Fokus ist. D.h. der Benutzer spart sich einen Mausklick bevor er die Abfrage eingeben kann. Weiters hat der Benutzer die Möglichkeit, seine Eingabe mit der Tastenkombination `STRG-Enter` direkt auszuführen, ohne mit der Maus auf den `Ausführen` Button klicken zu müssen.

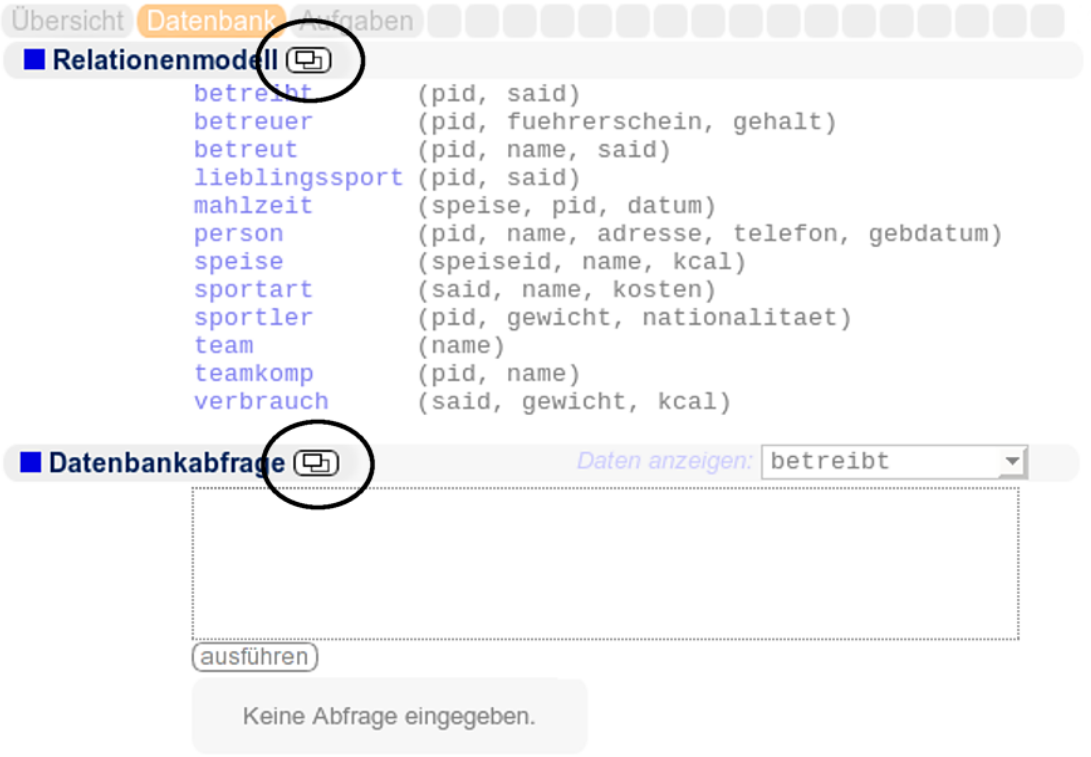


Abbildung 5.10: Komponenten als Popup Fenster öffnen

Popup Seiten

Die Komponenten zum „Browsen“ der Datenbank (lokaler Menüpunkt `Datenbank`) können unabhängig voneinander als Popupfenster geöffnet werden. Dies soll es dem User ermöglichen, eine Aufgabe zu bearbeiten und dazwischen schnell ins andere Fenster zu wechseln, um sich einen Überblick über die Datenbank zu machen.

Außerdem kann die Übersichtsseite eines Kurses, in der sich die Beschreibung der Datenbank befindet, in einem eigenen Fenster oder Browser-Tab geöffnet werden. Auch das kann dazu verwendet werden, zwischen Aufgabe und Übersicht zu wechseln.

Druckansicht

Für die Übersicht eines Kurses sowie für die Übersicht der Aufgaben (zusammen mit den eingegebenen Lösungen) können eigene Seiten angezeigt werden, auf denen die Information für den Drucker aufbereitet ist.

Hilfe

Es gibt eine globale Hilfeseite, in der die Navigation des Systems erläutert wird. Kontextsensitive Hilfe im breiteren Ausmaß ist nicht für die Bedienung des Systems nötig. Im kleineren

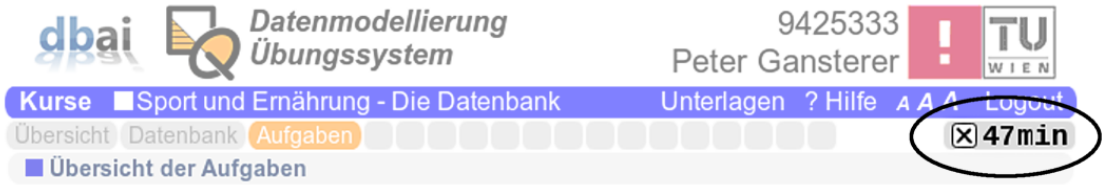


Abbildung 5.11: Anzeige der Prüfungszeit

Rahmen wird für alle Bedienungselemente, und manchmal auch Abkürzungen, ein Tooltip angezeigt, sobald der Benutzer mit der Maus über das Element fährt.

Zeitanzeige

Im Prüfungsmodus wird die verbleibende Prüfungszeit rechts oben angezeigt. Diese Anzeige kann auch vom Benutzer direkt deaktiviert werden. Dies ist sinnvoll für Studenten, die von einer tickenden Uhr von der Aufgabe abgelenkt werden.

Upload & Download

Im Übungsmodus (definitiv nicht im Prüfungsmodus) hat der Benutzer die Möglichkeit, die Lösungen der Aufgaben downzuloaden und in einem File abzuspeichern. Diese Lösungen können jederzeit wieder ins System upgeloadet werden, um an derselben Stelle weiterarbeiten zu können. Das gibt dem Benutzer die Möglichkeit, die Bearbeitung eines Kurses in mehreren Etappen anzugehen, bevor am Ende eine Auswahl von Aufgaben abgegeben wird. Andererseits erspart das (aus technischer Sicht) das Speichern und Verwalten von Usersessions.

5.3.5 Usability Tests

Usability Tests des Systems wurden in verschiedenen Phasen der Entwicklung durchgeführt. Beginnend mit den ersten HTML Page Designs bis zu funktionierenden Prototypen.

Bis zum produktiven Einsatz des Systems sind noch weitere Usability Tests vorgesehen.

In [Krug00] ist eine formalisierte Vorgehensweise für Usability Tests beschrieben, die als Vorlage für die hier durchgeführten Tests dienen soll, aber nicht vollständig angewendet wird. Der Grund liegt darin, dass zum einen im vorliegenden Fall nur eine einzelne Person am Userinterface arbeitet und zum anderen die Bedienung des Systems nicht sonderlich komplex ist. Entsprechend wurden die Tests zuerst informell (die Entwürfe wurden präsentiert und Feedback eingeholt) und später anhand eines funktionierenden Prototypen in einzelnen Testsessions mit verschiedenen Personen durchgeführt.

In den Testsessions sollten die Tester verschiedene Aufgaben erledigen um feststellen zu können, wie sie mit dem System zurechtkommen bzw. wo Probleme auftreten. Einige typische Aufgaben („key tasks“), die in den Tests gestellt werden, sind hier aufgelistet:

- Herausfinden, worum es auf der Seite (bei dem System) prinzipiell geht. Dieser Test kann natürlich nur einmal mit einer Person durchgeführt werden, die noch nicht über das System bescheid weiß.

- Den Inhalt einer bestimmten Datenbanktabelle ermitteln.
- Das Feedback einer Eingabe lesen und interpretieren.
- Eine Kursaufgabe lösen.
- Absolvieren einer gesamten Beispielprüfung.
- Abgeben von Übungsaufgaben.

Aufgrund von Unterschieden im Vorwissen konnten nicht alle Aufgaben mit allen Personen durchgeführt werden.

Die verschiedenen Erkenntnisse, die bisher aus Usability Tests gewonnen wurden, werden hier der Vollständigkeit halber angeführt (ohne spezielle Reihung oder Gruppierung):

- Für das Zurechtfinden im System ist entweder eine Startseite, die noch vor der Kursauswahl kommt, nötig. Alternativ könnte, auf der Seite der Kursauswahl, eine Einleitung stehen. Aufgrund der geplanten Integration mit dem LVA System wurde auf eine solche Startseite zuvor verzichtet.
- Bei der Darstellung der Ergebnistabelle auf der Aufgabenseite sollte nur die Differenztabelle, nicht aber die Ergebnis- und die Referenztabelle, eine Färbung der abweichenden Inhalte aufweisen.
- Die Größe des Eingabefelds für die Lösung der Aufgaben sollte zwischen 7 und 10 Zeilen sichtbaren Text ermöglichen. Längere Eingaben sind die Ausnahme und sollten nicht unnötig Platz auf der Seite verbrauchen.
- Die Angabe einer Aufgabe besteht nur aus dem Text und soll keinen eigenen Titel aufweisen.
- Die Anordnung des `Ausführen` Buttons links vom Eingabefeld erwies sich als verwirrend.
- Die Anzeige der Ergebnistabelle in mehreren Seiten ist nicht von Vorteil, wenn auf der Seite ohnehin gescrollt werden muss. Außerdem ist es der Übersicht dienlich, wenn das gesamte Ergebnis ausgegeben wird, zumal dieses sowieso im Umfang beschränkt sein wird.
- Im Feedback sollen Icons angeben, ob eine Meldung als positiv oder negativ zu werten ist.
- Die Farbgebung der Unterschiede in der Differenztabelle soll sich auch in den Feedbackmeldungen wiederfinden.

5.4 Systembeschreibung

In diesem Abschnitt wird das gesamte System im Überblick, mit Beschreibungen der Systemnavigation und der einzelnen Seiten, gezeigt. Ein Großteil der hier erwähnten Aspekte wurde

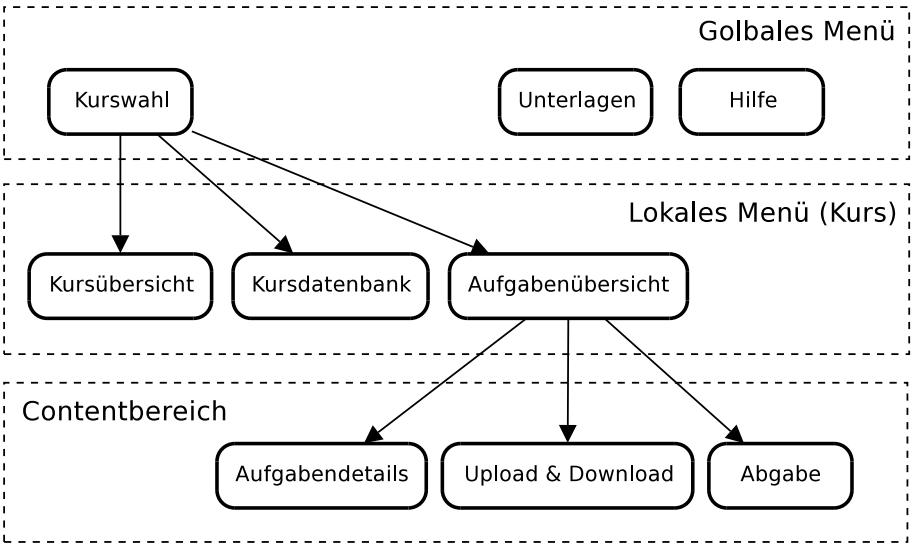


Abbildung 5.12: Navigationsübersicht

bereits in vorangegangenen Abschnitten detailliert beschrieben. Die kommenden Seiten sind demnach als Zusammenfassung zu sehen.

In Abbildung 5.12 wird ein Überblick über die vorhandenen HTML Seiten und die Navigationsmöglichkeiten gezeigt. Die in Abschnitt 5.3.1 angesprochene Hierarchie der Navigation ist hier wieder erkennbar. Die einzelnen Seiten in der Grafik werden in den folgenden Abschnitten beschrieben.

5.4.1 Einstiegsseite und Kurswahl

Diese Einstiegsseite dient einerseits der Zusammenfassung des Inhalts des Systems - der User soll über diese Seite informiert werden, worum es in dem System geht - andererseits wird dem Benutzer eine Liste der Kurse mit kurzer Beschreibung präsentiert. Der Benutzer hat die Möglichkeit, einen Kurs zur Bearbeitung auszuwählen. Dabei kann er einen Kurs als Übung (d.h. die vorgegebenen Übungsbeispiele lösen), oder „im Prüfungsmodus“ bearbeiten. In diesem Prüfungsmodus des Übungsmoduls erscheint das System so wie eine Prüfung im Prüfungsmodul. Der Unterschied ist, dass die „Prüfung“ nicht abgegeben und bewertet werden kann. Man kann sich dadurch ein Bild davon machen, wie die tatsächliche Prüfung aussehen wird.

Nach der Auswahl eines Kurses wird man auf die Übersichtsseite desselben weitergeleitet. Innerhalb des Kurses (im lokalen Menü) hat man die Möglichkeit, zwischen verschiedenen Seiten des Kurses (*Übersicht, Datenbank, Aufgaben* oder direkt auf eine *Aufgabendetailseite* (letzteres über die *Aufgabennavigation*, zu wechseln (siehe Abschnitt 5.3.3).

Ein Beispiel für die Einstiegsseite mit Kursauswahl ist in Abbildung 5.13 dargestellt. Bei jedem Kurs ist die Auswahl „Übung“ und „Prüfung“ zu sehen. Mit dem Button „Prüfung“ kann der beschriebene Prüfungsmodus im Übungsmodul gestartet werden.

Auf diese Einstiegsseite wird man auch zurückgeführt, wenn man auf das Logo des e-Learning Systems klickt.

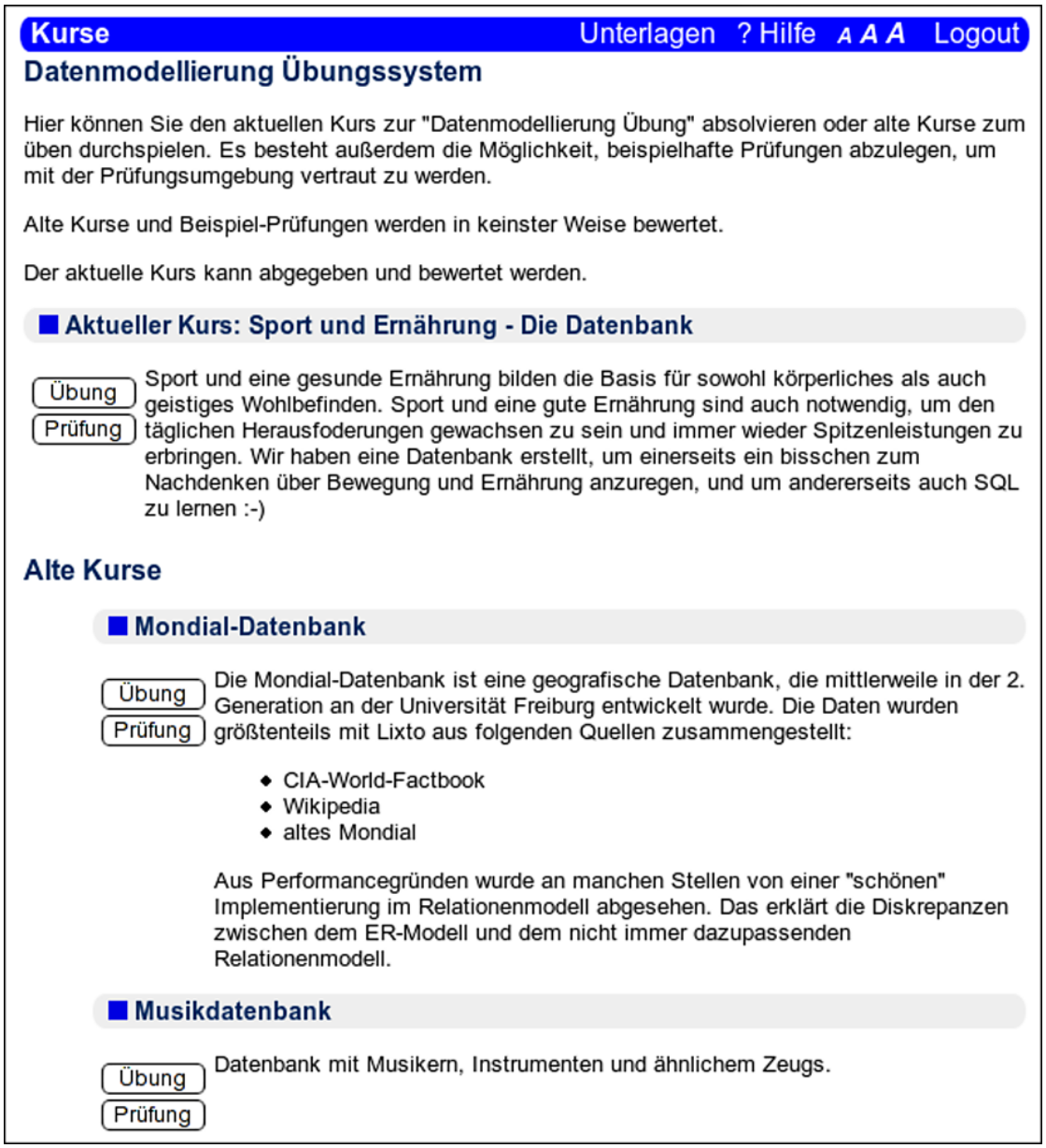


Abbildung 5.13: Einstiegsseite und Kursauswahl

5.4.2 Unterlagen

Auf dieser Seite werden dem Benutzer Unterlagen, die zum e-Learning System inhaltlich passen, oder für die Übung gebraucht werden, zur Verfügung gestellt. Das kann beispielsweise das Lehrveranstaltungsskriptum oder eine SQL Referenz sein.

Die Unterlagen-Seite ist als HTML File gestaltet, das jederzeit von einem Administrator angepasst werden kann.

5.4.3 Hilfe

Die Hilfe-Seite bietet dem Benutzer Unterstützung beim Zurechtfinden in der Applikation. Einerseits werden die wesentlichen Navigationselemente beschrieben, andererseits eine Struktur der Applikation.

Auch die Hilfe-Seite wird als HTML File umgesetzt, das jederzeit von einem Administrator angepasst werden kann.

5.4.4 Kursübersicht

Die Kursübersichtsseite (erreichbar unter `Übersicht` im lokalen Menü), gibt eine Beschreibung der Kursdatenbank und (im Normalfall) ein ER Diagramm.

Die Inhalte der Seite sind in Abschnitte unterteilt, die im Administrationsmodul bearbeitet werden können. Ein Abschnitt enthält entweder Text oder ein Bild (z.B. ein ER Diagramm).

Die Textabschnitte werden in der Administration als Plaintext eingegeben, der geringfügige Formatierungen erlaubt: Es können Absätze getrennt werden, Listenaufzählungen und Links eingefügt werden. Die Formatierungen richten sich nach dem Konzept von WIKI Seiten [5].

5.4.5 Kursdatenbank

Auf der Seite `Datenbank` kann man sich einen schnellen Überblick über die Relationen und Inhalte der Datenbank verschaffen. Siehe Abbildung 5.14 für eine beispielhafte Darstellung dieser Seite.

Es werden alle in der Datenbank vorhandenen Tabellen, zusammen mit den definierten Columns, aufgelistet. Bei Mausklick auf eine Tabelle bekommt der User eine Darstellung der Tabellendetails mit Columntypes, Primary Key und Foreign Key Definitionen, die direkt aus der vorhandenen Kursdatenbank ausgelesen werden.

Im zweiten Teil der Seite hat man die Möglichkeit, beliebige SQL Abfragen einzugeben und auszuführen. Das Ergebnis der Abfrage wird auf derselben Seite als Tabelle dargestellt.

Um möglichst schnell und einfach den Inhalt einer Tabelle anzeigen zu können, kann man auch direkt eine Tabelle im Dropdown-Feld wählen. Es wird dann der Inhalt der gewählten Tabelle im Ergebnisbereich angezeigt.

