Endbericht

Kooperatives Lernen mittels Internet-basierter Informationstechniken

Projektgruppe 301

SS 97 und WS 97/98

Veranstalter Sascha Dierkes
Petra Oberthür
Jörg Westbomke

Teilnehmer Tim Bahnes
Marcus Beyer
Manuel Brinkmann
Holger Höhl
Wolfgang Martens
Peter Mischke
Jörg Pleumann
Oliver Schröder
Birgit Sirocic
Olaf Strozyk
Lars Werbeck
Markus Zülch

Universität Dortmund
Lehrstuhl Informatik I
44221 Dortmund

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieses Dokument ist der Endbericht der Projektgruppe KOLIBRI, die im Sommersemester 1997 und im Wintersemester 1997/98 am Lehrstuhl I des Fachbereichs Informatik der Universität Dortmund durchgeführt wurde.

KOLIBRI steht für *Kooperatives Lernen mittels Internet-basierter Informationstechniken.* Die Projektgruppe hat das Konzept und den Prototyp einer Lernumgebung entwickelt, die wesentlich auf den Techniken des Internet, speziell auf dem *World Wide Web (WWW)*, basiert.

Der Endbericht ist wie folgt gegliedert:

- Kapitel 1 enthält diese Einleitung.
- Kapitel 2 stellt die Aufgabenstellung der Projektgruppe vor.
- Kapitel 3 zeigt den Verlauf der Projektgruppe, indem es die wesentlichen Meilensteine des Jahres, in dem die Projektgruppe tätig war, darstellt. Das Kapitel dokumentiert außerdem einige grundsätzliche Entscheidungen bezüglich des Konzeptes und der eingesetzten Hilfsmittel, die Einfluß auf den letztlich implementierten Prototyp hatten.
- Kapitel 4 erläutert die Architektur des KOLIBRI-Systems, im folgenden stets kurz KOLIBRI genannt.
- Kapitel 5 zeigt, wie Kurse für KOLIBRI aufgebaut sind, welche Möglichkeiten sie bieten und wie sie verwaltet werden.
- Kapitel 6 vergleicht die Anforderungen an die Projektgruppe mit dem, was letztendlich erreicht wurde.
- Der Anhang enthält technische Detail-Informationen zur Installation des Systems.

2 Aufgabe der Projektgruppe

2.1 Aktuelle Situation bei Lernsystemen

Computergestützte Lernsysteme konzentrieren sich bisher fast ausschließlich auf die Unterstützung individueller Lernsituationen, d. h. Lernende werden isoliert in ihrer Interaktion mit dem Gerät betrachtet. Beispiele hierfür sind Tutorensysteme und *Computer Based Training (CBT)*. Diese Form der Lernsysteme hat heute bereits eine große Verbreitung gefunden, besonders im PC-Bereich nimmt die Zahl der Lernsysteme, beispielsweise für Fremdsprachen, zu. Der Lernprozeß geschieht dabei offline und ermöglicht keinen direkten Erfahrungsaustauch mit anderen Lernenden.

Mittlerweile bewegt sich die weitere Entwicklung auf dem Gebiet der Lernsysteme von Einzelplatzlösungen hin zu Lernumgebungen zur Unterstützung kooperativer Lernprozesse. Es wird an solche Systeme die Anforderung gestellt, daß sie neben ihrer Funktionalität als Tutorensystem auch den Erfahrungsaustausch und die Gruppenarbeit ermöglichen. Dabei bleiben bestehende Anforderungen an moderne Lernsysteme erhalten, wie z. B. die multimediale Darstellung der Lerninhalte und das freie Navigieren durch diese Lerninhalte.

Die Fortschritte im Bereich der Informations- und Kommunikationstechnologie und besonders die rasante Entwicklung und Verbreitung des Internet schaffen die Basis, gerade das Internet als mögliche Grundlage für kooperative Lernumgebungen zu nutzen. Ein zentraler Begriff ist *Computer Supported Cooperative Learning (CSCL)*, der sich anlehnt an den Bereich *Computer Supported Cooperative Work (CSCW)*. Während Systeme zur Unterstützung kooperativen Arbeitens schon länger erforscht werden und damit Erfahrungen auf diesem Gebiet vorliegen, stehen die Entwicklungen auf dem Gebiet CSCL noch am Anfang. Annahmen, die der Entwicklung von CSCW-Systemen zugrunde liegen, sind nur teilweise auf kooperative Lehr- und Lernprozesse übertragbar. Zur Zeit vorhandene Online-Kurse sind zunächst auch nur für Einzelpersonen konzipiert und unterstützen noch keine interaktiven Lehr- und Lernsituationen.

2.2 Anforderungskatalog der Projektgruppe

Aufgabe der Projektgruppe war es, das Konzept einer CSCL-Umgebung für das WWW zu entwickeln und einen Prototypen mit einem konkreten Lehrbeispiel zu implementieren.

Dazu mußten in einem ersten Schritt die Anforderungen an ein solches System definiert werden, wobei zwei unterschiedliche Sichten Beachtung fanden. Einerseits galt es die Anforderungen zu berücksichtigen, die aus der Sicht des Anbieters eines solchen Systems zu erfüllen sind, etwa die leichte Erstellung von Kursmaterial mit Hilfe von Autorensystemen oder die leichte Verwaltung des gesamten Systems (technische Sicht). Auf der anderen Seite standen die Anforderungen, die sich auf die Vermittlung von Lerninhalten beziehen und die mehr der Sicht des Lernenden entsprechen (pädagogische Sicht).

Hieraus ergab sich folgender Anforderungskatalog für die Projektgruppe. Die einzelnen Punkte werden in den nächsten Abschnitten näher erläutert:

- Multimediale Darstellung von Lerninhalten (Hypermedia).
- Information/Teaching on Demand.
- Interaktives Arbeiten mit dem System.
- Wissensstandkontrolle.
- Unterstützung von kooperativem Arbeiten.
- Validierung des Systems durch Verhaltensanalyse von Lernenden.
- Persönliche Mappe/Arbeitsbereiche des Lernenden.

Ausgehend von diesen Punkten sollte im Rahmen der Projektgruppe ein Konzept für ein solches Lernsystem ausgearbeitet werden, das auf den Möglichkeiten des WWW aufsetzt. Zudem sollte ein Prototyp erstellt werden, der als Anwendungsbeispiel Lerninhalte aus dem Bereich der Fuzzy-Logik enthält. Hier bot sich eine Einführung in die Fuzzy-Logik für Ingenieure an, in deren Rahmen allgemeine Grundlagen zu diesem Thema vermittelt werden sollen. Mit der Entwicklung eines eigenen unscharfen Reglers inklusive durchzuführender Simulation und Optimierung sollte die Interaktion mit dem System, aber auch die Kooperation der Lernenden untereinander getestet werden.

2.2.1 Multimediale Darstellung von Lerninhalten

Gerade die Möglichkeit, unterschiedliche Medien (Text, Grafik, Ton, Video, etc.) in ein Dokument integrieren zu können und mit entsprechenden Strukturierungs- und Zugriffsmechanismen zu versehen, macht Multimedia-Dokumente zur interessanten Basis für Lernsysteme. Das von der Projektgruppe zu entwickelnde System sollte multimediales Kursmaterial unterstützen.

2.2.2 Information/Teaching on Demand

Das System sollte *Information* bzw. *Teaching on Demand* unterstützen. Ein Lernender sollte also jederzeit in der Lage sein, auf beliebige Lerninhalte und Informationen zuzugreifen. Ein solcher Zugriff kann sich auf vertiefende Information zum aktuellen Lerninhalt beziehen, aber auch den Rückgriff auf ältere Informationen ermöglichen, die zum Verständnis der aktuellen Lernsituation benötigt werden. Dieser Punkt war insbesondere im Hinblick auf interaktiv zu bearbeitende Aufgaben notwendig.

2.2.3 Interaktives Arbeiten mit dem System

Das System sollte interaktives Arbeiten unterstützen, eine Anforderung, die auch an bestehende CBT-Systeme gestellt wird. Ein Lernender sollte jederzeit in der Lage sein, auf beliebige Lerninhalte zugreifen zu können, und das System sollte in der Lage sein, sich an die Lerngeschwindigkeit des Lernenden anzupassen. Wissen sollte nicht nur durch reine Präsentation vermittelt, sondern auch durch Aktionen des Lernenden vertieft werden. Hier spielt auch der Punkt *Wissensstandkontrolle* (siehe Abschnitt 2.2.4) eine wichtige Rolle.

2.2.4 Wissensstandkontrolle

Das System sollte versuchen, den Wissensstand des Lernenden zu messen oder einzuschätzen. Wissensstandkontrolle kann bei einem interaktiven System sowohl explizit als auch implizit erfolgen. Bei der expliziten Erfassung werden dem Lernenden Aufgaben gestellt, um zu überprüfen, wie weit sein Lernstand fortgeschritten ist. Zusätzlich ermöglicht ein Online-System die implizite Wissensstandkontrolle, indem es das Verhalten des Lernenden beobachtet (etwa das Navigieren durch die Lerninhalte) und so ein Profil über ihn erstellt. Mit Hilfe des Profils kann dann zum Beispiel die Lerngeschwindigkeit für weitere Lerninhalte angepaßt werden. Dabei kann das System auch Analysen über das Lernverhalten mehrerer Lernender machen, um so eventuell Schwächen in einem Kurs aufzudecken und sie dem Anbieter mitzuteilen (siehe auch Abschnitt 2.2.6, *Validierung des Systems durch Verhaltensanalyse von Lernenden*), oder Klassen von Lernenden zu unterscheiden, durch die neue Lernende schneller eingeschätzt werden können.

2.2.5 Unterstützung von kooperativem Arbeiten

Das System sollte kooperatives Arbeiten unterstützen. Die einfachste Form des kooperativen Arbeitens stellt sicherlich die Möglichkeit des Erfahrungsaustausches dar, ähnlich wie dies in News-Gruppen realisiert ist. Des weiteren ist es aber auch möglich, daß mehrere Lernende zusammen eine Aufgabe bearbeiten. Dies kann sowohl eine vorgegebene Arbeit als auch eine selbstgestellte Aufgabe sein. Das System sollte die Gruppe bei dieser Arbeit unterstützen, indem es geeignete Kommunikationsmittel zur Verfügung stellt und

bei vorgegebenen Aufgabenstellungen auch eine Überprüfung der erarbeiteten Lösung ermöglicht. Problematisch hierbei ist insbesondere die mögliche weite räumliche Verteilung der Gruppenmitglieder. Es mußte ein geeignetes Konzept erarbeitet werden, das auch die asynchrone Aufgabenbearbeitung zuließ.

2.2.6 Validierung des Systems durch Verhaltensanalyse

Wie bereits beim Punkt *Wissensstandkontrolle* (siehe Abschnitt 2.2.4) angedeutet, kann bei einem Online-System die Qualität der Lerninhalte besser kontrolliert werden. Zum einen besteht für den Lernenden die Möglichkeit, Probleme direkt an den Anbieter zu melden, so daß dieser unmittelbar reagieren kann. Zum anderen kann der Anbieter anhand des Verhaltens von Benutzern des Lernsystems Verständnisschwierigkeiten aufdecken und Lerninhalte entsprechend überarbeiten. Die Projektgruppe sollte dies in ihrem Konzept berücksichtigen.

2.2.7 Persönliche Mappe/Arbeitsbereiche des Lernenden

Jedem Lernenden sollte ein persönlicher Arbeitsbereich zur Verfügung stehen, in dem er Notizen zu den Lerninhalten ablegen kann. Dabei sollte es auch möglich sein, Hyperlinks zu bestimmten Lerninhalten des Systems zu setzen, so daß er leicht zu den für ihn interessanten Punkten navigieren kann. Zusätzlich sollten vom System automatisch generierte Notizen berücksichtigt werden, die z. B. den Lernfortschritt dokumentieren oder den Lernenden auf bestimmte Lerninhalte hinweisen, etwa solche, die er ausgelassen hat.

3 Verlauf der Projektgruppe

3.1 Erstes Semester

Das erste Semester der Projektgruppe läßt sich grob als *konzeptionelle Phase* umschreiben. Nach einer Seminarfahrt zu Beginn des Semesters wurden zunächst bestehende Systeme gesichtet, bevor Ideen für das eigene Projekt gesammelt und konkretisiert wurden. Die Fertigstellung des schriftlichen Konzeptes markierte das Ende des ersten Semesters.

3.1.1 Seminarfahrt

Vom 7. bis zum 9. April 1997 fand eine Seminarfahrt der Projektgruppe zum „Haus Nordhelle“ in Meinerzhagen statt. Neben der Kennenlernphase für die einzelnen Mitglieder und Betreuer der Projektgruppe boten die Vorträge eine erste Möglichkeit, sich mit dem Themenspektrum und der gestellten Aufgabe auseinanderzusetzen.

Es wurden im einzelnen folgende Vorträge gehalten:

- Theorie der Fuzzy-Logik.
- Fuzzy-Logik-Controller – Entwurf und Simulation unter Unix.
- Übersicht Unix – Shells, Benutzerverwaltung und Tools.
- Sicherheit im Internet.
- Serveradministration am Beispiel `httpd` und Hyper-G.
- Distant Education I – Konzepte für den Bereich Distant Education.
- Distant Education II – State of the Art.
- Das Internet – Ein technischer Überblick.
- WWW I – Grafische Benutzeroberflächen im WWW unter Verwendung von HTML.
- WWW II – Interaktion im WWW.
- Erstellung interaktiver grafischer Oberflächen im WWW.

Weitere Details zu den Vorträgen können dem Seminarband [Bahn97a] der Projektgruppe, der schriftliche Ausarbeitungen zu allen Themen enthält, entnommen werden.

3.1.2 Sichten bestehender Systeme

Eine der ersten Aufgaben der Projektgruppe bestand darin, bestehende Lernsysteme zu sichten, zu bewerten und daraus Anregungen für das eigene System zu gewinnen. Nach anfänglichen technischen Schwierigkeiten mit dem Zugang zum Internet hat die Projektgruppe einige der größeren, kostenlos zugänglichen Lernsysteme im WWW betrachtet. Andere Systeme waren kommerzieller Natur und als solche nicht zugänglich.

Zu den näher betrachtenen Systemen gehörten u. a. folgende:

- die *Virtuelle Universität* der Fern-Uni Hagen (`http://vus.fernuni-hagen.de/`, zuletzt gesichtet am 5. März 1998).
- das Projekt *Lehre 2000* der Universität Saarbrücken (`http://lehre2000.iwi.uni-sb.de/`, zuletzt gesichtet am 5. März 1998).
- das System *Virtual Classroom* der Firma WBT Systems (`http://www.wbtsystems.com/`, zuletzt gesichtet am 5. März 1998).

Es fiel auf, daß die meisten Systeme versuchen, eine reale Lernumgebung virtuell nachzubilden. Dieser Ansatz hat den Vorteil, daß sich Studenten schnell in einer ihnen vertrauten Lernumgebung zurechtfinden. Der dargebotene Lehrstoff war in den meisten Fällen mit Hyperlinks angereicherter Stoff eines bestehenden Buches oder Vorlesungsskripts. Die Systeme boten außerdem Möglichkeiten, über das Lernen hinaus soziale Kontakte zu anderen Studenten aufzubauen.

3.1.3 Ideen für ein eigenes System

KOLIBRI sollte ähnliche Gedanken beinhalten wie die im vorangehenden Abschnitt genannten Systeme, jedoch speziell folgende Punkte berücksichtigen, die dort entweder komplett fehlten oder nur ansatzweise implementiert waren.

- Konsequentere Nutzung der Möglichkeiten von Hypermedia – Die Lerninhalte sollten nicht einfach aus statischen HTML-Seiten mit einigen Hyperlinks bestehen, da dies die Möglichkeiten von Hypermedia-Systemen bei weitem nicht ausschöpft. Insbesondere sollte bei diesem Punkt Berücksichtung finden, daß es nicht eine feste Sicht auf den vorhandenen Lehrstoff gibt, sondern daß das System dem Studenten Stoff entsprechend seinem Wissen und seinen Interessen präsentiert.
- Kooperatives Arbeiten – Keines der betrachteten Systeme bot Unterstützung für komplexere Aufgaben, die über das Internet von mehreren Studenten in Kooperation gelöst werden können. Die Projektgruppe hat insbesondere diesen Punkt als eine Art „Marktlücke“ betrachtet und entsprechend viel Arbeit in kooperative Konzepte und die schließlich realisierte Entwurfsaufgabe investiert.

3.1.4 Konzept

Im weiteren Verlauf des ersten Semesters wurden in verschiedenen Arbeitsgruppen Ideen für die Teilbereiche von KOLIBRI gesammelt und konkretisiert. Zum Ende des Semesters wurde ein schriftliches Konzept (siehe dazu [Bahn97b]) erstellt, das die zu implementierenden Funktionen und einige optionale Erweiterungsmöglichkeiten enthielt.

3.2 Zweites Semester

War das Ziel des ersten Semesters die Fertigstellung des Konzepts, so könnte man das zweite Semester am treffendsten mit dem Begriff *Implementierungsphase* kennzeichnen, denn am Ende stand die Fertigstellung eines funktionsfähigen Prototyps. Zuvor galt es jedoch, das bestehende Konzept zu verfeinern und einige Entscheidungen bezüglich der Arbeitsmethoden und -mittel zu treffen. Die Arbeit der Projektgruppe wurde mit dem vorliegenden schriftlichen Endbericht und einer öffentlichen Präsentation der Ergebnisse im Rahmen eines Fachgesprächs vorgestellt.

3.2.1 Softwaretechnische Entwurfsmethode

Zu Beginn des zweiten Semesters hat sich die Projektgruppe Gedanken über eine Entwurfsmethode gemacht, die sich für die gegebene Aufgabenstellung und das vorhandene, aus 12 Teilnehmern bestehende Team eignet. Die Gruppe hat sich schließlich für den Einsatz von *Rapid Prototyping* entschieden. Bei dieser Entwurfsmethode wird nicht sofort das gesamte Programm implementiert, sondern zunächst nur ein eingeschränkter Funktionsumfang. Im Sinne des Risikomanagements wird üblicherweise die Funktionalität zuerst realisiert, die bezüglich ihrer Implementierung das größte Risiko birgt. Aufbauend auf einem ersten Prototyp werden dann in einem rekursiven Prozeß weitere Prototypen erstellt.

Der Vorteil dieser Vorgehensweise liegt zum einen in einer Reduzierung des Risikos, da fundamentale Schwächen im geplanten System rechtzeitig erkannt und umgangen werden können. Zusätzlich ist es durch Rapid Prototyping möglich, dem Auftraggeber (in diesem Fall den Betreuern der Projektgruppe) frühzeitig sichtbare Ergebnisse vorzuweisen und das weitere Vorgehen mit ihm abzustimmen.

Im Fall von KOLIBRI wurden mehrere Prototypen parallel entwickelt, wobei jeder einen spezialisierten Teilbereich der gesamten Aufgabenstellung implementierte. Außer dem zentralen Prototypen für den Kern von KOLIBRI existierten Prototypen für die Kommunikationsbereiche des Systems (Chat und Pinnwand), das Aufnehmen von Kursen in die zugrundeliegende Datenbank, den Entwurf des unscharfen Reglers sowie das automatische Bewerten nicht-kooperativer Aufgaben. Im Laufe des weiteren Entwicklungsprozesses wurden diese Prototypen in den zentralen Prototypen integriert.

Der Einsatz von Rapid Prototyping hat sich aus der Sicht der Projektgruppe bewährt, da die Methode zum einen ermöglichte, die Entwicklung der Teilbereiche von KOLIBRI stark zu parallelisieren. Zum anderen gab es Problemstellungen innerhalb des Systems, die sich nicht durch eine rein theoretische Planung hätten lösen lassen, sondern stattdessen die unmittelbare Implementierung verschiedener Varianten erforderten. Als Beispiel sei hier die Wahl eines geeigneten Prinzips für die Kommunikation zwischen der Client- und der Server-Seite des Systems genannt.

3.2.2 Programmiersprachen

Für die Arbeit der Projektgruppe standen mehrere Programmiersprachen zur Auswahl. Da der Einsatz bestehender Autorensysteme mit den entsprechenden proprietären Skriptsprachen bereits sehr früh verworfen wurde, konzentrierte sich die Entscheidung auf folgende Sprachen:

- C oder C++ zur Programmierung der Server-Seite des Systems, insbesondere zum Zugriff auf die Datenbank und für Erweiterungen des in C implementierten WWW-Servers Apache, der zunächst zum Einsatz kommen sollte.
- Java zur Programmierung der Server-Seite des Systems, aber auch zur Programmierung von Applets, die auf dem WWW-Client eines Studenten ablaufen sollen.
- Perl zur Realisierung von CGI-Skripten, die zur Auswertung von Anfragen des WWW-Clients an den Server notwendig sind.

Die Entscheidung über die zu verwendenden Programmiersprachen stand im engen Zusammenhang mit dem schließlich eingesetzten WWW-Server. Da sich die Projektgruppe gegen Apache und zugunsten von Jigsaw entschieden hat, der sehr elegant Erweiterungen in Java unterstützt und komplett ohne „CGI-Bastellösungen“ auskommt, wurde der ausschließliche Einsatz von Java in folgender Weise beschlossen:

- Java 1.1 auf der Server-Seite des Systems. Es wurde das JDK 1.1.4 von Sun (siehe dazu [Sun97]) verwendet.
- Java 1.0.2 auf der Client-Seite des Systems, da die meisten WWW-Clients derzeit Java 1.1 gar nicht oder nur unvollständig unterstützen. Der Netscape Navigator bot zumindest die für das Projekt notwendige (siehe z. B. Abschnitt 4.4.3) Funktionalität.

Die Entscheidung für Java hat sich im nachhinein bewährt, da das entstandene System auf diese Weise sehr homogen geworden ist. Dadurch, daß alle Mitglieder der Projektgruppe in der gleichen Sprache programmierten, war insbesondere stets ein Ansprechpartner bei Problemen zur Stelle.

3.2.3 Spezifikation

Die Spezifikation von KOLIBRI begann mit dem zweiten Semester. Sie ergab sich zunächst aus dem im ersten Semester entstandenen Konzept, das weiter verfeinert und konkretisiert wurde. Die Spezifikation war zahlreichen Anpassungen an neue Ideen oder geänderte technische Randbedingungen unterworfen, so daß sie aus einer Reihe von Einzeldokumenten besteht, die von den jeweiligen Untergruppen oder Personen in Eigenverantwortung gepflegt und im Verlauf der Implementierung aktualisiert wurden.

3.2.4 Implementierung

Etwa ab November 1997 wurde mit der Implementierung einzelner Komponenten des Prototyps begonnen. Die Komponenten wurden zu diesem Zeitpunkt noch weitgehend unabhängig voneinander entwickelt und getestet, so daß sich erste sichtbare Ergebnisse auf einen lauffähigen WWW-Server und einzelne Servlets beschränkten, mit denen sich Studenten im System anmelden konnten.

Dem System lag bereits eine Datenbank zugrunde – anfangs noch mSQL 1.0.16, später mSQL 2.0.3 – in der sämtliche zu speichernden Daten verwaltet wurden. Die einzelnen Tabellen der Datenbank waren zu diesem Zeitpunkt noch starken strukturellen Änderungen unterworfen, bis sich schließlich eine effiziente Datenbankstruktur ergab, die alle anfallenden Aufgaben bewältigen konnte.

3.2.5 Test und Fehlerkorrektur

Im Dezember 1997 wurden nach und nach die Komponenten zusammengefügt, getestet und aufeinander abgestimmt. Zu Beginn und in der Mitte des Monats fanden zwei Projektgruppen-interne Vorstellungen des Systems statt, in denen sich die Betreuer vom Fortschritt der Arbeit überzeugen konnten.

Im Januar und den ersten beiden Wochen des Februar 1998 wurden Stabilität und Geschwindigkeit von KOLIBRI so weit verbessert, daß sich bis zum Fachgespräch ein Prototyp ergab, mit dem sowohl die Teilnehmer als auch die Betreuer der Projektgruppe zufrieden waren.

3.2.6 Fachgespräch und Endbericht

Am 19. Februar 1998 wurden die Ergebnisse der einjährigen Arbeit der Projektgruppe in einem Fachgespräch öffentlich vorgestellt. Die Konzepte von KOLIBRI wurden in einem etwa halbstündigen Vortrag erläutert und dann in einer ähnlich umfangreichen Rechnerpräsentation demonstriert. Anschließend ergab sich eine längere Diskussion zwischen den anwesenden Professoren, den wissenschaftlichen Mitarbeitern des Lehrstuhls und den Teilnehmern und Betreuern der Projektgruppe.

Im Anschluß an das Fachgespräch wurde mit der Erstellung des vorliegenden Endberichts begonnen.

4 Architektur des Systems

4.1 Überblick

Das System KOLIBRI teilt sich in eine Client-Seite und eine Server-Seite auf. Auf der Client-Seite befindet sich ein Benutzer – entweder ein Student oder ein Kursleiter –, der auf einen Kurs zugreift. Dieser Kurs wird ihm von der Server-Seite bereitgestellt.

Der Zugriff auf KOLIBRI geschieht über einen Java-fähigen WWW-Client. Der Benutzer findet sich anschließend in einer Lernumgebung wieder, die das reale Lernumfeld einer Universität nachzubilden versucht. Die angebotenen Kurse besitzen eine hierarchische, einem Buch ähnliche Struktur. Neben dem eigentlichen Lernstoff enthalten sie auch Aufgaben, die von einem oder mehreren Studenten zu lösen sind. Eine Pinnwand und ein Chat erlauben sowohl kursbezogene als auch private Kommunikation zwischen den Benutzern von KOLIBRI.

Basis der Server-Seite des Systems ist eine Datenbank, die sowohl die Daten der angebotenen Kurse als auch alle anfallenden Verwaltungsdaten speichert. Ein WWW-Server und eine Reihe von speziellen Servlets bereiten die Daten aus dieser Datenbank in Form von HTML-Seiten für den Benutzer auf. Dabei ist entscheidend, daß bis auf einige wenige Ausnahmen keine festen HTML-Seiten im System existieren. Stattdessen werden alle Seiten dynamisch auf eine Anfrage des Benutzers hin generiert.

Die Kommunikation zwischen Client und Server geschieht nicht nur über die üblichen HTTP-Anfragen (siehe dazu [Fiel97]), sondern auch über Kommunikationsmechanismen, die speziell entwickelt wurden, um den besonderen Anforderungen von KOLIBRI Rechnung zu tragen. Sie sind im Abschnitt 4.4.3 erläutert.

4.2 Client-Seite

Das in den Abbildungen 4.1 auf der nächsten Seite und 4.2 auf Seite 15 dargestellte Diagramm beschreibt mögliche Durchläufe von KOLIBRI-Benutzern durch das System. Den Knoten des Graphen entsprechen Java-Applet-Fenster oder HTML-Seiten, die zumeist dynamisch erzeugt werden. Jede Kante beschreibt einen möglichen Übergang von einer Seite zu einer anderen, wobei am Startpunkt jeder Kante ein Knopf oder ein Hyperlink steht, mit dem der Benutzer den Übergang anstoßen kann.