5.4.6 Aufgabenübersicht

Über den Menüpunkt `Aufgaben` im lokalen Menü gelangt man auf die Übersichtsseite der Aufgaben. Hier werden alle Aufgaben eines Kurses (entweder Übungsaufgaben oder Prüfungs-

[5] Die hier beschriebenen Textabschnitte stellen aber in keinem Fall ein WIKI System dar.
Siehe http://c2.com/cgi/wiki?WikiHistory für eine Entwicklungsgeschichte des WIKI Konzepts

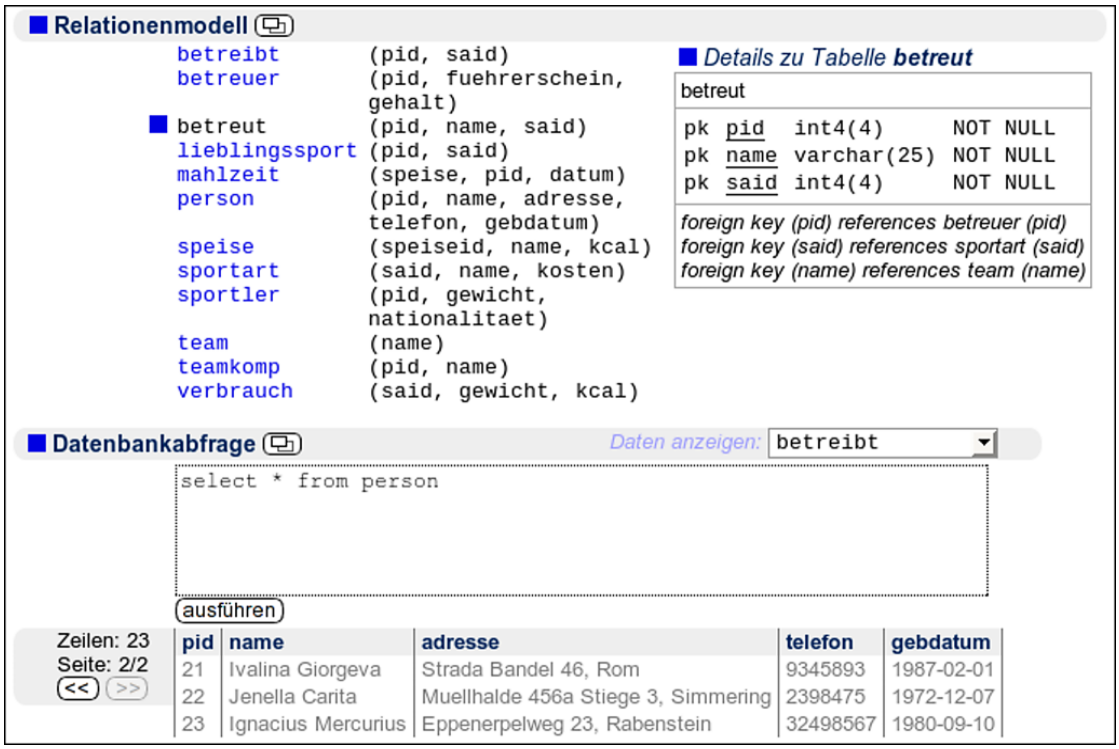


Abbildung 5.14: Kursdatenbank: Relationen und Abfrage

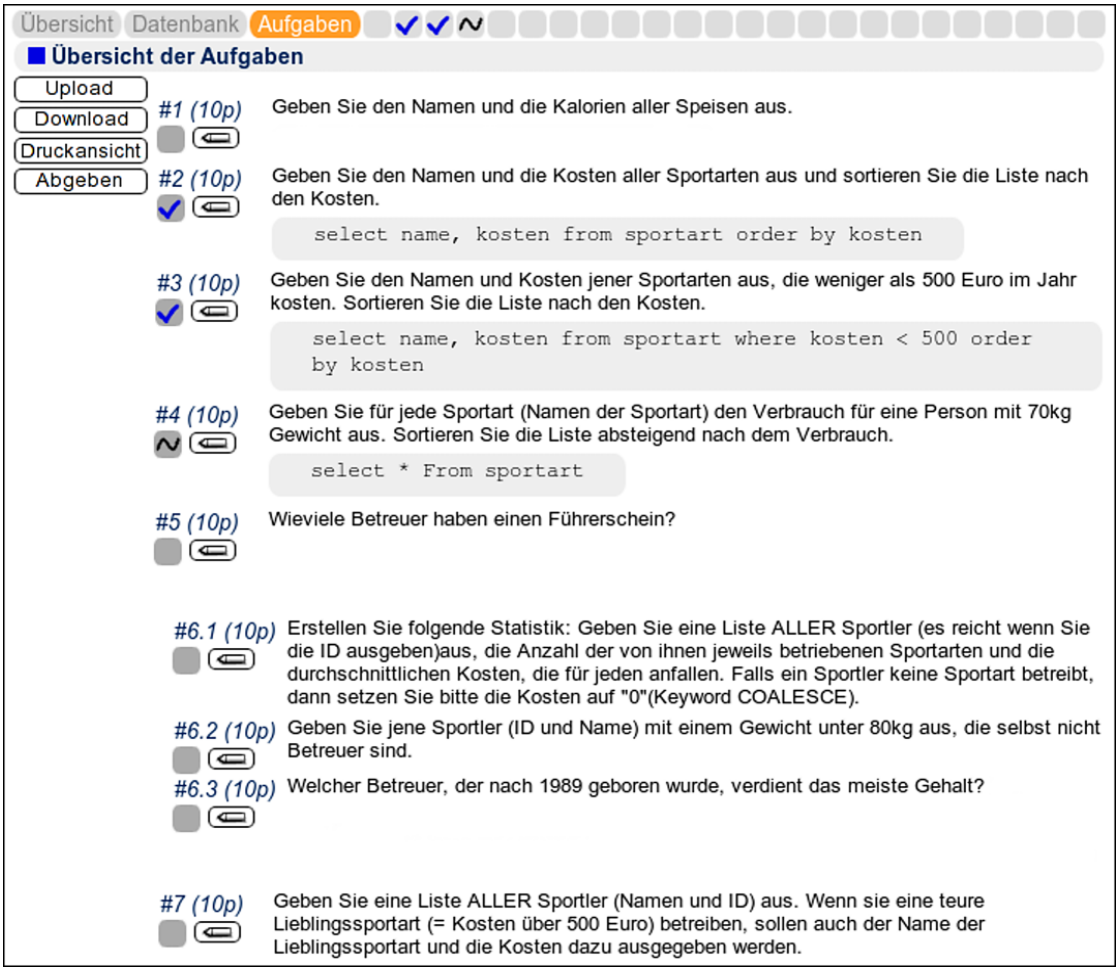


Abbildung 5.15: Aufgabenübersicht mit Aufgabengruppierung

aufgaben) aufgelistet.

Wenn zu einer Aufgabe bereits eine Eingabe gemacht wurde, dann wird diese hier ebenfalls angezeigt. Außerdem wird der Status der Aufgabe (die Korrektheit der Eingabe, siehe 5.3.3) als Icon und die Anzahl der erreichbaren Punkte für diese Aufgabe dargestellt.

Mehrere Aufgaben können (im Administrationsmodul) zu Gruppen zusammengefasst werden, wenn diese z.B. aufeinander aufbauen. Diese Gruppierung spiegelt sich hier in der Nummerierung der Aufgaben und auch in der Anordnung wider: durch unterschiedliche Abstände und Einrückungen werden gruppierte Aufgaben gekennzeichnet.

In Abbildung 5.15 ist ein Screenshot einer Aufgabenübersicht zu finden.

Von dieser Seite aus kommt man einerseits zur Detailseite der einzelnen Aufgaben, andererseits können von hier aus auch Aktionen gestartet werden, welche die Gesamtheit der Aufgaben betreffen. Das sind: Upload, Download und Abgabe. Mehr dazu in den weiteren Abschnitten.

5.4.7 Aufgabendetails

Auf der Aufgabendetailseite wird eine einzelne Aufgabe bearbeitet. Der Benutzer kann die Aufgabenstellung nachlesen, kann seine Lösungsversuche (SQL Queries) eingeben, und bekommt Feedback über die Richtigkeit der Eingabe.

Details zur Bearbeitung einer Aufgabe und entsprechendes Feedback wurden bereits in Abschnitt 5.3.2 beschrieben. In der dortigen Abbildung 5.8 ist eine Aufgabendetailseite mit Aufgabenstellung, Eingabe, Feedback und Ergebnistabelle dargestellt.

Zur Zeit steht noch nicht fest, ob auf der Seite auch Vor- und Zurückbuttons angezeigt werden sollen, die den Benutzer zur vorigen oder zur nächsten Aufgabe direkt weiterleiten. Die gewählte Form der Aufgabennavigation (siehe 5.3.3) macht das theoretisch unnötig. Diese Frage wird noch in Usability Tests erarbeitet werden.

Eine Erweiterungsoption auf dieser Seite wäre außerdem die Umsetzung einer Inputhistory, die es in irgendeiner Form ermöglicht die Eingaben des Benutzers zu speichern und wieder abrufbar zu machen.

5.4.8 Upload & Download

Über die Funktionen `Upload` und `Download` können die bisherigen Eingaben eines Kurses in ein XML File downgeloadet und später wieder upgeloadet werden.

Das ermöglicht dem Benutzer, die Arbeit auf mehrere Sitzungen aufzuteilen, ohne dabei ein komplexes System für die Verwaltung persistenter Sessions im System einführen zu müssen.

Die Möglichkeit des Aufgabendownloads und -uploads wird auch im Abschnitt 5.3.4 beschrieben.

5.4.9 Abgabe

Schließlich kann der Benutzer über den Menüpunkt `Abgeben` seine Lösungen abgeben. Dies kann entweder bei einer aktuellen Prüfung im Prüfungsmodul oder im Rahmen der aktuellen Übung gemacht werden. Alte Kurse oder Übungskurse im Prüfungsmodus können nicht abgegeben werden.

Bei der Abgabe wird immer der gesamte Kurs abgegeben. D.h. die Beispiele werden in dem Zustand gespeichert, in dem sie sich gerade befinden. Auch bei der Übung werden nicht einzelne Beispiele zur Abgabe ausgewählt, sondern es werden für die Übungsbewertung nur diejenigen Beispiele herangezogen, die gelöst wurden.

6 Resultset Vergleich

6.1 Motivation

Ein wesentlicher Bestandteil der Aufgabenbewertung im SQL e-Learning System ist der Resultset Vergleich. Hier geht es darum, dass das Ergebnis einer Query, die der Student als Lösung der Aufgabe eingibt, mit dem Ergebnis einer Referenzquery verglichen wird. Die Referenzquery ist vorgegeben als Teil der Aufgabendefinition.

Die Aufgabe wird nur dann als korrekt gelöst bewertet, wenn die beiden Ergebnisse (die wir hier im Sinne der JDBC Nomenklatur als *Resultset* bezeichnen), übereinstimmen[1].

Diese Art der Bewertung ist für SQL Eingaben die naheliegendste, vor allem auch wenn man bedenkt, dass es oft die unterschiedlichsten Wege gibt, wie man in SQL eine Abfrage formulieren kann, die zu demselben Ergebnis führt. Würde man sich nur auf die Vergleiche der eingegebenen Statements mit Referenzen beschränken, würde man voraussichtlich zuviele gültige und richtige Lösungen als falsch bewerten.

6.1.1 Gewünschte Ergebnisse

Es werden also die Queries auf der Kursdatenbank ausgeführt und deren Ergebnisse verglichen. Dieser Vergleich hat dabei zweierlei Aufgaben zu erfüllen:

1. Die Unterschiede der beiden Ergebnisse müssen festgestellt und in Zahlen gemessen werden. D.h. es ist zu ermitteln, wieviele Zeilen im Ergebnis zuviel oder zuwenig sind (wobei beide Werte größer 0 sein können), und wieviele und welche Spalten fehlen oder zuviel sind (kann ebenfalls beides zugleich vorkommen).

   Diese Ergebnisse fließen als Kriterien in das Bewertungssystem ein.

2. Die Differenzen müssen in einer Tabelle, die im Userinterface als Teil des Feedbacks dargestellt wird, aufbereitet werden. Die Tabelle ist so gestaltet, dass Zeilen, die zuviel oder zuwenig sind (im Vergleich zur Referenz) farblich markiert werden. Genauso werden unerwartete oder fehlende Spalten hervorgehoben.

Diese beiden Ergebnisse werden mit dem hier vorgestellten Konzept ermittelt. Auf die dabei auftretenden Probleme und deren Lösung wird im folgenden Abschnitt eingegangen.

[1] Genaugenommen kann es auch im Fall von übereinstimmenden Resultsets passieren, dass die Lösung nicht korrekt ist. Mehr dazu ist im Kapitel 8 nachzulesen

6.2 Konzept

6.2.1 Vorgehensweise beim Vergleich

Der Vergleich der beiden Resultsets arbeitet nach einem einfachen Prinzip: es werden beide nach denselben Kriterien sortiert und anschließend wird Zeile für Zeile verglichen.

Der Algorithmus dazu ist in Abbildung 6.1 dargestellt. Der Pseudocode sollte die Funktionsweise ausreichend verdeutlichen, darum soll hier nicht weiter darauf eingegangen werden.

```
Input : 2 gleich sortierte Listen inputList und refList
inpIdx ← 0;
refIdx ← 0;
inp ← inputList[inpIdx];
ref ← refList[refIdx];
while ¬(ref = nil ∧ inp = nil) do
    if ref = nil then
        unexpected(inp);
        incr(inpIdx);
    else if inp = nil then
        missing(ref);
        incr(refIdx);
    else if ref = inp then
        incr(refIdx);
        incr(inpIdx);
    else if ref > inp then
        unexpected(inp);
        incr(inpIdx);
    else
        missing(ref);
        incr(refIdx);
    // wir gehen davon aus, dass die Listen nil liefern,
    // wenn das Ende erreicht ist.
    inp ← inputList[inpIdx];
    ref ← refList[refIdx];
```

Abbildung 6.1: Algorithmus zum zeilenweisen Vergleich von Resultsets

Dieser Vergleichsalgorithmus selbst ist also kein weiteres Problem. Wenn man aber genauer hinsieht, ergibt sich ein Problem bei dem Vorbereitungsschritt: Sortieren der beiden Listen nach denselben Kriterien. Mit den Problemen, die dabei auftreten, wird sich dieser Abschnitt in Kürze beschäftigen, vorerst aber noch eine andere Anmerkung:

Der Resultset Vergleich wurde so implementiert, dass die Ergebnis-Resultsets vollständig in den Hauptspeicher geladen und dort sortiert und verglichen werden. Der Vergleich selbst könnte, wenn man den Vergleichsalgorithmus betrachtet, auch problemlos direkt von der Datenbank aus geschehen. Dazu müsste man nur die beiden SQL Statements so anpassen, dass sie gleich sortiert sind. Man könnte für beide Statements einen Cursor offen halten und so den Vergleich

ohne zusätzlichen Speicheraufwand durchführen.

Dieser Weg wurde nicht verfolgt, weil er einerseits den Nachteil hat, dass die Implementierung etwas komplexer ausfällt, da man beim Umgang mit den Cursor Ressourcen vorsichtiger sein muss. Außerdem war ursprünglich geplant, im Userinterface die Differenztabelle „gepaged“ aufzubauen, was bedeuten würde, dass man entweder die beiden Cursor solange offen halten muss, bis auf die Tabelle nicht mehr zugegriffen wird, oder man bei der Anzeige jeder Seite von neuem den gesamten Vergleich durchführen muss. Dieser Umstand macht die Implementierung noch um einen (unnötigen) Schwierigkeitsgrad anspruchsvoller.

Und schließlich kommt noch erschwerend hinzu, dass beide Ergebnisse möglicherweise ohnehin öfter vollständig abgearbeitet werden müssen, um zu einer sinnvollen Differenztabelle zu kommen. Wieso dem so ist, wird in Kürze ersichtlich sein.

Die Aufgabenstellungen in der Lehrveranstaltung mit Beispieltabellen in der Größenordnung von 100 Einträgen sind im Regelfall einfach genug, sodass der Vergleich im Hauptspeicher zu keinen Problemen führt.

Für größere Ergebnisse wurde ein Limit gesetzt, über das hinaus die Zeilen nicht gelesen werden. Dieses Limit führt natürlich dazu, dass der Vergleich eventuell falsche Ergebnisse liefert. Dieser Umstand stellt aber kein Problem dar, weil davon ausgegangen wird, dass die *korrekte* Lösung immer weit unter dem Limit liegt. Wird also durch eine Eingabe eines Studenten das Limit erreicht, dann ist diese Eingabe ohnehin „völlig falsch“.

Das Problem der richtigen Zuordnung

Nun zum Problem bei dem Vorbereitungsschritt des Sortierens: Um die beiden Resultsets gleich sortieren zu können, muss Klarheit darüber bestehen, welche *Spalten* in dem einen Resultset mit den Spalten im anderen Resultset gleichzusetzen sind. Diese Klarheit ist aber in einigen Fällen nicht gegeben.

Wenn man die Bezeichnungen der Spalten betrachtet, kann es einerseits passieren, dass die Bezeichnungen unterschiedlicher Spalten in zwei verschiedenen Tabellen dieselben sind. Eine Spalte *id* kann es sehr schnell in mehreren Tabellen geben. Werden diese zwei Tabellen mit einem Join zusammengeführt, dann kommen beide *id* Spalten mit demselben Namen im Resultset vor. Allein durch den Namen können diese beiden Spalten jetzt nicht mehr eindeutig zugeordnet werden.

Wir gehen von einer Implementierung auf Basis von JDBC aus. Hier besteht zusätzlich die Möglichkeit, den Table Namen zu einer Column auszulesen (sofern das für den Fall anwendbar, und die Column nicht aus einer Operation aus mehreren Columns zusammengesetzt ist). Die Information des Table Namens kann mir in einem Fall helfen in einem anderen Fall wiederum kann es störend sein. Helfen würde es in dem eben beschriebenen Fall, wo in beiden Resultsets zwei Spalten mit Namen *id* vorkommen. Durch Auslesen des Table Namens kann jetzt eine eindeutige Zuordnung getroffen werden.

Wird im selben Fall allerdings der Join über die *id* durchgeführt, und ins Ergebnis ebenfalls eine *id* aufgenommen, so hat man die Möglichkeit, die *id* Spalte aus einer der beiden Tabellen zu wählen. Wird im Referenzstatement die eine Tabelle gewählt und im Inputstatement die andere, dann ist es falsch, auf die Tabelleninfo zurückzugreifen.

Viel deutlicher wird das Problem aber dann, wenn man beispielsweise Aggregate betrachtet: wird in einer Query sowohl `AVG(col1)` als auch `AVG(col2)` selektiert, ist die Namensvergabe für die Spalten teils vom JDBC Driver und teils vom Datenbanksystem selbst abhängig.

In jedem Fall kann über die Zuordnung keine eindeutige Aussage mehr getroffen werden.

Was auch immer man betrachtet, man stößt immer wieder auf Fälle, wo die Zuordnung der Spalten nicht eindeutig getroffen werden kann. Der optimale Ansatz wäre also, einfach alle möglichen Zuordnungen zu probieren und aufgrund eines „Übereinstimmungsindikators“ jeder Zuordnung sich für die beste zu entscheiden. Dieser optimale Ansatz wird im folgenden Abschnitt behandelt.

6.2.2 Permutationen der Resultset-Spalten

Das Problem des Findens der besten Spaltenzuordnung soll hier formalisiert werden. Die grundlegende Vorgehensweise dabei ist:

1. Permutationen von Spaltenzuordnungen bilden,
2. jede Permutation bewerten
3. und am Ende jene Permutation mit der besten Bewertung auswählen.

Das Problem reduziert sich dadurch auf das Beschreiben einer möglichst aussagekräftigen Bewertung einer einzelnen Permutation.

Doch zunächst zu den Permutationen selbst. Sei n die Anzahl der Spalten im Referenz-Resultset und m die Anzahl der Spalten im Input-Resultset, dann ist die Spaltenpermutation p eine Funktion $p : \{1,..,n\} \rightarrow \{0,1,..,m\}$, die jedem Spaltenindex i einer Referenzzeile einen Spaltenindex $p(i)$ in einer Inputzeile zuordnet. Die beiden Matrizen

$$(inp_{i,j})...1 \leq i \leq k, 1 \leq j \leq m$$

und

$$(ref_{i,j})...1 \leq i \leq l, 1 \leq j \leq n$$

sollen die Resultsets darstellen. Dabei sind k und l die Anzahl der Zeilen im Input- (k) und Referenz-(l) Resultset.

Die Permutation $p(i)$ bildet auch auf den Index 0 ab. Eine Zuordnung $p(i) = 0$ bedeutet in dem Fall, dass die entsprechende Referenzspalte *nicht* zugeordnet wurde. Umgekehrt muss auch nicht jede Inputspalte in der Permutation enthalten sein. Abbildung 6.2 zeigt ein Beispiel für eine Spaltenzuordnung. In der gezeigten Permutation ist $p(3) = p(4) = 0$, d.h. nicht zugeordnet und $3 \notin \{p(i)\}$.

Um mit dieser Permutationsfunktion die permutierte Inputzeile $inp_{i,p(j)}$ vollständig darstellen zu können, wird $in_{0,j} := nil$ definiert. Wobei dieses *nil nicht* das `NULL` eines Datenbankwerts sein soll.

Darauf aufbauend kann eine Vergleichsrelation $eq(x,y)$ definiert werden mit

$$eq(x,y) := \begin{cases} 1 & x = nil \vee y = nil \\ 1 & x = y \\ 0 & sonst \end{cases}$$

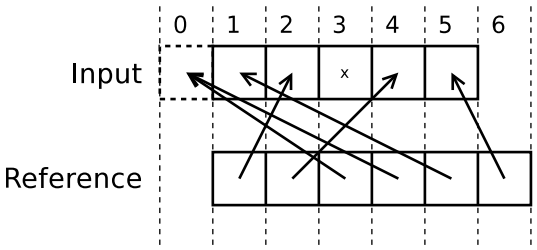


Abbildung 6.2: Beispiel einer Spaltenzuordnung

Mithilfe dieser Relation ergibt nun

$$match_p(inp_i, ref_i) := \prod_{j=1}^{n} eq(inp_{i,p(j)}, ref_{i,j})$$

eine Vergleichsfunktion für die beiden Zeilen inp_i und ref_i, unter Verwendung der Zuordnungspermutation p, die 1 ergibt, wenn die beiden Zeilen gleich sind und 0 wenn nicht.

Wir definieren damit

$$MatchValue(p) := \sum_{i=1}^{k} \prod_{j=1}^{l} match_p(inp_i, ref_j)$$

als einen Indikator für die Permutation p, welcher die Summe aller übereinstimmenden Zeilen in Input und Referenz unabhängig von der Sortierreihenfolge und unter Berücksichtigung der Permutation p darstellt.

Mit $MatchValue(p)$ wird der gewünschte „Übereinstimmungsindikator“ definiert, der zu maximieren ist. Gesucht ist also die Permutation p mit $max(MatchValue)$.

Auf der anderen Seite wird deutlich, dass diese Vorgehensweise sehr rechenaufwändig ist. Zum einen ist allein die Berechnung des *MatchValue* mit hohem Aufwand verbunden, weil dabei jedesmal das Referenz Resultset vollständig mit dem Input Resultset verglichen werden muss. Zum anderen steigt die Anzahl der möglichen Permutationen mit der Größenordnung $n!$ rasant an.[2]

Zum Glück sieht es in der Praxis nicht so dramatisch aus, denn, wie zuvor bereits beschrieben, lassen sich die meisten Zuordnungen aufgrund der vorhandenen Informationen mit hoher Sicherheit bereits vorab treffen.

6.2.3 Einschränkung des Aufwands

Um also den Aufwand beim Ermitteln der besten Spaltenzuordnung gering zu halten werden die Informationen genutzt, die vorab vorhanden sind. Das ist konkret Folgendes:

[2] Die tatsächliche Anzahl der möglichen Zuordnungen ist unter Berücksichtigung der möglichen 0-Zuordnungen: $\sum_{i=1}^{n} m!/(m-i)!$ wenn $m > n$. Wobei m und n die Anzahl der Spalten in (inp) und (ref) sind.

- Die Spalten werden nach Typklassen gruppiert. Es wird die Information des Datentyps in JDBC verwendet, um eine Spalte einer Typklasse zuzuordnen. Als Typklassen wird eine Zusammenfassung von Typen bezeichnet, die untereinander vergleichbar sind. Das sind in der konkreten Implementierung des Prototyps: INT, DOUBLE, CHAR, DATE, BOOL, OTHER. Weitere Klassen wurden bisher in den Tests nicht benötigt.

  Die Zuordnung der Spalten wird nach der Einteilung in Klassen nur noch innerhalb der einzelnen Klassen gemacht.

- Wenn durch den Spaltennamen die Zuordnung eindeutig getroffen werden kann (bei passendem Spaltentyp), dann wird diese Zuordnung fixiert.

- Für den Fall, dass Namen mehrfach vorkommen, wird versucht anhand des Tabellennamens eine eindeutige Zuordnung zu treffen.

Wenn nach Nutzung aller Informationen die Anzahl der nicht fixierten Spalten > 0 ist, wird mit dem Testen der Permutationen begonnen. Auch bei einer einzelnen verbleibenden Spalte, die nicht eindeutig zugeordnet werden konnte muss getestet werden, da es ja möglich wäre, dass diese Spalte nur in einem der beiden Resultsets vorkommt und damit gar nicht zuzuordnen ist.

In der Praxis zeigt sich allerdings, dass (vor allem bei der Art der gegebenen Angaben) der Fall, dass tatsächlich Zuordnungen zu finden sind, die nicht von vornherein fixiert wurden, nur noch selten eintritt. Was aber dennoch immer wieder vorkommt, sind die Fälle, in denen unterschiedliche Spalten im Input und im Result enthalten sind, die nicht zuzuordnen sind.

Weitere Verbesserung in der Performance kann erreicht werden, wenn vom Input Resultset nur ein limitiertes Subset der Zeilen verwendet wird um die Permutationstests durchzuführen. Dazu wird eine Zufallsauswahl der Zeilen verwendet, oder einfach die ersten n Zeilen herangezogen. Wichtig ist, dass der gesamte Test bis zur gefundenen Zuordnung mit demselben Subset von Zeilen gemacht wird.

Eine Optimierungsvariante, die nicht getestet wurde, wäre, die zu testenden Permutationen nach „Spaltenheuristiken“ zu sortieren. Solche Heuristiken könnten den Wertebereich der Spalten untersuchen. Man könnte beispielsweise die Unterschiede der Mittelwerte der Spalten verwenden, um zuerst Permutationen zu testen, in denen die Mittelwertdifferenzen minimal sind.

Zu guter Letzt kann „Ausreißern“ immer noch vorgesorgt werden, indem nach einer fixen Anzahl Permutationen abgebrochen wird. D.h. wenn zuviele Unsicherheiten bestehen, dann ist die Wahrscheinlichkeit auch hoch, dass die Ergebnisse ohnehin nicht übereinstimmen. In dem Fall ist wichtig, dass das System nicht aufgehalten wird durch ein „dummes“ Statement eines Studenten (schließlich kann es schon durch wenige einfache, aber falsche, Joins passieren, dass die Menge der möglichen Zuordnungen stark ansteigt). Darum wird ein solches Abbruchkriterium festgelegt.

6.3 Implementierung

Eine Übersicht der relevanten Klassen für den Resultset Vergleich wird in Abbildung 6.3 gegeben. Es sind nur die Klassen dargestellt, die unmittelbar verwendet und im Folgenden beschrie-

ben werden. Von diesen Klassen sind auch nur die für das Verständnis der Arbeitsweise nötigen Methoden angeführt.

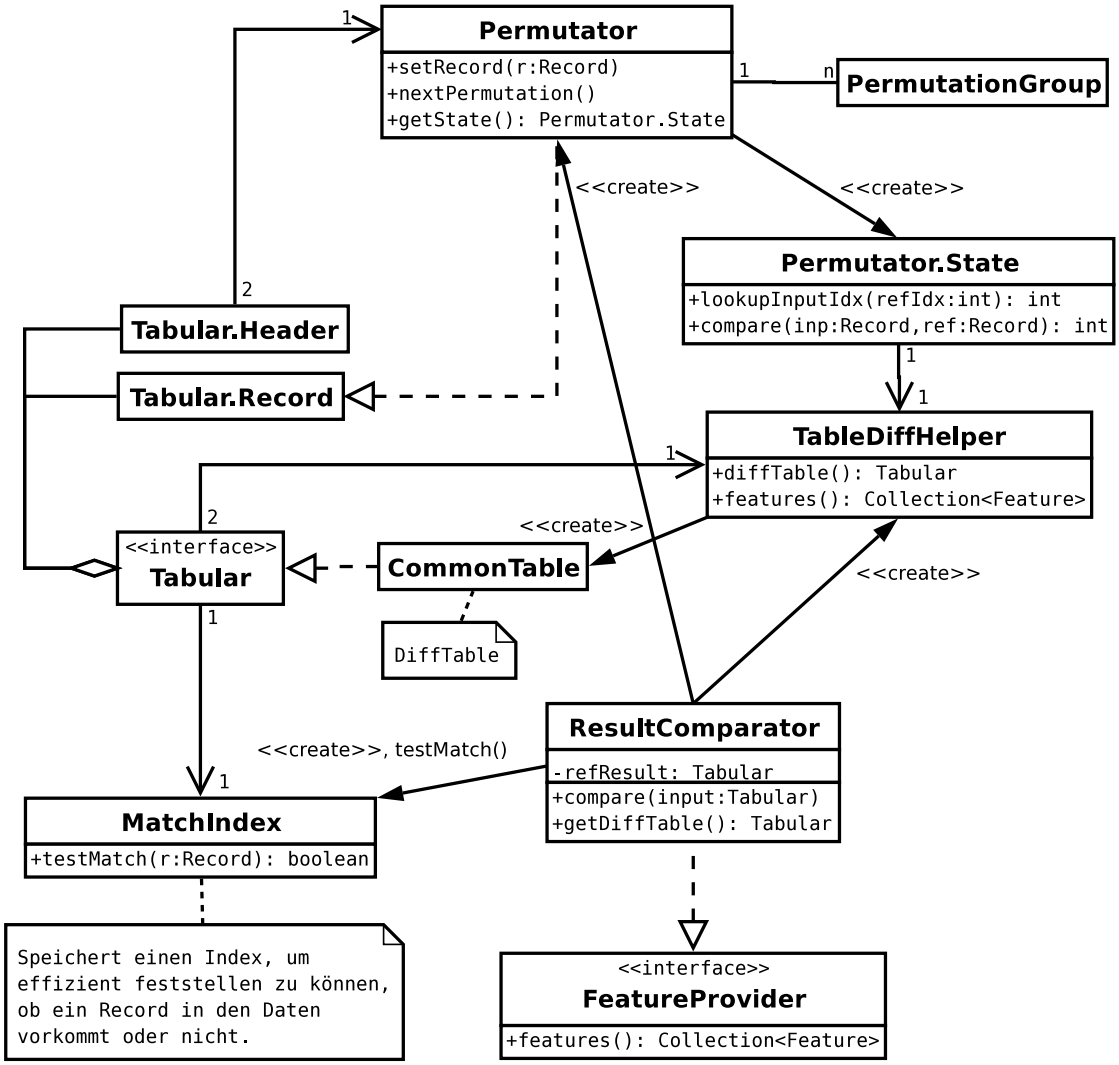


Abbildung 6.3: Relevante Klassen und Zusammenhänge beim Resultset Vergleich

Die Klasse `ResultsetComparator` spielt eine zentrale Rolle in der Implementierung des Resultset Vergleichs. Ein Objekt dieser Klasse bekommt mit dem Constructor die Ergebnistabelle des Referenz Statements mitgegeben. Beim Aufruf von `compare()` wird die Tabelle der Inputquery mitgegeben. Danach kann die Vergleichstabelle gelesen werden (mit der Methode `getDiffTable()`) und die Features für das Regelsystem (mit `features()`). Siehe Abschnitt 8.4 für Details über die Verwendung dieser Features. In den Features sind jedenfalls die Informationen über die Unterschiede der beiden Tabellen gespeichert.

Die Ergebnistabellen werden über das Interface `Tabular` zur Verfügung gestellt. Dieses Interface wird von einer Klasse geliefert, die Queries auf der Kursdatenbank absetzt und die Ergebnisse ausliest. Die Details des Interfaces sind nicht weiter relevant und demnach auch hier

nicht dargestellt. Wichtig sind allein die beiden enthaltenen Interfaces `Tabular.Header` und `Tabular.Record`, weil diese hier auch unabhängig verwendet werden.

Die Klasse `ResultComparator` selbst macht nicht viel und stellt eigentlich nur eine *Facade* zu den unterschiedlichen weiteren Klassen dar, die von hier aus verwendet werden.

Um die beste Spaltenzuordnung zu ermitteln werden die Klassen `Permutator` und `MatchIndex` verwendet. `Permutator` kapselt die komplexe Logik mit der die einzelnen Spaltenzuordnungen (Permutationen) der Reihe nach gebildet werden. Eine Permutation wird repräsentiert durch die Klasse `Permutator.State`. Der `Permutator` bekommt beim Erzeugen zwei Tabellen Header in Form von `Table.Header` Interfaces mit: den Header des Referenz Resultsets und denjenigen des Input Resultsets. Die Headerinformation beinhaltet alles, was der `Permutator` braucht, um alle möglichen (bzw. auch nach den im letzten Abschnitt beschriebenen Einschränkungen optimierten) Spaltenzuordnungen zu bilden und nacheinander zu liefern.

Das Interface eines `Permutator` Objekts stellt die Methode `nextPermutation()` zur Verfügung, mit der intern die nächste Permutation ermittelt wird. Interessant ist jedenfalls, dass die Klasse `Permutator` auch das `Record` Interface implementiert. Damit kann der `Permutator` als Record verwendet werden indem man mit der Methode `setRecord()` einen Record mitgibt. Der `Permutator` selbst kann dann als entsprechend permutierter Record verwendet werden.

Dieser Mechanismus wird verwendet vom `ResultComparator`. Um die Anzahl der übereinstimmenden Records für eine bestimmte Permutation zu ermitteln (also den *MatchValue*), wird jeder Record aus dem Input Resultset einmal an den `Permutator` übergeben. Damit wird der `Permutator` weiter übergeben an den `MatchIndex` um mit `testMatch()` festzustellen, ob der entsprechend zugeordnete Record im Referenz Resultset vorkommt oder nicht.

Der `Permutator` kann auch den aktuellen Zustand (`Permutator.State`) nach außen weitergeben. In dieser Klasse ist die Zuordnung gespeichert, die auch über ein entsprechendes Interface verwendet werden kann. Die Methoden `lookupInputIdx()` und `compare()` werden verwendet vom `TableDiffHelper`. Diese Klasse ist dafür verantwortlich, die Vergleichstabelle zu erzeugen und an den `ResultComparator` zu liefern. Dieser Schritt geschieht erst, wenn die beste Permutation ermittelt wurde. Dabei wird dem `TableDiffHelper` im Constructor diese Permutation mitgegeben. Die Vergleichstabelle wird erzeugt in Form einer Instanz von `CommonTable`, die eine Implementierung des `Tabular` Interfaces ist. Um die beiden Resultsets, die der `TableDiffHelper` ebenfalls in Form von `Tabular` Objekten vom `ResultComparator` bekommt, zu vergleichen, werden die Methoden aus dem `Permutator.State` verwendet.

Mit diesem Ansatz muss außerhalb der Implementierung der Klasse `Permutator` niemand über die komplexe Logik, mit der die Permutationen ermittelt werden Bescheid wissen. Dies ist im im Sinne des *Information Hiding* Konzepts[3] ist.

Wie im Diagramm auch ersichtlich ist der `MatchIndex` nichts anderes, als ein Index-Tree über die Zeilen des Referenz Resultsets. Das Referenz Resultset muss nur einmal gelesen werden. Aufgrund der eingelesenen Daten kann ein `MatchIndex` aufgebaut und gespeichert wer-

[3] *Information Hiding* wird das Konzept des Kapseln der Implementierung genannt, das das Ziel verfolgt, zukünftige Änderungen zu ermöglichen, ohne den Rest eines Systems ändern zu müssen. Der Begriff wurde erstmals erwähnt in [Parnas72].

den, was auch im `ResultComparator` gemacht wird. Das ist der Grund dafür, dass im `ResultComparator` das Referenz Resultset im Constructor mitgegeben wird, während das Input Resultset über eine eigene Methode gesetzt wird.

Durch Verwenden des `MatchIndex` wird das Ermitteln des *MatchValue* effizienter gestaltet.

Mit diesem Überblick sollte die Arbeitsweise und Implementierung des Resultset Vergleichs deutlich gemacht werden. Wie die Ergebnisse des Resultset Vergleichs im Bewertungssystem verwendet werden, wird im Kapitel 8 bzw. konkret im Abschnitt 8.4 gezeigt.

6.4 Recherche

Zum Schluss sollen noch die unterschiedlichen Resourcen aufgezeigt werden, die herangezogen wurden, um Anregungen zur Lösung des vorliegenden Problems einzuholen. Dieser Abschnitt wird deshalb am Ende angeführt, weil jetzt die Problemstellung deutlich gemacht wurde, und dadurch auch die unterschiedlichen Überlegungen einfacher nachvollziehbar sein sollten.

Die Anforderung, zwei Ergebnistupel von zwei verschiedenen Tabellen oder Abfragen vergleichen zu wollen, ist grundsätzlich nichts neues. Eine Reihe von Tools stehen einem zur Verfügung, um diese Aufgabe zu bewerkstelligen. Nennenswert darunter sind beispielsweise: *MySQL Query Browser*[4], *CompareData*[5], *Aqua Data Studio*[6] und *DBTimes - Relational Data Compare*[7]

Diese Tools verfolgen alle ähnliche Strategien bei der Wahl der übereinstimmenden Columns für den Vergleich:

- Der *MySQL Query Browser* verlangt zum Vergleich zwei Queries mit denselben Spaltennamen und derselben Reihenfolge der Spalten.
- *CompareData* verlangt vom Benutzer, einen „Comparison Key“ festzulegen, der angibt, welche Spalten für den Vergleich herangezogen werden. Ansonsten verwendet das Tool beliebige ODBC Datenquellen und kann damit auch Resultsets unterschiedlicher Datenbanken vergleichen. Außerdem hat man hier die Möglichkeit, große Datenmengen zu vergleichen indem das Tool die Sortierung in der Datenbank mit `ORDER BY` durchführt.
- *Aqua Data Studio*, ein umfangreiches Datenmanagement Tool, bietet ein Feature mit Namen „Results Compare Tool“. Das Tool vergleicht die Rows anhand eines Primary Key, der auch händisch vergeben werden kann.
- *Relational Data Compare* vergleicht die Resultsets entweder anhand eines „unique keys“ oder anhand der vorgegebenen Sortierreihenfolge.

All diesen Werkzeugen ist gemein, dass das Zuordnen der zu vergleichenden Spalten in irgendeiner Form dem User überlassen wird. Es wird entweder von Spaltennamen oder Primary

[4] http://dev.mysql.com/doc/query-browser/en/mysql-query-browser-using-compare.html
[5] http://www.zidsoft.com/
[6] http://www.aquafold.com/docs-compare-results.html
[7] http://www.dbtimes.com/rdcompare.htm

Key Definitionen in der Datenbank ausgegangen oder man kann selbst einen Schlüssel für den Vergleich definieren.

Das Problem der automatischen Spaltenzuordnung, das sich in der Anwendung hier stellt, wurde in keiner der genannten Quellen adressiert.

Weitere Überlegungen führen in den Bereich der Statistik. Die Aufgabe aus mathematischer Sicht besteht darin, für zwei Matrizen beliebigen Ranges mit dem Mittel beliebiger Zeilen- und Spaltenpermutationen die größtmögliche Übereinstimmung einzelner Werte zu erzielen. Wenn man eine Zeile als Vektor betrachtet, kann man die einzelnen Vektoren als Ereignisse einer Zufallsvariable betrachten. Man kann dann mithilfe der Korrelationsanalyse versuchen, Zusammenhänge zwischen den beiden Wertefolgen zu ermitteln.

Unterschiedliche Methoden der Vergleiche von Zufallsvariablen können dann näher betrachtet werden: verschiedene χ^2 Tests, der Pearson Korrelationskoeffizient oder Kreuzentropie und Kullback-Leibler-Divergenz.

Diese Methoden würden beispielsweise für eine gewählte Spaltenzuordnung einen Indikator liefern, wie gut die beiden Resultsets „korrelieren“.

Auffällig ist in diesem Zusammenhang auch der *Mantel Test* ([Mantel67]). Dieser Test verfolgt einen Monte-Carlo Ansatz, indem zufällige Zeilen- und Spaltenpermutationen verwendet werden um die Korrelation zwischen zwei Matrizen zu ermitteln (bzw. die Nullhypothese zu bestätigen, dass die beiden Matrizen nicht korrelieren). Der Mantel Test selbst ist zwar nicht direkt auf das vorliegende Problem anwendbar, der Ansatz des Tests von Permutationen ist aber ähnlich.

Die Entscheidung, einen eigenen Test für die Bewertung einer Spaltenzuordnung zu entwickeln, resultierte schließlich aus dem Gedanken, dem Problem nicht mehr Bedeutung beimessen zu wollen, als es tatsächlich hat. Es handelt sich im vorliegenden Fall immerhin nicht um Unmengen von möglichen Permutationen, die zu testen sind. Wenn die Übereinstimmungskriterien, die durch die JDBC Metadaten zur Verfügung stehen, vorab verwendet werden, ist im Normalfall die Zuordnung der übrigen Spalten nur noch ein relativ unaufwändiger Prozess und kann mit der beschriebenen Methode in Abschnitt 6.2.2 einfach durchgeführt werden.

7 Regelbasierte Systeme

Dieses Kapitel soll einen Überblick über einzelne Konzepte von Expertensystemen im Allgemeinen und regelbasierten Systemen im Speziellen geben. Es wird auf die allgemeine Struktur von Expertensystemen eingegangen, es werden verschiedene Beispiele von Regelsystemen gezeigt und kurz auf verbreitete Algorithmen in Regelsystemen eingegangen.

7.0.1 Aufbau von Expertensystemen

In [Beierle00] wird ein konzeptioneller Überblick über den typischen Aufbau von Expertensystemen gebracht. Die wesentlichen Komponenten eines jeden Expertensystems sind dort (vgl. Abbildung 7.1):

- Die *Wissensbasis*, welche in Form von Regeln gespeichert wird. Die Wissensbasis bildet die Grundlage für die nächste Komponente:
- Die *Wissensverarbeitung*. Hier wird vorhandenes Wissen verwendet um neues Wissen zu generieren.
- Das *fallspezifische Wissen* ist die Menge an Fakten (das Wissen), das von der Wissensverarbeitung als Input verwendet wird und anhand des regelhaften Wissens in der Wissensbasis verarbeitet wird.
- Die *Wissenserwerbskomponente* unterstützt den Aufbau der Wissensbasis.
- Die *Erklärungskomponente* ist dafür zuständig, zu „argumentieren“, warum das Wissen in einer bestimmten Art und Weise verarbeitet wurde.
- Die *Dialogkomponente* ist für die Interaktion mit dem User verantwortlich.
- Die „normale“ *Benutzerschnittstelle* ist die Schnittstelle für den regulären Benutzer des Systems. Für denjenigen, der vom Wissen des Expertensystems profitieren soll.
- Zusätzlich unterscheidet man eine *Schnittstelle für Experten*, die von denjenigen verwendet wird, die das Wissen ins System bringen.

Bezug zum SQL e-Learning System

Im Bezug auf das vorliegende System können diese Komponenten folgendermaßen betrachtet werden:

- Die Wissensbasis besteht aus den Regeln des Regelsystems für die Bewertung.
- Die Wissensverarbeitung wird in dem eigens implementierten Regelsystem durchgeführt, bzw. genaugenommen in der Inferenzkomponente des Regelsystems.

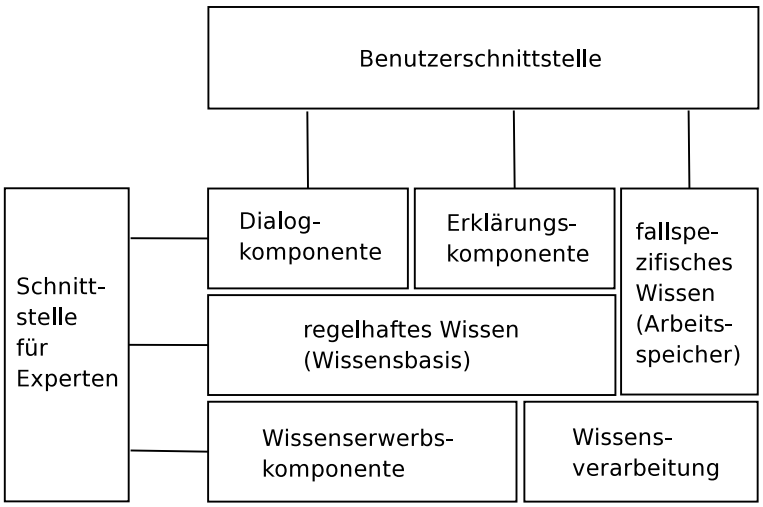


Abbildung 7.1: Schematischer Aufbau eines Expertensystems (aus [Beierle00])

- Das fallspezifische Wissen ist das Wissen über die Eigenschaften des eingegebenen Statements des Studenten. Das sind die Ergebnisse aus dem Resultset Vergleich oder der Statement Analyse.
- Die Wissenserwerbskomponente ist hier etwas rudimentär ausgeführt: Ein Administrator hat die Möglichkeit die Regeln im System in einem Textfile zu bearbeiten und ins System „upzuloaden“. Direktes Bearbeiten der Wissensbasis ist nicht möglich.
- Die Erklärungskomponente wird in diesem System so ausgeführt, dass man als Administrator die Möglichkeit hat, die zutreffenden Regeln bei einer Bewertung anzuzeigen.
- Die Dialogkomponente besteht aus der webbasierten Applikationsoberfläche. Die Eingabe einer Lösung und das darauf erhaltene Feedback kann als Dialog betrachtet werden.
- Die Benutzerschnittstelle ist ebenfalls die Applikationsoberfläche.
- Die Schnittstelle für Experten unterscheidet sich von der Benutzerschnittstelle nur darin, dass die Experten (Admins) mehr ausführen dürfen, wie beispielsweise auch neue Regeln „uploaden“.

7.0.2 Produktionssysteme

Im Bezug auf die Arbeitsweise der Inferenz von Expertensystemen unterscheidet man typischerweise zwei unterschiedliche Konzepte:

- *Vorwärtsverkettende Systeme* sind solche, in denen aus den vorhandenen Fakten im fallspezifischen Wissen auf neue Fakten geschlossen wird. Es wird dabei typischerweise überprüft, ob eine Regel zutrifft, und entsprechend neue Fakten dem fallspezifischen Wissen hinzugefügt.