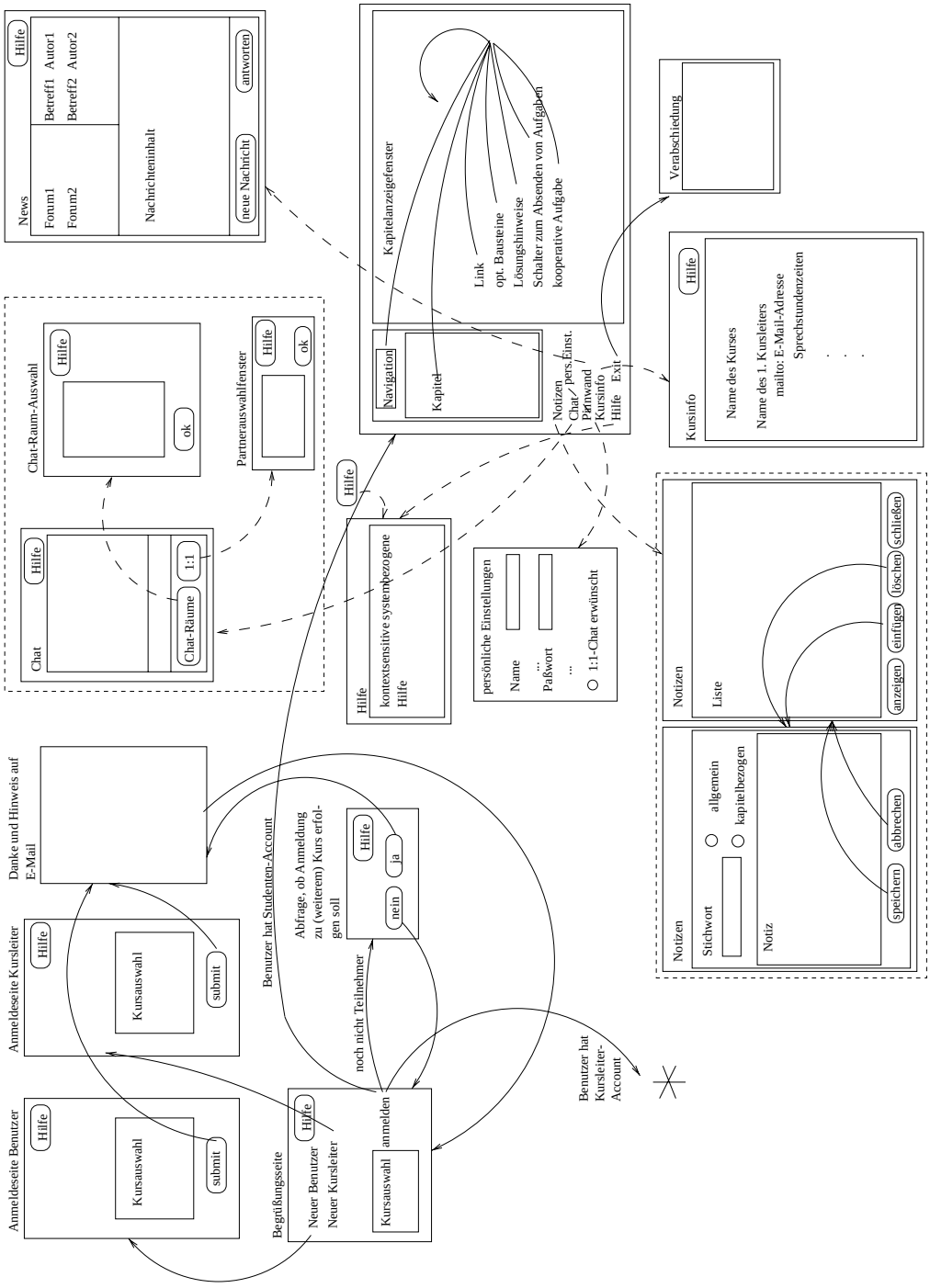

Abbildung 4.1: Überblick über das System (Teil 1 von 2)

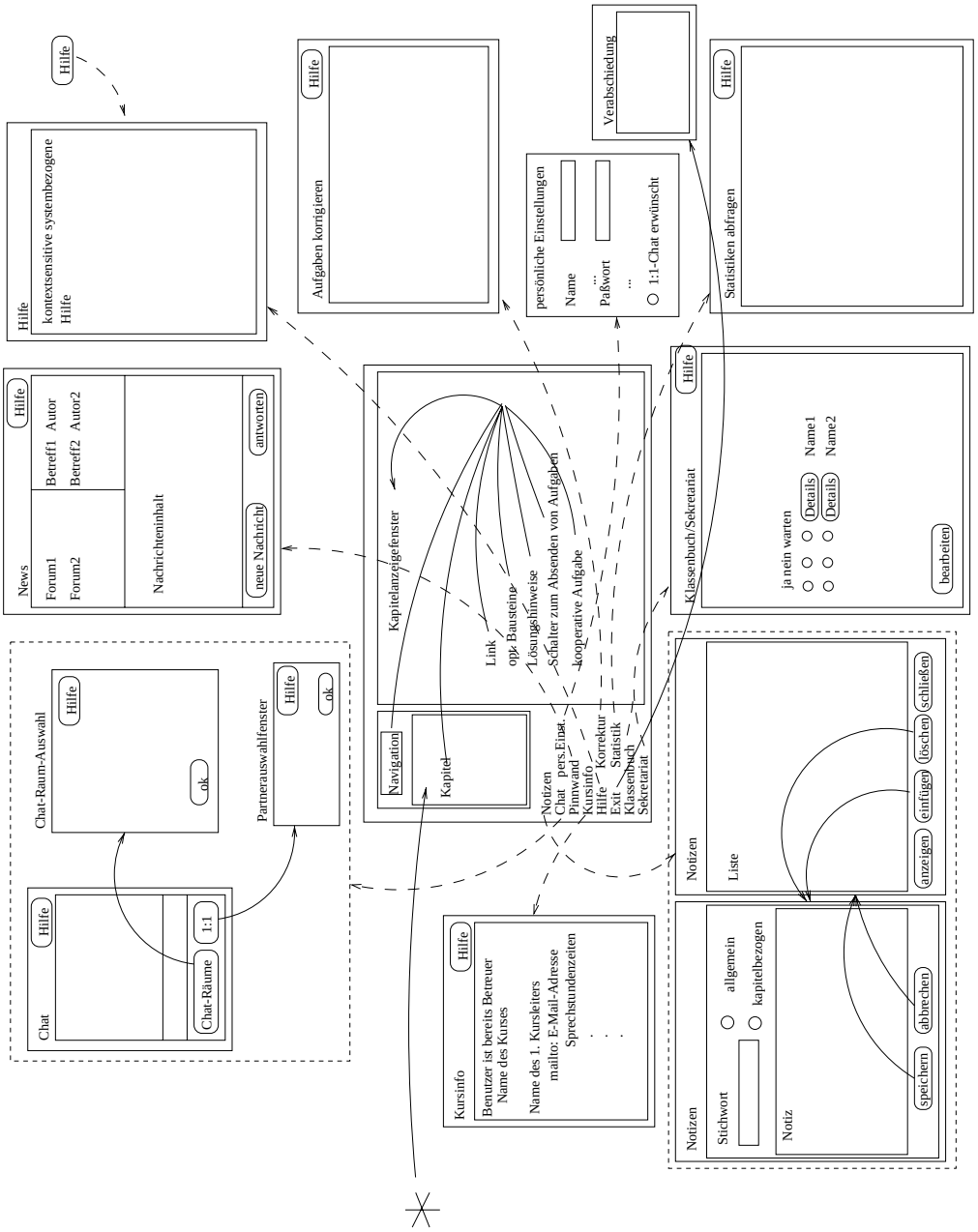

Abbildung 4.2: Überblick über das System (Teil 2 von 2)

In den meisten Fällen wird bei der Aktivierung einer Funktion der alte Inhalt des aktuellen Fensters überschrieben. Dies ist durch einen Pfeil dargestellt, dessen Linie durchgängig gezeichnet ist.

Falls bei der Aktivierung einer Funktion *zusätzlich* zum alten Fenster ein neues WWW-Client- oder Java-Applet-Fenster geöffnet wird, ist der Pfeil zwischen dem Knopf und dem Fenster mit einer gestrichelten Linie gezeichnet. In diesem Fall kann der Benutzer zwischen den beiden (und weiteren) Fenstern wechseln.

Ein mit gestrichelten Linien gezeichnetes Rechteck deutet an, daß alle innerhalb liegenden Fenster zum gleichen Teilbereich von KOLIBRI gehören.

Ein Benutzer beginnt seine KOLIBRI-Sitzung, indem er in seinem WWW-Client die Startseite des Systems lädt. Wenn ihm noch kein Account (als Student oder Kursleiter) zugewiesen wurde, hat er die Möglichkeit, einen solchen zu beantragen. Falls er bereits einen Account besitzt, kann er sich einen Kurs aussuchen und sich beim System anmelden (siehe Abschnitt 4.2.1).

Nach der Anmeldung erscheint der angewählte Kurs (siehe Abschnitt 4.2.2) in seinem Browser-Fenster. Das Fenster ist in zwei Bereiche unterteilt, die horizontal nebeneinander liegen. Rechts ist der Inhalt des gerade angewählten Kapitels dargestellt, während auf der linken Seite oben die Kapitelstruktur des Kurses angezeigt wird und darunter ein Menü erscheint.

Der Inhalt dieses Fensters unterscheidet sich bei Studenten und Kursleitern nur dadurch, daß Kursleitern mehr Menüfunktionen zur Verfügung stehen.

Über das Menü können Studenten und Kursleiter die Funktionen *Notizen* (siehe Abschnitt 4.2.5), *Chat* (siehe Abschnitt 4.2.3), *Pinnwand* (siehe Abschnitt 4.2.4), *Kursinfo* (siehe Abschnitt 4.2.6), *Hilfe* (siehe Abschnitt 4.2.7), *Persönliche Einstellungen* (siehe Abschnitt 4.2.12) und *Kolibri verlassen* aufrufen.

Kursleitern stehen zusätzlich die Optionen *Sekretariat* (siehe Abschnitt 4.2.8), *Klassenbuch* (siehe Abschnitt 4.2.9) und *Statistik* (siehe Abschnitt 4.2.11) zur Verfügung.

4.2.1 Anmeldung

Auf dieser Seite wird das System kurz erläutert. Der Benutzer wird aufgefordert, sich einen Kurs auszusuchen und sich entweder mit einem bestehenden Account anzumelden oder einen neuen Account zu beantragen (siehe Abbildung 4.3 auf der nächsten Seite).

Falls der Benutzer sich als Betreuer oder Kursleiter neu bei KOLIBRI anmeldet, wird er aufgefordert, seine persönlichen Daten in ein Anmeldeformular einzutragen. Nachdem diese Daten in der KOLIBRI-Datenbank gespeichert wurden, wird ihm für sein Interesse gedankt; er wird darüber informiert, daß er in Kürze, nach Verifikation der angegebenen Daten seitens eines Kursleiters, per E-Mail einen Account zugewiesen bekommt.

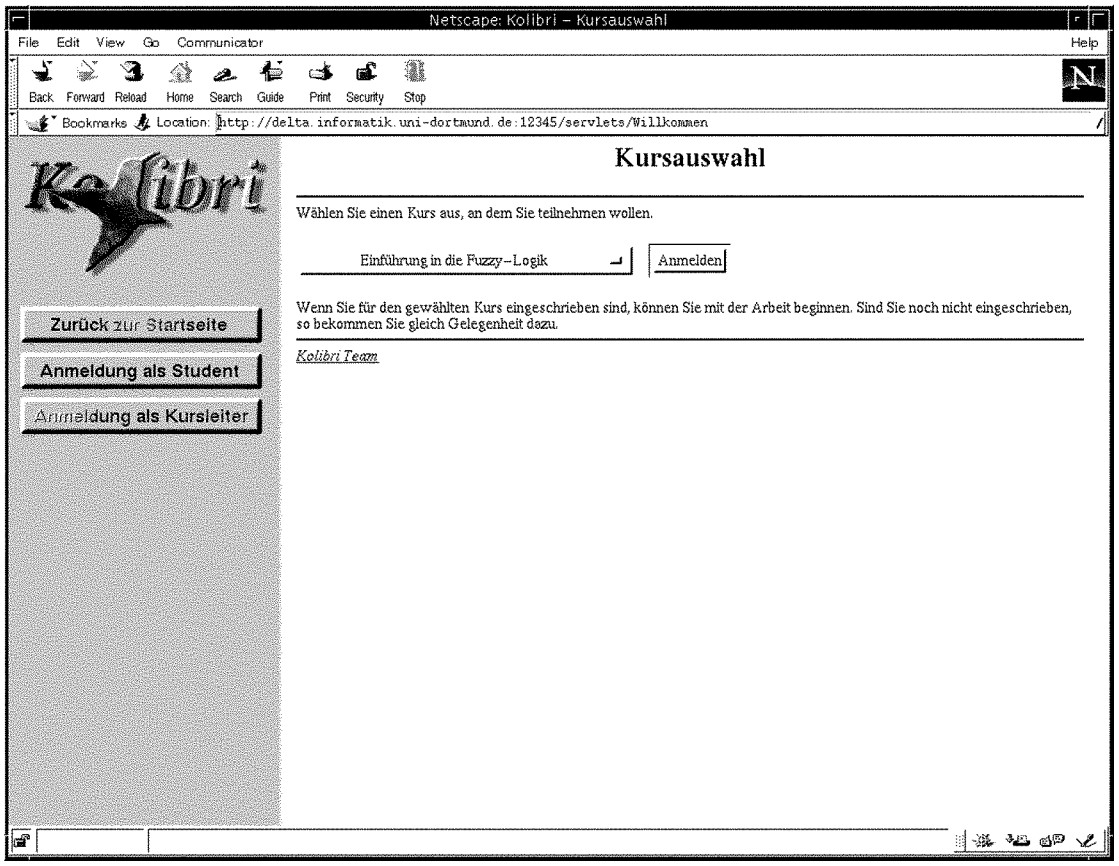

Abbildung 4.3: Anmeldung in KOLIBRI

Falls ein Benutzer zwar einen Account hat, aber noch nicht Teilnehmer des ausgewählten Kurses ist, erscheint eine Abfrage, ob er sich zu diesem Kurs neu anmelden möchte. Falls er diese Frage bejaht, wird ihm zunächst für sein Interesse gedankt und um sein Verständnis gebeten, daß er den neu ausgewählten Kurs erst nach Bearbeitung seiner Anmeldung durch einen Kursleiter besuchen kann.

4.2.2 Kurs

Die Darstellung eines Kurses setzt sich aus dem sogenannten *Kurs-Browser* und der Anzeige von Kapitelinhalten zusammen.

Der Kurs-Browser ist als Java-Applet realisiert und befindet sich auf der linken Seite des Fensters. Er zeigt das Inhaltsverzeichnis des aktuellen Kurses und den persönlichen Kursverlauf des Benutzers (siehe Abschnitt 5.1.3). Eine weitere Funktion des Kurs-Browsers ist das Navigieren durch die Kapitel des Kurses.

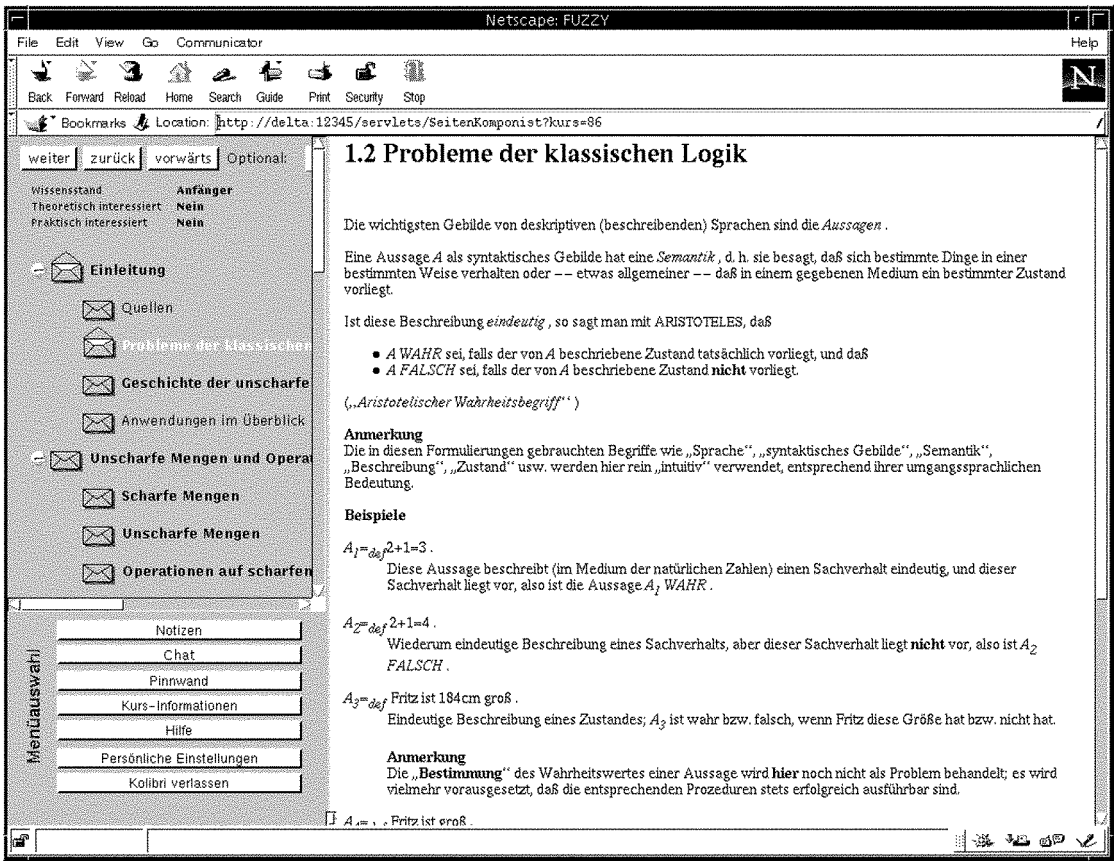

Abbildung 4.4: Kurs während des Lesens einer Lektion

Der Inhalt des aktuellen Kapitels wird auf der rechten Seite des Fensters in einem separaten Frame angezeigt.

4.2.2.1 Kurs-Browser

Der Kurs-Browser hat das in Abbildung 4.4 dargestellte Aussehen. Er ist während einer Sitzung immer sichtbar und wird ständig entsprechend den Benutzer- und Systemaktionen aktualisiert.

In der ersten Zeile des Kurs-Browsers befinden sich drei Schalter [Weiter], [Zurück] und [Vorwärts] sowie eine Auswahlliste, die mit dem Begriff *Optional* gekennzeichnet ist. Diese Bedienelemente helfen dem Benutzer, im Kurs zu navigieren und die für ihn relevanten Kapitel aufzusuchen. Durch Betätigen des Schalters [Weiter] gelangt er, vom aktuellen Kapitel im Baum aus gesehen, in das nächste obligatorische Kapitel, das noch nicht besucht wurde oder den Zustand *nicht verstanden* hat. Die Schalter [Vorwärts] und [Zurück] dienen der Navigation innerhalb der Kapitelhistorie des Benutzers. Die Kapitelhistorie enthält

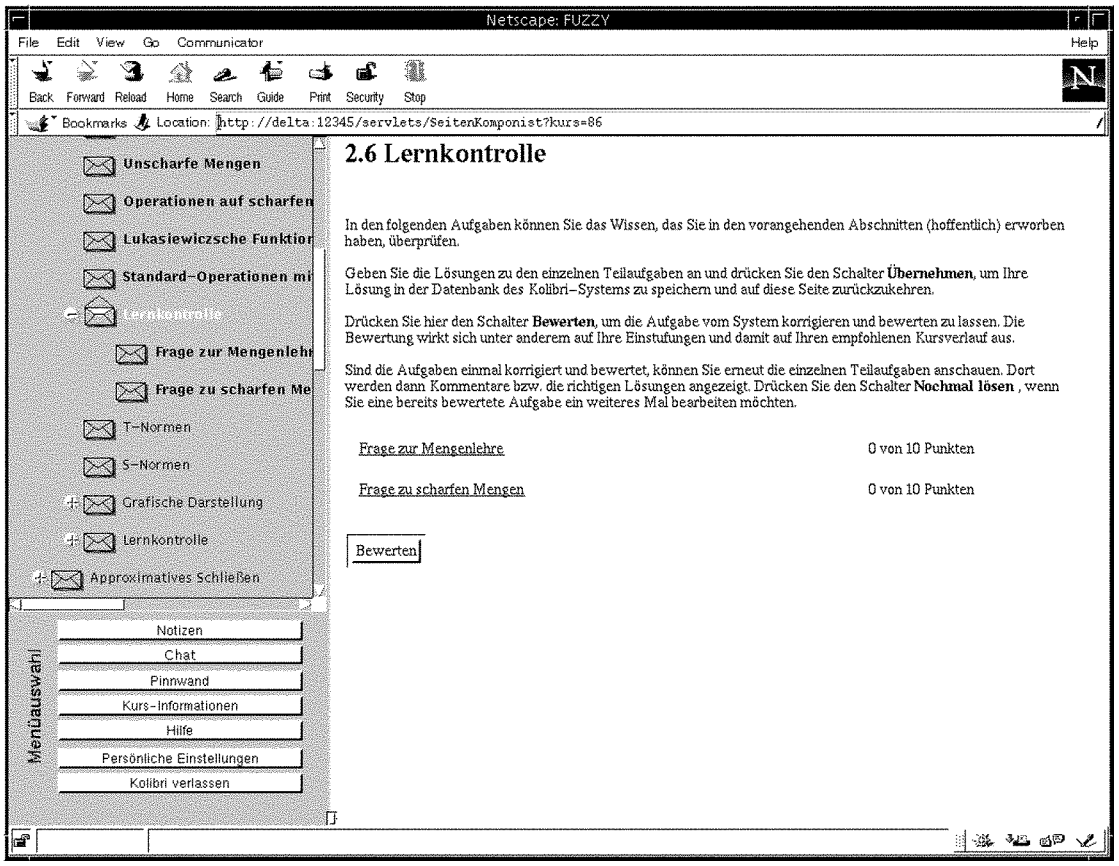

Abbildung 4.5: Kurs während der Bearbeitung einer Aufgabe

Informationen über die vom Benutzer betrachteten Seiten und die Reihenfolge, in der er sie aufgerufen hat. Diese Informationen werden vom System auch über eine KOLIBRI-Sitzung hinaus gespeichert. Mit dem Schalter [Zurück] springt der Benutzer einen Eintrag innerhalb seiner Historie zurück, mit dem Schalter [Vorwärts] gelangt er zum nächsten Eintrag. Die Auswahlliste enthält alle Sohn- und Nachfolger-Kapitel des aktuellen Kapitels, die für den Benutzer optional sind (siehe Abschnitt 5.1.3.2). Durch das Anwählen eines der Kapitel aus der Liste wird dieses zum aktuellen Kapitel, und der Kapitelinhalt wird auf der rechten Seite angezeigt.

In der zweiten Zeile, direkt unterhalb der Schalter, werden die Interessens- bzw. Wissensgruppen, in denen der Benutzer sich zum aktuellen Zeitpunkt befindet, angezeigt. Dadurch sind diese für den Benutzer zu Beginn einer Sitzung in jedem Fall sichtbar. Aufgrund bestimmter Aktionen wird er vom System möglicherweise anderen Gruppen zugeordnet werden. Diese Änderungen in den Zugehörigkeiten werden dem Kurs-Browser mitgeteilt, und dieser wird daraufhin aktualisiert.

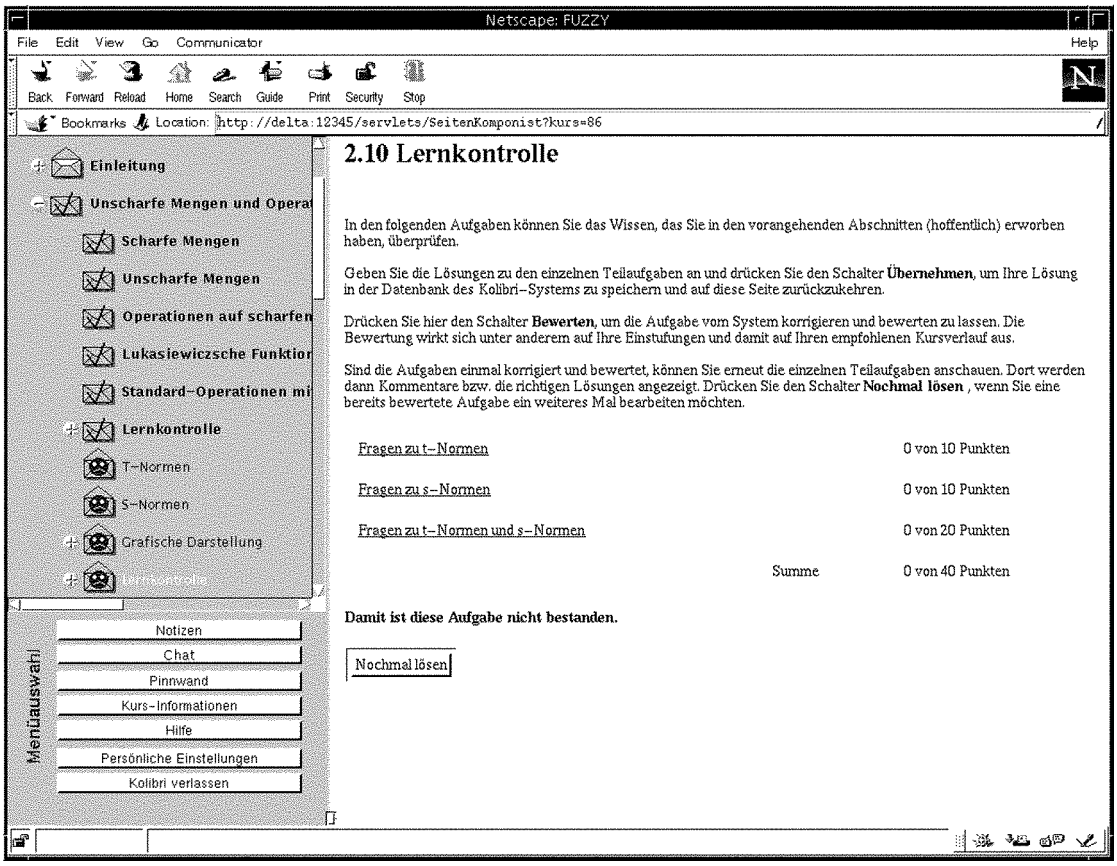

Abbildung 4.6: Kurs nach der Korrektur einer Aufgabe

Die einzelnen Kapiteltitel werden unterhalb der Gruppenanzeige als Baumstruktur dargestellt. Das Kapitel, in dem sich der Benutzer gerade befindet, wird durch weiße Schrift kenntlich gemacht.