- *Rückwärtsverkettende Systeme* sind daran interessiert, ob ein bestimmtes Faktum aufgrund des fallspezifischen Wissens erfüllt ist oder nicht. Das regelhafte Wissen wird „von hinten“ aufgearbeitet, indem nur diejenigen Regeln betrachtet werden, die als Zielknoten den untersuchten Knoten beinhalten.

Vorwärtsverkettung liefert einen allgemeinen Zustand des Systems ausgehend von einem Anfangszustand. Mit Rückwärtsverkettung wird die Gültigkeit bestimmter Fakten überprüft (vgl. [Beierle00]).

Vorwärtsverkettete regelbasierte Systeme werden auch als *Produktionssysteme* bezeichnet und stellen die Regelsysteme im engeren Sinn dar. Es wird hier nicht mehr weiter auf Rückwärtsverkettung eingegangen, stattdessen soll ein Überblick über einige bekannte Regelsysteme bzw. Produktionssysteme gegeben werden:

- *OPS5*
  OPS war der Name für eine Reihe von Produktionssytemen, die in den frühen 1980er Jahren von C. Forgy an der Carnegie Mellon University entwickelt wurden. Forgy brachte dabei einen effizienten matching Algorithmus zum Einsatz, den er RETE nannte ([Forgy82]) und der seither in zahlreichen Produktionssystemen zur Anwendung gekommen ist (siehe Abschnitt 7.1 für mehr zu RETE).

  OPS5 ist eine allgemeine Definition einer formalen Programmiersprache für Produktionssysteme ([ForgyOPS5-81]). Die Sprache ähnelt der LISP Syntax, und kann als eigenständige deklarative Programmiersprache verwendet werden.

- *CLIPS*
  CLIPS (C Language Integrated Production System) wurde um 1984 von der NASA am Johnson Space Center entwickelt. Eine Weiterentwicklung davon wird heute als Opensource Projekt von Gary Riley betreut[1].

  Für CLIPS gibt es zahlreiche Erweiterungen und vor allem unterschiedlichste Möglichkeiten, das System in andere Programmiersprachen einzubetten.

- *LISA*
  Eine Plattform für (Lisp-based Intelligent Software Agents). LISA ist ein in LISP implementiertes Produktionssystem, in dem die Regeln auf die Infrastruktur des Common Lisp Object Systems (CLOS) zurückgreifen können[2].

- *Soar*
  Soar Cognitive Architecture ist ein umfangreiches Framework für Expertensysteme, das von *Soar Technology, Inc.* angeboten wird[3]. Siehe [SoarOverview02]

Produktionssysteme werden heute auch häufig als *Business Rule Engines* eingesetzt. Beispiele dafür sind: ILOG BRMS[4] oder Hayley Office Rules[5].

[1] CLIPS ist derzeit zu finden unter: http://clipsrules.sourceforge.net/
[2] http://lisa.sourceforge.net/
[3] http://www.soartech.com/
[4] BRMS (Busniess Rule Management System) für Java, .NET und COBOL
http://www.ilog.com/products/businessrules/
[5] http://www.haley.com/products/products.html

Produktionssysteme für Java

Auch für die Programmiersprache Java steht eine Reihe von Systemen zur Verfügung, die als regelbasierte Engines oder alleinstehende Applikationen verwendet werden können.

Vorweg sei allerdings auf den JSR94 aufmerksam gemacht. „JSR94: Java Rule Engine API“ beschreibt eine Java Standard API für Regelsysteme in Java, mit der Regeln definiert werden können, das fallspezifische Wissen initialisert und eine *Runtime* verwendet werden kann, um die Regeln anzuwenden (siehe [Mahmoud05]).

Nennenswerte Java Implementierungen von Regelsystemen sind:

- *Jess (Java Experts System Shell)*[6] ist ein System, das sowohl als eigene Applikation verwendet, als auch als *Engine* in andere Applikationen eingebettet werden kann.
- *JBoss Rules (aka. Drools).*[7] Drools ist der alte aber immer noch bekannte Name des Projekts. Als das Projekt von JBoss übernommen wurde, wurde daraus *JBoss Rules*. Das System kann in JBoss beispielsweise als Business Rule Engine eingesetzt werden.
- *JLisa*[8] ist eine Implementierung des JSR94 auf Basis des LISA Projekts.
- *OPSJ*[9] ist eine native Java Implementierung eines Produktionssystems von *Production Systems Technologies, Inc.*

Andere interessante Ansätze für Logiksysteme bzw. Expertensysteme in Java, vor allem rückwärtsverkettete Systeme sind:

- *Interprolog* ist eine Brücke von Java zu unterschiedlichen Prolog Implementierungen.
- *Mandarax* ist ein Java basiertes Inferenz System das Rückwärtsverkettung implementiert.
- *Prova* stellt eine „Kombination“ aus Prolog und Java dar. Es handelt sich dabei um eine Syntaxerweiterung für Java um deklarative Elemente.

7.1 Algorithmen

Seit Forgy in [Forgy82] seinen RETE Algorithmus vorgestellt hat, bildet dieser in mehr oder weniger abgewandelten Formen die Grundlage für sehr viele Produktionssysteme. Der RETE Algorithmus versucht, das aufwändige Matching Problem bei großen Regelmengen und Faktenmengen zu optimieren, indem er verschiedenste Strukturen im Speicher behält. Das Grundprinzip ist ein Netzwerk aus Regeln, das die Redundanz in den Regeldefinitionen minimiert, indem überlappende Bereiche nur einmal im Netzwerk vorkommen. Es werden außerdem Teil-Matches im Netzwerk gespeichert, um so die Vergleiche beim Matching nicht nochmal durchführen zu müssen. Trotz der großen Mengen an gespeicherten Strukturen bietet der Algorithmus eine effiziente Möglichkeit, Fakten wieder aus dem Speicher zu entfernen. Der Algorithmus definiert α Memory und β Memory als Bereiche in denen Informationen zwischengespeichert werden (vgl. Abbildung 7.2).

[6] http://herzberg.ca.sandia.gov/
[7] http://www.jboss.com/products/rules
[8] http://jlisa.sourceforge.net/
[9] http://www.pst.com/opsj.htm

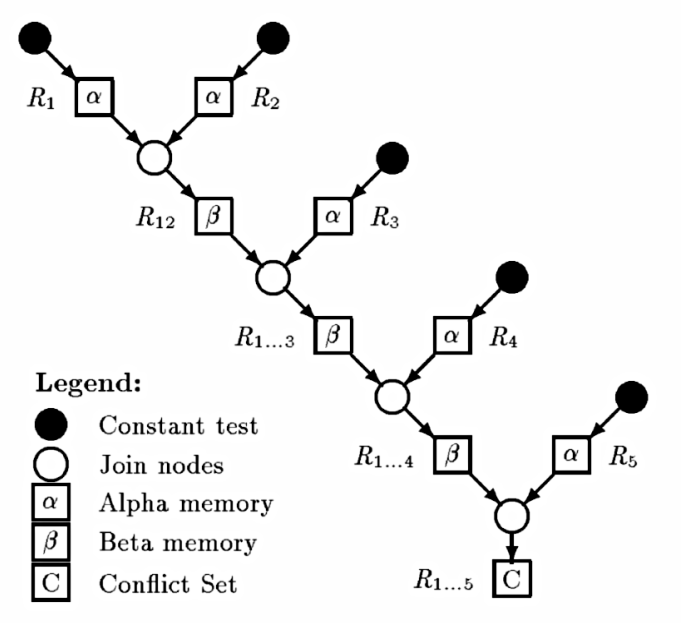


Abbildung 7.2: Ein RETE Netzwerk (aus [Wright03])

RETE Implementierungen finden sich beispielsweise in: OPS (die ursprüngliche Implementierung von Forgy), CLIPS, Jess, JBoss Rules oder Soar.

Ausgehend von RETE beschreibt [Miranker89] TREAT. Was in RETE erkennbar ist, ist die Tatsache, dass der Algorithmus nicht effizient ist, wenn sich große Mengen an Fakten ändern oder die Fakten sich sehr schnell ändern. Der Algorithmus ist ausgelegt auf große Mengen von Daten und effizienter Reaktion auf einzelne Änderungen.

TREAT hingegen bedient sich keiner Zwischenspeicherungen und ist deshalb im Fall von vielen Änderungen effizienter.

Als Extension zu TREAT ist LEAPS (Lazy Evaluation Algorithm for Production Systems) entstanden ([Miranker90]).

Unabhängig von TREAT/LEAPS hat Forgy seinen RETE Algorithmus zu RETE-2[10] weiterentwickelt. RETE-2 wurde nicht veröffentlicht, sondern nur kommerziell bei *Production Systems Technologies, Inc.* verwendet. Dort werden auch die Implementierungen CLIPS/R2, OPS/R2 und OPSJ angeboten.

Laut den dort gezeigten Benchmark Tests[11] sind diese Implementierungen grob um Faktor 100 schneller als RETE Systeme.

[10] http://www.pst.com/rete2.htm
[11] http://www.pst.com/benchcr2.htm

8 Bewertungssystem

8.1 Motivation und Überblick

Den Kern des Bewertungssystems bildet ein regelbasiertes System, das die Informationen, die von unterschiedlichen Quellen gesammelt werden, auswertet, um daraus eine Bewertung der eingegebenen Lösung zu ermitteln.

Diese Aufgabe könnte prinzipiell auch einfach direkt in Programmlogik abgebildet werden, aber es gibt doch gute Gründe, warum für diesen Zweck ein Regelsystem zum Einsatz kommen soll.

- Durch die Verwendung von Regeln anstelle von Programmlogik ist eine übersichtliche deklarative Darstellung des Bewertungskonzepts möglich. Wäre das Bewertungskonzept im Java Code direkt implementiert, würde das die Form von undurchsichtigen verschachtelten `if ... then` Statements annehmen.
- Das Regelsystem kann sehr gut als *Erklärungskomponente* eingesetzt werden. Ein LVA-Leiter kann die Regeln auflisten, die bei der Bewertung zugetroffen haben, und bekommt damit Einsicht in die Arbeitsweise des Systems, bzw. eine Begründung für die Bewertung.
- Das Bewertungssystem kann sich im Lauf der Zeit ändern. Einerseits sind vielleicht Verbesserungen vorzunehmen und andererseits ist durchaus vorstellbar, dass irgendwann das ganze Bewertungskonzept umgestellt wird.

  Mit dem Regelsystem kommt man dem entgegen und ermöglicht Änderungen durch Neudefinition der Regeln.
- Schließlich können die geänderten Regeln auch so im System gespeichert werden, dass bei alten abgelegten Prüfungen immer noch die alten Regeln verwendet werden und somit die Bewertung immer noch nachvollziehbar bleibt.

  D.h. die Verwendung des Regelsystems ermöglicht die Historisierung des Bewertungskonzepts zusammen mit den abgelegten Prüfungen.

Die Aufgaben des Regelsystems im Kontext des SQL Bewertungssystems sind in Abbildung 8.1 im Überblick dargestellt.

Als Input des Systems dienen die Ergebnisse aus dem Resultsetvergleich, der Analyse des Inputstatements und eines Referenzstatements und weitere Quellen auf die hier noch nicht näher eingegangen wird.

Die Ergebnisse am Ausgang des Regelsystems sind: die Bewertung des Statements selbst (d.h. eine Punktezahl für die aktuelle Aufgabe), eine Liste der Meldungen als Userfeedback sowie der Status der Aufgaben (korrekt, fehlerhaft, etc. Siehe Abschnitt 5.3.3).

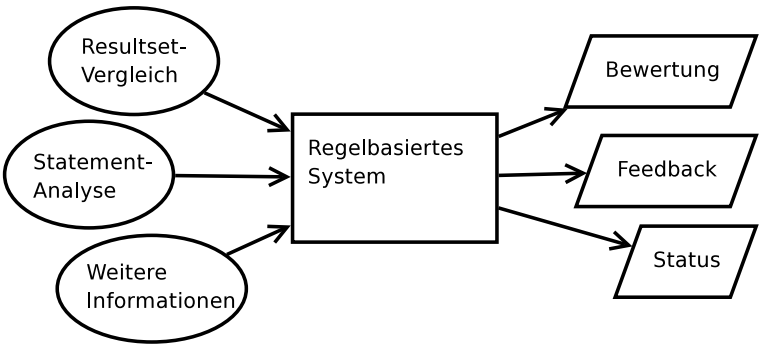


Abbildung 8.1: Aufgaben des Regelsystems

Für die Umsetzung des Regelsystems wurde entschieden, das System von Grund auf neu zu implementieren anstatt auf eine Java Implementierung eines allgemeinen Regelsystems zurückzugreifen.

Damit kann das Regelsystem auf die vorliegende Problemstellung optimal angepasst werden. Man muss keine Performanceeinbußen hinnehmen, die sich eventuell dadurch ergeben könnten, dass das System möglichst allgemein einsetzbar sein soll.

Vorhandene Implementierungen von Regelsystemen in Java sind meist für Szenarien ausgelegt, wo eine Faktenmenge gegeben ist, und sich die vorhandenen Fakten ändern. Für diesen Fall sind die Systeme optimiert. Im vorliegenden Fall wird aber die Faktenmenge bei jeder Auswertung neu gegeben, was alle Optimierungen des vorigen Falls unnötig macht (wenn nicht vielleicht sogar hinderlich).

Das gewählte Konzept des Systems ist außerdem einfach genug, um eine kleine überschaubare Implementierung zu ermöglichen. Die Auswertung wird in einer Form gemacht, die zu keinen Laufzeitproblemen führen kann.

In Abbildung 8.2 wird das Regelsystem schematisch mit den wesentlichen Komponenten dargestellt.

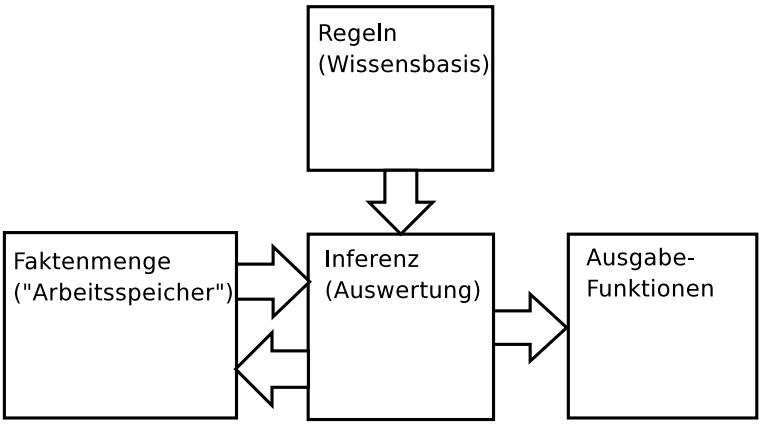


Abbildung 8.2: Grobe schematische Darstellung des Regelsystems

Ausgehend von einer Menge von Fakten (*Faktenmenge* stellt quasi den „Arbeitsspeicher“ des Regelsystems dar) werden bei der Inferenz durch *zutreffende Regeln* neue Fakten generiert und zusätzlich *Ausgabefunktionen* aufgerufen.

Mehr zum Konzept, der Arbeitsweise und der Implementierung des Regelsystems wird in den folgenden Abschnitten beschrieben.

8.2 Entwurf des Regelsystems

Das hier vorgestellte Regelsystem wurde eigens für den Einsatz im Bewertungssystem entworfen und implementiert. Bei der Implementierung wurde aber auf allgemeine Verwendbarkeit geachtet. Dadurch kann das Regelsystem auch in anderen Umgebungen zum Einsatz kommen und wird hier auch vorerst möglichst unabhängig beschrieben.

8.2.1 Definition der Strukturen

Die Grundelemente des Systems sind `Fakten`, die wahr oder falsch sein können. Es werden 2 Arten von Fakten unterschieden: *Konstanten* und darauf aufbauende n-stellige *Prädikate*. Konstanten und Prädikate sind „wahr“ (das bedeutet im Kontext des Bewertungssystem normalerweise *zutreffend*), wenn sie in der *Faktenmenge* enthalten sind. Die *Faktenmenge* ist demnach die Menge aller zutreffenden Fakten. Die Faktenmenge wird hier auch als *Faktenbasis* (in der Implementierung `FactBase`) bezeichnet.

Konstanten und Prädikate können so interpretiert werden, wie deren Entsprechungen in der Prädikatenlogik. Das vorgestellte Regelsystem unterliegt allerdings nicht den Regeln der Prädikatenlogik, sondern bedient sich nur einzelner Konzepte (bzw. Begriffe) aus diesem Bereich.

Beispiele für Konstanten sind Aussagen wie: `isExecutable` oder `hasSyntaxError`. Darauf aufbauende n-stellige Prädikate werden z.B. so geschrieben: `input(isExecutable)` oder `reference(hasSyntaxError)`. In der Faktenbasis werden auch die zutreffenden Prädikate genau in dieser Form gespeichert.

Durch eine *Regel* wird im System die Inferenz ermöglicht. D.h. es können neue Fakten aus den bestehenden Fakten anhand der definierten Regeln abgeleitet werden. Eine Regel besteht aus einer *linken Seite*, die als eine Konjunktion von *Bedingungen* aufgebaut ist, und einer *rechten Seite*, in der konjugierte Prädikate aufgelistet werden können. Wenn alle Bedingungen der linken Seite zutreffen, werden alle Prädikate der rechten Seite in die Faktenbasis mit aufgenommen.

Die Regel beschreibt also eine einfache **if** `x` **then** `y` Beziehung, die notiert wird als:

$$\bigwedge_i fact(i) \rightarrow \bigwedge_i pred(i)$$

Folgende Regel beispielsweise definiert ein neues Prädikat `missingFeature(x)`, wenn die beiden Bedingungen auf der linken Seite zutreffen:

$$\texttt{reference(x)} \wedge \neg \texttt{input(x)} \rightarrow \texttt{missingFeature(x)}$$

Dabei ist x eine Variable, die für beliebige Konstanten stehen kann. Ebenfalls in diesem Beispiel ersichtlich ist die Tatsache, dass auf der linken Seite auch Negationen erlaubt sind.

Diese Form der regelbasierten Inferenz, basierend auf den *modus ponens*, wird in [Beierle00] beschrieben. Die dort beschriebenen Regelsysteme sind allerdings zu unflexibel für den praktischen Gebrauch hier im Bewertungssystem. Das Buch geht nur sehr kurz auf regelbasierte Systeme im engeren Sinn ein und geht dann über zu komplexeren Systemen, wie Antwortmengenprogrammen oder TMS.

Tatsächlich stellt das hier vorgestellte Regelsystem einen recht einfachen Kompromiss aus den unkomplizierten Konzepten eines Regelsystems und den aufwändigen Methoden der logischen Resolution dar. Eine einfache Form der Resolution ist allerdings notwendig, da, wie im obigen Beispiel bereits angedeutet, auch Variablen in den Regeln verwendet werden können. Bei Zutreffen einer Regel müssen Substitutionen für die dort vorkommenden Variablen gefunden werden. Details zur gewählten Methode der Resolution werden im Abschnitt 8.2.3 beschrieben.

Weitere Konzepte

Die wesentlichen Konzepte des Systems sind damit eigentlich bereits beschrieben. Abgesehen von den noch fehlenden Details der Inferenz werden aber auch noch weitere strukturelle Konzepte definiert, die der Vereinfachung des Systems dienen und einen Spezialfall im hier verwendeten Kontext abbilden. Konkret sind das:

- *Typisierung von Konstanten.*
  Eine Konstante kann einem oder mehreren Typen zugeordnet werden. So kann z.B. definiert werden, dass die Fakten *group, having, subselect* vom Typ *structural* sind oder die Fakten *min, max, count, avg* vom Typ *aggregation*. Typen sind einfach nur Bezeichnungen, denen eine Menge von Konstanten zugeordnet ist.

  Das Konzept der Typisierung könnte man auch einfach im Rahmen des Regelsystems implizit abbilden, indem Regeln mit leere linker Seite definiert werden, wie z.B.:

  $$\rightarrow \texttt{isOfType}(\texttt{structural}, \texttt{group})$$

  Durch die Definition des Typsystems soll aber verhindert werden, die Regeln und die Faktenbasis unnötig wachsen zu lassen. Das führt zu einer übersichtlicheren Regelmenge und zu besserer Performance in der Auswertung.

  Typen können verwendet werden im Zusammenhang mit Variablen. Z.B.:

  $$\neg\texttt{missing}(\texttt{x}/\texttt{structural}) \rightarrow \texttt{structureOK}()$$

  trifft dann zu, wenn kein `missing(x)` existiert mit einem `x` vom Typ `structural`.

- *Datenlisten von Konstanten.*
  Jeder Konstante kann zusätzlich eine Liste von *Daten* zugewiesen werden. Diese Liste wird im Laufe der Verarbeitung mit der Konstante mitgeführt, hat aber keine direkte Auswirkung auf die Auswertung der Regeln. Ein Beispiel für eine Datenliste wäre eine Liste von Tabellennamen, die in einem SQL Statement verwendet werden, die einer Konstante `hasTables` zugewiesen wird. Die Konstante `hasTables` mit der Liste wird folgendermaßen notiert:

  $$\texttt{hasTables}[\texttt{person}, \texttt{sportart}]$$

Diese Datenlisten werden verwendet, um Informationen mit den Konstanten durch das Regelsystem zu „schleifen“.

Gleichheit der Konstanten für die Auswertung wird aber nur durch den Namen definiert. Das bedeutet, dass z.B. die beiden Konstanten `hasTables[person,sportart]` und `hasTables[mahlzeit]` trotz der unterschiedlichen Datenlisten als gleich gewertet werden.

Werden der Faktenbasis aufgrund von Regelauswertungen neue Konstanten hinzugefügt obwohl eine gleiche Konstante bereits vorhanden ist, werden die Datenlisten zusammengeführt und die Konstante wird nur einmal definiert.

- *Ausgabefunktionen.*
  Ebenfalls in der Auswertung nicht weiter von Bedeutung, aber wesentlich für die Verwendung des Regelsystems ist die Möglichkeit, spezielle Ausgabefunktionen zu definieren. Es wird als Ergebnis einer Auswertung (bzw. Ableitung) nicht die vollständig abgeleitet Faktenbasis verwendet, sondern die Ergebnisse werden über ein eigenes Ausgabemodul ermittelt.

  Auf der rechten Seite einer Regel können Funktionen verwendet werden, die vom Ausgabemodul ausgewertet werden. Notiert werden diese Funktionen mit einem „▷“ vor dem Namen:

  $$\texttt{structureOK}() \rightarrow \triangleright\texttt{maxScore}(4).$$

  Trifft eine solche Regel zu, wird dem Ausgabemodul die Funktion mitgegeben. Es wird in der Faktenbasis nichts erweitert. Das Ausgabemodul entscheidet, was aufgrund dieser Funktion zu tun ist. Im gezeigten Beispiel, im Kontext des SQL Bewertungssystems, würde die maximal erreichte Punktezahl auf 4 gesetzt.

Struktur einer Regel

Zur Vervollständigung nochmal eine Zusammenfassung der Struktur einer Regel:

- Eine Regel besteht aus einer linken Seite und einer rechten Seite. Auf der linken Seite steht eine Konjunktion von Bedingungen, die entweder Prädikate oder Konstanten sein können. Auf der rechten Seite steht eine Konjunktion von Aktionen. Eine Aktion kann die Definition eines neuen Prädikatwerts sein, oder der Aufruf einer Ausgabefunktion.

- Prädikatbedingungen können mit Variablen definiert werden. Es wird dann in der Faktenbasis nach passenden definierten Prädikaten gesucht und die Variablen werden durch die zutreffenden Konstanten substituiert. Die Variablen können durch einen Typ eingeschränkt werden. Dadurch können nur Konstanten dieses Typs für diese Variable substituiert werden.

- Bedingungen auf der linken Seite können auch negiert vorkommen. Die Interpretation von negierten Bedingungen ist abhängig von der Belegung der Variablen. Darauf wird im nächsten Abschnitt näher eingegangen.

Damit wurden die Grundstrukturen des Regelsystems definiert. In den nächsten Abschnitten soll näher auf die Vorgehensweise bei der Auswertung der Regeln (also der Inferenz) und der

Variablensubstitution eingegangen werden. Es wird sich zeigen, dass die Interpretation einer Bedingung teilweise davon abhängig ist, ob eine Variable in dieser Bedingung erstmals definiert wurde oder bereits in einer Bedingung „weiter links“ verwendet wurde. D.h. die relative Position einer Bedingung in der Regeldefinition hat Einfluss auf die Ergebnisse.

Weiters wird auf die Reihenfolge der Regelauswertung eingegangen und es wird definiert, dass die Regeln anhand eines azyklischen gerichteten Graphen zu sortieren sind.

8.2.2 Weitere Überlegungen und Einschränkungen

Reihenfolge der Regeln

Um den Aufwand der Inferenz gering zu halten wurde festgelegt, dass die Regeln zur Auswertung nur sequentiell abgearbeitet werden. Es wird das Prinzip der Vorwärtsverkettung angewandt indem ausgehend von der vorliegenden Faktenbasis die Regeln geprüft werden und dadurch die Faktenbasis erweitert wird. Im Gegensatz zu einer logisch vollständigen Auswertung wird allerdings der Prozess nach dem ersten Durchlauf bereits abgebrochen. Das bedeutet, jede Regel wird nur einmal überprüft und angewendet.

Wegen dieser Vereinfachung hat die Reihenfolge der Regelanwendung einen wesentlichen Einfluss auf das Ergebnis. Um trotz dieser Tatsache konsistente Ergebnisse zu ermöglichen ohne bei der Spezifikation der Regeln auf die Reihenfolge achten zu müssen, werden die Regeln vor der Verarbeitung sortiert.

Als Grundlage für diese Sortierung wird ein gerichteter Graph herangezogen, der aus den Regeln aufgebaut wird, indem die Konstanten und Prädikate als Knoten verwendet werden und von jedem Knoten in der linken Seite einer Regel zu jedem Knoten in der entsprechenden rechten Seite eine Kante gezogen wird.

Aus den Regeln

$$A(x) \wedge B(x) \rightarrow C(x) \wedge D(x)$$

$$const1 \wedge B(x) \rightarrow A(x)$$

$$C(x) \wedge const2 \rightarrow D(x) \wedge E(x)$$

$$A(x) \wedge E(x) \wedge const1 \rightarrow F()$$

entsteht so der Graph in Abbildung 8.3.

In dieser Abbildung ist auch die Nummerierung der Regeln in der passenden Reihenfolge angeführt. Jeder Knoten eines Prädikats wird in dem Graphen so nummeriert, dass seine Nummer kleiner ist als die Nummern aller Nachfolgeknoten. Um diese Nummerierung in jedem Fall zu ermöglichen, muss der Graph azyklisch sein!

Es wird also die Azyklizität des entsprechenden gerichteten Graphen als Bedingung für die Regeln dazu genommen. Dadurch kann auf iterative Auswertung verzichtet werden, jede Regel muss nur einmal betrachtet werden und die Ergebnisse sind dennoch konsistent.

Reihenfolge der Variablen

Bei der Auswertung einer Regel werden die vorkommenden Variablen von links nach rechts substituiert. Sei $\{A(const1), A(const2), B(const1), const2, C(const2)\}$ eine angenommene Faktenbasis. Die Regel

$$A(x) \wedge B(x) \rightarrow C(x)$$

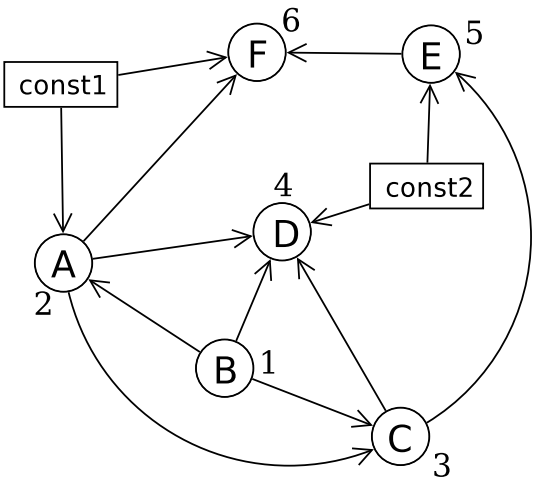


Abbildung 8.3: Beispiel: Gerichteter Graph aus 4 Regeln

trifft zu mit der Substitution $x = const1$ und es wird $C(const1)$ der Faktenbasis hinzugefügt. Um dieses Ergebnis zu ermitteln, wird von links nach rechts substituiert. D.h. es wird zuerst $A(x)$ betrachtet. Es gibt 2 Elemente der Basis, die zutreffen: $A(const1)$ und $A(const2)$. Damit ist die Substitution zunächst $x \in \{const1, const2\}$. Mit dieser Substitution wird die zweite Komponente $B(x)$ betrachtet. Die Substitution wird jetzt nur noch weiter eingeschränkt. D.h. es wird überprüft, ob $B(const1)$ oder $B(const2)$ in der Basis enthalten sind. Nur $B(const1)$ ist enthalten und damit ergibt die Substitution $x = const1$.

Allgemein wird jede Variable bei ihrem ersten Vorkommen mit allen möglichen Substitutionen belegt und danach nur noch weiter eingeschränkt. Das führt bei positiven Bedingungen immer zum selben Ergebnis, unabhängig von der Variablenreihenfolge.

Anders verhält sich die Sache bei negativen Bedingungen. Betrachten wir die Regel

$$A(x) \wedge \neg B(x) \rightarrow D(x)$$

bei der selben Faktenbasis, dann ist zunächst offensichtlich, dass die einzige gültige Substitution $x = const2$ ist und damit $D(const2)$ der Faktenbasis hinzugefügt wird. Die Auswertung von links nach rechts kann genau gleich erfolgen, wie im positiven Fall.

Wenn aber andererseits die Regel lautet:

$$\neg B(x) \wedge A(x) \rightarrow D(x)$$

dann stellt sich die Situation ganz anders dar. Im Sinne der Prädikatenlogik müsste das Ergebnis dasselbe sein. Allerdings wird dieser Fall hier anders definiert: um die Substitution in einem solchen Fall zu vereinfachen, wird festgelegt, dass eine Variable, die in einer negierten Bedingung erstmals verwendet wird, als *implizit existenzquantifiziert* betrachtet wird. In diesem Fall wird nicht x mit allen Werten substituiert, für die $\neg B(x)$ zutrifft, sondern es wird festgestellt, ob ein beliebiges $B(x)$ in der Faktenbasis existiert, und wenn das der Fall ist, trifft die Regel nicht zu. Sollte kein $B(x)$ existieren und weiter ausgewertet wird, dann wird für x nichts substituiert. x wird also als lokal gebundene Variable behandelt.

Durch diese Sonderbehandlung wird die Substitution weiter vereinfacht, und es wird gleichzeitig eine Nichtexistenzprüfung ermöglicht. Positive Existenzprüfung kann ja ohnehin einfach abgebildet werden, was beispielsweise die Regel $A(x) \rightarrow Aexists()$ verdeutlicht.

Es wäre möglich gewesen, die linke Seite einer Regel automatisch umzureihen, so dass die erste Komponente, in der eine Variable vorkommt positiv ist, sofern dies möglich ist. Zusätzlich könnte man einschränken, dass Variablen, die existenzquantifiziert verwendet werden, in einer Regel eindeutig sein müssen, oder man könnte den Existenzquantor explizit definieren und den oben beschriebenen Fall anders behandeln bzw. ungültig machen.

Es wurde aber für die Implementierung dieser Weg gewählt, weil er einfach umzusetzen und auch unkompliziert zu verwenden ist.

Widersprüche

Dadurch, dass durch das Regelsystem auf der rechten Seite nur positive Prädikate erlaubt sind, können in der Faktenbasis immer nur Fakten hinzugefügt werden. Die Art der Inferenz in dem System zusammen mit den getroffenen Einschränkungen ermöglicht somit keine Widersprüche in den Regeln.

Das so definierte Regelsystem ist vielleicht nicht besonders flexibel in der Definition der Regeln und die Auswertung der Regeln erinnert stark an prozedurales Abarbeiten von Programmlogik. Aber es bietet so wie es im Bewertungssystem verwendet wird dennoch den Vorteil eines Regelsystems und bleibt durch seine Einfachheit gut umsetzbar.

8.2.3 Inferenz

Während im vorigen Abschnitt die Strukturen definiert wurden und die prinzipielle Arbeitsweise des Systems und die sich daraus ergebenden Einschränkungen vorgestellt wurden, wird jetzt im Detail auf die Algorithmen beim Aufbau der Regelbasis und der Inferenz eingegangen. Es werden teilweise bereits besprochene Themen vom Gesichtspunkt der Algorithmen behandelt.

Vorwärtsverkettung

Die Aufgabe des Regelsystems besteht darin, aus einer gegebenen Faktenmenge eine erweiterte Faktenmenge abzuleiten und dabei Ausgabefunktionen aufzurufen. Das System arbeitet also prinzipiell nach der Methode der *Vorwärtsverkettung*, indem die Basis durch Auswerten von Regeln erweitert wird. Der Ansatz der *Rückwärtsverkettung* würde im gegebenen Fall nicht zum Ziel führen, weil nicht bestimmte Aussagen gegeben sind, von denen überprüft werden soll, ob sie zutreffen oder nicht. Im Gegenteil: die zutreffenden Aussagen bzw. die aufgerufenen Ausgabefunktionen müssen nichtmal vollständig definiert sein. Man denke z.B. an eine mögliche Ausgabefunktion $\triangleright maxScore(i)$. Es ist nicht von vornherein festgelegt, welche Werte für i verwendet werden.

Die Arbeitsweise kann damit vereinfacht folgendermaßen dargestellt werden:

Es werden dabei die Regeln in einer festgelegten Reihenfolge abgearbeitet und im Gegensatz zum allgemeinen Algorithmus, in dem alle Regeln so lange weiter ausgewertet werden, bis die Wissensbasis nicht weiter vergrößert werden kann, wird in diesem Algorithmus jede Regel nur einmal betrachtet und der Algorithmus endet, wenn alle Regeln abgearbeitet sind.

Algorithmus : Arbeitsweise der Vorwärtsverkettung

Input : RuleSet, FactBase
Result : erweiterte FactBase, Aufrufe von OutputFunctions
foreach rule ∈ RuleSet **do**
  if `ruleApplies(`rule,FactBase`)` **then**
    foreach pred ∈ *predicates(*rule.*rightHandSide())* **do**
      `addToFactBase(`FactBase, pred`);`
    foreach func ∈ *functions(*rule.*rightHandSide())* **do**
      `callOutputFunction(`func`);`

Diese Vorgehensweise wird ermöglicht durch die Zusatzbedingung, dass die Regeln anhand eines azyklischen gerichteten Graphen sortiert sein müssen. In der Praxis bedeutet das, dass jedes Prädikat erst in einer anderen Regel auf der linken Seite verwendet werden kann, wenn alle Regeln abgearbeitet wurden, die dieses Prädikat „erzeugen“ könnten (d.h., die dieses Prädikat in der rechten Seite enthalten). Damit wird sichergestellt, dass ein Prädikat nicht nach seiner Abfrage in einer Bedingung noch durch eine andere Regel in der Faktenbasis aufgenommen werden kann.

Die iterative Auswertung, in der alle Regeln solange nochmal getestet werden, bis die Faktenbasis nicht mehr erweitert wird, wird dadurch unnötig.

Es wird zwar die Flexibilität des Systems eingeschränkt, aber für den gedachten Anwendungsbereich ist das Konzept ausreichend und es wird der Auswertungsaufwand extrem vereinfacht; vor allem im Bereich der Variablensubstitution.

Um die Bedingung sicherzustellen, wird beim Initialisieren der Regelbasis der entsprechende Graph aufgebaut. Bei jeder Regel, die hinzugefügt wird, wird die Sortierung neu ermittelt. Sobald durch eine Regel die Sortierung nicht mehr ermittelt werden kann, ist die Azykliziätsbedingung verletzt und man kann die letzte Regel als erste fehlerhafte Regel ausgeben.

Diese Vorgehensweise erleichtert das Auffinden von Regeln, die die Bedingungen verletzen.

Der Graph wird aus allen Prädikaten, die in den Regeln vorkommen, aufgebaut. Konstanten werden ignoriert, genauso wie Ausgabefunktionen. Letztere haben ja auch keinen Einfluss auf die Arbeitsweise des Regelsystems selbst und Konstanten können nicht auf der rechten Seite einer Regel angeführt werden.

Ein Prädikatname identifiziert eindeutig einen Knoten im Graphen. Auf die Stelligkeit wird nicht weiter eingegangen. Ein Knoten eines Graphen führt eine Liste aller Regeln, die das entsprechende Prädikat in der rechten Seite verwenden.

Zusätzlich werden noch folgende Daten verwendet:

- `nodeList(graph)` liefert die Liste aller Knoten des Graphen, ohne bestimmte Reihenfolge.
- `inList(node)` liefert eine Liste aller Knoten, von denen eine Kante zum betrachteten Knoten führt. Das sind also alle Prädikate, die in der linken Seite einer Regel vorkommen, in deren rechten Seite das betrachtete Prädikat vorkommt.
- `outList(node)` liefert eine Liste aller Knoten, die vom betrachteten Knoten aus direkt erreichbar sind. Das sind alle Prädikate, die in der rechten Seite von Regeln vorkom-

men, welche das betrachtete Prädikat in der linken Seite enthalten.

- `nodeRules(node)` liefert eine Liste aller Regeln, die in der rechten Seite den Knoten enthalten.
- `node.active` und `node.inputCount` sind Hilfsstrukturen für den Sortieralgorithmus. *active* sind alle Knoten, die noch nicht „einsortiert“ wurden, und *inputCount* ist eine Variable, die die Anzahl der aktiven Knoten in der *inList*() angibt.

Mithilfe dieser Daten wird der Algorithmus definiert, der bei der Definition einer neuen Regel die Reihenfolge der Regeln neu ermittelt, zu sehen in Abbildung 8.4. Der Algorithmus ist eine Erweiterung der Suche nach der topologischen Reihenfolge in einem azyklischen gerichteten Graphen.

Algorithmus : Neue Reihenfolge von Regeln ermitteln

Input : graph, newRule
Result : ruleList (=sortierte Liste von Regeln), graph (erweitert)

```
// die neue Regel zum Graphen hinzufügen
addRule(graph,newRule);
// Initialisieren für die Sortierung
foreach node ∈ nodeList(graph) do
    node.inputCount ← |inList(node)|;
    node.active ← true;
// Erzeugen der neuen sortierten Regelliste
ruleList ← ();
while hasActiveNodes(graph) do
    workNode ← nil;
    foreach node ∈ nodeList(graph) do
        if node.inputCount = 0 ∧ node.active then
            workNode ← node;
            break;
    if workNode = nil then
        // die neue Regel verletzt die Azyklizitätsbedingung
        error();
    // den workNode deaktivieren
    foreach neighbour ∈ outList(workNode) do
        decrement(neighbour.inputCount);
    workNode.active ← false;
    foreach rule ∈ nodeRules(workNode) do
        if ¬rule.used then
            append(ruleList,rule);
            rule.used ← true;
```

Abbildung 8.4: Algorithmus zur Ermittlung der neuen Regelreihenfolge bei Einfügen einer zusätzlichen Regel

Substitution

Bei der Auswertung einer Regel müssen die vorkommenden Variablen substituiert werden. Wie bereits beschrieben, wird dabei von links nach rechts vorgegangen: die Komponenten der linken Seite einer Regel werden von links nach rechts abgearbeitet. Dabei wird für jede Komponente überprüft, ob es entsprechende Fakten in der Faktenbasis gibt, und welche das sind. Gibt es zutreffende Fakten, dann werden für die vorkommenden Variablen alle passenden Substitutionen ermittelt. Sind bereits Substitutionen vorhanden („von weiter links“), dann werden diese nur weiter eingeschränkt.

Wie ebenfalls bereits beschrieben, werden negierte Regelkomponenten anders behandelt als nicht negierte.

In der rechten Seite einer Regel, werden Variablen nur mit den von links ermittelten Substitutionen verwendet. Die Prädikate der rechten Seite werden für jede ermittelte Substitution der Faktenbasis hinzugefügt.

Bei der schrittweisen Auswertung einer Regel wird bei jeder Komponente entschieden, welche Methode angewendet wird, um die zutreffenden Substitutionen zu ermitteln. Bei Konstanten wird eine einfache Suche in der Faktenbasis verwendet, die keinen Einfluss auf die Substitution hat. Bei positiven Prädikaten wird der Algorithmus in Abbildung 8.5 verwendet, bei negativen Prädikaten der Algorithmus in Abbildung 8.6. In beiden Fällen wird auch ermittelt, ob die Komponente überhaupt zutrifft. Trifft die Komponente nicht zu (d.h. es gibt keine Entsprechungen in der Faktenbasis), dann trifft die Regel nicht zu und die Auswertung wird nicht fortgesetzt.

In den beiden Algorithmen kommt jeweils eine Liste von Substitutionen (`substList`) zum Einsatz. Diese Liste wird von Komponente zu Komponente weitergereicht und enthält einzelne Substitutionsobjekte. Ein Substitutionsobjekt (bzw. eine Substitution) enthält $(variable, wert)$ Paare. D.h. es wird den unterschiedlichen Variablen je ein Wert zugeordnet. Kann eine Variable unterschiedliche Werte annehmen, sodass die Regelkomponente gültig ist, werden mehrere Substitutionsobjekte erzeugt.

Die `substList` beinhaltet also immer die Substitutionen der Komponenten, die bereits ausgewertet wurden.

Eine Substitution muss nicht vollständig sein. Eine Variable x, die zwar in der betrachteten Regel verwendet wird, muss in einer Substitution nicht enthalten sein. Erst wenn alle Komponenten ausgewertet wurden, ist sichergestellt, dass jede Substitution aus der `substList` auch alle vorkommenden Variablen enthält.

Mit dieser Liste der ermittelten Substitutionen kann dann die rechte Seite der Regel ausgewertet werden (sofern die Regel nach Auswertung der linken Seite als zutreffend befunden wurde). Für jede verbleibende Substitution in der `substList` werden die entsprechenden Prädikate aus der rechten Seite der Faktenbasis hinzugefügt.

Besteht die rechte Seite der Regel z.B. aus $A(x) \wedge B(x)$, und es sind 2 Substitutionen in der `substList` mit den Inhalten $\{(x, const1), (x, const2)\}$, dann werden $A(const1)$, $B(const1)$, $A(const2)$, $B(const2)$ in die Faktenbasis aufgenommen.

Außerdem werden folgende Funktionen in den Algorithmen verwendet, die vielleicht näher erläutert werden sollten:

- `findMatches(FactBase,Template)` sucht alle zutreffenden Prädikate in der Faktenbasis `FactBase` nach der Vorlage `Template`. Die Vorlage ist ein teilweise spezifiziertes Prädikat. D.h. die Parameter können Konstanten oder (optional typisierte) Va-

```
Input : pred (Prädikat der betrachteten Komponente), substList, FactBase
Result : substList (erweitert), matchResult (true/false, Komponente trifft zu)

newSubstList ← ();
foreach subst ∈ substList do
    // Initialisieren eines Templates, das zur Suche in der
       Faktenbasis verwendet wird.
    template ← pred;
    // varList() liefert alle Variablen eines Prädikats
    foreach var ∈ varList(pred) do
        if subst.contains(var) then
            // ersetze Variable im Template mit Wert aus Subst.
            template.setValue(var, subst.getValue(var));
    foreach found ∈ findMatches(FactBase, template) do
        newSubst ← subst;
        compareVariables(pred, found, newSubst);
        addToList(newSubstList, newSubst);
if |newSubstList| = 0 then
    matchResult ← false;
    return;
substList ← newSubstList;
matchResult ← true;
```

Abbildung 8.5: Substitutionsalgorithmus für positive Prädikat-Regelkomponenten

riablen sein. $ref(group,x)$ ist ein solches Template-Prädikat, das beispielsweise auf die Prädikate $ref(group,C1)$ und $ref(group,C2)$ zutreffen würde. Letztere würden von der Funktion `findMatches()` geliefert, wenn sie in der `FactBase` enthalten sind.

- `compareVariables(pred1,pred2,subst)` vergleicht die beiden Prädikate `pred1` und `pred2` und nimmt die Variablen aus `pred1`, die in `pred2` als Konstanten angeführt sind in die Substitution `subst` auf.

  Ein Beispiel soll Klarheit verschaffen: Wenn $pred1 = A(x,y,const1,const2)$ und $pred2 = A(cccc,y,const1,const2)$, dann wird die Zuordnung $x = cccc$ in die Substitution `subst` aufgenommen. y hingegen wird nicht aufgenommen, weil in $pred2$ keine Konstante dafür ersetzt wurde.

8.2.4 Entwicklung des Systems

Bei dem Entwurf des Regelsystems wurden verschiedene Überlegungen angestellt, die schließlich zu dem hier beschriebenen System geführt haben.