Der Benutzer kann ein Kapitel durch einen Mausklick auf dessen Titel direkt anwählen. Es wird dadurch zum aktuellen Kapitel, und der Inhalt wird auf der rechten Seite angezeigt. Die Kapiteleinordnungen werden durch verschiedene Schriftarten und unterschiedliche Helligkeitsgrade der Standardschriftfarbe dargestellt. Die nicht empfohlenen Kapitel werden in einem helleren Ton angezeigt. Damit wird eine Nichtzugehörigkeit dieser Kapitel zum empfohlenen Kursverlauf suggeriert, sie sind aber trotzdem vom Benutzer anwählbar. Hat ein Wechsel innerhalb der Interessens- bzw. Wissensgruppen stattgefunden, kann dies auch Änderungen in den Kapiteleinordnungen nach sich ziehen, so daß das Aussehen des Kurs-Browsers angepaßt werden muß.

Bei Kapiteln, die Unterkapitel besitzen, wird vor dem Titel ein Expandierungssymbol angezeigt. Dieses Symbol ist, wie z. B. in Abbildung 4.6 dargestellt, eine Kugel mit einem Plus- oder Minuszeichen. Durch das Anwählen des Pluszeichens wird das entsprechen-

de Kapitel expandiert, und die Unterkapiteltitel werden angezeigt. Dies kann durch das Drücken des Minussymbols wieder rückgängig gemacht werden: Die Unterkapitel verschwinden aus der Anzeige, ihr Status (s. u.) bleibt aber in der Datenbank gespeichert.

Jedes Kapitel befindet sich in einem bestimmten Zustand, der den Bearbeitungsstatus wiedergibt. Die Kapitelzustände werden durch entsprechende Symbole neben den Kapiteltiteln angezeigt (siehe Abbildung 4.6 auf der vorherigen Seite). Wurde ein Kapitel noch nicht vom Benutzer besucht, so ist das zugehörige Zustandssymbol ein geschlossener Briefumschlag. Wurde ein Kapitel vom Benutzer angewählt, also besucht, dann ist das zugehörige Zustandssymbol ein geöffneter Briefumschlag. Hat der Benutzer Lernkontrollen erfolgreich bearbeitet, dann werden die relevanten Kapitel als verstanden markiert. Hat der Benutzer hingegen die Lernkontrollen nicht bestanden, dann werden die relevanten Kapitel als nicht verstanden markiert. Auch für diese beiden Zustände gibt es passende Zustandssymbole (siehe Abbildung 4.6 auf der vorherigen Seite). Die Änderung in den Kapitelzuständen werden im Kurs-Browser unmittelbar angezeigt.

Zu Beginn der ersten Sitzung in einem Kurs werden nur die Titel der obersten Kapitelebene angezeigt. Alle Kapitel, die Unterkapitel enthalten, sind geschlossen. Das erste Kapitel ist dann automatisch das aktuelle Kapitel. Alle Kapitel gelten als noch nicht besucht und haben das zugehörige Zustandssymbol. Die Einordnungen innerhalb der möglichen Interessens- bzw. Wissensgruppen sind nach den vom Kursautor vorgegebenen Initialisierungen vorgenommen worden und werden auch so im Kurs-Browser angezeigt.

In jeder weiteren Sitzung wird zu Beginn wieder nur die oberste Kapitelebene angeboten, und alle Kapitel sind geschlossen. Alle anderen Einstellungen werden so angezeigt, wie sie eingestellt waren, als der Student den Kurs verlassen hat. Das bedeutet, das aktuelle Kapitel ist das zuletzt besuchte Kapitel, es wird als solches markiert und sein Inhalt wird angezeigt. Die Einordnungen des Benutzers innerhalb der Interessens- bzw. Wissensgruppen sind dieselben wie vor dem Verlassen des Kurses. Dasselbe gilt auch für die Kapitelzustände.

4.2.2.2 Kapitelinhalt

Der Kapitelinhalt des aktuellen Kapitels wird auf der rechten Seite des Fensters in einem separaten Frame angezeigt. Der Kapitelinhalt besteht aus einer Reihe von Bausteinen unterschiedlichen Typs. Ein Baustein kann für den Benutzer komplett sichtbar sein oder durch einen Verweis repräsentiert werden. Ein solcher Verweis ist mit einem Pluszeichen gekennzeichnet. Über das Anwählen dieses Pluszeichens wird der Baustein aufgeklappt und dadurch angezeigt. Der offene Baustein wird dann mit einem Minuszeichen gekennzeichnet. Durch das Anwählen des Minuszeichens wird der Baustein wieder geschlossen.

Innerhalb der Kapitelinhalte können auch Verweise auf andere Kapitel existieren. Durch das Anwählen eines solchen Kapitelverweises wird in das entsprechende Kapitel verzweigt.

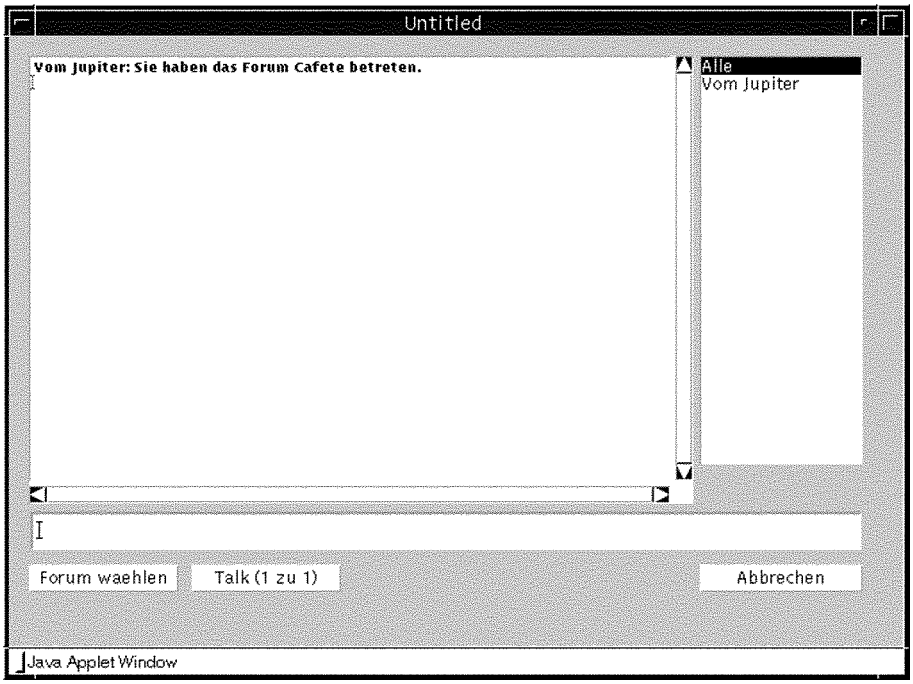

Abbildung 4.7: Chat-Bereich von KOLIBRI

4.2.3 Chat

Im Chat-Bereich können zwei oder mehr Benutzer miteinander kommunizieren, indem sie sich abwechselnd eine kurze Nachricht zusenden. Jeder Benutzer, der sich in einem der verschiedenen Chat-Räume befindet, kann alle Nachrichten lesen, die andere Benutzer im selben Chat-Raum geschrieben haben.

Der Chat ist auf der Client-Seite durch ein Applet realisiert worden. Wenn das Applet startet, wird ein Ereignis gesendet, das den Benutzer dem System bekannt macht. Die Anbindung des Chats an das System erfolgt über den Startschalter [Chat] in der Menüauswahl. Das gesamte Applet erscheint in der ersten Phase als einzelner Schalter. Hier bekommt der Chat seine beiden Parameter übergeben: die Identität des Benutzers und den aktuell besuchten Kurs, um damit das aktuelle Forum auswählen zu können.

Das Chat-Applet setzt sich aus vier Teilen zusammen:

- **Startschalter:** Die Menüauswahl [Chat] öffnet das Chat-Fenster und trägt den Benutzer in das zum Kurs gehörige Forum ein. Falls ein Benutzer von einem anderen angesprochen wird, zeigt der Startschalter dies an. Ein Druck auf den Schalter führt den Benutzer in diesem Fall direkt zur 1-zu-1-Kommunikation.

  Es gibt vier verschiedene Forentypen: In der *Cafete* werden allgemeine Dinge besprochen, oder es wird einfach nur „Smalltalk“ betrieben. Die *Kursforen* sollen

Diskussionen zum Kurs unterstützen und dadurch ermöglichen, daß sich die einzelnen Studenten gegenseitig helfen, im Kurs voranzukommen. Die Gruppen, die kooperative Aufgaben lösen, erhalten eigene Foren, in denen die Teilnehmer die Lösung gemeinsam erarbeiten können. Der *1-zu-1-Chat* soll die direkte Kommunikation zweier Studenten untereinander oder die Kommunikation zwischen einem Studenten und einem Kursleiter ermöglichen.

- **Chat-Fenster**: Das Chat-Fenster ist der eigentliche Kommunikationsplatz. Hier wird in den verschiedenen Foren der Kurse oder in der Cafete diskutiert und Informationsaustausch betrieben.

  Im Chat gibt es ein Textfenster, in dem die Nachrichten angezeigt werden. Hiermit sind sowohl die Nachrichten der Benutzer als auch die Nachrichten des Systems gemeint. Zusätzlich gibt es ein Textfeld, in das die Nachricht, die der Benutzer absenden will, eingetragen wird. Sobald Eingabe gedrückt wird, ist die Nachricht abgesendet. Das dritte Element ist die Auswahlliste, in der bestimmt wird, an wen die abzusendende Nachricht gehen soll. Die Schalter sind für den Wechsel des Forums oder für den Aufbau einer 1-zu-1-Kommunikation zuständig.

- **Forenauswahlfenster**: Wenn im Chat-Fenster die Forenauswahl gewählt worden ist, wird ein weiteres Fenster geöffnet, in dem die Foren und kooperativen Aufgaben aufgelistet werden zu denen der Benutzer zugangsberechtigt ist. Der Benutzer wählt das neue Forum aus und bestätigt diese Wahl mit dem zugehörigen Schalter. Danach wird das Fenster geschlossen und im Chat-Fenster erscheint das neue Forum mit den aktuellen Benutzern.

- **1-zu-1-Kommunikationsfenster**: Wenn im Chat-Fenster die 1-zu-1-Kommunikation gewählt wurde, wird ein Fenster geöffnet, in dem alle derzeit im System angemeldeten Benutzer aufgelistet sind. Der Benutzer wählt einen Partner aus, und das System meldet dem Partner in dessen Startschalter oder im Schalter für die 1-zu-1-Kommunikation den Gesprächswunsch des anderen Benutzers. Falls der Partner nicht gestört werden will (siehe Abschnitt 4.2.12), erhält der Benutzer eine entsprechende Nachricht.

4.2.4 Pinnwand

Die Pinnwand realisiert ein News-System („Schwarzes Brett“) zur asynchronen Kommunikation innerhalb von KOLIBRI. Sie bietet dem Kursteilnehmer die folgenden Möglichkeiten:

- Auswahl eines Forums.
- Anzeigen einer Nachricht.
- Erstellen einer neuen Nachricht.

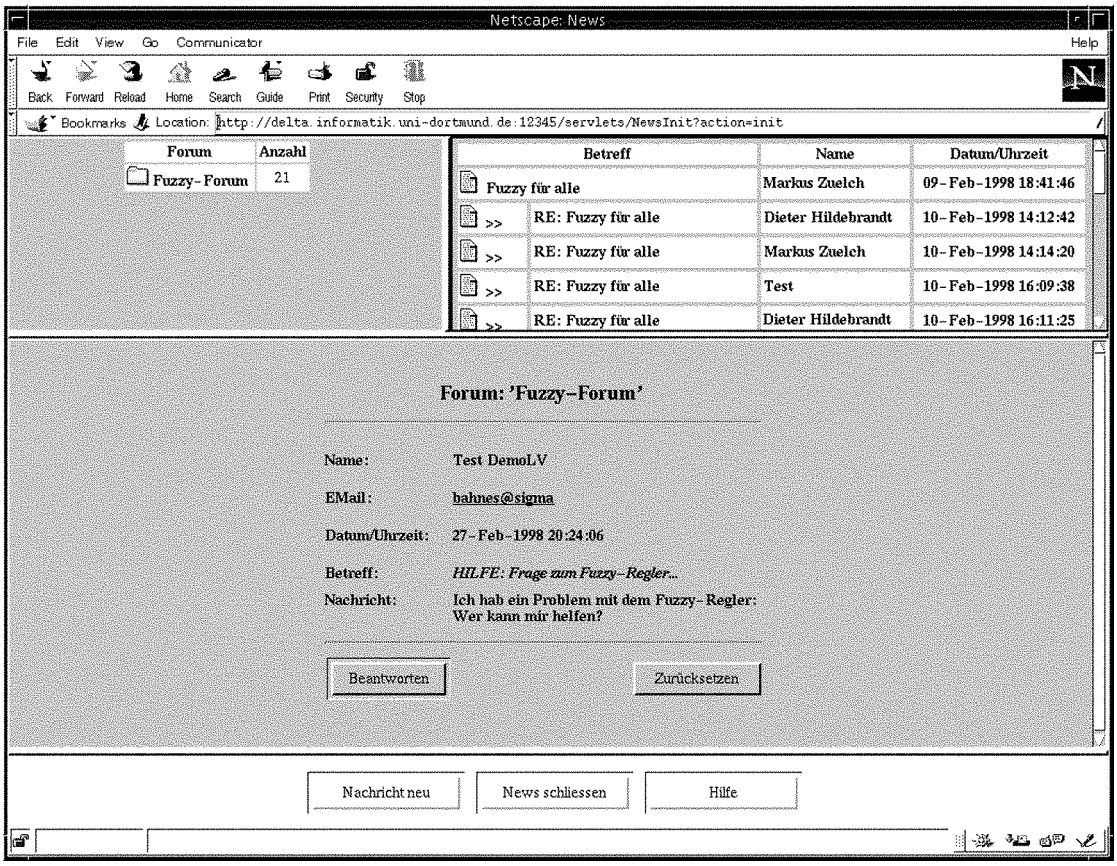

Abbildung 4.8: Pinnwand von KOLIBRI

- Antworten auf eine vorhandene Nachricht.

Dem Kursleiter stehen zusätzlich die folgenden Optionen zur Verfügung:

- Anlegen eines Forums.
- Löschen eines Forums (inklusive der enthalten Nachrichten).

Das Browser-Fenster ist viergeteilt (siehe Abbildung 4.8). Oben links werden die Foren angezeigt, auf die der Benutzer Zugriff hat. Oben rechts werden die Betreffzeilen und Verfasser der Nachrichten (sortiert nach Thema, Datum und Zeit) dargestellt, die dem oben links ausgewählten Forum hinzugefügt wurden. Nachrichten, auf die bereits geantwortet wurde, sind auf die folgende Weise visuell von den Original-Nachrichten abgegrenzt:

- Ihnen sind zwei Größerzeichen (») vorangestellt.

- Sie sind farblich gelb unterlegt.

Im mittleren Bereich wird der Nachrichtentext der ausgewählten Nachricht angezeigt. Dieses Fenster enthält im unteren Bereich einen Schalter [Beantworten], um auf die zuvor ausgewählte Nachricht zu antworten. Wird dieser Schalter aktiviert, leert sich das mittlere Fenster, und eine Eingabemaske erscheint. Diese enthält u. a. den Titel des ausgewählten Forums, Name und E-Mail-Adresse des Original-Verfassers, Name und E-Mail-Adresse des aktuellen Benutzers, die Betreff-Zeile, die Original-Nachricht und ein Antwort-Eingabefeld. Weiterhin sind im unteren Bereich zwei Schalter [Einfügen] und [Zurücksetzen] vorhanden. Die Informationen des Benutzers, z. B. Name und E-Mail-Adresse, werden zur Laufzeit durch Datenbank-Abfragen bestimmt und in die entsprechenden Felder eingesetzt. Die dafür notwendigen Informationen werden den Tabellen *accounts*, *studenten* und *kursleiter* entnommen (siehe Abschnitt 4.3.3.4 auf Seite 50).

Wird der Schalter [Einfügen] aktiviert, werden die zuvor vom Benutzer eingegebenen Daten in die entsprechende Datenbank-Tabelle *news* eingetragen. Tritt ein Fehler auf, wird dies dem Benutzer in Form einer Fehlernummer mit entsprechender Beschreibung angezeigt, andernfalls erfolgt eine Bestätigung, daß die Nachricht erfolgreich eingefügt wurde. Mit dem Schalter [Zurücksetzen] werden alle vorhandenen Eingabefelder wieder gelöscht (diese Aktion bezieht sich nur auf die vorhandenen Eingabefelder und nicht auf Datenbank-Aktionen).

Im unteren Fenster befindet sich die Schalterleiste, die je nach Benutzerart (Kursteilnehmer oder Kursleiter) eine andere Anzahl von Schaltern und Beschriftungen besitzt. Wird die Pinnwand aufgerufen, so sind für einen Kursteilnehmer die Schalter [Nachricht neu], [News schließen] und [Hilfe] sichtbar, für einen Kursleiter zusätzlich [Forum anlegen] und [Forum löschen].

4.2.5 Notizen

Die Idee für die Notizen war es, eine Arbeitsumgebung für den Benutzer zu schaffen, in der er – ohne weitere Hilfsmittel wie z. B. Papier und Bleistift – ausschließlich mit dem System arbeiten kann. Es wurde dabei im Konzept angedacht, die Hilfsmittel, die beim konventionellen Lernen zur Verfügung stehen, ähnlich auch in KOLIBRI abzubilden. Gleichzeitig sollten die Möglichkeiten, die Computer heute bieten, als Erweiterungen einfließen. Im Konzept wurde dieser Punkt unter dem Begriff „Mappe“ abgehandelt.

Exemplarisch wurden im Prototyp die *Notizen* implementiert. Sie sind als Karteikarten, die sich ein Lernender anlegt, zu verstehen. Der Benutzer hat während des Kursdurchlaufs die Möglichkeit, durch Aufruf des Menüpunktes *Notizen* in diesen Teil des Systems zu gelangen.

Durch den Aufruf startet der Benutzer das Applet *Notizen*. Dieses Applet nutzt die Funktionalitäten des Kommunikationmanagers (siehe Abschnitt 4.4.3.1), um die Kommunikation mit der Datenbank zu realisieren. Dies geschieht durch den *NotizenManager*

Abbildung 4.9: Liste der Notizen eines Studenten

(siehe Abschnitt 4.4.3.7). Eine Notiz, die von einem Benutzer des Systems angelegt wird, wird in der Datenbank mit einer eindeutigen Kennung in der Tabelle *notizen* (siehe Tabelle 4.16 auf Seite 46) abgelegt. Da jeder Benutzer ebenfalls eine eindeutige Kennung besitzt, ist die eindeutige Zuordnung der Notizen zu einem Benutzer gewährleistet.

Will der Benutzer nun eine Notiz anlegen, startet er das Applet *Notizen* aus der Menüzeile. Es erscheint ein separates Applet-Fenster neben dem eigentlichen WWW-Client (siehe Abbildung 4.9). Im oberen Teil dieses Fensters erscheint eine Liste der bereits von ihm angelegten Notizen. Jede Zeile enthält dabei das Stichwort und das Kapitel, zu dem er die Notiz gemacht hat. Der Benutzer kann Notizen zu einem bestimmten Kapitel oder allgemein zum Kurs anlegen. Im letzteren Fall bezieht sich die Notiz auf kein spezielles Kapitel. In der Liste werden dem Benutzer immer alle Notizen angezeigt, auch wenn er diese in einem anderen Kurs gemacht haben sollte.

Im unteren Teil des Fensters erscheinen die Schalter [Anzeigen], [Einfügen], [Löschen] und [Schließen].

Durch Auswahl einer Notiz aus der Liste und Drücken des Schalters [Anzeigen] erscheint im Applet-Fenster ein neu aufgebautes Menü (Doppelklicken auf eine Notiz aus der Liste bewirkt das gleiche). Im oberen Teil des Fensters erscheint das Stichwort, unter dem die Notiz gespeichert wurde, und rechts daneben, um welche Art von Notiz es sich handelt, also entweder kapitelbezogen oder allgemein (siehe Abbildung 4.10 auf der nächsten Seite). Darunter wird in einem Textfeld die eigentliche Notiz angezeigt. Dieses Feld ist editierbar, so daß der Benutzer seine Notiz nach Bedarf verändern und erweitern kann. Veränderbar sind ebenfalls das Stichwort und die Art der Notiz.

Abbildung 4.10: Bearbeiten einer Notiz

Im unteren Teil des Fensters stehen ihm die Schalter [Speichern] und [Abbrechen] zur Auswahl. Mit [Speichern] übernimmt er seine Änderungen in die Datenbank, mit [Abbrechen] verwirft er sie. Danach gelangt er wieder in das vorherige Fenster, das ihm die Auswahl seiner Notizen zeigt.

Mit Auswahl des Schalters [Einfügen] erhält der Benutzer das gleiche Fenster, das auch bei der Auswahl von [Anzeigen] erscheint, jedoch mit leeren Vorgaben in den entsprechenden Feldern. Die Art der Notiz wird mit [Allgemein] vorgeschlagen.

Das Löschen einer Notiz erfolgt durch Auswahl einer Notiz aus der Liste und Drücken des Schalters [Löschen]. Es erfolgt keine weitere Sicherheitsabfrage, die Notiz wird direkt gelöscht.

Der Notizenbereich wird verlassen durch Auswahl des Schalters [Schließen].

4.2.6 Kursinfo

Über diese Funktion kann sich der Benutzer anzeigen lassen, welche Kursleiter für den aktuellen Kurs zuständig sind. Zu jedem Kursleiter sind dessen E-Mail-Adresse und Sprechstunden-Zeiten im Chat-Bereich angegeben.

Zu den Zeiten, die als Sprechstunden eines Kursleiters angegeben sind, sollte dieser im System angemeldet sein und auf 1-zu-1-Chat-Anforderungen seitens der Benutzer reagieren.

4.2.7 Hilfe

Das Konzept sah eine kontextsensitive Hilfe vor: Auf jeder Seite, die dem Benutzer präsentiert wird, sollte ein Hilfe-Schalter anwählbar sein, bei dessen Betätigung sich ein neues Fenster öffnet, in dem eine kurze Beschreibung der Optionen dargestellt wird.

Aus Zeitmangel konnte diese Funktion jedoch nicht mehr implementiert werden.

4.2.8 Sekretariat

Mit dieser Funktion, die den Kursleitern vorbehalten ist, kann Studenten, die sich neu im System angemeldet haben, ein Account zugewiesen oder deren Teilnahme verweigert werden.

Wenn die Funktion aufgerufen wird, erscheint eine Liste der neuen Studenten. Der Kursleiter kann sich nun über einen Knopf entweder Details einer Anmeldung anschauen oder den Status eines Studenten über Schalter von [Abwarten] auf [Akzeptieren] oder [Ablehnen] ändern. Wenn der Kursleiter mit der Bearbeitung fertig ist, kann er seine Änderungen absenden oder verwerfen.

Ein Student wird, nachdem ein Kursleiter ihn akzeptiert oder abgelehnt hat, darüber automatisch per E-Mail informiert. Mit dieser E-Mail wird auch das Paßwort des neuen Benutzers übertragen, welches vom System automatisch erzeugt wird.

4.2.9 Klassenbuch

Auch diese Funktion ist den Kursleitern vorbehalten. Sie erlaubt es, Studenten für den aktuell angezeigten Kurs die Teilnahme zu erlauben oder zu verweigern. Diese Funktion ist ähnlich wie die Funktion *Sekretariat* (siehe Abschnitt 4.2.8) aufgebaut.

Auch hier wird dem Kursleiter zunächst eine Liste der Studenten präsentiert, die an dem aktuellen Kurs teilnehmen möchten. Der Kursleiter kann detaillierte Informationen über die Studenten anfordern und deren Status von [Abwarten] auf [Akzeptieren] oder [Ablehnen] ändern.

Der Student wird, nachdem ein Kursleiter ihn akzeptiert oder abgelehnt hat, vom System darüber automatisch per E-Mail informiert.

4.2.10 Korrektur

Mit dieser Funktion sollen Kursleiter die Aufgabenlösungen der Kursteilnehmer präsentiert bekommen, um dann entscheiden zu können, ob die Lösungen korrekt oder fehlerhaft sind. Dies ist jedoch nur bei solchen Aufgabentypen vorgesehen, bei denen das System nicht automatisch eine Korrektur vornehmen kann.

Da am Ende der Projektgruppe keine Zeit mehr übrig war, konnte diese Funktion nicht mehr in der im Konzept angedachten Form implementiert werden. Im Moment werden dem zuständigen Kursleiter die Lösungen der Studenten per E-Mail zugestellt. Er kann dann das Ergebnis direkt in der Datenbank vermerken.

4.2.11 Statistik

Diese Funktion ist den Kursleitern vorbehalten. Aufgrund von Zeitmangel konnte sie jedoch nicht mehr implementiert werden.

Laut Konzept sollen dem Kursleiter hier allgemeine Statistiken präsentiert werden:

- Wieviele Studenten sind immatrikuliert?
- Wieviele Studenten warten darauf, immatrikuliert zu werden?
- Wieviele Teilnehmer sind im Kurs angemeldet?
- Wieviele Studenten warten darauf, daß ihre Anmeldung bearbeitet wird?
- Wie sind die Teilnehmer innerhalb der Lerngruppen verteilt?
- Welche Aufgaben wurden wie gelöst?
- Wieviele Benutzer sind gerade im System aktiv?
- Wieviele Benutzer haben sich im letzten Monat nicht mehr bei KOLIBRI angemeldet?

4.2.12 Persönliche Einstellungen

Der Benutzer kann nachträglich seine persönlichen Daten ändern. Lediglich sein Vor- und Nachname sowie sein Account-Name sind nicht editierbar. Bei Eingabe eines neuen Paßworts muß dieses zur Verifikation zweimal korrekt eingegeben werden. In dem Auswahlfeld *Chat zulassen* kann angegeben werden, ob eine Kommunikationsaufforderung eines anderen Benutzers angezeigt werden soll oder nicht.

Für Kursleiter besteht zusätzlich die Möglichkeit, die Zugehörigkeiten seiner Wissensgruppen zu ändern, um z. B. die Auswirkungen auf die Darstellung des Kurses zu überprüfen.

4.3 Server-Seite

Dieser Abschnitt erläutert die auf der Server-Seite von KOLIBRI eingesetzten Elemente. Eine Übersicht liefert die Abbildung 4.11 auf der nächsten Seite.

Abbildung 4.11: Übersicht über den KOLIBRI-Server

4.3.1 WWW-Server

Nach einer Evaluierungsphase von verschiedenen WWW-Servern fiel die Wahl der Projektgruppe auf den Jigsaw-Server. Dies ist der neue offizielle Server des W3-Konsortiums und gleichzeitig die Referenzimplementierung von HTTP (siehe dazu [Fiel97]). Vorteile wurden in der mitgelieferten Dokumentation [W3C98], die u. a. Beschreibungen zum Erweitern des Servers enthält, dem freien Quellcode und den vielfältigen Erweiterungsmöglichkeiten gesehen. Außerdem ist der Server komplett in Java implementiert, so daß auch an dieser Stelle keine weitere Programmiersprache verwendet werden mußte. Für die Entscheidung war auch die Unterstützung von Techniken zum dynamischen Aufbau von HTML-Seiten maßgebend (z. B. Servlets).

Als problematisch wurde das frühe Beta-Stadium des Servers angesehen, in der Testphase zeigten sich aber keine ernsten Schwächen. Außerdem war die mitgelieferte Dokumentation an einigen Stellen nicht umfassend genug, so daß hier Experimente nötig wurden.

Nach einer Vorauswahl stand letztendlich noch der Apache-Server als Alternative zur Diskussion. Dieser Server ist häufig im Einsatz und besitzt einen ausgereiften Aufbau. Er ist über eigene Module zu erweitern und bietet ebenfalls Unterstützung für die Erzeugung dynamischer Seiten. Dies wird über sogenannte *Server Side Includes (SSI)* realisiert, die Verwendung von Servlets ist nur durch ein zusätzliches Modul möglich. Dieses Servlet-Modul befand sich zu der Zeit, in der die Entscheidung über den Server getroffen wurde, im Alpha-Stadium. Kleinere Tests zeigten aber keine offensichtlichen Schwächen. Apache ist vollständig in C implementiert, dies hätte die Verwendung einer zusätzlichen Programmiersprache für Server-Erweiterungen nötig gemacht. Die Dokumentation war nur in einer Online-Version im Internet verfügbar. Sie besteht hauptsächlich aus einer Zusammenfassung der in den Konfigurationsdateien zu verwendenden Schlüsselworte.

Die endgültige Entscheidung für den Jigsaw-Server fiel aufgrund der guten Unterstützung von dynamisch erzeugten Seiten und der guten Erweiterbarkeit mittels Java.

Im Verlauf der Implementierungs- und Testphase machte Jigsaw einen stabilen Eindruck. Kurz vor Ende des zweiten Semesters erschien eine zweite, fehlerbereinigte Beta-Version, die problemlos eingesetzt werden konnte.

4.3.2 Servlets

Servlets erlauben wie CGI-Skripte die dynamische Erzeugung von HTML-Seiten. Dies ist ein grundlegender Bestandteil von KOLIBRI, da sich Seiten durch Benutzeraktionen wie z. B. das Lösen von Aufgaben oder das Auf- und Zuklappen von Bausteinen verändern. Für die Entscheidung zugunsten von Servlets und gegen CGI-Skripte gab es zwei Gründe: Durch die Wahl der Programmiersprache Java stand ein Paket zur Entwicklung von Servlets bereit, das bereits wichtige grundlegende Funktionen zur Verfügung stellte, wie z. B. das Parsing von in einem *Uniform Resource Locator (URL)* kodierten Parametern. Der andere Grund war der Geschwindigkeitsvorteil, denn Servlets unterstützen

die Multithreading-Fähigkeit von Java. Dadurch muß nicht für jede Anfrage eine neue Instanz des Programms (wie bei CGI-Skripten), sondern nur ein weiterer Thread erzeugt werden. Zeitkritische Routinen, wie z. B. das Aufbauen einer Datenbank-Verbindung, werden einmal beim Starten des Servlets ausgeführt. Im folgenden finden dann lediglich die Operationen zur Erzeugung der dynamischen Seiten statt. Da einige Servlets, etwa der in Abschnitt 4.4.2 beschriebene *Seitenkomponist*, von mehreren Benutzern gleichzeitig angesprochen werden, wurde dies als Vorteil gegenüber den CGI-Skripten angesehen. Leider stellte sich im Verlauf der Implementierungsphase heraus, daß – wahrscheinlich durch einen Fehler im eingesetzten JDBC-Treiber – das einmalige Aufbauen und mehrfache Verwenden einer Datenbankverbindung nicht möglich war. Die Antwortzeiten waren aber dennoch durch das Ausnutzen des Multithreading geringer als bei einer sequentiellen Bearbeitung von Anfragen.

Im folgenden werden ausgewählte Servlets beschrieben, die häuptsächlich die Verwaltung von Studenten und Kursleitern übernehmen. Das komplexere Servlet *Seitenkomponist* wird im Abschnitt 4.4.2 besprochen.

Häufig ähnelt sich die Funktionalität bestimmter Servlets für Studenten und Kursleiter. Diese Servlets werden dann zusammen in einem Abschnitt beschrieben, wobei auf die wesentlichen Unterschiede kurz eingegangen wird.

4.3.2.1 Servlet *Willkommen*

Das *Willkommen*-Servlet erzeugt eine Begrüßungsseite, auf der der Benutzer verschiedene Auswahlmöglichkeiten angeboten bekommt: Es gibt die Möglichkeit, sich als Student oder Kursleiter zu bewerben, dies würde zum Aufruf der Servlets *StudentenAnmeldung* oder *KursleiterAnmeldung* führen. Außerdem wird eine Liste aller angebotenen Kurse präsentiert, aus der Studenten und Kursleiter den Kurs auswählen können, den sie bearbeiten möchten. Der Kurs wird dann entweder angezeigt, falls der Benutzer bereits akzeptiert ist, oder es erscheint eine Seite, auf der der Benutzer die Möglichkeit erhält, sich für den ausgewählten Kurs anzumelden. Beides wird vom Seitenkomponisten realisiert.

4.3.2.2 Servlets *StudentenAnmeldung* und *KursleiterAnmeldung*

Durch diese Servlets wird die Erstanmeldung von Studenten bzw. Kursleitern, die noch nicht in KOLIBRI registriert sind, vorgenommen (vergleichbar mit der Immatrikulation von Studenten an einer Universität).

Die Servlets unterscheiden zwischen einem Aufruf über die Get- und Post-Methoden von HTTP (siehe dazu [Fiel97]). Die Auswahl des zugehörigen Schalters, z. B. auf der KOLIBRI-Hauptseite, führt zu einem Get-Aufruf. Der Student bzw. Kursleiter bekommt daraufhin ein Formular gesendet, in das die persönlichen Daten und gewünschten Kurse eingetragen werden sollen. Das Absenden des ausgefüllten Formulars führt zu einem Aufruf desselben Servlets, diesmal aber über die Post-Methode. In diesem Fall werden einige

Überprüfungen der eingegebenen Daten vorgenommen (z. B. ob es sich bei der Postleitzahl auch um eine Zahl handelt) und, wenn diese erfolgreich verlaufen, die Daten in die Datenbank eingefügt. Für Kursleiter finden Eintragungen in die Tabellen *kursleiter* und *kursleiter_kurse* statt, bei Studenten in die Tabellen *studenten* und *teilnehmer_kurse*.

Das Feld *account* der Tabellen *studenten* und *kursleiter* wird bei der ersten Anmeldung mit dem Wert Null belegt. Dieser Wert bedeutet, daß der Student bzw. Kursleiter noch keine Zugangsberechtigung erhalten hat. Die Berechtigung wird durch das Sekretariat (Servlet *NeueStudentenBearbeiten*, siehe Abschnitt 4.3.2.3) für Studenten oder durch einen Kursleiter (mit besonderen Rechten) mit dem Servlet *NeueKursleiterBearbeiten* für andere Kursleiter erteilt.

In den Tabellen *kursleiter_kurse* bzw. *teilnehmer_kurse* erhält das Feld *aktuellKapitel* bei der ersten Anmeldung den Wert *null*. Dadurch kann festgestellt werden, für welche Kurse die Anmeldung gelten soll. Die Anmeldung kann durch die Servlets *Klassenbuch* und *Kursleiterbuch* akzeptiert werden.

Anmeldungen zu weiteren Kursen werden für eingetragene Benutzer durch das Anmeldungsmodul des Seitenkomponisten vorgenommen.

4.3.2.3 Servlets *NeueStudentenBearbeiten* und *NeueKursleiterBearbeiten*

Nach der bereits beschriebenen Erstanmeldung von Studenten und Kursleitern kann ihnen über die Servlets *NeueStudentenBearbeiten* bzw. *NeueKursleiterBearbeiten* ihre Zugangsberechtigung erteilt werden. Die Rolle von *NeueStudentenBearbeiten* kann mit der eines Studentensekretariats verglichen werden, das Servlet wird daher auch über einen Schalter [Sekretariat] im Menü des Kursleiters aufgerufen.

Beide Servlets unterscheiden zwischen einem Aufruf über die Get- bzw. Post-Methode. Der erste Aufruf erfolgt über die Get-Methode. Hierbei erzeugen beide Servlets ein Formular, in dem die jeweiligen Neuanmeldungen aufgeführt sind. Neuanmeldungen können über den Wert Null im Feld *account* der Tabellen *studenten* bzw. *kursleiter* identifiziert werden. Über Auswahlschalter innerhalb des Formulars kann jede Anmeldung angenommen, abgelehnt oder zurückgestellt werden.

Das Absenden des Formulars führt zum Aufruf der Post-Methode des jeweiligen Servlets, das abhängig vom Zustand der Auswahlknöpfe die Anmeldung verarbeitet. Eine akzeptierte Anmeldung erzeugt einen neuen Eintrag in der Tabelle *accounts* und eine Anpassung des Feldes *account*. Über E-Mail wird der Benutzer über seine erfolgreiche Bewerbung informiert. Er erhält gleichzeitig Benutzerkennung und Paßwort, beides wird automatisch erzeugt. Eine Ablehnung führt zur Versendung einer entsprechenden E-Mail und zur Vernichtung der zugehörigen Daten in der Datenbank. Eine zurückgestellte Anmeldung wird nicht verändert.

Detaillierte Daten einer Neuanmeldung können im Bearbeitungsformular über einen speziellen Knopf angefordert werden. Hierdurch wird wiederum die Get-Methode aufgerufen, diesmal jedoch mit einem Parameter. Übergeben wird die Identifikationsnummer des

Studenten bzw. Kursleiters, und die zugehörigen Daten aus den Tabellen *studenten* bzw. *kursleiter* werden zurückgesendet.

Das Servlet *NeueStudentenBearbeiten* sollte nur von Kursleitern aufgerufen werden können, *NeueKursleiterBearbeiten* nur von einem speziellen Administrator. Beide Zugangsbeschränkungen werden über Authentifizierungsfilter realisiert, Details hierzu finden sich im Abschnitt 4.4.1 und im Anhang A.

4.3.2.4 Servlets *Klassenbuch* und *Kursleiterbuch*

Mit diesen Servlets können bereits eingeschriebene Studenten bzw. Kursleiter für die Teilnahme an einem Kurs oder die Betreuung eines Kurses eingetragen werden. *Klassenbuch* ist über einen entsprechenden Schalter im Menü der für den Kurs zuständigen Kursleiter zu erreichen. Diese erhalten eine Liste aller eingeschriebenen Studenten, die sich neu um die Teilnahme am Kurs beworben haben. Der Aufbau des erzeugten Formulars und die hinter dem Servlet stehende Realisierung ist der des im Abschnitt 4.3.2.3 beschriebenen Servlets *NeueStudentenBearbeiten* ähnlich und wird deshalb nicht weiter beschrieben. Für akzeptierte Studenten wird eine Reihe von Datenbankeinträgen vorgenommen, die u. a. die Grundeinstellung der Einstufungen des Studenten darstellen.

Kursleiterbuch kann nur von einem Kursleiter mit speziellen Administrationsrechten aufgerufen werden, dieser nimmt die Eintragung bzw. Ablehnung aller Anmeldungen von Kursleitern für sämtliche Kurse zentral vor. Bis auf diesen Unterschied ist *Kursleiterbuch* dem Servlet *Klassenbuch* ähnlich. Die vorgenommenen Grundeinstellungen können von jedem Kursleiter später frei geändert werden, um sämtliche Sichten eines Studenten auf einen Kurs in den unterschiedlichen Wissensgruppen nachvollziehen zu können. Die Berechtigung zum Aufruf des Servlets wird über Authentifizierungsfilter vorgenommen, hierzu sei auf die Abschnitte 4.4.1 und Anhang A verwiesen.

4.3.2.5 Servlet *KursInfo*

Das Servlet *KursInfo* erzeugt eine Seite mit wichtigen Informationen über einen Kurs. Angezeigt werden die Beschreibung des Kurses und eine Liste aller zuständigen Kursleiter mit deren Namen, E-Mail-Adressen und Sprechstunden.

4.3.2.6 Servlet *NewsInit*

Aufgabe dieses Servlets ist es, dem Kursteilnehmer bzw. Kursleiter den Zugang zu den Diskussionsforen von KOLIBRI zu ermöglichen. Zu diesem Zweck wird beim Aufruf des Servlets ein neues Fenster des WWW-Clients geöffnet, das in vier Frames unterteilt ist.

Initialisierung und Anzeigen der Forenliste Initialisiert wird das Servlet über den Schalter *Pinnwand* in der Menüzeile, wobei dem Servlet *NewsInit* per Get-Methode als Parameter der Name der auszuführenden Aktion (*init*) übergeben wird.

Ist der Aufruf erfolgt, so werden im oberen linken Frame die aktuell vorhandenen Foren und die Anzahl der zugehörigen Nachrichten angezeigt. Die vorhandenen Foren werden aus der Datenbank-Tabelle *foren* geladen, die Anzahl der Nachrichten pro Forum wird aus der Datenbank-Tabelle *news* ermittelt.

Anzeigen der Nachrichtenliste Wurde vom Benutzer durch einen Mausklick ein Forum ausgewählt, so werden die zu diesem Forum gehörenden Nachrichten in Tabellenform innerhalb des rechten oberen Frames ausgegeben. Antworten auf eine ältere Nachricht erkennt man daran, daß ihnen zwei Größer-Zeichen (») vorangestellt sind. Zu diesem Zweck wird dem Servlet die Foren-Kennung als URL-Parameter übergeben, und zwar einschließlich der auszuführenden Aktion `action=ShowNews`. Anhand der Kennung erfolgt eine SQL-Abfrage der Datenbank-Tabelle *news*. Zu beachten ist, daß Original-Nachrichten innerhalb des Feldes *betreff_id* der Tabelle *news* den Wert Null, und Antworten auf eine Nachricht einen Wert größer Null besitzen. Dieser Wert ist der Verweis auf die eindeutige Kennung der entsprechenden Original-Nachricht.

Anzeigen einer Nachricht Hat der Benutzer eine Nachricht ausgewählt, so werden folgende Parameter an das Servlet übergeben:

- Name der auszuführenden Aktion (*ShowMessage*).
- Kennung *Forum_id* des Forums.
- Eindeutige Kennung *id* der Nachricht.
- Kennung *betreff_id* des Betreffs der Nachricht.

Anhand dieser Informationen wird eine SQL-Abfrage erstellt und ausgeführt. Anschließend wird die HTML-Seite der Nachricht generiert und im mittleren Frame angezeigt.

Beantworten einer Nachricht Damit auf eine Nachricht geantwortet werden kann, muß zuvor eine Nachricht aus der Nachrichtenliste ausgewählt worden sein. Durch Drücken des Schalters [Beantworten] innerhalb der HTML-Seite der Original-Nachricht werden an das Servlet folgende Parameter übergeben:

- Name der auszuführenden Aktion (*ShowReplyMessage*).
- Eindeutige Kennung *id* des Forums.

- Eindeutige Kennung *news_id* der Nachricht.

Anhand dieser Informationen wird eine SQL-Abfrage erstellt und ausgeführt. Danach wird die HTML-Seite der Antwortnachricht generiert und im mittleren Frame angezeigt. Sie enthält neben den Informationen zur Original-Nachricht u. a. die Eingabefelder, die vom Benutzer auszufüllen sind. Wurden alle benötigten Daten eingegeben und der Schalter [Antworten] gedrückt, erfolgt ein Eintrag in die Datenbank-Tabelle *news*. Ist der Datenbank-Eintrag erfolgreich verlaufen, wird vom *NewsInit*-Servlet automatisch eine positive Bestätigungs-Seite erzeugt und angezeigt. Andernfalls wird eine Fehler-Seite erzeugt und angezeigt. Diese enthält eine SQL-Fehlernummer und (wenn vorhanden) eine kurze Beschreibung des Fehlers. Anschließend werden automatisch die Forenliste und die Nachrichtenliste mit den neuen Datenbankinhalten gefüllt.

Anlegen eines Forums Diese Funktionalität ist nur für Kursleiter sichtbar bzw. verfügbar. Wird der Schalter [Forum einfügen] aktiviert, wird an das Servlet der Parameter *ShowInsertForum* übergeben. Dies ist der Name der auszuführenden Aktion. Das Servlet wertet den Parameter aus und erzeugt eine HTML-Seite, die im mittleren Frame angezeigt wird. Diese enthält u. a. ein Eingabefeld für den Namen des neu zu erstellenden Forums. Nachdem der Kursleiter die benötigten Informationen eingetragen hat und der Schalter [Einfügen] betätigt wurde, werden diese Informationen an das Servlet übergeben und in die Datenbank-Tabelle *foren* eingefügt.

Löschen eines Forums Diese Funktionalität ist nur für Kursleiter sichtbar bzw. verfügbar. Wird der Schalter [Forum löschen] aktiviert, werden an das Servlet folgende Parameter übergeben:

- Name der auszuführenden Aktion (*DeleteForum*).
- Kennung *id* des zu löschenden Forums.

Das Servlet wertet diese Parameter aus, löscht zuerst alle Nachrichten und dann das gewünschte Forum aus der Datenbank.

4.3.3 Datenbank

KOLIBRI benötigt zur Verwaltung der Benutzerdaten und Kursinhalte sowie Kursstrukturen eine zentrale Datenbank. Da als Programmiersprache Java zum Einsatz kam, wurde eine Datenbank mit einer Schnittstelle nach *Java Database Connectivity (JDBC)* benötigt. Des weiteren sollte die Datenbank für Einsätze in Forschung und Lehre kostenfrei sein. Durch den Einsatz der JDBC-Schnittstelle sollte außerdem ein nachträglicher Wechsel der eingesetzten Datenbank möglich sein. Die Datenbank sollte die *Standard*

Query Language (SQL) zumindest in Form eines Teiles von ANSI-SQL unterstützen. Diese Bedingungen werden z. Z. von zwei Datenbank-Servern gewährleistet, zum einen *Mini-SQL (mSQL)* von Hughes Technologies sowie *PostgreSQL* der PostgreSQL Organisation. Die Entscheidung für mSQL ist aus Effizienzgründen erfolgt, da ein deutlicher Geschwindigkeitsvorteil gegenüber PostgreSQL vorliegt. Der im Vergleich zu PostgreSQL eingeschränkte SQL-Befehlssatz ist für das System KOLIBRI ausreichend.

4.3.3.1 Verwendete Version von mSQL

Zum Einsatz kam zunächst mSQL Version 1.0.16 von Hughes Technologies (siehe dazu [Hugh97]). Im Verlauf des Projektes wurde die Datenbank auf mSQL Version 2.0.3 umgestellt, da die neuere Version das Beta-Stadium verlassen hatte. Einige Vorteile wie zusätzliche Feldertypen und die bessere Schlüsselverwaltung rechtfertigten die Umstellung. Die Vorteile der Verwendung von JDBC zeigten sich in Form der reibungslosen Umstellung.

4.3.3.2 Verwendeter Treiber für JDBC

Zum Einsatz kam die Version JDBC-mSQL 1.0b3 von Imaginary (siehe dazu [Imag97]). Im Verlauf des Projektes kamen die Versionen 1.0a4 und 1.0b1 zum Einsatz.

4.3.3.3 Datenbankschema

Die Abbildung 4.12 auf der nächsten Seite zeigt die Tabellenabhängigkeiten der Datenbank *kolibri*. Dabei repräsentiert jedes Rechteck eine Tabelle. Der Name der Tabelle steht jeweils in der oberen Zeile des Rechtecks. Ein Pfeil von einer Tabelle zu einer anderen ist überall dort eingezeichnet, wo innerhalb einer Tabelle (Ursprung des Pfeiles) auf einen Datensatz einer anderen Tabelle verwiesen wird (Ziel des Pfeiles).

4.3.3.4 Tabellenbeschreibungen

Im folgenden werden die Tabellen der Datenbank sowie deren Spalten beschrieben.

Tabelle *accounts* In der Tabelle *accounts* werden alle Accounts der das System nutzenden Personen eingetragen. Das Feld *id* ist eindeutig und wird als Verweis auf einen Eintrag in dieser Tabelle verwendet. Außerdem wird in *talk* vermerkt, ob der Benutzer zum 1-zu-1-Chat bereit ist (*Y*) oder nicht (*N*). Im Feld *aktKurs* wird die Kennung des Kurses gespeichert, in dem sich der Student angemeldet hat oder *null*, falls der Student gerade nicht angemeldet ist. Wann er sich zum letzten Mal angemeldet hat, wird in dem Wertepaar (*tag, uhr*) gespeichert. Das Feld *student* legt fest, ob es sich bei dem Account um den eines Studenten (*Y*) oder Kursleiters (*N*) handelt.

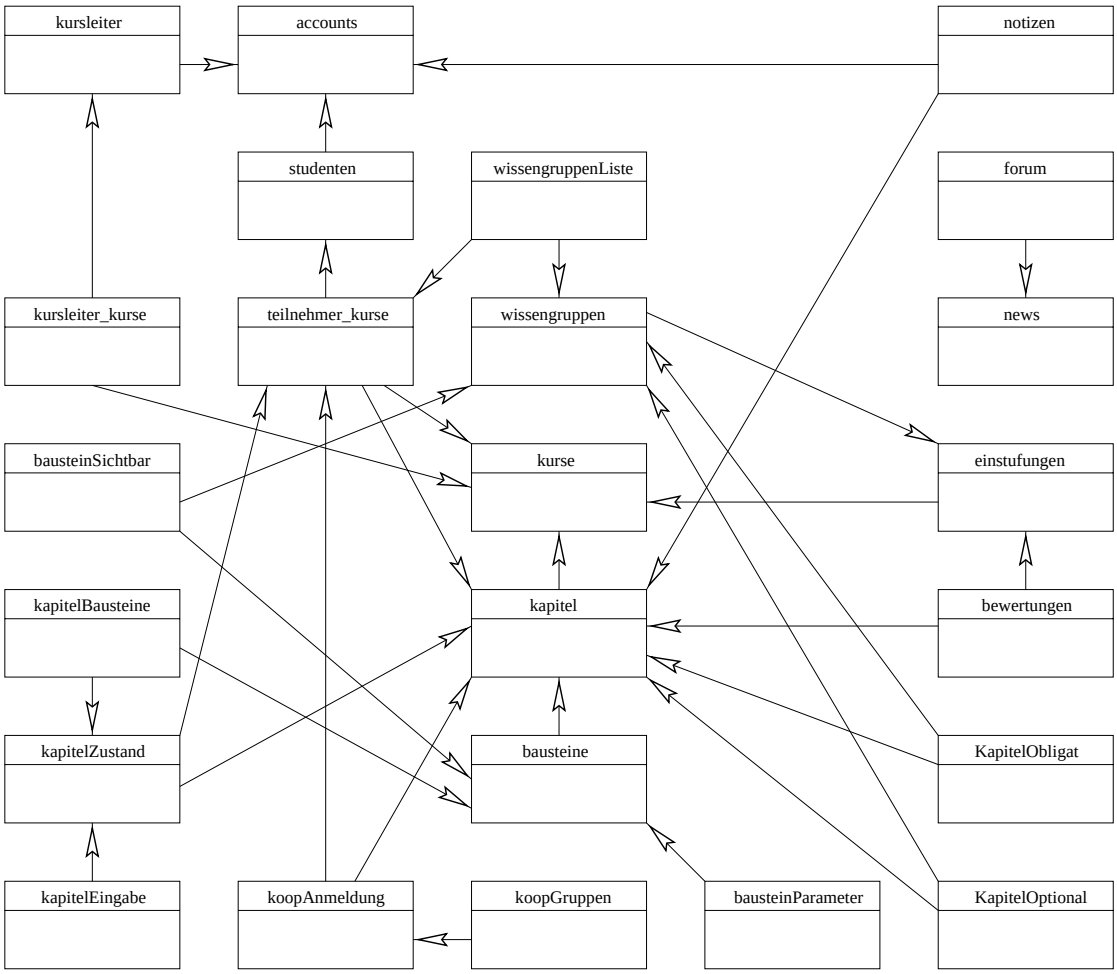

Abbildung 4.12: Tabellenabhängigkeiten

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
name	char	30	J
password	char	30	J
talk	char	1	N
student	char	1	N
aktKurs	int	4	N
zeitpunkt	char	30	N
tag	date	4	N
uhr	time	4	N

Tabelle 4.1: Schema der Tabelle *accounts*

Tabelle *studenten* Diese Tabelle enthält sowohl die eingeschriebenen Studenten als auch die angemeldeten Studenten. Letztere haben noch keinen Account, d. h. das Feld *account* hat den Wert Null. Ein Account wird nach Sichtung durch einen Administrator vergeben. Es muß dem Studenten jedoch vorher ein Account in der Tabelle *accounts* eingerichtet werden, auf den in dieser Tabelle verwiesen werden kann. Eine Eindeutigkeit von *id* wird vom System sichergestellt, da in anderen Tabellen mittels genau dieser *id* auf einzelne Studenten verwiesen werden können muß. Alle weiteren Felder sind dem Schema zu entnehmen, die Semantik ergibt sich jeweils unmittelbar aus dem Namen.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
name	char	30	J
vorname	char	30	J
strasse	char	30	N
hausnummer	char	5	N
plz	int	4	N
ort	char	30	N
telefon	char	30	N
staat	char	30	N
email	char	50	N
schule	char	50	N
semester	int	4	N
matrikelnummer	char	30	N
motivation	char	255	N
wissen	char	255	N
account	int	4	N

Tabelle 4.2: Schema der Tabelle *studenten*

Tabelle *teilnehmer_kurse* In dieser Tabelle wird vermerkt, welche Studenten für welche Kurse eingeschrieben sind. Das Feld *id* bildet den eindeutigen Schlüssel. Falls der Student mit *id* X im Kurs mit *id* Y eingeschrieben ist, gibt es in dieser Tabelle einen entsprechenden Eintrag. Zusätzlich wird in dieser Tabelle vermerkt, welches das aktuelle Kapitel von Student X in Kurs Y ist. Falls er neu in den Kurs aufgenommen wird, wird sein aktuelles Kapitel auf das erste Kapitel des Kurses gesetzt.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kurs_id	int	4	J
student_id	int	4	J
aktuellKapitel	int	4	N

Tabelle 4.3: Schema der Tabelle *teilnehmer_kurse*

Tabelle *kursleiter* In dieser Tabelle werden die Daten der Kursleiter gespeichert. Die Bedeutung des Feldes *account* ist wie bei der Tabelle *studenten*.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
name	char	30	J
vorname	char	30	J
strasse	char	30	N
hausnummer	char	5	N
plz	int	4	N
ort	char	30	N
staat	char	30	N
email	char	50	N
schule	char	50	N
sprechstunden	char	255	N
account	int	4	N

Tabelle 4.4: Schema der Tabelle *kursleiter*

Tabelle *kursleiter_kurse* In dieser Tabelle wird vermerkt, welche Kursleiter für welche Kurse verantwortlich sind. Die Felder *kurs_id* und *kursleiter_id* bilden gemeinsam den eindeutigen Schlüssel. Falls der Kursleiter mit *id* X einer der Kursleiter des Kurses mit *id* Y ist, gibt es in dieser Tabelle einen entsprechenden Eintrag.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kurs_id	int	4	J
kursleiter_id	int	4	J
aktuellKapitel	int	4	N

Tabelle 4.5: Schema der Tabelle *kursleiter_kurse*

Tabelle *bausteine* In dieser Tabelle sind alle Bausteine gespeichert. Das Feld *id* ist eindeutiger Schlüssel. Für jeden Baustein ist vermerkt, welchem Kapitel er zugeordnet ist und an welcher Position im Kapitel er erscheinen soll (*kapitel_id, pos*). Dem Baustein ist ein Bausteintyp *typ_id* zugewiesen, in *verweis* ist der zugehörige Dateiname vermerkt. Im Feld *beschreibung* wird der Inhalt des Bausteins kurz beschrieben (dieser Text wird dem Benutzer als Hyperlink präsentiert, wenn der Baustein für ihn nicht sichtbar ist). Außerdem ist der *autor* des Bausteins vermerkt. Das Feld *spezial_typ* enthält den Verwendungszweck eines Bausteins (siehe Abschnitt 5.1.1.9). Das Feld *name* speichert den Bezeichner eines Bausteins, der zum Beispiel zum Auflösen von Verweisen dient. Die Felder *breite* und *hoehe* enthalten Informationen zu den Abmessungen des Bausteins in der erzeugten HTML-Seite.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kapitel_id	int	4	N
pos	int	4	J
spezial_typ	int	4	N
typ_id	int	4	J
verweis	char	255	N
beschreibung	char	255	N
name	char	255	N
autor	char	100	N
breite	int	4	N
hoehe	int	4	N

Tabelle 4.6: Schema der Tabelle *bausteine*

Tabelle *bausteinParameter* In dieser Tabelle werden (*name, wert*)-Paare gespeichert, die über das Feld *baustein_id* auf Datensätze der Tabelle *bausteine* verweisen. Die Parameter werden genutzt, um zusätzliche Informationen an Applets zu übergeben, die in eine HTML-Seite eingebettet werden.