Ausgegangen wurde dabei von einer relativ fixen Vorstellung, was das Regelsystem können soll, und welche Regeln typischerweise zum Einsatz kommen sollten.

Bei konkretem Ausformulieren der Regeln, konnte schnell festgestellt werden, dass man mit einfachen Aussagen nicht auskommt: Prädikate und Variablen mussten dazu definiert werden.

```
Input : pred (Prädikat der betrachteten Komponente), substList, FactBase
Result : substList (nur eingeschränkt), matchResult (true/false, Komponente trifft zu)

newSubstList ← ();
foreach subst ∈ substList do
    // Initialisieren eines Templates, das zur Suche in der
       Faktenbasis verwendet wird.
    template ← pred;
    // varList() liefert alle Variablen eines Prädikats
    foreach var ∈ varList(pred) do
        if subst.contains(var) then
            // ersetze Variable im Template mit Wert aus Subst.
            template.setValue(var, subst.getValue(var));
    if |findMatches(FactBase, template)| = 0 then
        // Im negierten Fall wird nur getestet, ob es
           zutreffende Prädikate gibt.
        // Gibt es keine, ist die Substitution zutreffend.
        addToList(newSubstList, subst);
if |newSubstList| = 0 then
    matchResult ← false;
    return;
substList ← newSubstList;
matchResult ← true;
```

Abbildung 8.6: Substitutionsalgorithmus für negierte Prädikat-Regelkomponenten

Damit war es möglich, allgemeine Regeln zu definieren, welche auf einen Vergleich von zwei SQL Statements ausgelegt sind. Die Regel

$$reference(x) \wedge \neg input(x) \rightarrow missingFeature(x)$$

definiert beispielsweise ein neues Prädikat, das für alle fehlenden Features im Input-Statement gesetzt wird. Bei weiteren Überlegungen wurde klar, dass es möglich sein muss, die Existenz einzelner Prädikate zu überprüfen. Regeln der Form

$$\neg\exists x\ missingFeature(x) \rightarrow completeFeatures()$$

wurden benötigt.

Die konsequente Fortführung der Konzeption ergab bald das Bild eines vollständigen logisches Systems mit Resolution von Antwortsubstitutionen. Das war allerdings nicht im Sinne der Aufgabe. Ziel war es schließlich, ein möglichst simples Regelsystem zum Einsatz zu bringen, einfach zu dem Zweck der besseren Wartbarkeit des Bewertungskonzepts im Laufe der Zeit.

Das System sollte weder in die Richtung gehen, Prolog nachzuimplementieren, noch mit aufwändiger Resolution beschäftigt sein. Schließlich müssen die Bewertungen der Usereingaben sofort, möglichst weniger als eine Sekunde verzögert, vorliegen.

Vereinfachungen mussten gefunden werden:

- Die Auswertung wird nur *linear* durchgeführt. Dabei wird jede Regel nur ein einziges Mal ausgewertet.
- Der Anspruch eines logischen Systems wurde fallen gelassen. Der Gedanke geht jetzt in Richtung „matching“: Wenn die linke Seite zutrifft (*matcht*), wird die rechte Seite ausgeführt.
- Das Typsystem wurde zur einfachen Beschreibung solcher öfter vorkommender Konstrukte, und mit dem Gedanken, dass diese Form weniger Performance bei der Auswertung kostet, eingeführt. Die implizite Typdefinition der Form

$$\rightarrow isOfType(structural, group)$$

$$missingFeature(x) \wedge isOfType(structural, x) \rightarrow missingStructure()$$

wird so verkürzt auf

$$type(structural, group)$$

$$missingFeature(x/structural) \rightarrow missingStructure().$$

Wobei die Typdefinitionen in eigener Syntax festgelegt werden.

Andere Überlegungen

Eine andere Überlegung zur Vereinfachung des Systems ging in die Richtung, das Regelsystem in mehreren Stufen aufzubauen; so, dass jede einzelne Stufe mit einfacheren Regeln arbeiten könnte. Stufen könnten z.B. sein: *Vorbereitung von Features*, *Vergleich von Features aus zwei Statements* und *Auswerten der Ergebnisse*.

Wenn die Vergleichsstufe nur zum Vergleich von zwei SQL Statements verwendet wird, dann können die Regeln in dem Bereich voraussichtlich sehr spezialisiert aber einfach ausfallen.

Bei der linearen Abarbeitung der Regeln besteht außerdem die Möglichkeit, die Regeln explizit mit einer Reihenfolge zu versehen, die vom User, im Gegensatz zur impliziten Ermittlung der Reihenfolge beim Initialisieren der Regelbasis, direkt angegeben wird.

Dieser Ansatz hätte aber keinen Vorteil gegenüber dem gewählten, außer etwas weniger Implementierungsaufwand.

8.3 Implementierung

8.3.1 Datenstrukturen

In Abbildung 8.7 ist eine Übersicht der Klassen dargestellt, welche den Kern des Regelsystems ausmachen. Es handelt sich um einen Auszug der Klassen, die für die Repräsentation der Faktenmenge und der Regeln relevant sind. Es werden in den Klassen weiters nur die für das Interface der Klasse relevanten Methoden dargestellt (d.h. vor allem: private Methoden entfallen, Konstruktoren werden in dieser Übersicht ebenfalls nicht angeführt) um das Diagramm übersichtlich zu halten.

Die Strukturen teilen sich im wesentlichen in zwei Bereiche: Regeln und Fakten. In den folgenden Abschnitten wird auf einzelne wesentliche Aspekte aus beiden Bereichen eingegangen.

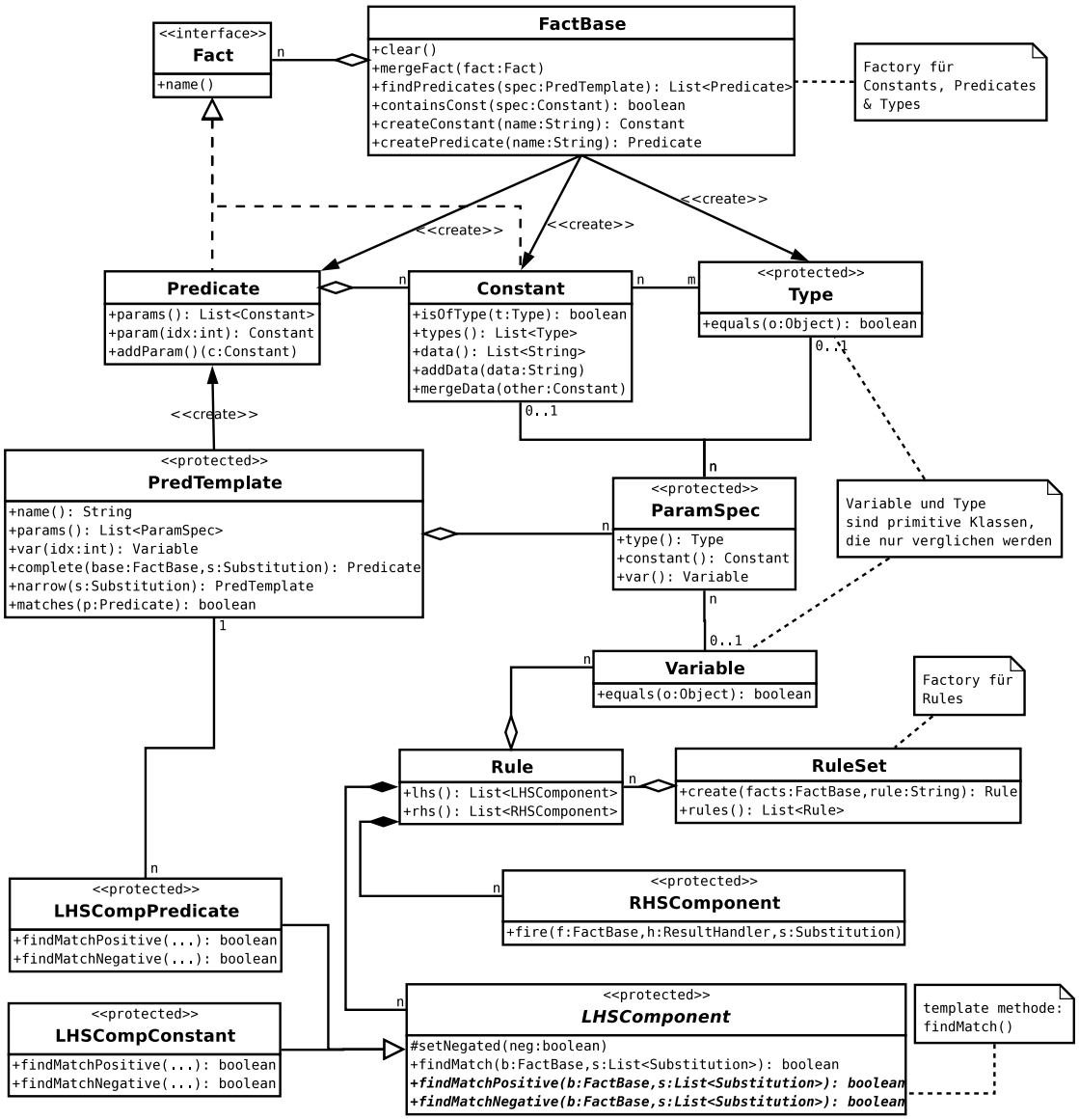


Abbildung 8.7: Datenstrukturen - Kern des Regelsystems

Facts

Ausgehend von der `FactBase` Klasse fallen in diesen Bereich die Klassen und Interfaces: `Fact, Predicate, Constant` und `Type`. `FactBase` ist Factory für die drei Klassen `Constant, Predicate` und `Type`. D.h. für diese Klassen werden nur Objekte über die entsprechenden Methoden in `FactBase` instanziert. Das ermöglicht eine effiziente Verwaltung von Fakten und Typen innerhalb der `FactBase`.

Die Hauptaufgabe der `FactBase` ist das Verwalten einer Liste von Fakten. Fakten werden implementiert durch das Interface `Fact` und zwar von den beiden Klassen `Predicate` und `Constant` (ein Faktum kann ein Prädikat oder eine Konstante sein, siehe Abschnitt 8.2.1). `Predicate` ist aufgebaut aus einem Namen und einer Liste von Parametern, welche durch Konstanten (`Constant`) abgebildet werden. Die Klasse `Predicate` ist die Implementierung von konkreten Ausprägungen eines Prädikats. Das ist der Grund, warum hier keine Variablen vorkommen. Für die Darstellung eines Prädikats mit Variablen wird die Klasse `PredTemplate` verwendet.

Typen von Konstanten werden durch die Klasse `Type` dargestellt. Ein Typ besteht nur aus seinem Namen. Ein `Type` Objekt wird daher nur zum Vergleich mit anderen `Type` Objekten verwendet. Die Typen und Typ-Konstanten Zuordnungen werden in der `FactBase` intern verwaltet.

Die Klasse `Constant` stellt außerdem das Interface zu den Datenlisten einer Konstante zur Verfügung: `data(), addData()` und `mergeData()`.

Rules

Die Klasse `RuleSet` stellt einerseits eine Sammlung von Regeln dar und ist andererseits Factory für die Regeln (Klasse `Rule`). Es wird hier also das selbe Prinzip angewendet, wie für `FactBase` und `Fact`.

`Rule` ist aufgebaut aus einer Liste von Komponenten auf der linken und rechten Seite der Regel. Komponenten auf der linken Seite werden implementiert von der Klasse `LHSComponent` und entsprechend auf der rechten von `RHSComponent`.

Das Interface von `RHSComponent` besteht im Wesentlichen nur aus einer Methode, die aufgerufen wird, wenn die Regel zutrifft: `fire()`. Die Komponente sorgt dann dafür, dass gemacht wird, was gemacht werden muss (im Normalfall ist das: Eintragen eines Prädikats in der Faktenmenge).

Das Interface von `LHSComponent` stellt die Methode `findMatch()` zur Verfügung, die beim Auswerten der Regel aufgerufen wird, um festzustellen, ob die Komponente mit der gegebenen Faktenmenge zutrifft oder nicht. `setNegated()` wird gesetzt beim Aufbau der Regel, wenn die Regelkomponente negiert verwendet wird.

`LHSComponent` ist eine abstrakte Klasse, von welcher unterschiedliche Klassen abgeleitet werden, je nachdem welche Komponente auf der linken Seite der Regel vorkommt: eine Konstante (`LHSCompConstant`) oder ein Prädikat (`LHSCompPredicate`). Beim Aufbau der Regel wird ein Objekt der konkreten Klasse initialisiert. Die `findMatch()` Methode von `LHSComponent` wird als *template Methode* ausgeführt: je nachdem, ob die Komponente negiert oder nichtnegiert verwendet wird, wird eine der beiden abstrakten Methoden `findMatchNegative()` oder `findMatchPositive()` aufgerufen. In der eigentlichen Implementierung der beiden Methoden können damit die unterschiedlichen Aspekte der

Substitution bei positiven oder negativen Komponenten berücksichtigt werden.

Die Klasse `PredTemplate` wird verwendet um in einer Prädikatkomponente das Prädikat mit Variablen und Konstanten zu repräsentieren. Die Klasse besteht aus dem Namen des Prädikats und einer Liste von Parameterspezifikationen (Klasse `ParamSpec`). Eine Parameterspezifikation kann entweder eine Konstante sein, oder eine Variable, die zusätzlich noch mit einem Typ eingeschränkt sein kann. Die Klasse ParamSpec hat entsprechend je eine optionale Referenz auf die Klassen `Constant, Variable` und `Type`.

`PredTemplate` stellt die beiden Methoden `complete()` und `narrow()` zur Verfügung, die für die Substitution relevant sind. `complete()` liefert ein `Predicate` indem die Variablensubstitution herangezogen wird und die Variablen durch die entsprechenden Konstanten ersetzt werden. `narrow()` macht im Grunde dasselbe, nur dass hier nicht davon ausgegangen wird, dass alle Variablen substituiert sind. Demnach wird hier wieder ein `PredTemplate` geliefert.

8.3.2 Initialisierung

Beim Initialisieren des Regelsystems werden Fakten und Regeln erzeugt. Dabei werden die `Fact` Instanzen von der `FactBase` erzeugt und verwaltet und die `Rule` Instanzen analog vom `RuleSet`. Die Initialisierung von Fakten (`Constant` und `Predicate`) geschieht schrittweise, indem zuerst die Objekte mit dem Namen (in der Factory Methode) erzeugt und dann weiter durch die Methoden `addParam()` im Prädikat bzw. `addData()` in der Konstante spezifiziert werden.

Anders bei den Regeln: Hier wird ein `Rule` Objekt mit einer einzigen Methode durch Parsen eines Regel-Strings erzeugt. Die Methode `RuleSet.create()` bekommt einen String und liefert eine fertig aufgebaute Regel zurück. Implementiert ist der Parser allerdings in der `Rule` Klasse selbst. Das `RuleSet` ruft nur den Constructor von `Rule` auf, der von außen nicht zugängig ist. Der Regelparser selbst ist auf möglichst einfache Weise mithilfe von Regular Expressions implementiert. Diese Designentscheidung hat (wenn auch nur geringfügigen) Einfluss auf die Syntax einer Regelrepräsentation: Eine Regel muss ohne Aufwand mit Regular Expressions zu parsen sein. Es wurden daher vor allem Darstellungsformen gewählt, die jeden Typ von Objekt eindeutig durch entsprechende Präfixes identifizieren: „#“ für Konstanten, „>“ für Ausgabefunktionen, „:“ für Funktionen (diese werden weiter unten definiert, siehe Abschnitt 8.3.4).

Zur Übersicht wird hier die Syntax in EBNF angegeben (inklusive *Funktionen*):

Rule	→	*LHS* '**->**' *RHS*
LHS	→	*LComp* ?('**+**' *LComp*)*
RHS	→	*RComp* ('**+**' *RComp*)*
LComp	→	*Const* | *Pred* | *Func*
Const	→	'**#**' *word*
Pred	→	*word* '**(**' *PParams* '**)**'
PParams	→	*Const* | *Var* ('**,**' *PParams*)*
Var	→	*word* ('**/**' *Type*)?
Type	→	*word*
Func	→	'**:**' *word* '**(**' *FParams* '**)**'
FParams	→	*Var* ('**,**' *FParams*)*
RComp	→	*Pred* | *OutFunc*
OutFunc	→	'**>**' *word* '**(**' *OFParams* '**)**'

OFParams wird nicht näher festgelegt, sondern ist ein beliebiger String ohne ')'.

Das Sortieren der Regeln und Überprüfen der Azyklizität ist im `RuleSet` implementiert. Die Methode `create()` legt eine neue Regel an, fügt sie in den Graphen, der intern verwaltet wird, ein und prüft anschließend ob der Graph immer noch azyklisch ist (wie beschrieben im Algorithmus in Abbildung 8.4).

8.3.3 Implementierung der Algorithmen

In Abbildung 8.8 wird ein Überblick über die Klassen dargestellt, die bei der Verarbeitung der Regeln (Auswertung) relevant sind.

Die bereits beschriebenen Klassen `FactBase` und `RuleSet` bilden die Basisdaten für die Auswertung. In der Klasse `Engine` ist der Algorithmus für die lineare Auswertung der Regeln implementiert (Algorithmus der Vorwärtsverkettung, beschrieben in Abschnitt 8.2.3).

Auswertung

Bei der Auswertung werden die Regeln einzeln in ihrer Reihenfolge betrachtet (wie sie in der Liste `rules()` vorliegen) und überprüft. Zum Überprüfen einer Regel gegen die FactBase werden deren Komponenten der linken Seite einzeln herangezogen (von links nach rechts, d.h. ebenfalls in der vorgegebenen Reihenfolge). Sobald eine Komponente nicht zutrifft, wird die Regel als nicht zutreffend betrachtet und die Verarbeitung geht über zur nächsten Regel. Das Zutreffen einer Komponente wird mit der Methode `findMatch()` überprüft. Diese Methode sucht in der Faktenmenge nach zutreffenden Fakten (je nach Implementierung - `Constant` oder `Predicate` - werden dabei die `FactBase` Methoden `findPredicates()` oder `containsConst()` aufgerufen).

Trifft eine Regel zu, wird für jede Komponente der rechten Seite die Methode `fire()` aufgerufen, die dafür sorgt, dass im Fall eines Prädikats, das Prädikat zur `FactBase` hinzugefügt oder im Fall einer Ausgabefunktion der entsprechende Handler aufgerufen wird.

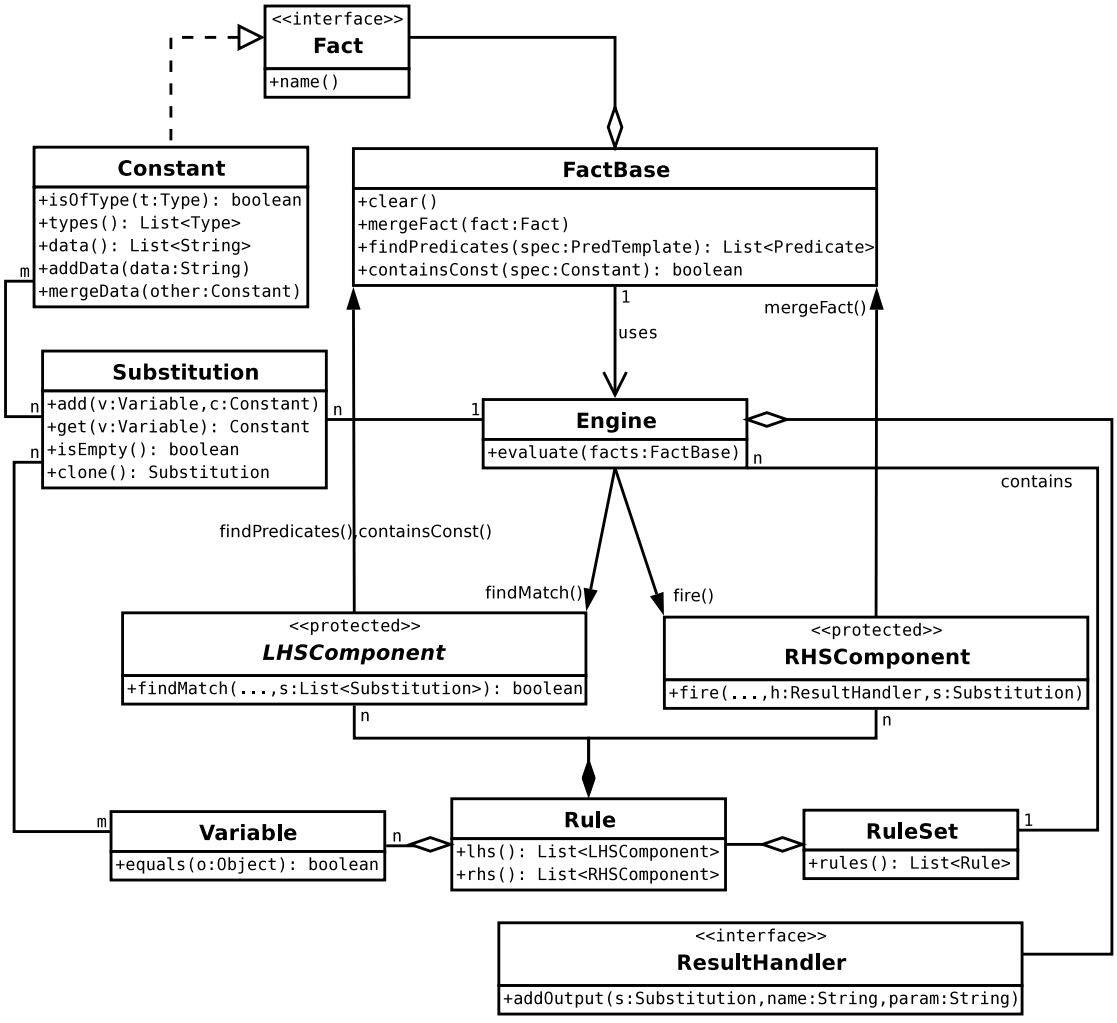


Abbildung 8.8: Relevante Klassen für die Auswertung

Substitution

Die Variablensubstitution wird mithilfe der Klasse `Substitution` und einer daraus aufgebauten Liste implementiert. Die Klasse `Substitution` speichert eine mögliche Variablenbelegung. Dabei wird die Variable und der entsprechende Wert gespeichert. Eine Substitution muss nicht vollständig sein. D.h. es müssen nicht alle Variablen einer Regel enthalten sein. Beim Auswerten einer Regel wird die Liste der Substitutionen von Komponente zu Komponente weitergereicht. Bei jedem Aufruf von `findMatch()` einer Komponente werden die Substitutionen potenziell mit zusätzlichen Variablen erweitert oder durch Testen der vorhandenen Substitutionen gegen die `FactBase` eingeschränkt.