Feld	Typ	Länge	*null* nicht erlaubt
baustein_id	int	4	J
name	char	50	J
wert	text	50	N

Tabelle 4.7: Schema der Tabelle *bausteinParameter*

Tabelle *kapitel* In dieser Tabelle werden alle Kapitel verwaltet. Jeder Datensatz ist über das Feld *id* eindeutig. Jedes Kapitel hat außerden einen eindeutigen Namen und eine anzuzeigende Kapitelnummer (Feld *kapitel_nr*, enthält z. B. „1.2.3“). Das Feld *spezial_typ* enthält den Typ eines Kapitels (siehe Abschnitt 5.1.2). Eine Referenz auf die Tabelle *kurse* wird durch das Feld *kurs_id* hergestellt. Das Feld *auswertung* dient zur Speicherung von Informationen bezüglich der Auswertung des Kapitels, enthält also z. B. den Namen einer spezialisierten Klasse zur Auswertung von Aufgaben (siehe Abschnitt 5.1.2.4). Die Felder *punkte* und *erfolg* geben die Punkte und die Erfolgsgrenze eines Kapitels an (verwendet etwa bei Aufgaben und Projekten).

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kurs_id	int	4	J
preorder	int	4	J
name	char	50	N
beschreibung	char	255	N
autor	char	255	N
kapitel_nr	char	30	J
spezial_typ	int	4	N
vater_id	int	4	N
nachfolger_id	int	4	N
auswertung	text	100	N
punkte	int	4	N
erfolg	int	4	N
lernkontrolle_id	int	4	N

Tabelle 4.8: Schema der Tabelle *kapitel*

Außerdem beschreibt diese Tabelle die Baumstruktur der Kapitel aller von KOLIBRI verwalteten Kurse, also einen Wald. Das Feld *vater_id* ist bei allen Kapitelknoten innerhalb eines Baumes außer der Wurzel ein Verweis auf den Vaterkapitelknoten, bei Wurzelknoten ist es *null*. Das Feld *nachfolger_id* beschreibt die Ordnung von Knoten, die den gleichen Vater haben. Falls ein Kapitel keinen Nachfolger hat, ist der Wert des Feldes *nachfolger_id null*. Das Feld *preorder* beschreibt für jedes Kapitel innerhalb eines Kurses seine Preorder-Nummer. Über dieses Feld sortiert können alle Kapitel eines Kurses

effizient in geordneter Form ausgelesen werden. Abbildung 4.13 zeigt die Semantik des Feldes an einem Beispiel.

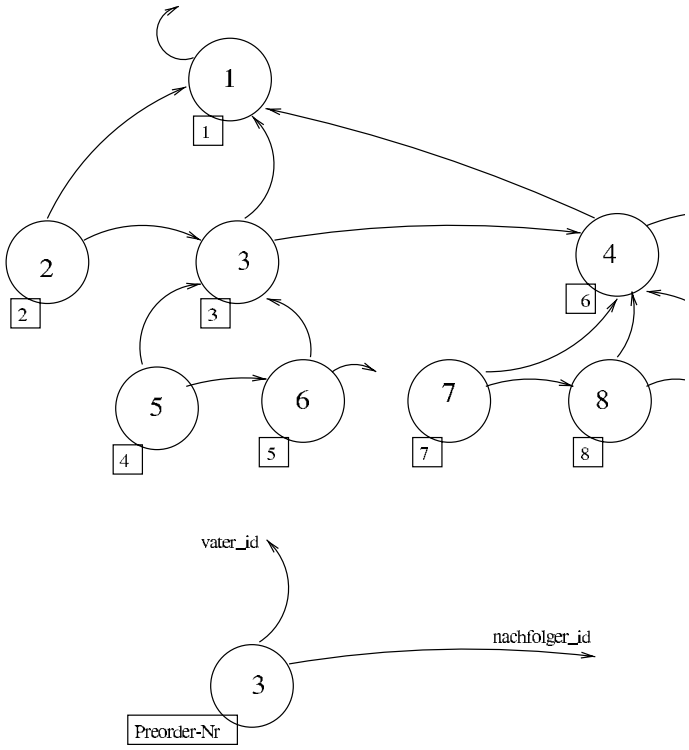

Abbildung 4.13: Kursstrukturbeispiel (Graph der Kapitel)

Angenommen in der Tabelle *kapitel* stünden entsprechende *kapitel_nr*, dann wäre die Struktur wie folgt:

kapitel_nr	kapitel_id
	1
1	2
2	3
2.1	5
2.2	6
3	4
3.1	7
3.2	8
3.3	9

Tabelle 4.9: Kursstrukturbeispiel (Hierarchie der Kapitel)

Der daraus resultierende Inhalt der Tabelle *kapitel* ist der Tabelle 4.10 zu entnehmen.

id	kurs_id	vater_id	nachfolger_id	preorder	...
1	1	*null*	*null*	0	
2	1	1	3	1	
3	1	1	4	2	
4	1	1	*null*	5	
5	1	3	6	3	
6	1	3	*null*	4	
7	1	4	8	6	
8	1	4	9	7	
9	1	4	*null*	8	

Tabelle 4.10: Kursstrukturbeispiel (Tabelleninhalt)

Tabelle *kurse* In dieser Tabelle werden die Kurse verwaltet, die über einen KOLIBRI-Server angeboten werden. Jeder Kurs hat eine eindeutige Kennung (*id*), einen Namen (*name*), eine Beschreibung (*beschreibung*) und einen Autor (*autor*).

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
name	char	30	N
beschreibung	char	255	N
autor	char	255	N
dateibasis	char	255	N
wwwbasis	char	255	N

Tabelle 4.11: Schema der Tabelle *kurse*

Tabelle *wissengruppen* In dieser Tabelle sind die Daten aller Wissensgruppen gespeichert. Das Feld *id* ist eindeutig, das Feld *einstufung_id* ein Verweis auf einen Eintrag in der Tabelle *einstufungen*. Das Feld *pos* gibt die Position einer Wissensgruppe innerhalb ihrer Einstufung an. Das Feld *beschreibung* enthält eine kurze Beschreibung einer Wissensgruppe. Das Feld *name* dient als interner Bezeichner der Wissensgruppe und wird im wesentlichen zum Formulieren der Sichtbarkeitsregeln innerhalb einer Kursdatei verwendet.

Die Felder *obereSchranke* und *untereSchranke* geben an, bei welchem (Wissens-)Punktestand ein Kursteilnehmer von der expliziten oder impliziten Wissensstandkontrolle in die nächsthöhere bzw. -niedrigere Wissensgruppe eingestuft wird. Eine detailliertere Beschreibung ist Abschnitt 5.1.3.1 zu entnehmen.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
einstufung_id	int	4	J
pos	int	4	J
beschreibung	char	255	N
name	char	255	N
obereSchranke	int	4	N
untereSchranke	int	4	N

Tabelle 4.12: Schema der Tabelle *wissengruppen*

Für einen fiktiven Kurs mit *id* 1 könnte diese Tabelle folgendermaßen gefüllt sein:

id	einstufung_id	pos	beschreibung	obereSchranke	untereSchranke
1	1	1	Anfänger	10	-10
2	1	2	Fortgeschrittener	25	-10
3	1	3	Experte	40	-10
4	2	1	Nicht-Theoretiker	15	-10
5	2	2	Theoretiker	10	-10
6	3	1	Nicht-Praktiker	10	-10
7	3	2	Praktiker	30	-30

Tabelle 4.13: Beispieltabelle für Wissensgruppen

Tabelle *einstufungen* In dieser Tabelle werden die Einstufungen gespeichert. Das Feld *id* ist eindeutiger Schlüssel jedes Datensatzes. Das Feld *kurs_id* verweist auf einen Eintrag in der Tabelle *kurse*. Das Feld *beschreibung* dient der Namensgebung von Einstufungen. Das Feld *name* dient als interner Bezeichner der Einstufung und wird im wesentlichen zur Beschreibung von Regeln der expliziten und impliziten Wissensstandkontrolle innerhalb einer Kursdatei verwendet.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
beschreibung	char	255	N
name	char	255	N
kurs_id	int	4	J

Tabelle 4.14: Schema der Tabelle *einstufungen*

Tabelle *wissengruppenListe* Diese Tabelle gibt an, wieviele Punkte ein Student innerhalb einer Wissensgruppe hat. Die Felder *teilnehmer_id* und *gruppe_id* sind zusammen eindeutig. Im Feld *punkte* sind sowohl positive als auch negative Werte erlaubt. Das Feld

gruppe_id ist ein Verweis auf einen Eintrag in der Tabelle *wissengruppen*.

Feld	Typ	Länge	*null* nicht erlaubt
teilnehmer_id	int	4	J
gruppe_id	int	4	J
punkte	int	4	J

Tabelle 4.15: Schema der Tabelle *wissengruppenListe*

Tabelle *notizen* In dieser Tabelle werden im Feld *notiz* Daten gespeichert, die ein Systemteilnehmer *account_id* als Notiz bezüglich dem Kapitel *kapitel_id* gespeichert hat. Einträge ohne Kapitel werden als allgemeine Notizen interpretiert. Das Feld *beschreibung* soll eine Überschrift zur Notiz enthalten, die der Systemteilnehmer bestimmen kann.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
account_id	int	4	J
kapitel_id	int	4	N
beschreibung	char	100	N
notiz	text	100	N

Tabelle 4.16: Schema der Tabelle *notizen*

Tabelle *bewertungen* In dieser Tabelle wird zu Kapiteln (*kapitel_id*) vermerkt, wieviele Punkte (*punkte*) für die jeweilige Einstufung (*einstufung_id*) vergeben werden. Die Tabelle wird sowohl für explizite als auch implizite Wissensstandkontrollen verwendet. Aus dem Typ eines Kapitels ist implizit klar, welcher Mechanismus greift.

Feld	Typ	Länge	*null* nicht erlaubt
kapitel_id	int	4	J
einstufung_id	int	4	J
punkte	int	4	J

Tabelle 4.17: Schema der Tabelle *bewertungen*

Tabelle *kapitelOptional* Diese Tabelle enthält die *Optional*-Relation der Kapitel. Ist ein Kapitel X für Wissensgruppe Y optional, so existiert hier ein Eintrag mit *kapitel_id* = X und *gruppe_id* = Y. *kapitel_id* ist Verweis auf einen Eintrag in der Tabelle *kapitel*, *gruppe_id* Verweis auf einen Eintrag in der Tabelle *wissengruppen*. Aus dem Wert des Feldes *kapitel_id* ergibt sich der zugehörige Kurs, so daß diese Information in der Tabelle nicht explizit aufgeführt werden muß.

Feld	Typ	Länge	*null* nicht erlaubt
kapitel_id	int	4	J
gruppe_id	int	4	J

Tabelle 4.18: Schema der Tabelle *kapitelOptional*

Tabelle *kapitelObligat* Diese Tabelle enthält die *Obligatorisch*-Relation der Kapitel. Ist ein Kapitel X für Wissensgruppe Y obligatorisch, so existiert hier ein Eintrag mit *kapitel_id* = X und *gruppe_id* = Y. *kapitel_id* ist Verweis auf einen Eintrag in der Tabelle *kapitel*, *gruppe_id* Verweis auf einen Eintrag in der Tabelle *wissengruppen*. Aus dem Wert des Feldes *kapitel_id* ergibt sich der zugehörige Kurs, so daß diese Information in der Tabelle nicht explizit aufgeführt werden muß.

Feld	Typ	Länge	*null* nicht erlaubt
kapitel_id	int	4	J
gruppe_id	int	4	J

Tabelle 4.19: Schema der Tabelle *kapitelObligat*

Tabelle *bausteineSichtbar* Diese Tabelle enthält die Sichtbarkeitsrelation für Bausteine. Ist ein Baustein X für Wissensgruppe Y sichtbar, existiert hier ein Eintrag mit *baustein_id* = X und *gruppe_id* = Y. Aus dem Wert des Feldes *baustein_id* ergeben sich sowohl das zugehörige Kapitel als auch der zugehörige Kurs, so daß diese Informationen in der Tabelle nicht explizit aufgeführt werden müssen.

Es sei darauf aufmerksam gemacht, daß dies sowohl Kurs als auch Kapitel impliziert.

Feld	Typ	Länge	*null* nicht erlaubt
baustein_id	int	4	J
gruppe_id	int	4	J

Tabelle 4.20: Schema der Tabelle *bausteineSichtbar*

Tabelle *kapitelZustand* In dieser Tabelle wird protokolliert, wie (*zustand*) und wann (*tag, uhr*) welcher Student (*besucher_id*) in welchem Kapitel (*kapitel_id*) eine Eingabe gemacht hat, die den Zustand des Kapitels beeinflußt hat. In der Tabelle *kapitelEingabe* sind Einträge gespeichert, die auf die eindeutige *id* der Datensätze dieser Tabelle verweisen. Das Feld *besucher_id* wird entweder zur Speicherung der Kennung eines Studenten oder einer kooperativen Gruppe benutzt. Die Semantik ergibt sich durch die *kapitel_id*, die impliziert, ob das Kapitel ein *Gruppenkapitel* ist. Im Feld *bausteine* wird eine Liste der vom Benutzer zusätzlich expandierten Bausteine gespeichert. Dabei werden die Baustein-Kennungen durch Kommata getrennt in dem Text-Feld gespeichert. Dies ist aus

Effizienzgründen einer relationalen Datenhaltung vorzuziehen, da sonst unnötige Umwandlungen im Hinblick auf Teil-URLs, die genau diesen Typ beinhalten, durchgeführt werden müßten. Der Eintrag bleibt leer, wenn der Student die Bausteinsichtbarkeiten nicht verändert.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
besucher_id	int	4	J
kapitel_id	int	4	J
tag	date	4	J
uhr	time	4	J
zustand	char	1	J
punkte	int	4	N
istStudent	char	1	N

Tabelle 4.21: Schema der Tabelle *kapitelZustand*

Tabelle *kapitelEingabe* In dieser Tabelle werden (*name, wert*)-Paare gespeichert, die über das Feld *zustand_id* auf Datensätze der Tabelle *kapitelZustand* verweisen.

Feld	Typ	Länge	*null* nicht erlaubt
zustand_id	int	4	J
name	char	50	J
wert	text	50	N

Tabelle 4.22: Schema der Tabelle *kapitelEingabe*

Tabelle *koopAnmeldung* In dieser Tabelle werden Anmeldungen von Studenten gespeichert, die eine Teilaufgabe einer kooperativen Aufgabe bearbeiten möchten. Das Feld *id* ist eindeutiger Schlüssel. Das Feld *kapitel_id* gibt an, welches Kapitel die Teilaufgabe enthält. Dabei repräsentiert das Vaterkapitel automatisch die kooperative Aufgabe. Die *teilnehmer_id* weist auf einen eindeutigen Kursteilnehmer der Tabelle *teilnehmer_kurse*. Die Felder *anmeldetag* und *anmeldezeit* werden mit der Systemzeit zur Anmeldung belegt. Die Felder *vergabetag* und *vergabezeit* sind zunächst mit dem Wert *null* zu belegen. Sie werden mit der Systemzeit, zu der eine Bearbeitungsgruppenbildung erfolgt, gefüllt.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kapitel_id	int	4	J
teilnehmer_id	int	4	J
anmeldetag	date	4	J
⋮	⋮	⋮	⋮

⋮	⋮	⋮	⋮
anmeldezeit	time	4	J
vergabetag	date	4	N
vergabezeit	time	4	N
istStudent	char	1	N

Tabelle 4.23: Schema der Tabelle *koopAnmeldung*

Tabelle *koopGruppen* In dieser Tabelle werden Bearbeitungsgruppen für kooperative Aufgaben gespeichert. Das Feld *id* ist eindeutiger Schlüssel für eine Bearbeitungsgruppe. Einen eindeutigen Schlüssel bildet das Paar (*id*, *kAnmeldung_id*). Dabei ist das Feld *kAnmeldung_id* Verweis auf einen Datensatz der Tabelle *koopAnmeldung*.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
kAnmeldung_id	int	4	J
kapZustand_id	int	4	N

Tabelle 4.24: Schema der Tabelle *koopGruppen*

Tabelle *forum* In dieser Tabelle werden die Foren verwaltet, in die die Benutzer Nachrichten schreiben können. Das Feld *id*, das ein Forum eindeutig macht, wird von der Tabelle *news* referenziert. *beschreibung* ist eine kurze Beschreibung des Forums und *tage_loeschen* gibt an, nach wievielen Tagen Nachrichten, auf die keine neueren Antworten geschrieben wurden, automatisch gelöscht werden sollen.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
beschreibung	char	50	J
tage_loeschen	int	4	N

Tabelle 4.25: Schema der Tabelle *forum*

Tabelle *news* In dieser Tabelle werden die Nachrichten selbst gespeichert. *id* ist ein eindeutiger Schlüssel jeder Nachricht. *forum_id* gibt an, in welches Forum eine Nachricht geschrieben wurde, (*tag_alt, uhr_alt*) gibt an, wann die Nachricht verfaßt wurde, und (*tag_neu, uhr_neu*), wann die letzte Antwort auf diese Nachricht geschrieben wurde. *name* ist der (reale) Name des Studenten, Kursleiters oder Gastes, der die Nachricht verfaßt hat, *email* seine E-Mail-Adresse, *betreff_id* ist die *id* der Nachricht, auf die diese Nachricht eine Antwort ist oder *null*. *kopf* ist der Betreff einer Nachricht. Die eigentliche Nachricht wird im Feld *dokument* gespeichert.

Feld	Typ	Länge	*null* nicht erlaubt
id	int	4	J
forum_id	int	4	J
tag_alt	date	4	J
uhr_alt	time	4	J
tag_neu	date	4	J
uhr_neu	time	4	J
name	char	30	N
email	char	50	N
betreff_id	int	4	N
kopf	char	80	N
dokument	text	250	N

Tabelle 4.26: Schema der Tabelle *news*

4.3.3.5 Verwendung von Indizes

Durch Benutzung von Indizes in einer Tabelle sind Anfragen auf Datensätze dieser Tabelle effizienter, falls in der Bedingung Felder mit Index verwendet werden. Intern verwaltet mSQL diese Indizes durch AVL-Bäume. Weiterhin dienen Indizes zur Festlegung eindeutiger Felder. Die folgende Tabelle zeigt die in der Datenbank *kolibri* verwendeten Indizes aller Tabellen. Eine Benutzung der Indexnamen in SQL-Anweisungen ist nicht erforderlich. Sämtliche beschriebenen Vorteile werden automatisch genutzt.

Tabellenname	Indexname	eindeutig	Felder
studenten	idx1	J	id
accounts	idx1	J	id
bausteine	idx1	J	id
bausteinParameter	idx1	N	baustein_id
kurse	idx1	J	id
	idx2	J	name
kapitel	idx1	J	id
	idx2	N	vater_id
kursleiter	idx1	J	id
teilnehmer_kurse	idx1	J	id
kursleiter_kurse	idx1	J	id
wissengruppen	idx1	J	id
wissengruppenListe	idx1	N	teilnehmer_id
notizen	idx1	J	id
	idx2	N	account_id
kapitelZustand	idx1	J	id
⋮	⋮	⋮	⋮

⋮	⋮	⋮	⋮
forum	idx1	J	id
news	idx1	J	id
	idx2	N	tag_neu, uhr_neu
	idx3	N	forum_id
einstufungen	idx1	J	id
koopAnmeldung	idx1	J	id
	idx2	N	kapitel_id
	idx3	N	teilnehmer_id
	idx4	N	istStudent
koopGruppen	idx1	J	id

Tabelle 4.27: Tabellen mit Indizes

4.3.3.6 Verwendung von Sequenzen

Jede Tabelle, in der eindeutige Schlüssel benötigt werden, ist mit einem virtuellen Feld namens *_seq* versehen. Um den nächsten Schlüssel zu erhalten, greift man lesend darauf zu:

```
SELECT _seq FROM studenten
```

mSQL erhöht den Wert des Feldes dann automatisch für den nächsten Zugriff und stellt vor allem sicher, daß nicht zwei Anfragen mit der selben Nummer beantwortet werden. Dieser Mechanismus wird innerhalb von mSQL als *Sequenz* bezeichnet.

Folgende Tabellen verwenden Sequenzen:

- *studenten*
- *kursleiter*
- *accounts*
- *bausteine*
- *news*
- *bausteine_typen*
- *kurse*
- *teilnehmer_kurse*
- *kapitel*
- *wissengruppen*
- *einstufungen*
- *forum*
- *koopGruppen*
- *koopAnmeldung*
- *notizen*
- *kapitelZustand*
- *kursleiter_kurse*

4.4 Kommunikation

4.4.1 Autorisierung

Die meisten Seiten von KOLIBRI sollen nicht von Benutzern eingesehen werden können, die nicht als Studenten oder Kursleiter angemeldet sind. Dies ermöglicht beispielsweise Abrechnungsverfahren und den Schutz der angebotenen Dokumente.

Da die Benutzer des Systems in einer Datenbank gespeichert sind, können als Filter nicht die üblichen Zugriffsbereiche (*Realms*) des WWW-Servers Jigsaw verwendet werden, da diese auf eigene Ressourcen zugreifen und ein doppeltes Eintragen der Benutzer vermieden werden soll. Da Jigsaw bereits Basisklassen für die Entwicklung eigener Filter bietet, werden diese verwendet, um einen speziellen Autorisierungsfilter zu implementieren.

Die Autorisierung soll über das Authentifizierungsverfahren des *Hypertext-Transfer-Protokolls (HTTP)* abgewickelt werden (siehe dazu [Fiel97]). Dazu wird für den WWW-Server ein Filter entwickelt, der HTTP-Anfragen abfängt und überprüft, ob sie autorisiert sind: der `MsqlAuthFilter`. Ist eine Anfrage nicht autorisiert, so wird ein Fehler (401) an den WWW-Client zurückgesendet, der dann den Benutzer auffordert, sich zu identifizieren. Bricht der Benutzer diesen Vorgang ab, wird der vom Filter zusammen mit der Fehlernachricht gesendete HTML-Code mit der Fehlerbeschreibung angezeigt.

Ist eine Anfrage autorisiert, so wird noch überprüft, ob der Benutzer in der Tabelle *accounts* der KOLIBRI-Datenbank (siehe Abschnitt 4.3.3.4) eingetragen ist und ob das

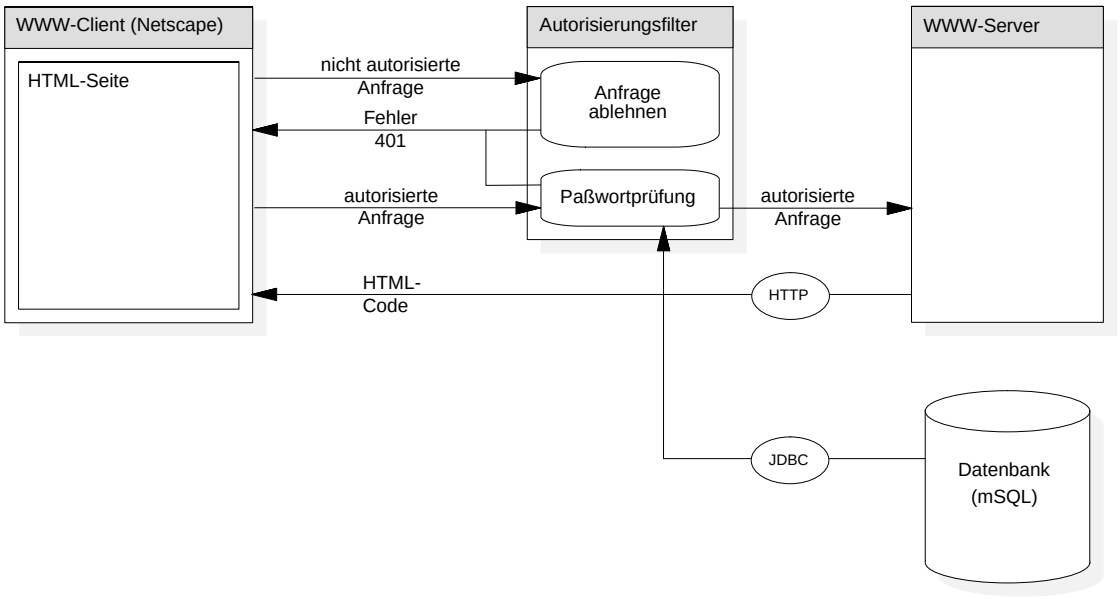

Abbildung 4.14: Autorisierungsverfahren

übermittelte Paßwort mit dem in der Datenbank übereinstimmt. Erst dann wird die autorisierte Anfrage an den WWW-Server weitergeleitet.