Wird eine Substitution mit einer neuen Variable erweitert, so entstehen dadurch soviele neue Objekte von `Substitution` wie es mögliche Belegungen der neuen Variable gibt.

Wird auf eine Variablenbelegung getroffen, die in der Komponente nicht gültig ist (d.h. nicht in der Faktenmenge enthalten), so fällt diese Substitution aus der Liste raus.

Jede Implementierung der Komponenten kann die Einschränkung und Erweiterung der Substitutionsliste selbst implementieren (auch abhängig von negiert oder nicht). Die Implementierung der Komponente für Konstanten beispielsweise ignoriert die Substitutionsliste gänzlich.

Die Implementierung der Substitution für Prädikatkomponenten ist in den Abbildungen 8.6 und 8.5 zu sehen.

8.3.4 Eine Erweiterung: Funktionen

Wie sich beim Entwurf des regelbasierten Systems bereits herausgestellt hat, was aber dort nicht beschrieben wurde, sind im vorliegenden Fall Konstanten und Prädikate nicht ausreichend. Konkret wird eine Möglichkeit benötigt, die Datenlisten von Konstanten zu manipulieren. Um dieses Problem zu adressieren und um gleichzeitig die Flexibilität zu erhöhen, wurde das Konzept der erweiterbaren Funktionen ins Leben gerufen.

Eine Funktion ist eine mögliche Komponente auf der linken Seite einer Regel, die ähnlich aufgebaut ist, wie ein Prädikat. Von Seiten der Syntax ist eine Funktion ein Prädikat, das mit einem „:“ startet.

Die Implementierung von Funktionen wird in Abbildung 8.9 dargestellt.

Es wird eine neue Ableitung von `LHSComponent` implementiert: `LHSCompFunction`. Diese Klasse verwendet eine Reihe von `FunctionHandler` Objekten um aufgrund der in der Regel vorkommenden Namen die entsprechenden Methoden zu suchen. Bei der Auswertung wird das Interface `Method` verwendet, um die Methodenaufrufe abzuwickeln.

Wird eine Funktionskomponente ausgewertet, dann wird für jede Substitution einmal die `call()` Methode im `Method` Interface aufgerufen. Dabei werden vorher mit der Methode `bind()` die Werte der Variablen an die Funktion übergeben. Nach dem Aufruf können geänderte Variablenwerte mit der Methode `get()` gelesen werden.

Mit diesem Interface kann eine Funktion Manipulationen an den übergebenen `Constant` Objekten vornehmen oder neue `Constant` Objekte erzeugen. Die Methode `call()` liefert außerdem zurück, ob die Substitution zutrifft oder nicht. Dadurch wird auch die Substitutionsliste eingeschränkt.

Beispiele für Funktionen, sind die bis dato benötigten Funktionen `:sameFeature(x,y)` und `:diffData(x,y,z)`. Erstere ist dann zutreffend, wenn die beiden in x und y übergebenen Konstanten denselben Namen haben. Zweitere trifft zu, wenn die beiden in x und y

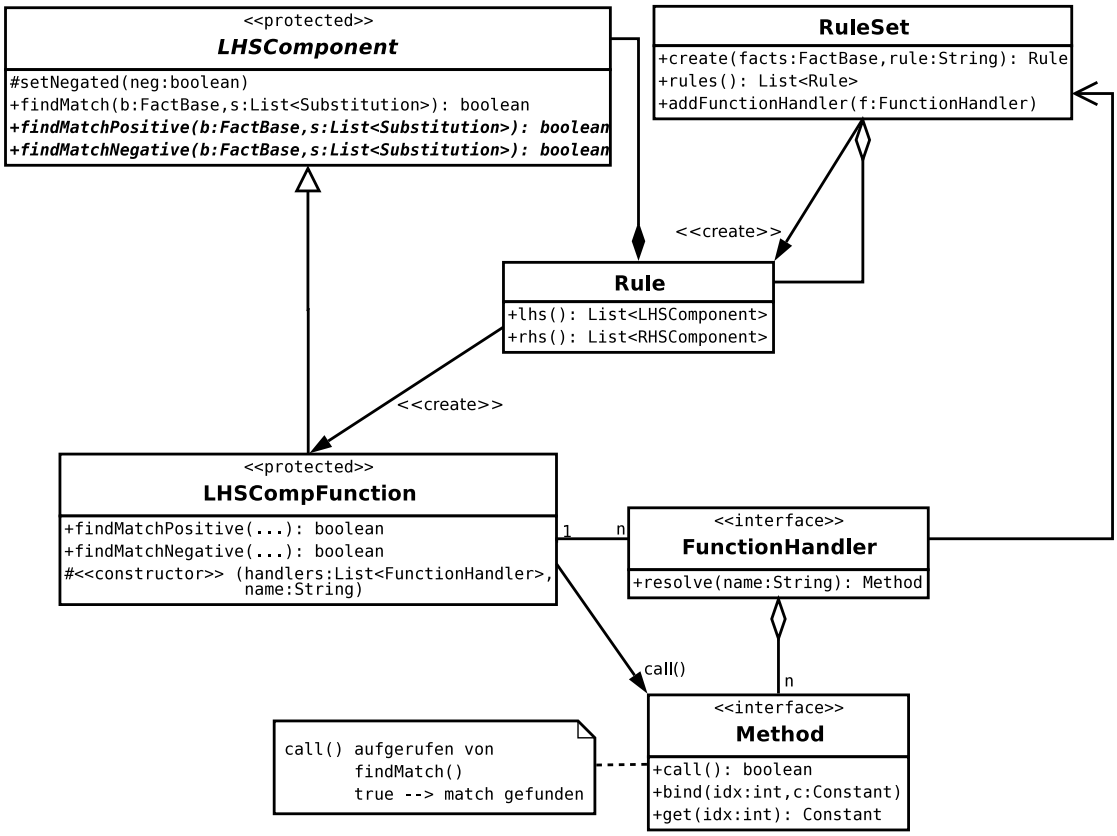


Abbildung 8.9: Implementierung von Funktionen

übergebenen Konstanten eine unterschiedliche Datenliste haben. Dabei wird die Differenz der Datenlisten in einer neuen Konstante z zurückgeliefert.

8.3.5 Einsatz im Bewertungssystem

Wie bereits zu Anfang erwähnt, wurde das Regelsystem so entworfen, dass es auch unabhängig vom SQL Bewertungssystem anderweitig zum Einsatz kommen könnte. In den bisherigen Beschreibungen haben sich nur wenige Beispiele auf den Einsatz hier als Bewertungssystem bezogen.

In Abbildung 8.10 wird gezeigt, wie das allgemeine Regelsystem konkret im System zur Bewertung von SQL Statements verwendet wird. Diese Abbildung veranschaulicht, welche Klassen als Interface zum Regelsystem verwendet werden.

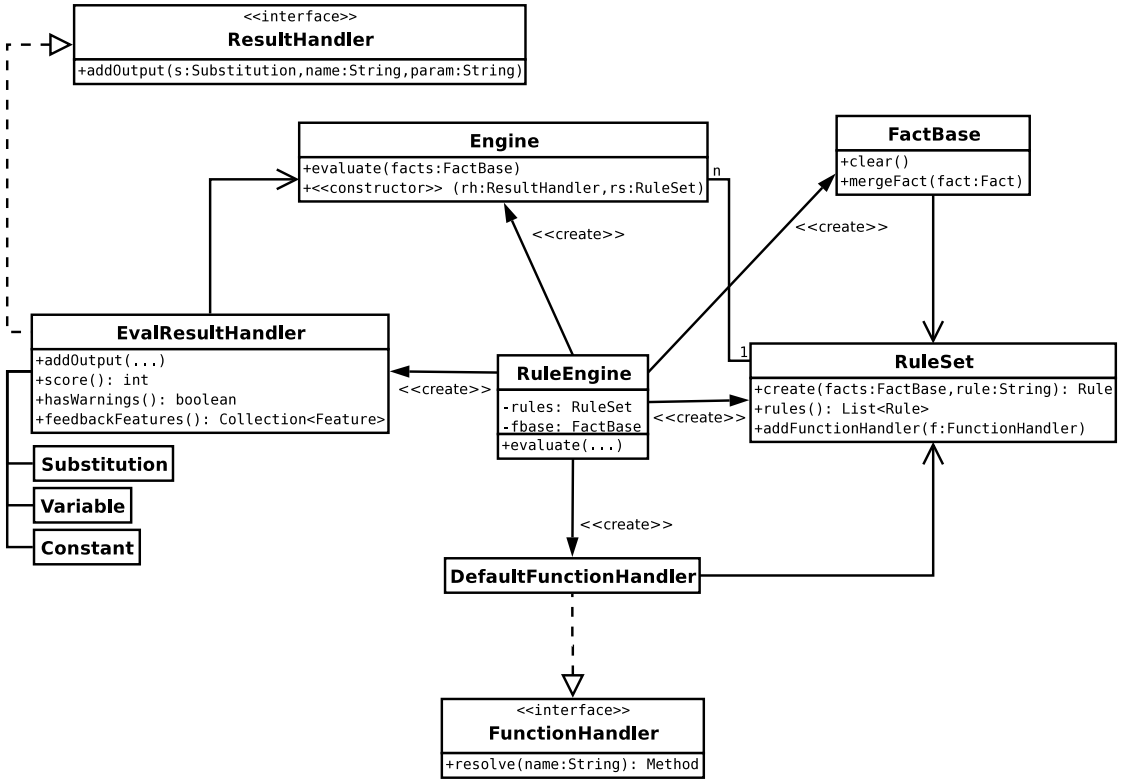


Abbildung 8.10: Verwendung der Kernstrukturen im Bewertungssystem

Außerhalb des Kernsystems werden die Klassen `RuleEngine, EvalResultHandler` und `DefaultFunctionHandler` implementiert. Dabei sind die beiden Handler Klassen jeweils Implementierungen von zwei Handler Interfaces. Die `RuleEngine` Klasse ist der Anlaufpunkt für das Bewertungssystem innerhalb der Applikation und übernimmt das Erzeugen und Verwalten des Regelsystems.

Von hier aus werden alle für das Regelsystem relevanten Klassen erzeugt, verwaltet und die entsprechenden Methodenaufrufe weitergeleitet. Nach außen ist damit kein zusätzliches Wissen über die Implementierung des Regelsystems nötig. Die Methode `RuleEngine.evaluate()`

sorgt für die Auswertung der SQL Features und liefert ein `EvalResultHandler` Objekt zurück, das vorher intern erzeugt worden ist und als `ResultHandler` für die `Engine` gesorgt hat.

Der `EvalResultHandler` interpretiert die Aufrufe der Ausgabefunktionen und stellt ein Interface für die Ergebnisse, wie z.B. die erreichte Punktezahl in der Methode `score()`, zur Verfügung. Vom `EvalResultHandler` aus wird auf die Klassen `Substitution`, `Variable` und `Constant` des Kernsystems zugegriffen (weil die entsprechenden Objekte beim Aufruf einer Ausgabefunktion mitgegeben werden).

8.3.6 Optimierungsmöglichkeiten

Abgesehen von den Maßnahmen, die in der Entwicklung des Systems berücksichtigt wurden (wie z.B. das Konzept der linearen Auswertung) wurde in der Implementierung selbst nicht weiter auf Performanceoptimierungen geachtet.

Das Ergebnis im Prototyp ist ausreichend in der Performance, zumal auch die verwendete Regelmenge und die Möglichkeiten der Fakten in der Faktenmenge relativ gering sind (ca. 40 Regeln im Testsystem und im Normalfall nicht mehr als 100 Fakten).

Eine mögliche, relativ einfache Verbesserung würde sich dennoch erzielen lassen, wenn die Menge von Stringvergleichen, die in der Auswertung gemacht werden, geändert würde auf Integervergleiche. Die Stringvergleiche ergeben sich aus den Vergleichen der Namen der Prädikate, Konstanten und Variablen. Wenn man diese Klassen jeweils beim Erzeugen mit einem Index versieht, könnte man bei der Auswertung sämtliche Vergleiche nach Indizes anstellen und eine entsprechende Verbessserung ist zu erwarten.

Dieser Ansatz wurde im Prototyp noch nicht umgesetzt, weil bisher die Notwendigkeit nicht gegeben war. Eine Umsetzung für das produktive System ist allerdings angedacht.

Eine andere Überlegung wäre die Optimierung der Auswertung durch entsprechende Strukturen in den Regeln.

8.4 Arbeitsweise des Bewertungssystems

In diesem Abschnitt soll das Bewertungssystem als Ganzes dargestellt werden. Die bisher beschriebenen Komponenten *Regelsystem* und *Resultset Vergleich* sind darin die wesentlichen Bestandteile. Was um diese beiden herum noch passiert, wird in Abbildung 8.11 als logischer Überblick dargestellt.

Hier ist zu erkennen, wie ausgehend von der eingegebenen Query und einer Referenzquery (links) die Ergebnisse auf der rechten Seite erarbeitet werden (Vergleichstabelle, Bewertung, Feedback und Beispielstatus).

Eine wesentliche Rolle in diesem Prozess spielen die sogenannten *Features*. Die Features sind Eigenschaften der Queries, der Beispielangabe oder der Vergleichstabelle, die als Grundlage bei der Bewertung herangezogen werden. Ein Feature besteht aus einem Namen und zusätzlichen Daten, ähnlich wie die Konstanten im Regelsystem. Tatsächlich werden die Features auch als Fakten im Regelsystem abgelegt. Dabei kann allerdings aus einem Feature auch ein Prädikat erzeugt werden, nicht nur Konstanten.

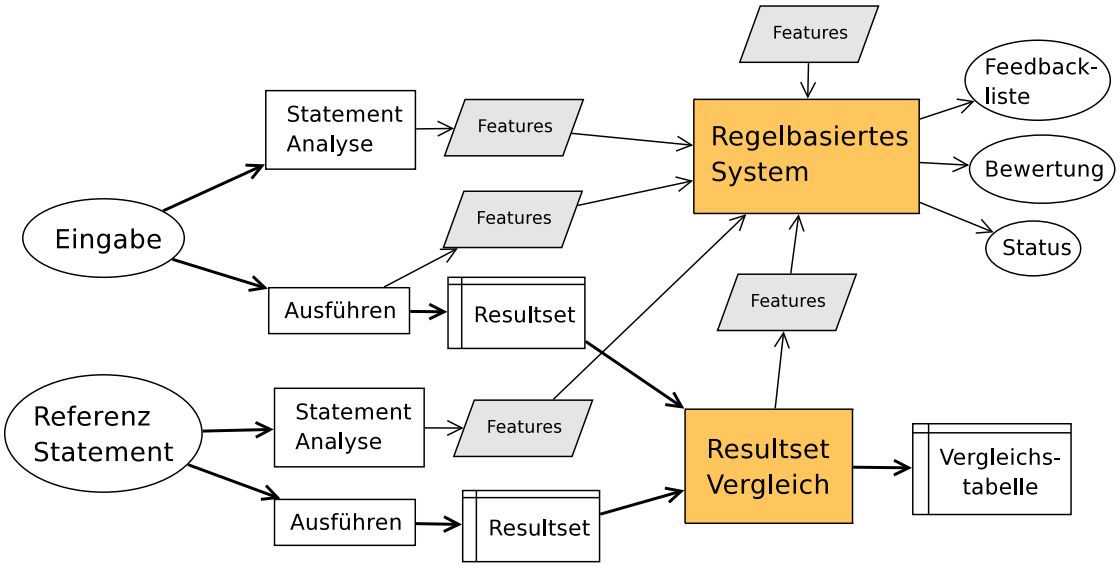


Abbildung 8.11: Logischer Überblick des Bewertungssystems

Die Features bilden also damit die gesamte Faktenmenge im Regelsystem, aufgrund der das Regelsystem bewertet.

Features werden aus den unterschiedlichen Quellen zusammengeführt. Das sind:

- Beim Ausführen der Eingabequery entstehen Features, die besagen, ob die Query ausführbar ist, oder welcher Fehler vom Datenbanksystem gemeldet wird.
- Sowohl aus der Eingabequery als auch aus der Referenzquery werden Features aus dem Statement selbst *extrahiert*. Wie das gemacht wird, wird in Abschnitt 8.4.1 beschrieben. Es sei nur gesagt, dass dabei strukturelle Informationen über die Statements gesammelt werden.
- Der Resultset Vergleich liefert Features, welche die Unterschiede der beiden Ergebnisse beschreiben. Dabei wird z.B. angegeben, ob und wieviele Zeilen zuviel im Input sind oder welche Spalten fehlen oder zuviel sind.
- Schließlich kommen noch Features dazu, welche z.B. aus der Aufgabendefinition kommen. Der Schwierigkeitsgrad der Aufgabe kann so z.B. mitgegeben werden. Oder es kann die Information mitgegeben werden, ob es sich um Übungsbeispiele oder Prüfungsbeispiele handelt, etc.

In Abbildung 8.12 wird ein Überblick über die wesentlichen Klassen im Bewertungssystem gegeben.

Darin sind auch die wesentlichen Quellen für Features ersichtlich: `ResultComparator`, `Extractor` und `ExceptionAnalyzer`. Die `RuleEngine`, die weiter oben schon beschrieben wurde, erzeugt Instanzen von diesen Klassen und steuert somit den gesamten Bewertungsprozess. Die `RuleEngine` stellt auch ein Objekt der Klasse `EvalResultHandler`

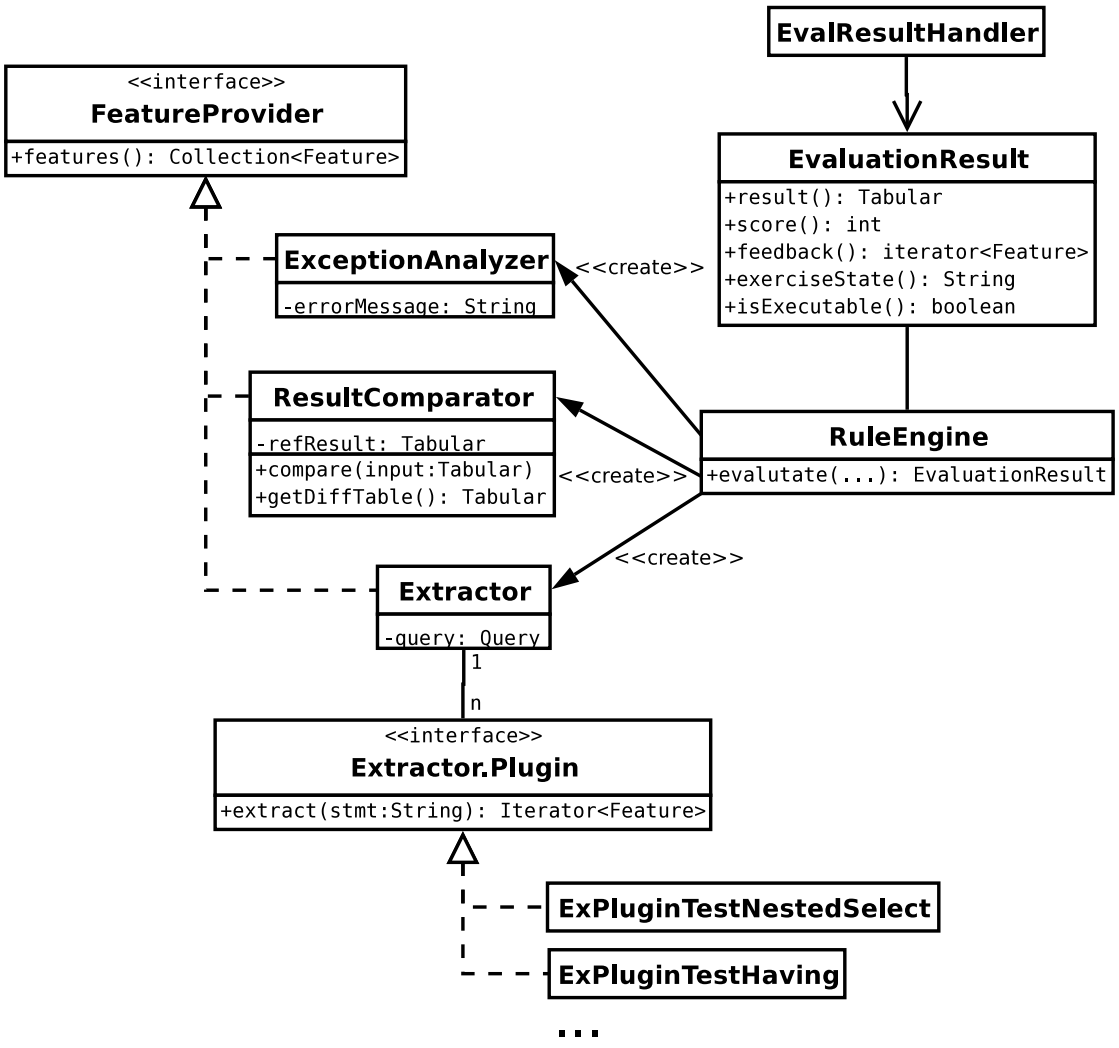


Abbildung 8.12: Wesentliche Klassen im Bewertungssystem

zur Verfügung, das die Ergebnisse des Regelsystems auswertet. Dieses Objekt wird weitergegeben an ein `EvaluationResult`, das als *Decorator* für den `EvalResultHandler` dient und zusätzlich auch noch die Vergleichstabelle beinhaltet. Dieses `EvaluationResult` Objekt wird an das Userinterface weitergegeben.

8.4.1 Extractors

Die *Extractors* sind verantwortlich für die Analyse eines SQL Statements. Ein Objekt der Klasse `Extractor` wird mit einer Query erzeugt. Das `Extractor` Objekt erzeugt einzelne `Plugin` Objekte, welche die eigentliche Aufgabe der Analyse des Statements übernehmen.

Durch dieses Plugin-Konzept ist auch die Analyse des Statements relativ einfach erweiterbar. Ein Plugin ist eine Java Klasse, die ein bestimmtes Interface, nämlich `Extractor.Plugin` implementiert und im System registriert werden muss (in einer Konfigurationsdatei). Das Interface besteht aus nicht mehr als der Methode `extract()`, die mit dem Query-String aufgerufen wird und dann eine Liste von `Feature` Objekten zurück liefert.