Der Filter wird über sämtliche Ressourcen gelegt, auf die nicht frei zugegriffen werden soll. Ein Beispiel ist der *Seitenkomponist*, ein Servlet, das den dynamischen Aufbau von HTML-Seiten erledigt. Der *Seitenkomponist* soll nur angemeldeten Benutzern Kursseiten anzeigen, ein direktes Aufrufen durch Eingabe der Kurs-URL von nicht autorisierten Personen wird durch den Filter verhindert.

Erläuterungen zu Begriffen wie *Filter* und *Ressource* können der Jigsaw-Dokumentation entnommen werden (siehe dazu [W3C98]). Abbildung 4.14 stellt das Authentifizierungsverfahren grafisch dar, wobei die Pfeile den Datenfluß zwischen den einzelnen Komponenten repräsentieren.

4.4.2 Seitenkomponist

Der *Seitenkomponist* sorgt für die individuelle dynamische Erstellung der Kursseiten jedes Studenten oder Kursleiters. Er ist dadurch die zentrale Anlaufstelle für Seitenanforderungen innerhalb des Kurses. Er wurde in Form eines Servlets realisiert, das auf verschiedene Untermodule mit speziellen Aufgaben zurückgreift. Der schematische Aufbau ist in Abbildung 4.15 dargestellt.

Da dieses Servlet von vielen Benutzern gleichzeitig verwendet wird, wurde der Ausnutzung der Multithreading-Fähigkeit hier besondere Beachtung geschenkt (siehe Abschnitt 4.3.2).

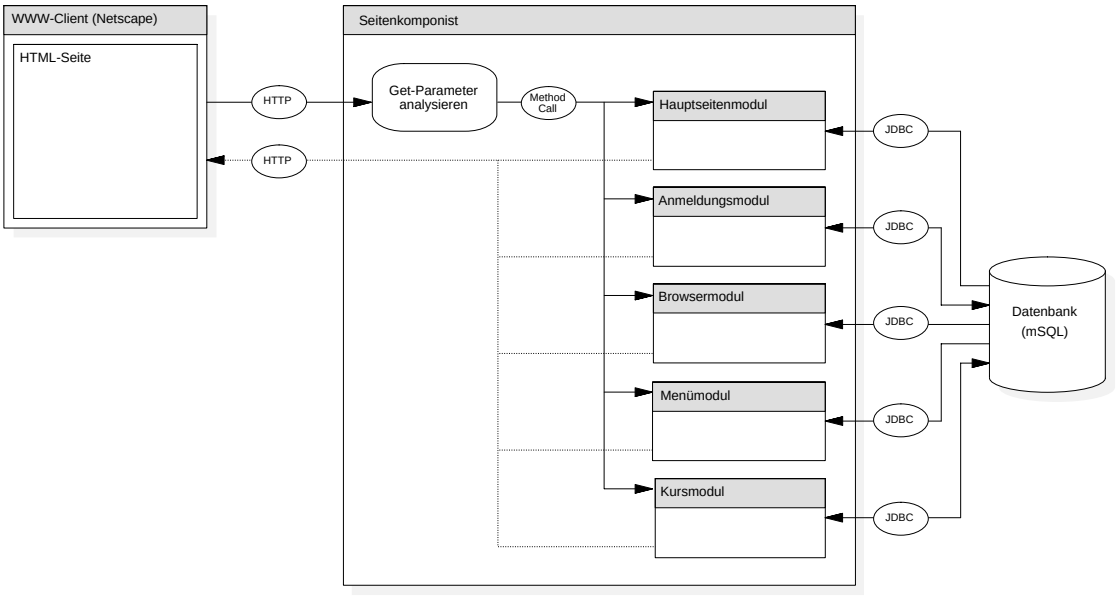

Abbildung 4.15: Schematischer Aufbau des *Seitenkomponist*-Servlets

Durch die Benutzung eines Autorisierungsfilters wird sichergestellt, daß auf dieses Servlet (und damit auf den Kurs) nur eingeschriebene Benutzer zugreifen können. Nach der Auswahl eines Kurses auf der Willkommen-Seite des Systems und der erfolgreichen Authentifizierung sorgt der Seitenkomponist für den Aufbau der Benutzeroberfläche mit Kurs-Browser, Menüzeile und Textbereich.

Welche Aktion auszuführen ist, bekommt der Seitenkomponist über einen URL-kodierten Parameter mitgeteilt. Je nach Wert dieses Parameters übergibt der Seitenkomponist die Bearbeitung an das entsprechende Untermodul. Im folgenden sollen die Untermodule mit ihren Aufgaben beschrieben werden.

4.4.2.1 Hauptseitenmodul

Das Hauptseitenmodul wird sofort nach der Authentifizierung des Benutzers aufgerufen. Es überprüft anhand der Kennung, ob es sich um einen Studenten oder einen Kursleiter handelt. Ist der Benutzer für den gewählten Kurs noch nicht angemeldet, erhält er durch eine spezielle Seite die Möglichkeit hierzu. Über diese Seite wird ggf. das Anmeldungsmodul (s. u.) des Seitenkomponisten aktiviert.

Akzeptierte Benutzer erhalten eine HTML-Seite, über die der weitere Aufbau der Kursoberfläche via *Seitenkomponist* vorgenommen wird.

4.4.2.2 Anmeldungsmodul

Wählt ein eingeschriebener Benutzer einen Kurs, für den er noch nicht angemeldet ist, wird über das Hauptseitenmodul die Seite erzeugt, über die er dies nachholen kann. Anschließend wird über den Seitenkomponisten das Anmeldungsmodul aufgerufen, das die entsprechenden Datenbankeinträge vornimmt. Die Anmeldung von Studenten wird über das Servlet *Klassenbuch* von einem Kursleiter des gewählten Kurses vorgenommen, für Kursleiteranmeldungen erledigt dies ein Administrator mit dem Servlet *Kursleiterbuch* (siehe Abschnitt 4.3.2.4).

4.4.2.3 Browsermodul

Über dieses Modul wird eine HTML-Seite erzeugt, die das Applet für den Kurs-Browser lädt. Diese Seite muß dynamisch erstellt werden, da über einen Applet-Parameter die Kennung des Benutzers an den Kurs-Browser weitergegeben wird (nötig z. B. zur Darstellung der individuellen Kursstruktur).

4.4.2.4 Menümodul

Dieses Modul erzeugt, analog zum Browsermodul, eine HTML-Seite, die das Menüzeilen-Applet für die Benutzeroberfläche lädt. Auch hier muß die Seite dynamisch erstellt wer-

den, um dem Applet die Kennung des Benutzers über einen Parameter mitteilen zu können.

4.4.2.5 Kursmodul

Dieses Modul erzeugt eine HTML-Seite, die die Kursinhalte enthält. Sie wird im rechten Frame angezeigt. Grundlage für den Aufbau der HTML-Seite sind die im Abschnitt 5 beschriebenen Mechanismen. Ein Parameter in der HTTP-Anfrage bestimmt, durch welche Aktion des Benutzers die Seite mit dem Kursinhalt angefordert wird. Dabei wird zwischen den folgenden Aktionen unterschieden:

- Der Benutzer ist gerade in das System eingestiegen. Das Kapitel, bei dem er das letzte Mal zu lesen aufgehört hat, wird angezeigt. Besucht er den Kurs zum ersten Mal, wird das erste Kapitel angezeigt.
- Der Benutzer fordert ein neues Kapitel an. Das kann entweder durch Auswählen im Kurs-Browser oder durch Benutzen eines Verweis-Bausteins geschehen. Das angeforderte Kapitel wird angezeigt und in der Historie des Benutzers vermerkt. Eventuell wird eine implizite Wissensstandkontrolle durchgeführt.
- Der Benutzer fordert über die Historie ein bereits besuchtes Kapitel an. Das Kapitel wird angezeigt.
- Der Benutzer blendet einen Baustein ein oder aus. Das aktuelle Kapitel wird erneut angezeigt. Diesmal jedoch mit bzw. ohne den gewählten Baustein.
- Der Benutzer fordert einen Lösungshinweis an. Das aktuelle Kapitel – eine Teilaufgabe oder ein Teilprojekt – wird erneut angezeigt. Diesmal jedoch mit den als Lösungshinweisen markierten Bausteinen.
- Der Benutzer möchte in einer Teilaufgabe oder einem Teilprojekt die von ihm gemachten Eingaben übernehmen. Seine Eingaben werden in der Datenbank gespeichert. Es wird wieder die Hauptseite der Aufgabe bzw. des Projektes angezeigt.
- Der Benutzer gibt eine Aufgabe oder ein Projekt zur Bewertung frei. Wenn möglich, wird die Aufgabe bzw. das Projekt korrigiert. Es wird wieder die Hauptseite angezeigt.
- Der Benutzer möchte eine Aufgabe oder ein Projekt erneut lösen. Der Zustand der Aufgabe bzw. des Projektes wird zurückgesetzt. Es wird wieder die Hauptseite angezeigt.
- Der Benutzer meldet sich zu einem Projekt an. Seine Anmeldung wird aufgenommen. Es wird wieder die Hauptseite des Projektes angezeigt.

4.4.3 Kommunikationsmanager

Der Kommunikationsmanager vermittelt zwischen dem Server (mit Server ist im folgenden der gesamte KOLIBRI-Server gemeint) und den Clients (WWW-Clients). Er wickelt alle Kommunikation ab, die nicht über HTTP realisiert werden kann. Im WWW-Client des Benutzers stellen Applets die Kommunikationspartner des Managers dar.

4.4.3.1 Konzept

Die Kommunikation zwischen den Applets auf der Clientseite und dem Kommunikationsmanager auf der Server-Seite soll einen einfachen Nachrichtenaustausch ermöglichen. Dabei soll eine Kommunikation nicht nur dann stattfinden, wenn diese durch den Client ausgelöst wird (wie dies bei normalen HTTP-Anfragen der Fall ist), sondern es soll auch möglich sein, daß der Server von sich aus mit dem Client kommuniziert.

Eine Realisierung der Client-Server-Kommunikation muß so flexibel sein, daß sie leicht an unterschiedliche Funktionalitäten angepaßt werden kann. Berücksichtigt werden müssen z. B. die Anforderungen des Kurs-Browsers (siehe Abschnitt 4.2.2.1), eines Applets, das die Kursstruktur grafisch darstellt und dem Benutzer eine Übersicht über den zur Verfügung stehenden Stoff liefert. Der Kurs-Browser soll aber ebenfalls zur Navigation dienen, was bedeutet, daß sowohl Daten an das Applet fließen (z. B. die kodierte Kursstruktur, die vom Applet als Baum dargestellt wird), das Applet aber auch Aktionen auf der Server-Seite auslösen können muß (z. B. die Erzeugung einer neuen Seite veranlassen). Ein ähnliches Anforderungsprofil hat die Chat-Komponente des Systems. Nachrichten, die ein Benutzer eingibt, müssen zum Kommunikationsmanager und von da aus weiter an den oder die adressierten Benutzer geleitet werden (und umgekehrt). Außerdem müssen sämtliche Benutzer informiert werden, wer für ein Online-Gespräch zur Verfügung steht und wer sich von dieser Gesprächsbereitschaft gerade abgemeldet hat.

Es soll auch eine Kommunikation zwischen mehreren Applets innerhalb einer WWW-Seite ermöglicht werden. Diese Kommunikation erfolgt dabei immer über den Server, damit dieser ebenfalls auf die Nachrichten reagieren kann. Eine direkte Kommunikation der Applets untereinander ist nicht vorgesehen.

Der Kommunikationsmanager besteht aus einzelnen Manager-Objekten, die jeweils einen eigenen Dienst zur Verfügung stellen. Im folgenden werden auch die einzelnen Objekte als Kommunikationsmanager bezeichnet; die Vereinigung aller (realen) Kommunikationsmanager-Objekte bildet den (logischen) Kommunikationsmanager.

4.4.3.2 Realisierung

Die Kommunikationsmanager, die jeweils einen bestimmten Dienst zur Verfügung stellen, und die Applets, die auf den entsprechenden Manager zugreifen, werden zu einem Client-Server-Paar (bzw. Applet-Kommunikationsmanager-Paar) zusammengefaßt. Ein

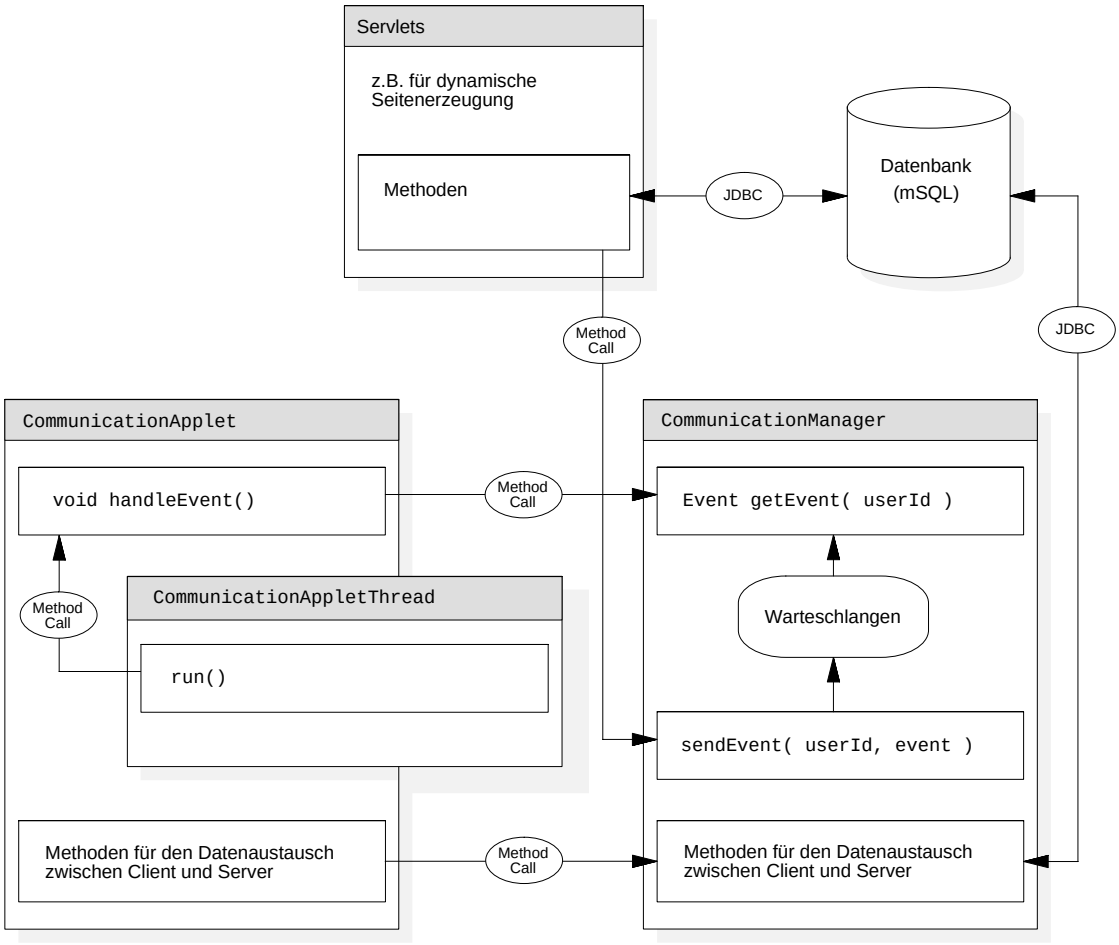

Abbildung 4.16: Kommunikation zwischen Client und Server

Kurs-Browser-Applet auf der Clientseite greift z. B. auf den *KursbrowserManager* auf der Serverseite zu. Für jeden Dienst gibt es Applets, die mit dem Manager des entsprechenden Dienstes Daten austauschen.

Der Austausch der Daten wird über die Schnittstelle der *Remote Method Invocation (RMI)* von Java realisiert. RMI hat gegenüber der direkten Kommunikation über Sockets den Vorteil, daß die gewünschte Funktionalität sehr einfach implementiert werden kann, da Methoden des RMI-Servers (also des Kommunikationsmanagers) genauso aufgerufen werden wie normale Java-Methoden von lokal existierenden Objekten.

Der Datenaustausch zwischen den Applet-Kommunikationsmanager-Paaren ist bei allen Diensten gleich, deshalb wird dieser von zwei Basisklassen bereitgestellt, die die entsprechende Funktionalität an Unterklassen vererben. Die Unterklassen realisieren dann die speziellen Anforderungen der jeweiligen Dienste.

Der Kommunikationsmanager steuert das Verhalten des Applets über Ereignisse (Events). Das Applet wartet darauf, daß ihm sein Manager ein Ereignis übermittelt, und reagiert entsprechend. Applets können durch Senden einer Nachricht an ihren Manager Ereignisse für andere Applet-Instanzen des Dienstes erzeugen. Da die einzelnen Dienste verschiedene Ereignisse benötigen, werden diese erst in den Unterklassen festgelegt. Die Semantik eines Ereignisses muß natürlich bei Applet und Manager gleich sein.

Die Ereignisse sind dabei immer an einen bestimmten Benutzer von KOLIBRI gerichtet. Der Manager verwaltet für jeden Benutzer eine eigene Warteschlange.

Ereignisse können auch von anderen Komponenten auf der Server-Seite im Kommunikationsmanager erzeugt werden. Ein Modul des Seitenkomponisten kann z. B. dem *KursbrowserManager* mitteilen, daß sich durch eine implizite Wissensstandkontrolle die Wissensgruppe eines Benutzers geändert hat und deshalb der Kurs-Browser auf Client-Seite eine neue Darstellung erzeugen muß. Diese Mitteilung erfolgt über die gleiche Schnittstelle, mit der auch die Applets ein Ereignis beim Manager erzeugen.

Der Nachrichtenaustausch zwischen Client und Server wird realisiert, indem das Applet in regelmäßigen Abständen bei seinem Kommunikationsmanager nachfragt, ob neue Nachrichten vorhanden sind. Ist eine Nachricht vorhanden, so ruft der Thread, der die Nachrichten vom Server abholt, die Methode `handleEvent()` auf. In dieser Methode muß das Applet auf die Nachrichten reagieren. In Abbildung 4.16 auf der vorherigen Seite wird dieser Kommunikationsfluß dargestellt.

4.4.3.3 Aufgaben des *LoggingManager*

Die Aufgabe des *LoggingManagers* ist das Überwachen aller z. Z. angemeldeten Benutzer. Sobald ein Benutzer sich über die Kursauswahl in einem Kurs angemeldet hat, sendet das Servlet *Seitenkomponist* eine Nachricht an den *LoggingManager*, der dann Zeit und Datum der Anmeldung in der Datenbank vermerkt (siehe Abschnitt 4.3.3.4). Dadurch wird ein Benutzer als „online“ gekennzeichnet. Das Anmelden muß über das Servlet abgewickelt werden und kann nicht durch ein Applet vorgenommen werden. Sonst wäre es nämlich möglich, daß ein anderes Applet bereits geladen ist und Daten vom Server anfordert, bevor der Benutzer als angemeldet gekennzeichnet wurde. Dies kann zu Problemen beim Datenbankzugriff führen, da auch der aktuelle Kurs bei der Anmeldung vermerkt wird.

Zu dem *LoggingManager* gehört das Applet *LoggingApplet*. Seine primäre Funktion ist die Bereitstellung eines Schalters [Kolibri verlassen], mit dem sich der Benutzer beim System abmelden kann. Der *LoggingManager* nimmt daraufhin die entsprechenden Einträge in der Datenbank vor (siehe ebenfalls Abschnitt 4.3.3.4), die den Benutzer als abgemeldet kennzeichnen.

Für den Fall, daß der Benutzer den Schalter [Kolibri verlassen] nicht verwendet und den WWW-Client einfach schließt oder dieser abstürzt, muß ebenfalls ein Mechanismus vorhanden sein, der dies bemerkt. Dazu wird auf der Server-Seite in der Warteschlange, die

die Nachrichten verwaltet, vermerkt, wann der Client zuletzt die Methode `getEvent()` aufgerufen hat. Der Manager prüft nun in regelmäßigen Abständen, wie lange der Client sich schon nicht mehr gemeldet hat. Wird eine bestimmte Zeitspanne überschritten, so wird der Benutzer automatisch abgemeldet. Es ist aber zu beachten, daß die Zeitspanne nicht zu kurz gewählt werden darf, da es je nach Verbindung einige Zeit dauern kann, bis die Applets geladen sind und sich bei ihrem zugehörigen Manager melden. Zur Zeit prüft der *LoggingManager* alle drei Minuten, ob sich ein Client länger als zwei Minuten nicht gemeldet hat.

4.4.3.4 Aufgaben des *KursbrowserManager*

Der *KursbrowserManager* stellt für das Applet *Kurs-Browser* in erster Linie die Funktionalität der Kommunikation mit der Datenbank zur Verfügung.

Dem Applet *KursBrowser* müssen die komplette Kursstruktur, die Daten des persönlichen Kursverlaufs und die Interessens- bzw. Wissensgruppen mitgeteilt werden, in denen sich der Benutzer gerade befindet. Der *KursbrowserManager* stellt eine Reihe von Methoden zur Verfügung, die diese Informationen aus der Datenbank lesen und an den Client übertragen.

Das Applet *KursBrowser* wird von verschiedenen anderen Komponenten des Systems benachrichtigt, wenn sich im persönlichen Kursverlauf des Benutzers etwas geändert hat. Das bedeutet zum Beispiel: Der Benutzer wurde in eine andere Interessens- bzw. Wissensgruppe eingeordnet, der Benutzer hat ein neues Kapitel ausgewählt, oder er hat eine Lernkontrolle bestanden bzw. nicht bestanden. Über all diese Ereignisse wird das Applet benachrichtigt, was zur Folge hat, daß die Bildschirmanzeige des Kurs-Browsers aktualisiert wird.

4.4.3.5 Aufgaben des *ChatManager*

Der Chat-Manager stellt alle serverseitigen Funktionen des Chat-Applets bereit. Der Manager erhält von den Clients die Ereignisse, die vom Benutzer ausgelöst wurden. Er verwaltet die Benutzer und die Foren und teilt den anderen Clients die abgesendeten Ereignisse mit und liefert die Ergebnisse (Nachrichten an andere Benutzer) aus.

Ereignisse Es werden Ereignisse für folgende Fälle benötigt:

- **Anmeldung beim Manager:** Beim Start des Chat-Forums wird ein Ereignis an den Manager gesendet, das den Benutzer in die Liste der Ereignisbehandlung des RMI einträgt. Dies ist notwendig, damit bei der Anfrage zur 1-zu-1-Kommunikation auch die Benutzer angezeigt werden, die sich zwar auf dem Server befinden, jedoch den Chat nicht betreten haben.

- **Anmeldung in ein Forum:** Der Client sendet das Ereignis zum Manager. Dieser trägt den Benutzer und das Forum in seine Liste ein und benachrichtigt die Clients über den neuen Partner.
- **Abmelden von einem Forum:** Der Client sendet das Ereignis zum Manager. Dieser trägt den Benutzer und das Forum aus seiner Liste aus und benachrichtigt die Clients über das Verlassen des Partner.
- **Ummelden des Forums:** Der Client sendet das Ereignis zum Manager. Dieser ändert die Eintragungen in der Liste und benachrichtigt die Clients des alten Forums über das Ausscheiden und die Partner in dem neuen Forum über den Eintritt des Benutzers.
- **Anfrage an einen Kommunikationspartner:** Der Client sendet das Ereignis zum Manager. Dieser leitet die Anfrage an den Partner weiter.
- **Antwort des Kommunikationspartners:** Der Client sendet das Ereignis zum Manager. Der Partner antwortet, indem er die Verbindung aufbaut oder die Verbindung ablehnt.
- **Versenden einer Nachricht:** Der Client sendet das Ereignis zum Server. Der Server verteilt die Nachricht entweder an alle Teilnehmer eines Forums oder an eine vorher gewählte Person.

4.4.3.6 Aufgaben des *KoopFuzzyArrangeManager*

Der *KoopFuzzyArrangeManager* stellt für die Applets der kooperativen Aufgabe (Entwurf des unscharfen Reglers) sämtliche Funktionen zur permanenten Speicherung dynamischer Daten in der Datenbank zur Verfügung. Einzelheiten sind der Dokumentation des Managers zu entnehmen.

4.4.3.7 Aufgaben des *NotizenManager*

Der *NotizenManager* stellt für das Applet *Notizen* die Funktionalität der Kommunikation mit der Datenbank zur Verfügung.

So ermöglicht der Manager das Auflisten aller Notizen, die ein Benutzer bereits angelegt hat. Dazu holt er die entsprechenden Notizen aus der Datenbank. Die geänderten, gelöschten oder neu eingefügten Notizen speichert er in dieser. Zum Anzeigen einer Notiz greift das Applet *Notizen* nicht wieder auf den Manager zu, denn bereits beim Holen der Liste mit den Notizen sind auch die Inhalte übertragen worden. Dies mag bei einer größeren Anzahl von Notizen mit großen Textmengen problematisch sein, stellt für den im Rahmen der Projektgruppe implementierten Prototypen aber eine ausreichende Lösung dar.

Die Ereignisbehandlung wird durch diesen Manager nicht genutzt. In einer Erweiterung wäre es aber denkbar, über ein spezielles Ereignis den Benutzer zu benachrichtigen, daß er ein Kapitel besucht, zu dem er sich bereits eine Notiz angelegt hat.

5 Kurse

5.1 Struktur von Kursen

Sinn von KOLIBRI ist es, einem Studenten über das Internet Wissen in Form von *Kursen* zu vermitteln. Ein mit KOLIBRI realisierter Kurs ahmt die klassischen Teilbereiche *Vorlesung*, *Übung* und *Praktikum* einer universitären Lehrveranstaltung mit Hilfe eines interaktiven Hypermediasystems nach.

Jeder Kurs ist – ähnlich einem Buch oder Vorlesungsskript – in Abschnitte gegliedert, die *Kapitel* genannt werden.

Ein Kapitel verfügt über einen Titel und einen Typ, der Aufschluß über seine Verwendung gibt. Zur Zeit werden die Kapiteltypen *Lektion*, *Aufgabe* und *Projekt* unterstützt. Eine Lektion ist das Pendant zu einer Vorlesung, es wird also Lehrstoff präsentiert. Aufgaben nehmen die Funktion von Übungszetteln wahr, die ein Student alleine lösen soll. Ein Projekt schließlich ist eine mehrteilige Aufgabe, die von einer Studentengruppe in Kooperation gelöst wird. Je nach Typ müssen Aufbau und Inhalt eines Kapitels bestimmten Anforderungen genügen.

Der Inhalt eines Kapitels setzt sich aus einer beliebigen Anzahl von sogenannten *Bausteinen* zusammen. Bausteine sind verschiedenartige multimediale Informationseinheiten, die von KOLIBRI einzeln verwaltet und für jeden Studenten *sichtbar* oder *unsichtbar* sein können.

Zur Überwachung und Bewertung des Verhaltens und der Lernerfolge eines Studenten wird ein Mechanismus verwendet, der auf den Begriffen *Einstufung* und *Gruppe* basiert. Sowohl die Informationen des Kurses als auch die teilnehmenden Studenten werden mit Hilfe dieses Systems klassifiziert. Besuchte Lektionen, gelöste Aufgaben und bearbeitete Projekte wirken sich auf die Einstufungen und Gruppen eines Studenten aus. Gleichzeitig paßt das System den empfohlenen Weg durch den Kurs den aktuellen Einstufungen und Gruppen eines Studenten an, indem es den Kurs für ihn in *obligatorische*, *optionale* und *obsolete* Kapitel unterteilt. So wird dem Studenten jederzeit eine Hilfe zur Orientierung gegeben, die ihn jedoch in keiner Weise in seinen Navigationsmöglichkeiten beschränkt.

Abbildung 5.1 auf der nächsten Seite verdeutlicht den schichtenartigen Aufbau der verschiedenen Bestandteile eines KOLIBRI-Kurses. Die folgenden Abschnitte enthalten detaillierte Informationen zu den einzelnen Bestandteilen.

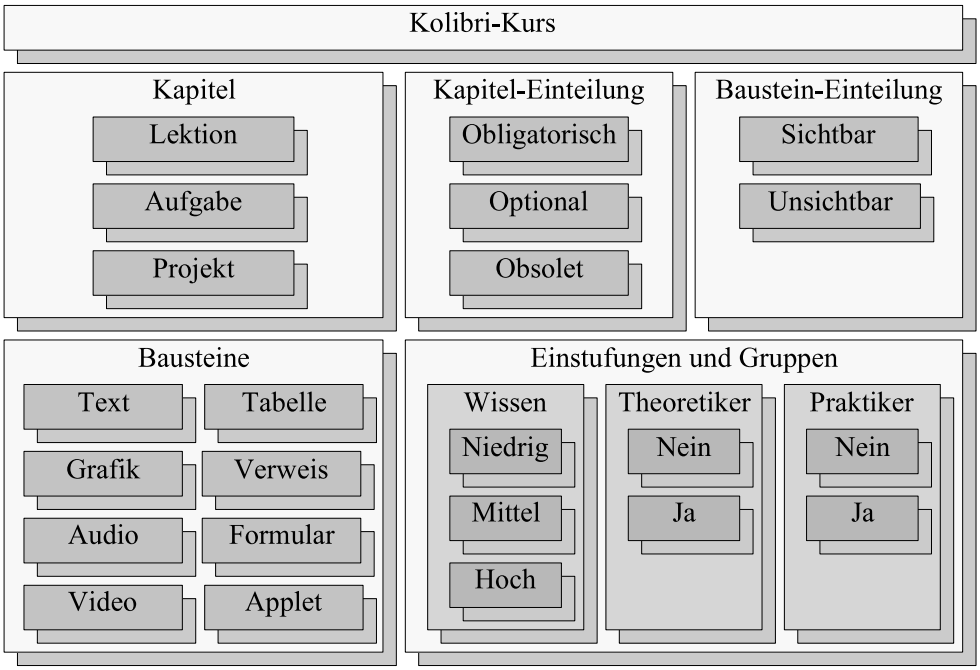

Abbildung 5.1: Die Bestandteile eines KOLIBRI-Kurses

5.1.1 Bausteine

Der Inhalt eines Kapitels setzt sich aus einer beliebigen Anzahl von *Bausteinen* zusammen.

Ein Baustein ist die kleinste Informationseinheit, die von KOLIBRI verwaltet und gezielt für einen Studenten angezeigt oder vor ihm „versteckt“ werden kann. Jeder Baustein korrespondiert im Normalfall mit einer Datei im Dateisystem des Servers, auf dem KOLIBRI läuft. Grafiken, Audio- und Video-Bausteine können alternativ auch von einem entfernten Rechner bezogen werden, sofern dieser einen WWW-Service anbietet.

Da nicht der Baustein selbst in die KOLIBRI-Datenbank aufgenommen wird, sondern nur der Pfad der entsprechenden Datei oder der URL der entfernten Ressource, können kleinere Änderungen direkt am Inhalt eines im System befindlichen Kurses vorgenommen werden. Dabei darf allerdings nicht die Struktur des Kurses betroffen sein. Bausteine sind zudem wiederverwertbar, d. h. sie können an mehreren Stellen eines Kurses oder sogar in völlig verschiedenen Kursen verwendet werden.

Die folgenden Baustein-Typen werden derzeit unterstützt:

- Text
- Grafik
- Audio
- Video

- Tabelle
- Applet
- Formular
- Verweis

KOLIBRI setzt zur Laufzeit aus den verschiedenen Bausteinen eines Kapitels HTML-Seiten zusammen, die an den WWW-Client des Studenten übertragen werden. Für jeden Baustein können zudem spezielle Attribute spezifiziert werden, die bestimmen, wie mit dem Baustein verfahren wird.

5.1.1.1 Bausteintyp Text

Text-Bausteine repräsentieren solche Abschnitte des Kurses, die im wesentlichen aus Fließtext bestehen. Text-Bausteine sind die am häufigsten verwendeten Bausteine. Ein Text-Baustein sollte einen sinngemäß zusammengehörigen Absatz oder Abschnitt umfassen.

Dateiformat Dateien für Text-Bausteine müssen Text enthalten, der in der Zeichentabelle ISO-8859-1 (auch ISO-Latin-1 genannt) kodiert ist. Zusätzlich ist die Verwendung sogenannter *Entities* zugelassen, mit denen die 256 erlaubten Zeichen unter Zuhilfenahme von nur 7 Bit kodierbar sind.

Außer dem reinen Text sind HTML-Tags zur logischen und physischen Hervorhebung von Zeichen sowie zur Bildung von Absätzen, Aufzählungen und Listen erlaubt.

Die Verwendung von HTML-Tags und Entities folgt dem HTML-Standard (siehe dazu [Bern95]).

Nicht erlaubt sind Hyperlinks und Überschriften. Eingebettete Grafiken sollten nur dann verwendet werden, wenn es sich nicht vermeiden läßt (etwa zur Darstellung einer Formel im Fließtext). Im Normalfall sollte für eine Grafik ein entsprechender Baustein (siehe Abschnitt 5.1.1.2) angelegt werden. Der bei HTML-Dateien übliche Rahmen darf nicht Teil des Bausteins sein, da er vom System generiert wird.

Falls im Baustein ein Verweis enthalten ist – etwa zu einer eingebetteten Grafik – dann muß dieser einer der beiden folgenden Varianten entsprechen:

- **Relativ:** Im Normalfall werden Verweise relativ zu der Datei angegeben, in der sich der Verweis befindet.

  Verweise dieser Art werden von KOLIBRI erkannt und zur Laufzeit automatisch in absolute Verweise umgewandelt. Nur so ist gewährleistet, daß die eingebetteten Ressourcen vom WWW-Client des Studenten wirklich gefunden werden. Da die übertragene HTML-Seite physikalisch gar nicht existiert, sondern dynamisch vom *Seitenkomponisten* (siehe Abschnitt 4.4.2) für den Benutzer generiert wird, würden relative Verweise vom WWW-Client sonst fälschlicherweise relativ zum URL des Seitenkomponisten und nicht zum URL des Bausteins interpretiert.

- **Absolut:** Falls eine Ressource verwendet werden soll, die von einem anderen Server stammt, muß deren kompletter URL – inklusive Protokoll – angegeben werden.

  Verweise dieser Art werden von KOLIBRI nicht modifiziert, sondern unverändert in die generierte HTML-Seite übernommen.

Absolute Verweise innerhalb des lokalen Dateisystems sollten nicht verwendet werden, da Pfade, die mit einem Schrägstrich (/) beginnen, im Dateisystem und im WWW-Server in den meisten Fällen eine unterschiedliche Bedeutung haben. Dies führt zur Laufzeit zu Problemen, weil Dateien nicht gefunden werden können.

5.1.1.2 Bausteintyp Grafik

Diese Art von Bausteinen repräsentiert Grafiken, die Bestandteil eines Kurses sind. Die Grafiken werden unmittelbar in eine Seite eingebettet und – falls vorhanden – mit einem Titel versehen, der unterhalb der Grafik angezeigt wird.

Wenn die Grafik in einer bestimmten Größe in die erzeugte HTML-Seite eingebunden werden soll, müssen dem Baustein entsprechende Informationen in Form von Attributen (siehe Abschnitt 5.1.1.9) hinzugefügt werden.

Dateiformat Dateien für Grafik-Bausteine müssen in einem Format vorliegen, das von üblichen WWW-Clients verarbeitet werden kann. Folgende Grafikformate werden als Standard für KOLIBRI betrachtet:

- Das *Graphics Interchange Format* (Endung *.GIF*).
- Das Format der *Joint Photographics Experts Group* (Endung *.JPG*).
- Das *Portable Network Graphics*-Format (Endung *.PNG*).

5.1.1.3 Bausteintyp Audio

Audio-Bausteine repräsentieren digitale Audiodaten, die in einem Kurs vorkommen. Die Audiodaten werden unmittelbar in eine erzeugte HTML-Seite eingebettet. KOLIBRI stellt keine speziellen Bedienelemente zum Abspielen der Audiodaten bereit, sondern verläßt sich stattdessen darauf, daß der WWW-Client auf Benutzerseite über die benötigte Funktionalität oder über ein geeignetes Plug-In verfügt.

Dateiformat Dateien für digitale Audio-Bausteine müssen in einem Format vorliegen, das von üblichen WWW-Clients oder deren Plug-Ins verarbeitet werden kann. Folgende digitale Audioformate werden als Standard für KOLIBRI betrachtet:

- Das *Waveform Audio*-Format (Endung *.WAV*).
- Das *Creative Labs Voice*-Format (Endung *.VOC*).
- Das *Sun Raw Audio*-Format (Endungen *.AU* und *.SND*).

5.1.1.4 Bausteintyp Video

Video-Bausteine repräsentieren digitale Videodaten, die Bestandteil eines Kurses sind. Für die Unterstützung von Video-Bausteinen durch KOLIBRI gilt im wesentlichen das gleiche wie für die zuvor erwähnten Audio-Bausteine.

Dateiformat Dateien für digitale Video-Bausteine müssen in einem Format vorliegen, das von üblichen WWW-Clients oder deren Plug-Ins verarbeitet werden kann. Folgende digitale Videoformate werden als Standard für KOLIBRI betrachtet:

- Das *Audio Video Interleaved*-Format (Endung *.AVI*).
- Das Format der *Moving Pictures Experts Group* (Endungen *.MPG* und *.MPEG*).
- Das *Quicktime*-Format von Apple (Endung *.MOV*).

5.1.1.5 Bausteintyp Tabelle

Tabellen-Bausteine repräsentieren Tabellen, die Bestandteil eines Kurses sind. Die Tabellen werden unmittelbar in eine Seite eingebettet und – falls vorhanden – mit einem Titel versehen. Tabellen sind statisch, d. h. ihre Inhalte liegen bereits fest, wenn ein Kurs erstellt wird. Zur Realisierung von dynamischen Tabellen müssen geeignete Applets verwendet werden.

Dateiformat Dateien für Tabellen-Bausteine müssen als HTML-kodierte Tabellen vorliegen. Für die Zeichenkodierung gelten die gleichen Einschränkungen wie bei Text-Bausteinen (siehe Abschnitt 5.1.1.1). Innerhalb der Zellen einer Tabelle sind alle Formatierungen erlaubt, die HTML für Tabellen zuläßt. Überschriften und Verweise sind nicht zugelassen. Die Datei darf keinen HTML-Rahmen enthalten.

5.1.1.6 Bausteintyp Formular

Formular-Bausteine erlauben dem Studenten eine Eingabe, die entweder vom System oder von einem Kursleiter weiterverarbeitet wird.

Der häufigste Verwendungszweck von Formularen sind Aufgaben und Projekte, bei denen sie zur Eingabe der Lösung benutzt werden. In einem Kapitel, das mindestens einen Formular-Baustein enthält, wird automatisch ein Schalter angezeigt, mit dem die Eingabe des Benutzers an das System übergeben werden kann. Die Eingabe wird in der Datenbank gespeichert. Die Tabelle *kapitelEingabe* (siehe Tabelle 4.22 auf Seite 48) kann für jeden Studenten und jedes Kapitel eine beliebige Anzahl solcher Eingaben speichern. Eine Eingabe setzt sich aus einem Namen und einem Wert zusammen. Jedes Paar aus Name und Wert korrespondiert mit einem Eingabeelement des Formulars.

Dateiformat Dateien für Formular-Bausteine müssen als HTML-kodierte Formulare vorliegen. Das heißt: die Datei muß zunächst den gleichen Anforderungen genügen wie ein Text-Baustein (siehe Abschnitt 5.1.1.1). Zusätzlich können an beliebigen Stellen des Bausteins HTML-kodierte Eingabeelemente plaziert werden. Der für HTML-Formulare übliche Rahmen darf nicht in der Datei enthalten sein, da er zur Laufzeit vom System selbst generiert wird.

Jedem Eingabeelement muß über das HTML-Attribut `NAME` ein Bezeichner zugeordnet werden. Die Bezeichner der Eingabeelemente aller Formular-Bausteine eines Kapitels müssen eindeutig sein, damit die Eingaben korrekt gespeichert und verarbeitet werden können.

Eingabeelementen kann auf verschiedene Weise ein Vorgabewert zugeordnet werden (siehe dazu [Bern95]). Bei Formularen, die zur Lösungseingabe von Aufgaben verwendet werden, dient dieser Vorgabewert zur Aufnahme der richtigen Lösung. Diese wird vor der Versendung an den WWW-Client des Studenten automatisch herausgefiltert. Sind mehrere Lösungen korrekt (was bei Textaufgaben der Fall sein kann), müssen sie durch einen senkrechten Strich (|) voneinander getrennt werden.

5.1.1.7 Bausteintyp Applet

Applet-Bausteine repräsentieren Java-Applets, die Bestandteil eines Kurses sind. Über Applets kann nahezu jede beliebige Interaktion eines Studenten mit KOLIBRI realisiert werden. Für komplexe Anwendungen ist eventuell die Implementierung eines speziellen Kommunikationsmanagers (siehe Abschnitt 4.4.3) notwendig.

Die Applets werden unmittelbar in eine erzeugte HTML-Seite eingebettet. Der WWW-Client des Studenten fordert ein Applet automatisch zusammen mit der Seite an und startet es anschließend. Ein Applet, das seinen Zustand oder Eingaben des Benutzers speichern möchte – etwa um sich beim nächsten Betreten der Seite restaurieren zu können – hat die Möglichkeit, ebenso wie ein Formular Paare aus Namen und Werten in die Tabelle *kapitelEingabe* (siehe Tabelle 4.22 auf Seite 48) zu schreiben.

Dateiformat Dateien für Applet-Bausteine müssen als compilierte Java-Applets (Endung *.class*) vorliegen. Alle Anforderungen, die an Java-Applets innerhalb von HTML-Seiten gestellt werden, müssen erfüllt sein. Detailliertere Informationen hierzu enthält die Dokumentation zum *Java Development Kit (JDK)* von Sun (siehe dazu [Sun97]).

5.1.1.8 Bausteintyp Verweis

Verweis-Bausteine erlauben sowohl Verweise innerhalb eines KOLIBRI-Kurses als auch solche Verweise, die das System verlassen. Es wurde ein spezieller Bausteintyp für Verweise eingeführt, weil HTML-Verweise innerhalb von Text-Bausteinen nicht verwendet werden dürfen. Dies hätte eine Reihe von technischen Problemen mit sich gebracht:

- Die Seiten eines KOLIBRI-Kurses werden dynamisch generiert. Deshalb ist es nicht möglich, innerhalb eines Text-Bausteins auf eine Seite zu verweisen: Zu dem Zeitpunkt, da der Autor den Kurs entwirft, existiert diese noch gar nicht. Außerdem sollen Bausteine in verschiedenen Kursen verwendet werden können. Eine korrekte und vollständige Lösung dieses Problems würde ein überarbeitetes Kursmodell erfordern, das über das im Rahmen der Projektgruppe entwickelte hinausgeht.
- Für das korrekte Funktionieren des Systems ist es notwendig, daß dem Server bekannt ist, auf welcher Seite sich der Student befindet. Enthält ein Text-Baustein einen Verweis, der das System verläßt, so wird das Ziel dieses Verweises innerhalb eines HTML-Frames angezeigt, der für KOLIBRI verwendet wird. Das sieht zum einen unpassend aus, zum anderen wird es den Studenten möglicherweise verwirren.

Dateiformat Verweis-Bausteine basieren nicht auf einzelnen Dateien. Sie werden vollständig in der Beschreibung der Kursdatei spezifiziert. Der Dateiname eines Verweis-Bausteins gibt das Ziel des Verweises an. Ist ein Titel festgelegt, dann wird dieser als sichtbarer Text für den Verweis angezeigt.

Folgende Syntax wird für Verweise innerhalb von KOLIBRI verwendet:

```
<Kapitel-Name> [ "." <Baustein-Name> ]
```

`<Kapitel-Name>` gibt den Namen des gewünschten Kapitels an. Es wird angenommen, daß sich das Kapitel im gleichen Kurs befindet.

`<Baustein-Name>` ist optional und gibt den Namen eines Bausteins im gewünschten Kapitel an. Auf diesem Weg kann direkt zu einem bestimmten Baustein in einem Kapitel verzweigt werden.

Verweise, die einen gültigen URL enthalten, werden als Verweise erkannt, die das System verlassen. Für solche Verweise wird ein zweites Fenster geöffnet.

5.1.1.9 Attribute

Für einen Baustein können verschiedene zusätzliche Attribute spezifiziert werden, die bestimmen, wie das System mit dem Baustein verfahren soll. Einige dieser Attribute müssen festgelegt werden, andere sind optional.

Datei Für alle Bausteine mit Ausnahme der Verweis-Bausteine (siehe Abschnitt 5.1.1.8) muß als minimale Information die Datei angegeben werden, in der die Bausteindaten zu finden sind. Die Datei sollte vorhanden sein, während ein Kurs in das System aufgenommen wird. Andernfalls gibt das Hilfsprogramm *KursAdmin* (siehe Abschnitt 5.3) eine Warnung aus. Bei Verweis-Bausteinen legt diese Information das Ziel des Verweises fest. Da das Ziel sowohl innerhalb als auch außerhalb des Kurses (und damit außerhalb des Systems) liegen kann, wird bei diesen Bausteinen keine Existenzprüfung durchgeführt, während der Kurs in das System eingefügt wird.

Name Für jeden Baustein kann ein interner Name vergeben werden. Dieser interne Name wird z. B. zum Auflösen von Verweisen verwendet. Der interne Name wird normalerweise nicht angezeigt. Bausteinnamen gehorchen den gleichen Regeln wie Bezeichner in der Programmiersprache Pascal (z. B. keine Unterscheidung von Groß- und Kleinschreibung). Die Namen der Bausteine eines Kapitels müssen eindeutig sein.

Titel Für jeden Baustein kann und soll ein Titel vergeben werden. Der Titel enthält eine kurze Beschreibung des Bausteininhaltes. Bei einigen Bausteinen – etwa bei Grafiken – wird der Titel als Beschriftung unter dem Baustein angezeigt. Der Titel wird außerdem angezeigt, während ein Baustein unsichtbar ist und durch einen speziellen Verweis in die Seite eingeblendet werden kann.

Autor Für jeden Baustein können der Name und die E-Mail-Adresse des Autors angegeben werden. Diese Information wird derzeit zwar gespeichert, zur Laufzeit aber nicht ausgewertet. Es gibt verschiedene Möglichkeiten, die Angabe des Autors in einer späteren Version von KOLIBRI zu nutzen. Zum Beispiel kann neben oder unter dem Baustein ein kleiner Verweis in Form eines Briefumschlags angezeigt werden, mit dem ein Student dem Autor des Bausteins eine Nachricht senden kann, etwa um Fehler zu melden oder Fragen zu stellen. Wenn in einem Kurs Bausteine von Fremdanbietern verwendet werden, beispielsweise Grafiken, kann das Autor-Feld zudem zur Speicherung von Copyright-Informationen dienen.

Sichtbarkeit Für jeden Baustein einer Lektion kann spezifiziert werden, unter welchen Umständen er innerhalb seines Kapitels für einen Studenten sichtbar oder unsichtbar ist. Die Regeln hierzu werden mit Hilfe der Gruppen gebildet, die für den Kurs definiert sind. Bei Aufgaben und Projekten kann keine Sichtbarkeit festgelegt werden. Abschnitt 5.1.3.3 enthält detaillierte Informationen über die zugrundeliegenden Konzepte.

Verwendungszweck Für jeden Baustein einer Aufgabe oder eines Projektes kann ein Verwendungszweck festgelegt werden. Der Verwendungszweck bestimmt – ähnlich wie die Sichtbarkeit von Bausteinen in Lektionen – wann dieser Baustein angezeigt wird. Hier wird jedoch nur zwischen einigen fest implementierten Fällen unterschieden. Es können keine Regeln in Abhängigkeit von den Gruppen des Kurses angegeben werden. Kombinationen der folgenden vier Markierungen sind möglich:

- **Normal**: Dieser Baustein wird angezeigt, wenn die Fragestellung einer Aufgabe oder eines Projektes dargestellt wird.
- **Hinweis**: Dieser Baustein wird angezeigt, wenn der Student einen Hinweis zur Lösung einer Aufgabe oder eines Projektes anfordert.
- **Richtig**: Dieser Baustein wird angezeigt, wenn das System die Bewertung einer richtig gelösten Aufgabe oder eines richtig gelösten Projektes übermittelt.
- **Falsch**: Dieser Baustein wird angezeigt, wenn das System die Bewertung einer falsch gelösten Aufgabe oder eines falsch gelösten Projektes übermittelt.

Wird kein Verwendungszweck für einen Baustein festgelegt, dann wird als Vorgabe die Kombination aller vier möglichen Markierungen angenommen, mit dem Ergebnis, daß der Baustein immer sichtbar ist.

Detaillierte Informationen zu den Markierungen finden sich in den Abschnitten zu *Aufgaben* und *Projekten* (siehe Abschnitt 5.1.2.2 und Abschnitt 5.1.2.3).

Parameter Für jeden Baustein kann eine beliebige Anzahl von Parametern spezifiziert werden. Jeder Parameter setzt sich aus einem Namen und einem Wert zusammen. Diese Parameter werden derzeit nur für Applet-Bausteine genutzt (siehe Abschnitt 5.1.1.7). Sie werden mit dem HTML-Tag `PARAM` unmittelbar an das Applet übergeben.

Breite und Höhe Für jeden Baustein kann angegeben werden, mit welcher Breite und Höhe er in die erzeugte HTML-Seite eingebettet werden soll. Diese Information wird nicht für alle Bausteintypen tatsächlich genutzt. Für Applet-Bausteine ist es eine erforderliche Information. Für Grafiken (siehe Abschnitt 5.1.1.2) ist die Angabe von Breite und Höhe optional, aber trotzdem sinnvoll, da diese Information dem WWW-Client eine schnellere Darstellung der Seite ermöglicht. Alle anderen Bausteintypen ignorieren Größenangaben.

5.1.2 Kapitel

Bausteine werden zu sogenannten *Kapiteln* zusammengefaßt. Ein Kapitel entspricht der Informationsmenge, die einem Studenten gleichzeitig angezeigt werden kann. Die HTML-Seiten der Kapitel liegen zu keinem Zeitpunkt vollständig in KOLIBRI vor. Stattdessen werden sie dynamisch zur Laufzeit auf eine Anfrage des Benutzers hin generiert.

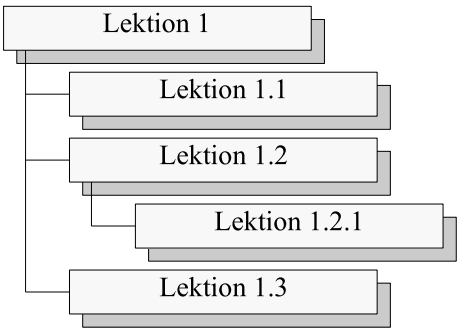

Abbildung 5.2: Beispiel für die Struktur von Lektionen

Jedes Kapitel verfügt über einen Typ, der Aufschluß über seine Verwendung gibt. Zur Zeit werden die folgenden Kapiteltypen unterstützt:

- Lektion.
- Aufgabe und Teilaufgabe.
- Projekt und Teilprojekt.

Je nach Typ müssen Aufbau und Inhalt eines Kapitels bestimmten Anforderungen genügen. Der Kapiteltyp wirkt sich auch darauf aus, wie ein Kapitel für einen Studenten aufbereitet wird und welche Interaktionsmöglichkeiten ihm darin angeboten werden.

Während ein Student einen Kurs besucht, ändert sich möglicherweise der Zustand einzelner Kapitel. Dies reflektiert die Bearbeitung von Aufgaben und Projekten sowie das Verstehen oder Nichtverstehen einzelner Kursabschnitte durch den Studenten. Der Zustand eines Kapitels wirkt sich auf die angezeigten Bausteine sowie die Interaktionsmöglichkeiten des Benutzers aus.

Für jedes Kapitel können spezielle Attribute spezifiziert werden, die bestimmen, wie das System mit dem Kapitel verfahren soll. Diese Attribute ähneln denen, die auch für Bausteine verwendet werden. Ihre Beschreibung folgt im Anschluß an die einzelnen Kapiteltypen.

5.1.2.1 Kapiteltyp Lektion

Lektionen sind Kapitel, in denen einem Studenten Lehrstoff präsentiert wird. Eine Lektion ist die einfachste Form eines Kapitels, da hier keine besonderen Anforderungen an den Inhalt oder die untergeordnete Struktur gestellt werden. Lektionen dürfen beliebige Bausteine enthalten und beliebig tief geschachtelt sein. Abbildung 5.2 verdeutlicht dies.

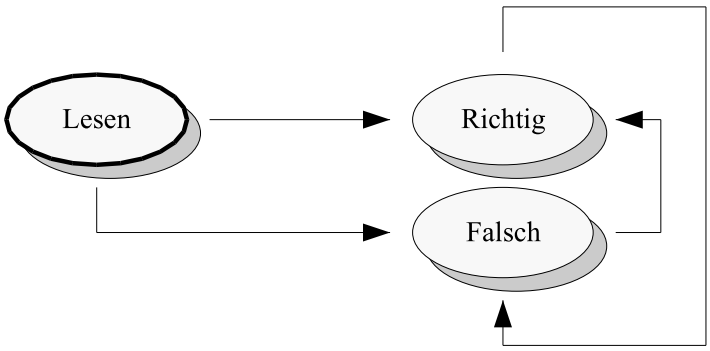

Abbildung 5.3: Zustandsdiagramm für Lektionen

Lektionen können sich durch *implizite Wissensstandkontrolle*, nicht jedoch durch *explizite Wissensstandkontrolle* auf die Einstufungen eines Studenten auswirken. Zur Nutzung der impliziten Kontrolle wird einer Lektion eine Anzahl von Punkten sowie eine Liste von Einstufungen zugeordnet. Diese Information kann mehrfach vergeben werden, so daß sich die Seite auf verschiedene Einstufungen unterschiedlich stark auswirkt. Detaillierte Informationen zum Einfluß der impliziten Kontrolle finden sich in Abschnitt 5.1.3.4.