Im Prototyp wurden nur Extractors implementiert, die auf Basis von Regular Expressions das Statement analysieren. Das Parsen des SQL Statements wurde für die Bewertung nicht benötigt. Die Analyse des Statements mit Regular Expressions hat (abgesehen von der Einfachheit) auch einen weiteren Vorteil: Die Query muss nicht syntaktisch korrekt oder ausführbar sein, damit sie analysiert werden kann. Damit können auch noch Eigenschaften des Statements ermittelt werden, auch wenn es nicht ausführbar ist, und evtl. kann das Bewertungssystem „entscheiden“, dass auch Punkte vergeben werden für nicht ausführbare Statements.

Es ist aber dennoch nicht ausgeschlossen, dass Extractor Plugins implementiert werden, die das SQL Statement parsen und aufgrund des Parse-Trees Features ermitteln.

Folgende Plugins kommen im Testsystem zum Einsatz:

- Analyse von *strukturellen Eigenschaften*. D.h. es wird festgestellt, ob ein `GROUP BY, HAVING, WHERE, ORDER, JOIN`, etc. im Statement enthalten ist und zwar so, dass nicht einfach nach den Keywords gesucht wird, sondern diese auch groben Anforderungen entsprechen müssen.

- Liste der *Aggregatfunktionen*. Es werden Funktionen wie `MIN(), MAX(), AVG()` im Statement gesucht und als Features gelistet.

- Ermitteln der *Konstanten* im Statement. String- oder numerische Konstanten werden gelesen und als ein Feature mit der Liste der Konstanten zurückgegeben. Dieses Feature ist besonders hilfreich wenn es darum geht, „Abkürzungen“ in einer Lösung zu erkennen (d.h. wenn eine Lösung ein Subselect verlangen würde, aber der Student einfach stattdessen eine Konstante einsetzt).

- Ermitteln der verwendeten *Tables*. Es wird ein Feature mit einer Liste der Tables in den `FROM` Teilen des Statements geliefert.

- Außerdem noch ein Plugin, das generell feststellt, ob das Statement vom Aufbau her die Struktur eines `SELECT` Statements besitzt. D.h. es werden die Keywords `SELECT, FROM, WHERE` überprüft.

8.4.2 Auswerten der Ergebnisse

Die Ergebnisse aus dem Regelsystem werden in der Klasse `EvalResultHanlder` ausgewertet. Diese Klasse übernimmt drei Aufgaben:

1. Feststellen des Aufgabenstatus (einer aus: INCORRECT, INCOMPARABLE, WARNING, CORRECT, ERROR; siehe Abschnitt 5.3.3 für eine Erläuterung der unterschiedlichen States).

2. Angabe der Feedbackmeldungen für den User. In der Methode `feedback()` liefert das `EvaluationResult` eine Reihe von Keywords, die als Meldungen im Userinterface dargestellt werden sollen. Diese Keywords werden genau so vom dahinterliegenden `EvalResultHandler` aufbereitet.

3. Ermitteln der zu vergebenden Punktezahl. Das geschieht im `EvalResultHandler` durch Auswerten der Funktionen `scoreUp()`, `scoreDown()`, `maxScoreUp()`, `maxScoreDown()` und `maxScore()`. Es werden dazu intern drei Werte gespeichert: der *score*, der *maxScore* und das *maxScoreDelta*. *score* wird beeinflusst von `scoreUp()` und `scoreDown()`, *maxScore* nur durch die Funktion `maxScore()`. Die Funktionen `maxScoreUp()` und `maxScoreDown()` steuern den Wert von *maxScoreDelta*.

   `maxScore` wird während der Auswertung so gehalten, dass der minimale Wert von `maxScore` behalten wird. Für die beiden anderen Werte werden die `Up()` und `Down()` Aufrufe summiert.

   Am Ende wird der Punktestand errechnet durch:

   $$min(maxScore + maxScoreDelta, score)$$

   Die Errechnung des Punktestands im Prototyp kann relativ einfach durch Anpassung der Klasse `EvalResultHandler` geändert werden.

Die gewählte Art der Implementierung für die Ermittlung des Punktestands umgeht Komplexitäten im Regelsystem, die notwendig wären um *Defaultfälle* [1] abzubilden. Dies ist dadurch gewährleistet, dass die Logik der Defaultfälle in den `ResultHandler`, der das Benötigte einfach in Java implementiert, ausgelagert werden. Wie man sieht, können so auch sehr einfach Funktionen implementiert werden, die einen Punktestand ändern, anstatt nur absolute Werte zu setzen.

Insgesamt werden im Prototyp folgende Ausgabefunktionen für das Regelsystem definiert:

- Für die Punktevergabe: `scoreUp()`, `scoreDown()` `maxScore()`, `maxScoreUp()`, `maxScoreDown()`.
- Für den Status der Aufgabe: `state()`.
- Für die Anzeige von Feedback: `show()` und `warning()`.

[1] Der Defaultfall besteht hier beispielsweise darin, dass „im Normalfall“ die maximal erreichbare Punktezahl 10 beträgt, aber im Falle eines SQL Fehlers nur noch maximal 3 Punkte gegeben werden. Im Regelsystem selbst könnte man solche Fälle nur durch nichtmonotone Logik implementieren.

8.4.3 Auswahl der Vergleichsquery

Für viele Aufgaben gibt es mehrere korrekte Lösungen, die völlig unterschiedliche Strukturen in der SQL Query verwenden. Beispielsweise kann oft in einem Ansatz gruppiert und summiert werden und in einem anderen Ansatz dasselbe Ergebnis mit nested Selects erreicht werden.

Diesem Umstand wird Rechnung getragen, indem pro Beispiel mehrere Referenz Queries angegeben werden können. Hier stellt sich aber folgendes Problem: mit welchem Referenz Statement soll nun der Input für die Bewertung verglichen werden?

In einer ersten Lösung dieses Problems wurde die Vergleichsquery aus der Reihe von Referenzqueries in einer Vorstufe ausgewählt, indem die Features verglichen wurden und jene Query ausgewählt wurde, für welche die größte Übereinstimmung von Features mit den Features der Inputquery gegeben war.

Dieser Ansatz hat funktioniert, hat aber in einer wesentlichen Stufe das Regelsystem außer Acht gelassen. Um auch diesen Aspekt möglichst flexibel zu gestalten, wurde der erste Ansatz schließlich verworfen und ein noch viel einfacherer Ansatz gewählt:

Die Bewertung wird für jedes Referenzstatement unabhängig durchgeführt. Am Ende wird verglichen und es wird das Ergebnis des Referenzstatements gewählt, in welchem die höchste Punktezahl mit den wenigsten Warnings erzielt wurde. Dabei wird im Grunde der ganze Prozess, wie er in Abbildung 8.11 dargestellt ist, pro Referenzstatement durchlaufen (mit Ausnahme des Resultset Vergleichs). Dort wird davon ausgegangen, dass jedes Referenzstatement dasselbe Resultset liefert.

8.4.4 Ein Set von Regeln als Beispiel

Schließlich soll noch eine vollständige Regelmenge gezeigt werden um sich ein Bild vom Bewertungskonzept machen zu können. Die gezeigten Regeln wurden im Prototyp für Tests verwendet und sind mit den Lehrveranstaltungsleitern nicht abgestimmt. Für den produktiven Einsatz müssen die Regeln auf jeden Fall in der Praxis erarbeitet werden.

Das Beispiel soll einen Überblick über die Möglichkeiten geben und veranschaulichen, welche Gedanken hinter den Konzepten stecken.

```
// Zunächst eine Menge von Typdefinitionen
type(#group,     struct)
type(#having,    struct)
type(#order,     struct)
type(#subselect, struct)

type(#avg,   aggr)
type(#count, aggr)
type(#sum,   aggr)
type(#min,   aggr)
type(#max,   aggr)

type(#constants, const)
type(#tables, tables)
type(#hasSyntaxErrors, error)
type(#hasErrors, error)

// Defaultwerte für die maximale Punktezahl
```

```
#exLevelLight --> >maxScore(5)
#exLevelMedium --> >maxScore(10)
#exLevelHard --> >maxScore(15)

// Vergleichsregeln - Unterschiede in den Datenlisten
ref(x) + in(y) + :sameFeature(x,y) + :diffData(x,y,z) -->
                                                  missingData(z)
ref(x) + in(y) + :sameFeature(x,y) + :diffData(y,x,z) -->
                                                    plusData(z)

// Vergleichsregeln - unterschiedliche Vorkommnisse von Features
ref(x) + !in(x) --> missingFeature(x)
in(x) + !ref(x) --> plusFeature(x)

// Diverses Fehlerhandling
!in(#isCorrectSelect) --> >maxScore(0) + >show(#notSelect)
#hasSyntaxErrors --> Error() + >scoreDown(1)
#hasErrors --> Error()

// Konstanten und Tabellen in Ordnung?
!missingData(#constants) + !plusData(#constants) -->
                                constantsOK() + >show(#constantsOK)
!missingData(#tables) + !plusData(#tables) -->
                                tablesOK() + >show(#tablesOK)
tablesOK() --> >scoreUp(1)
constantsOK() --> >scoreUp(1)

// Maximaler Score bei Fehler
Error() + #exLevelLight --> >maxScore(3)
Error() + #exLevelMedium --> >maxScore(5)
Error() + #exLevelHard --> >maxScore(8)

// Struktur und Aggregate in Ordnung?
!missingFeature(x/struct) + !plusFeature(y/struct) --> structureOK()
structureOK() --> >scoreUp(1) + >show(#structOK)
!missingFeature(x/aggr) + !plusFeature(y/aggr) --> aggregateOK()
aggregateOK() --> >scoreUp(1) + >show(#aggrOK)

missingFeature(x) --> >show(#missing,x)

// Unvergleichbare Ergebnisse (aus RS Vergleich)
#isExecutable + #inComparable --> >show(#incomparable)
#isExecutable + #inComparable + #exLevelLight --> >maxScore(3)
#isExecutable + #inComparable + #exLevelMedium --> >maxScore(7)
#isExecutable + #inComparable + #exLevelHard --> >maxScore(10)

// weitere Anpassungen bei fehlenden oder zuvielen Daten
#isExecutable + missingData(x/const) -->
                              >maxScoreDown(x) + >show(#missingConst,x)
#isExecutable + plusData(x/const) -->
                                  >scoreDown(2) + >show(#plusConst,x)
#isExecutable + missingData(x/tables) -->
```

```
                              >maxScoreDown(2) + >show(#missingTables,x)
#isExecutable + plusData(x/tables) -->
                                >maxScoreDown(x) + >show(#plusTables,x)
#isExecutable + #resultMissingCols -->
                                 >maxScoreDown(2) + >show(#missingCols)
#isExecutable + #resultMissingRows -->
                                 >maxScoreDown(3) + >show(#missingRows)
#isExecutable + #resultPlusRows -->
                                    >maxScoreDown(3) + >show(#plusRows)
#isExecutable + #resultPlusCols --> >show(#plusCols)

// Ergebnis korrekt
#isExecutable + #resultIsCorrect + #exLevelLight --> >scoreUp(2)
#isExecutable + #resultIsCorrect + #exLevelMedium --> >scoreUp(7)
#isExecutable + #resultIsCorrect + #exLevelHard --> >scoreUp(12)

// Ergebnis korrekt, aber das Statement ist nicht in Ordnung
#isExecutable + #resultIsCorrect + !structureOK() --> >warning(unknown)
#isExecutable + #resultIsCorrect + !tablesOK() --> >warning(shortcut)
#isExecutable + #resultIsCorrect + !constantsOK() --> >warning(shortcut)
#isExecutable + #resultIsCorrect + !aggregateOK() --> >warning(shortcut)
```

9 Zusammenfassung und Ausblick

Die wichtigsten Aspekte des e-Learning Systems wurden damit ausführlich dargestellt. Umgesetzt wurde ein Prototyp zum Testen der Konzepte und zum Erarbeiten des Userinterfaces. Dieser Prototyp bildet auch die Grundlage der hiesigen Ausführungen.

Im ersten Teil wurde auf die Architektur des Systems eingegangen, auf die Struktur der Packages im System sowie auf das zugrundeliegende Datenmodell. Dabei wurde teilweise vorgegriffen, indem die geplanten Konzepte für das fertige System vorgestellt wurden. Insbesondere die Gesamtarchitektur wurde im Prototyp noch nicht in der Form umgesetzt. Es wurde beispielsweise nicht auf Oracle entwickelt, sondern auf einem Tomcat Server mit PostgreSQL Datenbanken.

Die vorgestellten Konzepte sind allerdings als Designvorgaben zu verstehen und werden im Produktivsystem umgesetzt. Dennoch ist nicht auszuschließen, dass sich im Laufe der weiteren Entwicklung noch Details ändern werden. Der Entwicklung wird in iterativer Weise durchgeführt, mit Usability Tests und möglichen Erweiterungen oder Änderungen der Anforderungen zwischen den Implementierungsphasen. Auch durch die noch bevorstehende Migration auf das Oracle System könnten sich noch Änderungen ergeben.

Änderungen und Erweiterungen sind besonders im Datenmodell möglich. Neue Anforderungen spiegeln sich in der Regel in den Datenstrukturen wider. Erweiterungen in Form von zusätzlichen Attributen in verschiedenen Tabellen werden mit Sicherheit vorgenommen werden.

Diese erwarteten Änderungen sind Bestandteil der gewählten Vorgehensweise: mit dem Prototyp werden die Konzepte getestet und festgelegt; danach wird – aufbauend auf den Prototyp– ein produktives System implementiert.

Im Kapitel über das Userinterface Design wurden die Überlegungen geschildert, die zur Entwicklung der Oberflächen der Webapplikation geführt haben. Auch hier wird auf das Userinterface des Prototypen eingegangen, was konkret das Userinterface des Übungsmoduls ist.

Es wurden durch das Userinterface des Übungsmoduls die Richtlinien für die Umsetzung des restlichen Systems vorgegeben. Im Übungsmodul sind alle wesentlichen Konzepte vorhanden: die Kursübersicht, die Möglicheiten, sich über den Aufbau und die Inhalte der Kursdatenbank zu informieren, die Aufgabennavigation, Feedback und Differenztabelle.

Was das Userinterface betrifft, wird es auch in den weiteren Modulen keine Neuigkeiten mehr geben. Dieses Kapitel ist demnach als Dokumentation der Konzepte und Richtlinienvorgabe zu verstehen.

Beim Bewertungssystem wurde die vollständige Struktur der Implementierung im Prototypen beschrieben. Die Probleme, die sich beim Resultset Vergleich ergeben haben, wurden im Detail behandelt sowie der im Endeffekt pragmatische Ansatz für die Lösung. Diese Implementierung wird so im Produktivsytem zum Einsatz kommen. Genauso wie das Regelsystem und das Konzept der „Feature Extractors“, die sich allesamt in den Tests sehr gut bewährt ha-

ben. Erweiterungen sind natürlich auch hier nicht ausgeschlossen. Vor allem Extractors können jederzeit zusätzliche implementiert werden.

Die verwendeten Bewertungsregeln sind als Ausgangsbasis für die Bewertung im Produktivsystem zu verstehen. Die tatsächlichen Regeln sind jedenfalls noch im Zuge von umfangreichen Tests zu erarbeiten.

Es wurden im Prototyp keine Performanceprobleme festgestellt. Die Stellen, wo im Multi-User Betrieb möglicherweise Probleme auftreten können, wurden identifiziert. Es gibt zu allen erkannten potenziellen Problemstellen auch gute Verbesserungsmöglichkeiten. Hier wird der Ansatz verfolgt, das Problem erst dann zu lösen, wenn es tatsächlich auftritt.

Schließlich sei noch darauf hingewiesen, dass in der Implementierung des Prototypen nicht auf Standards im e-Learning Umfeld (beispielsweise SCORM) rücksicht genommen wurde. Der Schwerpunkt liegt allein am Konzept des Bewertungssystems und dem Entwurf eines Userinterfaces, das intutitiv verwendbar sein soll. Die Arbeitserleichterung am Institut soll durch den Einsatz dieses Systems möglichst bald erreicht werden. Wenn das System produktiv ist, kann evtl. auch über eine Integration des Systems mit TUWEL[1] als Folgeprojekt nachgedacht werden.

[1] http://tuwel.tuwien.ac.at

Literaturverzeichnis

[ADL06] Advanced Distributed Learning- ADL: Sharable Content Object Reference Model (SCORM) 2004 3rd Edition Documentation Suite.
<http://www.adlnet.gov/downloads/DownloadPage.aspx?ID=237>
[abgerufen: 3.5.2008] 2006

[Bankwitz03] Bankwitz Johannes: Elektronisch unterstütztes Lernen (E-Learning). Diplomarbeit an der FernUniversität Hagen Deutschland, 2003

[Baumgartner97] P. Baumgartner, S. Payr: Erfinden lernen. In: Konstruktivismus und Kognitionswissenschaft. Kulturelle Wurzeln und Ergebnisse. K. H. Müller und F. Stadler. (hrsg.) pp. 89–106. Wien-New York: Springer, 1997

[Beierle00] Christoph Beierle, Gabriele Kern-Isberner: Methoden wissensbasierter Systeme – Grundlagen, Algorithmen, Anwendungen. Vieweg, 2000

[Blumstengel98] Astrid Blumstengel: Entwicklung hypermedialer Lernsysteme.
<http://dsor.upb.de/%7Eblumstengel/main_index_titel.html>
[abgerufen: 4.5.2008] 1998

[Brown74] J.S. Brown: Sophisticated Instructional Environment for Teaching Electronic Troubleshooting. Final Report. Technical Training Division, Lowry Air Force Base, 1974

[Carbonell70] J.R. Carbonell: AI in CAI: An artificial intelligence approach to computer-assisted instruction. IEEE Trans. Man-Machine Systems 11 (4) pp. 190–202, 1970

[Cisco99] Cisco Systems: Reusable Learning Object Strategy: Definition, Creation Overview and Guidelines (Version 3.0).
<http://www.cisco.com/warp/public/779/ibs/solutions/learning/whitepapers/el_cisco_rio.pdf>
[abgerufen: 4.5.2008] 1999

[Downes05] Steven Downes: E-learning 2.0
<http://www.elearnmag.org/subpage.cfm?section=articles&article=29-1>
[abgerufen: 4.5.2008] 2005

[Draper05] S.W. Draper: Feedback <http://www.psy.gla.ac.uk/%7Esteve/feedback.html> [abgerufen: 4.5.2008] 2005

[Forgy82] C. Forgy: RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19, pp. 17–37, 1982

[ForgyOPS5-81] C. Forgy: OPS5 User's Manual. Technical Report CMU-CS-81-135 (Carnegie Mellon University) 1981

[Heyer05] Susanne Heyer: Vergleich von Lernobjektmodellen nach pädagogischen Gesichtspunkten. FernUniversität Hagen, 2005

[Hodgins03] E. Duval, W. Hodgins: A LOM Research Agenda. WWW2003 Conference, May 20-24, 2003, Budapest, Hungary.
<http://www2003.org/cdrom/papers/alternate/P659/p659-duval.html.html>
[abgerufen: 4.5.2008] 2003

[Hugentobler04] U. Hugentobler: Interaktion und Dialog im computerunterstützten Lernen - Entwicklung eines Kostenmodells. Diplomarbeit im Fach Informatik, Universität Zürich, 2004

[J2EEPat01] Deepak Alur et al.: Core J2EE Patterns: Best Practices and Design Strategies. Pearson Education, 2001

[Khazaeli05] C. D. Khazaeli: Systemisches Design - Intelligente Oberflächen für Information und Interaktion. Rowolt, 2005

[Knall05] Knall Thomas: Automatische Adaptierung von SCORM-basierenden Lerninhalten. Magisterarbeit an der Technischen Universität Graz, 2005

[Krug00] Steve Krug: Don't Make Me Think! A Common Sense Approach to Web Usability. circle.com Library, 2000

[L'Allier97] J.J. L'Allier: Frame of Reference: NETg's Map to the Products, Their Structure and Core Beliefs. <https://www.netg.com/Upload/uk_Frame_Reference.pdf>
[abgerufen: 4.5.2008] 1997

[Lelouche99] Ruddy Lelouche: Intelligent Tutoring Systems from Birth to Now. KI Vol. 13 No. 4, pp. 5–11, 1999

[Mahmoud05] Qusay H. Mahmoud: Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications.
<http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html>
[abgerufen: 4.5.2008] 2005

[Mantel67] Nathan Mantel: The detection of disease clustering and a generalized regression approach. Cancer Research Vol. 27, pp. 209–220, 1967

[Martens04] Alke Martens, Andreas Harrer: Lehr-/Lernsysteme – Welche Rolle spielt die Künstliche Intelligenz gestern, heute und morgen? KI 2004, pp. 396–409, 2004

[Miranker89] D.P. Miranker: TREAT: A new and efficient match algorithm for AI production systems. Pittman/Morgan Kaufman, 1989

[Miranker90] D.P. Miranker, D.A. Brant, B. Lofaso, D. Gadbois: On the performance of lazy matching in production systems. Proceedings of the Eigth National Conference on AI, pp. 685–692, 1990

[Parnas72] D.L. Parnas: On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, 1972

[Polsani03] Pithamber R. Polsani (2003): Use and Abuse of Reusable Learning Objects. <http://jodi.tamu.edu/Articles/v03/i04/Polsani/> [abgerufen: 4.5.2008] 2003

[Rotenhofer93] D. Rothenhofer, C. Herzog: SYPROS – an intelligent tutoring system for parallel programming. In Chan, T.–W., editor, Proc. ICCE93, pp. 300–305, Taiwan, 1993

[Sauerstein07] Gert Sauerstein: KI Ansätze zur Lerneradaption in Lern-Management-Systemen. Diplomarbeit, Technische Universität Ilmenau, 2007

[SoarOverview02] Soar Technology, Inc.: Soar – An Overview.
<http://www.soartech.com/projects/16%20SoarOverviewWP.pdf>
[abgerufen: 4.5.2008] 2002

[Tang03] Tiffany Ya TANG, Gordon MCCALLA: Smart Recommendation for an Evolving E-Learning System. In 11th Int. Conference on Artificial Intelligence in Education (AIED'2003). Sydney, Australia. 2003. pp. 699–710, 2003

[Wright03] Ian Wrigt, James Marshal: RETE*, A Faster RETE with TREAT as a Special Case. Int. Journal of Intelligent Games and Simulation, 2(1). ISSN 1477-2043, pp. 36–48, 2003