Zustände Lektionen haben im Gegensatz zu Aufgaben und Projekten einen überwiegend statischen Charakter, weshalb sie mit wenigen Zuständen auskommen. Eine Lektion befindet sich beim ersten Besuch durch einen Studenten im Zustand *Lesen*. Kann im Laufe des Kurses durch eine Lernkontrolle festgestellt werden, daß der Student die Lektion verstanden oder nicht verstanden hat, so wechselt der Zustand der Lektion in *Richtig* oder *Falsch*. Dies kann sich durch erneutes Bearbeiten der Lernkontrolle auch wieder ändern. Abbildung 5.3 verdeutlicht dies. Lektionen, die sich im Zustand *Richtig* oder *Falsch* befinden, werden im Inhaltsverzeichnis des Kurs-Browsers speziell hervorgehoben.

Welche Bausteine innerhalb einer Lektion angezeigt werden, ist unabhängig vom Zustand der Lektion. Allein die Sichtbarkeitsregeln legen fest, welche der enthaltenen Bausteine für den Studenten fest in die Seite integriert sind und welche hinzugeschaltet werden können.

5.1.2.2 Kapiteltyp Aufgabe/Teilaufgabe

Aufgaben sind Kapitel, in denen ein Student zur Lösung einer oder mehrerer vorgegebener Problemstellungen aufgefordert wird. Er tut dies allein, d. h. nicht in Kooperation mit anderen Studenten. Aufgaben können sich durch *explizite Kontrolle*, nicht jedoch durch *implizite Kontrolle* auf die Einstufungen eines Studenten auswirken.

Eine Aufgabe besitzt eine feste zweistufige Struktur. Sie setzt sich aus einer Hauptseite und einer beliebigen Anzahl von direkten Unterseiten zusammen. Die Hauptseite reprä-

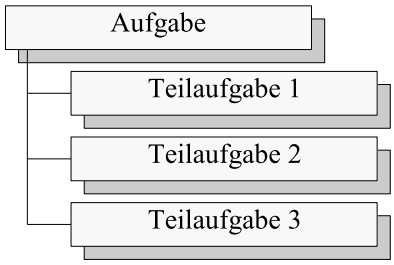

Abbildung 5.4: Beispiel für die Struktur von Aufgaben

sentiert eine Art Klausur oder Aufgabenzettel. Hier finden sich allgemeine Informationen zur Aufgabenstellung sowie eine Liste der *Teilaufgaben*. Die Unterseiten enthalten die einzelnen Teilaufgaben, die vom Studenten zu bearbeiten sind. Abbildung 5.4 verdeutlicht dies.

Jede Teilaufgabe sollte mindestens einen Formular-Baustein oder ein Applet enthalten, damit sie in der Lage ist, die Lösungen des Benutzers entgegenzunehmen und in der Datenbank abzulegen. Diese Lösungen werden entweder automatisch durch das System oder manuell durch einen Kursleiter korrigiert und bewertet.

Das Lösen, Bestehen oder Nichtbestehen sowohl jeder einzelnen Teilaufgabe als auch der gesamten Aufgabe kann – muß aber nicht – verschiedene Bewertungen zur Folge haben. Die möglichen Auswirkungen sind im Abschnitt über explizite Wissensstandkontrolle (siehe Abschnitt 5.1.3.5) beschrieben.

Zustände Im Gegensatz zu Lektionen, die nur zur Präsentation von Lehrstoff dienen, kommt bei Aufgaben (wie auch bei Projekten) eine stärkere dynamische Komponente ins Spiel. Diese soll die verschiedenen Stadien wiedergeben, die ein solches Aufgabenkapitel „durchlebt“. Hier besteht eine Analogie zu einem klassischen Aufgabenzettel, der zunächst nur die Aufgabenstellung enthält, dann zusätzlich die Lösung des Studenten und anschließend noch die Kommentare des Tutors.

Der Zustand, in dem sich eine Aufgabe oder Teilaufgabe befindet, wirkt sich darauf aus, welche Bausteine innerhalb der entsprechenden HTML-Seite angezeigt werden. Die Sichtbarkeitsregeln für Bausteine spielen hier keine Rolle. Die Zustände der Aufgabenseiten, die vom Studenten bereits besucht wurden, werden explizit in der Datenbank gespeichert. Ein Wechsel wird entweder durch Aktionen des Studenten oder durch das System angestoßen. Abbildung 5.5 auf der nächsten Seite verdeutlicht dies.

Die Bedeutung der einzelnen Zustände ist folgende:

- **Bearbeiten**: Der Zustand *Bearbeiten* repräsentiert die Bearbeitung einer Aufgabe bzw. einer Teilaufgabe. In diesem Zustand befinden sich sowohl die Aufgabe als auch deren Teilaufgaben, wenn eine Aufgabenseite zum ersten Mal betreten wird.

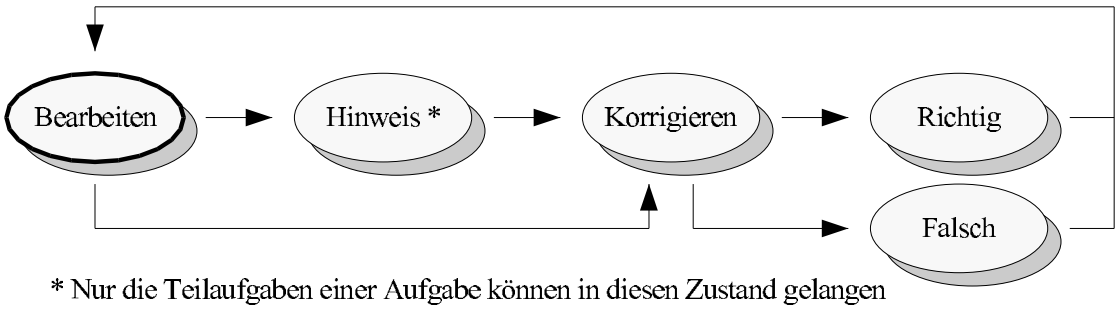

Abbildung 5.5: Zustandsdiagramm für Aufgaben und Teilaufgaben

Es werden alle Bausteine angezeigt, die mit dem Verwendungszweck *Normal* markiert sind. Die Eingabeelemente der Teilaufgaben zeigen die letzten gespeicherten Eingaben des Studenten an. Beim ersten Betreten einer Teilaufgabe sind die Eingabeelemente folglich leer.

Auf der Hauptseite wird eine Liste von Verweisen angezeigt, die unmittelbar zu den einzelnen Teilaufgaben führen. Diese Liste beinhaltet auch Informationen darüber, wieviele Punkte in den jeweiligen Teilaufgaben erreicht werden können. Außerdem enthält die Hauptseite einen Schalter [Bewerten], mit dem die komplette Aufgabe zur Bewertung abgesendet werden kann. Wird dieser Schalter gedrückt, dann wechselt die Aufgabe mit allen Teilaufgaben (auch den nicht bearbeiteten) in den Zustand *Korrigieren*.

Eine Unterseite enthält Schalter zum [Übernehmen] und [Zurücksetzen] der Eingaben, wenn mindestens ein Formular-Baustein enthalten ist. Bietet die Unterseite Lösungshinweise an, so wird außerdem ein Schalter [Hinweis] angezeigt, mit dem diese Lösungshinweise angefordert werden können. Die Anforderung von Hinweisen hat einen Wechsel in den Zustand *Hinweis* zur Folge.

- **Hinweis:** Auch der Zustand *Hinweis* repräsentiert die Bearbeitung einer Aufgabe bzw. einer Teilaufgabe. Es werden alle Bausteine angezeigt, die mit dem Verwendungszweck *Hinweis* markiert sind. Die Hinweise sollen den Benutzer bei der Lösung der Aufgabe unterstützen.

  Die Hauptseite einer Aufgabe kann sich nicht im Zustand *Hinweis* befinden.

  Jede der Unterseiten enthält Schalter zum [Übernehmen] und [Zurücksetzen] der Eingaben.

- **Korrigieren:** Der Zustand *Korrigieren* repräsentiert die Bewertung einer gelösten Aufgabe durch einen Kursleiter oder durch das System. Bei einer automatisch korrigierten Aufgabe wird dieser Zustand fast augenblicklich wieder verlassen. Wichtiger ist er bei Aufgaben, die von einem Kursleiter korrigiert werden, da die Korrekturen hier möglicherweise einige Zeit in Anspruch nehmen.

Auf der Hauptseite werden alle Bausteine angezeigt, die mit dem Verwendungszweck *Normal* markiert sind. Zusätzlich wird eine Liste von Verweisen angezeigt, die unmittelbar zu den einzelnen Teilaufgaben führen. Es wird vermerkt, daß die Aufgabe momentan korrigiert wird, und es werden keine Schalter angezeigt.

Auf jeder Unterseite werden die Bausteine angezeigt, die mit dem Verwendungszweck *Normal* markiert sind. Alle Eingabeelemente enthalten die letzte Eingabe des Studenten. Es werden keine Schalter angezeigt, so daß keine Änderungen mehr möglich sind.

Aufgaben und deren Teilaufgaben wechseln – abhängig vom jeweiligen Ergebnis – in den Zustand *Richtig* oder *Falsch*, sobald die Bewertung vorliegt. Der Zustandswechsel wird also durch das System angestoßen. Bei diesem Wechsel können außerdem Markierungen für verstandene oder nicht verstandene Kursteile vergeben und Änderungen an den Wissensgruppen des Studenten vorgenommen werden (siehe Abschnitt 5.1.3.5).

- **Richtig und Falsch**: Die Zustände *Richtig* und *Falsch* repräsentieren das Ergebnis einer Aufgabe oder Teilaufgabe. Hiermit liegt also eine Bewertung vor, die eine endgültige Aussage über die Lösung des Studenten erlaubt.

  Abhängig vom Zustand werden die Bausteine angezeigt, die mit dem Verwendungszweck *Richtig* oder *Falsch* markiert sind.

  Die Liste auf der Hauptseite enthält für jede Teilaufgabe die Anzahl der erreichten und erreichbaren Punkte. Unterhalb der Liste wird die Summe der Punkte des Studenten angezeigt, außerdem ein Text, der Aufschluß darüber gibt, ob damit die gesamte Aufgabe bestanden wurde oder nicht. Schließlich enthält die Seite noch einen Schalter [Nochmal lösen], mit dem die Aufgabe ein weiteres Mal bearbeitet werden kann.

  Alle Eingabeelemente der Unterseiten enthalten die letzte Eingabe des Studenten. Es werden keine Schalter angezeigt, so daß keine Änderungen mehr möglich sind.

  Die Zustände *Richtig* und *Falsch* sind normalerweise Endzustände einer Aufgabe. Die Aufgabe wechselt wieder in den Zustand *Bearbeitung*, wenn der Schalter zum erneuten Lösen gedrückt wird.

5.1.2.3 Kapiteltyp Projekt/Teilprojekt

Projekte sind Kapitel, in denen sich mehrere Studenten zur Lösung einer gemeinsamen Problemstellung zusammenfinden. Das Projekt ist also eine kooperative Aufgabe, das Team der daran arbeitenden Studenten wird auch als kooperative Gruppe bezeichnet. Es existiert für jeden Studenten ein *Teilprojekt*, das nur von ihm gelöst wird. Im Gegensatz zu einer Teilaufgabe ist dazu jedoch eine Absprache mit den anderen Mitgliedern der kooperativen Gruppe notwendig.

Projekte besitzen – wie Aufgaben – eine feste zweistufige Struktur. Ein Projekt setzt sich aus einer Hauptseite und einer beliebigen Anzahl von direkten Unterseiten zusammen.

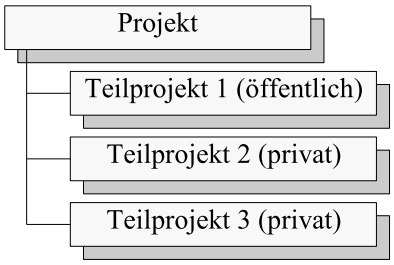

Abbildung 5.6: Beispiel für die Struktur von Projekten

Die Hauptseite repräsentiert das Projekt an sich. Hier finden die Benutzer allgemeine Informationen zur Problemstellung und Verweise auf die Teilprojekte, die sich auf den Unterseiten befinden.

Jedes Teilprojekt sollte mindestens einen Formular-Baustein oder ein Applet enthalten, damit es in der Lage ist, die Lösungen des Benutzers entgegenzunehmen und in der Datenbank abzulegen. Diese Lösungen werden später entweder automatisch durch das System oder manuell durch einen Kursleiter korrigiert und bewertet.

Es wird unterschieden zwischen *öffentlichen Teilprojekten*, an denen alle beteiligten Studenten gleichzeitig arbeiten können, und *privaten Teilprojekten*, an denen nur ein Student Änderungen vornehmen kann. Abbildung 5.6 verdeutlicht die Struktur eines Projektes.

Das Lösen, Bestehen oder Nichtbestehen sowohl der einzelnen Teilprojekte als auch des gesamten Projektes kann – muß aber nicht – verschiedene Bewertungen zur Folge haben. Dazu wird weitgehend der gleiche Mechanismus wie bei Aufgaben und Teilaufgaben verwendet. Bei bewerteten Projekten ergibt sich die Gesamtbewertung für alle Studenten aus der Summe der Teilbewertungen. Gerade deshalb ist eine Absprache der Teilnehmer untereinander besonders wichtig.

Zustände Genau wie Aufgaben besitzen auch Projekte eine starke dynamische Komponente, die die verschiedenen Stadien wiedergibt, in denen sich das Projekt befindet. Dadurch werden auch die zugehörigen Seiten beeinflußt. Projekte und Teilprojekte besitzen im wesentlichen die gleichen Zustände wie Aufgaben. Es sind jedoch noch zwei weitere Zustände „vorgeschaltet“, die für die Vergabe der Teilprojekte an Studenten notwendig sind. Abbildung 5.7 auf der nächsten Seite verdeutlicht dies.

Die Bedeutung der beiden zusätzlichen Zustände ist folgende:

- **Anmeldung**: Im Zustand *Anmeldung* kann ein Student seinen Teilnahmewunsch für ein Projekt äußern. In diesem Zustand befinden sich alle Seiten eines Projektes, wenn es zum ersten Mal betreten wird.

  Es werden alle Bausteine angezeigt, die mit dem Verwendungszweck *Normal* markiert sind.

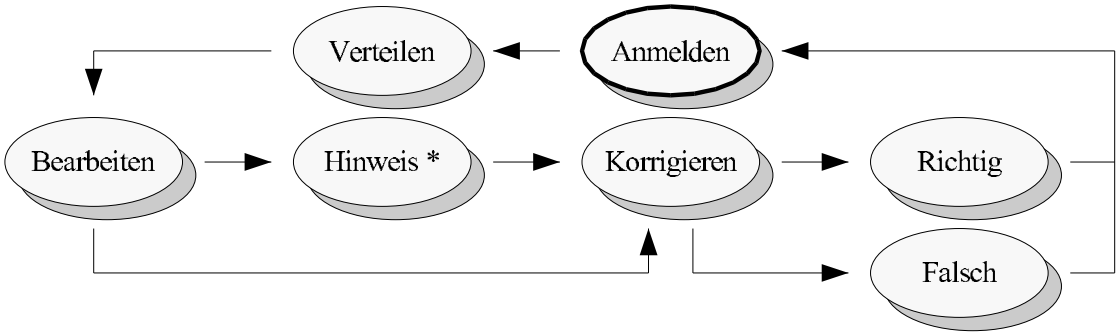

Abbildung 5.7: Zustandsdiagramm für Projekte und Teilprojekte

Auf der Hauptseite wird zusätzlich eine Liste von Verweisen angezeigt, die zu den einzelnen Teilprojekten führen. Jedem Teilprojekt ist dabei ein Markierungsfeld vorangestellt, mit dem der Student festlegen kann, welches Teilprojekt er bearbeiten möchte. Außerdem enthält die Seite einen Schalter [Anmelden], über den die Anmeldung für das Projekt abgesendet werden kann.

Falls eine der Unterseiten Eingabeelemente oder Applets enthält, so existiert in diesem Zustand kein Schalter, mit dem die Eingaben in Datenbank geschrieben werden könnten. Der Student kann sich so zwar einen Überblick über jedes Teilprojekt verschaffen, aber keinerlei Änderungen durchführen.

Die Hauptseite und alle Unterseiten wechseln in den Zustand *Verteilung*, sobald der Student seine Anmeldung absendet.

- **Verteilung:** Der Zustand *Verteilung* repräsentiert die Verteilung der einzelnen Teilprojekte an Studenten. Alle Seiten eines Projektes bleiben in diesem Zustand, solange nicht genügend Anmeldungen vorhanden zur Bildung einer kooperativen Gruppe vorhanden sind.

  Es werden alle Bausteine angezeigt, die mit dem Verwendungszweck *Normal* markiert sind.

  Auf der Hauptseite wird zusätzlich eine Liste von Verweisen angezeigt, die zu den einzelnen Teilprojekten führt. Außerdem wird unterhalb der Liste vermerkt, daß das System noch weitere Anmeldungen für die Aufgabe benötigt, bis eine kooperative Gruppe gebildet werden kann.

  Falls eine der Unterseiten Eingabeelemente oder Applets enthält, so existiert kein Schalter, mit dem die Eingaben in die Datenbank geschrieben werden können. Der Student kann sich also einen Überblick über jedes Teilprojekt verschaffen, aber keinerlei Änderungen durchführen.

  Die Hauptseite und alle Unterseiten wechseln in den Zustand *Bearbeitung*, sobald eine Gruppe für dieses Projekt gebildet werden kann. Der Zustandswechsel wird

durch das System ausgelöst.

5.1.2.4 Attribute

Für ein Kapitel können verschiedene zusätzliche Attribute spezifiziert werden, die bestimmen, wie das System mit dem Kapitel verfahren soll. Einige dieser Attribute müssen festgelegt werden, andere sind optional.

Name Für jedes Kapitel können ein interner Name vergeben werden. Dieser interne Name kann z. B. zum Auflösen von Verweisen verwendet werden. Er wird normalerweise nicht angezeigt. Kapitelnamen gehorchen den gleichen Regeln wie Bezeichner in der Programmiersprache Pascal (z. B. keine Unterscheidung zwischen Groß- und Kleinschreibung). Die Namen aller Kapitel eines Kurses müssen eindeutig sein.

Titel Für jedes Kapitel kann ein Titel vergeben werden. Der Titel entspricht der Überschrift des Kapitels und sollte dementsprechend den Kapitelinhalt in wenigen Worten wiedergeben. Der Titel wird im Inhaltsverzeichnis des Kurs-Browsers angezeigt.

Autor Für jedes Kapitel kann der Name und die E-Mail-Adresse des Autors angegeben werden. Wird ein Autor angegeben, dann kann an einer geeigneten Stelle der entsprechenden Seite ein kleiner Verweis in Form eines Briefumschlags angezeigt werden. Mit diesem Verweis erhält der Student die Möglichkeit, dem Autor des Kapitels eine Nachricht zu senden, z. B. um Fehler zu melden. Dies ist derzeit nicht implementiert. Die Information wird zwar gespeichert, aber nicht ausgewertet.

Sichtbarkeit Für Lektionen, Aufgaben und Projekte, nicht jedoch für die Unterseiten von Aufgaben und Projekten kann eine Sichtbarkeit festgelegt werden. Die Sichtbarkeit ist eine komplexe Relation zwischen dem jeweiligen Kapitel und den Gruppen des Kurses. Sie wird verwendet, um zur Laufzeit jedes Kapitel in eine der folgenden drei Kategorien einordnen zu können:

- Ist das Kapitel für einen Studenten *obligatorisch*?
- Ist das Kapitel für einen Studenten *optional*?
- Ist das Kapitel für einen Studenten *obsolet*?

Wird für ein Kapitel keine Sichtbarkeit festgelegt, dann erbt es seine Sichtbarkeitsregeln vom übergeordneten Kapitel. Die Kapitel auf der obersten Ebene eines Kurses sind als Vorgabe für alle Gruppen optional. Weitere Informationen zur Kapitelsichtbarkeit befinden sich in Abschnitt 5.1.3.2.

Auswertung Jedes Kapitel, das Formulare oder Applets enthält, kann Eingaben des Studenten entgegennehmen und diese in der Tabelle *kapitelEingabe* (siehe Tabelle 4.22 auf Seite 48) ablegen. Wenn diese Eingaben nicht nur gespeichert, sondern auch ausgewertet werden sollen (inbesondere bei Aufgaben und Projekten), dann muß festgelegt werden, über welchen Mechanismus diese Auswertung und die anschließende Punktevergabe erfolgt. Der Mechanismus wird zum Beispiel dann angestoßen, wenn eine Aufgabe oder ein Projekt zur Korrektur freigegeben wird.

Das System unterstützt zwei einfache Standard-Mechanismen zur Auswertung von Aufgaben:

- Es kann ein paarweiser Vergleich zwischen den Vorgabewerten einer Aufgabe und den vom Studenten eingegebenen Lösungen durchgeführt werden. Dies ist nur dann möglich, wenn die Aufgabe komplett aus Formular-Bausteinen besteht, weil nur dort Vorgabewerte festgelegt werden können (siehe Abschnitt 5.1.1.6).
- Die Eingaben des Studenten können dem Autor der Aufgabe oder des Kurses in einer E-Mail zugesandt werden. Die Eingaben müssen manuell bearbeitet und bewertet werden. Anschließend sollten die erreichten Punkte in die Datenbank geschrieben werden, wozu allerdings derzeit keine komfortable Schnittstelle existiert.

5.1.3 Persönliche Verläufe

KOLIBRI unterstützt persönliche Kursverläufe, also Verläufe, die sich dynamisch sowohl den Interessen als auch dem Wissensstand eines Studenten anpassen. Dies entspricht den Punkten *Teaching On Demand*, *Interaktives Arbeiten mit dem System* und *Wissensstandkontrolle* aus dem Anforderungskatalog der Projektgruppe (siehe Kapitel 2).

Der Sinn der persönlichen Kursverläufe ist, einem Studenten zu helfen, die für ihn interessanten und relevanten Informationen zu finden. Sie stellen also eine Orientierungshilfe dar, sollen aber keinesfalls zur Restriktion dienen. Regeln der Art „Ein Student darf Kapitel fünf erst betreten, wenn er zuvor eine bestimmte Anzahl von Aufgaben erfolgreich gelöst hat“ sind nicht Ziel des Systems und werden auch nicht unterstützt. Der Student hat jederzeit die Möglichkeit, sich den gesamten Inhalt des Kurses anzuschauen, sofern er dies wünscht.

5.1.3.1 Einstufungen und Gruppen

Zur Klassifikation sowohl der Studenten als auch der angebotenen Informationen wird ein Konzept verwendet, das auf zwei Begriffen basiert: *Einstufungen* und *Gruppen*.

Eine Einstufung dient als Maß für den Wissensstand oder das Interesse eines Studenten. Sie kann sich auf den gesamten Lehrstoff beziehen oder nur auf einen speziellen Teilbereich. Beispielsweise könnte eine Einstufung verwendet werden, um das Interesse des

Studenten an im Kurs enthaltenen anwendungsorientierten Informationen zu messen. Eine weitere Einstufung könnte dazu dienen, den allgemeinen Fortschritt eines Studenten zu erfassen.

Jede Einstufung setzt sich aus einer Anzahl von (mindestens zwei) Gruppen zusammen, die die möglichen Werte der Einstufung definieren. Die Gruppen sind disjunkt und geordnet. Der Student muß für jede der vom Kurs verwendeten Einstufungen zu jedem Zeitpunkt in genau einer Gruppe sein. Eine Einstufung ähnelt sehr einem Aufzählungstyp in einer Programmiersprache wie etwa Pascal. Die Gruppen entsprechen den möglichen Werten dieses Aufzählungstyps.

Mit jeder Einstufung i ist zudem für jeden Studenten ein Zähler z_i assoziiert. Für jede Gruppe j sind eine untere Schranke u_j und eine obere Schranke o_j bestimmt. Die Zähler z_i haben zu Beginn den Wert Null. Während ein Student einen Kurs bearbeitet, werden die Werte der Zähler z_i durch die Seiten, die der Student besucht, sowie seine erbrachten Leistungen bei Aufgaben beeinflußt (siehe Abschnitte 5.1.3.4 und 5.1.3.5). Dabei sind folgende Regeln zu beachten. Ausgangspunkt ist immer ein Student, der bezüglich der Einstufung i in der Gruppe j ist.

- Überschreitet der Zähler z_i die obere Schranke o_j, dann wechselt der Student in die nächsthöhere Gruppe der Einstufung i, und der Zähler z_i wird auf Null zurückgesetzt. Gibt es keine nächsthöhere Gruppe, bleibt der Student in Gruppe j, und der Zähler behält den Wert o_j.
- Unterschreitet der Zähler z_i die untere Schranke u_j, dann wechselt der Student in die nächsttiefere Gruppe Einstufung i, und der Zähler z_i wird auf Null zurückgesetzt. Gibt es keine nächsttiefere Gruppe, bleibt der Student in Gruppe j, und der Zähler behält den Wert u_j.

5.1.3.2 Einteilung der Kapitel

KOLIBRI verwaltet drei Relationen zwischen den Kapiteln und den Gruppen eines Kurses. Diese Relationen geben an, ob ein Kapitel für eine bestimmte Gruppe *obligatorisch*, *optional* oder *obsolet* ist. Die Relationen sind disjunkt, d. h. kein Kapitel befindet sich in mehr als einer dieser Relationen. Dafür wird beim Einfügen eines Kurses in das System Sorge getragen. Es ergibt sich also jede der Relationen aus den beiden anderen, so daß nur zwei Relationen wirklich in der Datenbank verwaltet werden müssen.

Zur Laufzeit, also beim Betrachten des Kurses, wird aus den konkreten Gruppen eines Studenten ermittelt, wie ein Kapitel in seinen Kursverlauf einzuordnen ist. Dabei gelten die folgenden drei Regeln:

1. Befindet sich der Student in mindestens einer Gruppe, für die das Kapitel obligatorisch ist, dann ist das Kapitel auch für ihn obligatorisch. Andernfalls wird Regel 2 betrachtet.

2. Befindet sich der Student in mindestens einer Gruppe, für die das Kapitel optional ist, dann ist das Kapitel auch für ihn optional. Andernfalls gilt automatisch Regel 3.

3. Befindet sich der Student in keiner Gruppe, für die das Kapitel obligatorisch oder optional ist, dann ist das Kapitel für ihn obsolet.

Alle für einen Studenten obligatorischen Kapitel bilden seinen minimalen Kursverlauf. Hat der Student diese Kapitel erfolgreich durchgearbeitet, so hat er auch die Lernziele seiner Gruppen erreicht.

In jedem obligatorischen Kapitel führt der Schalter [Weiter] des Kurs-Browsers den Studenten zu seinem nächsten obligatorischen Kapitel. Dies ist das nächste Kapitel, das für ihn als obligatorisch gekennzeichnet ist und das noch nicht gelesen oder noch nicht verstanden wurde. In optionalen und obsoleten Kapiteln führt der Schalter [Weiter] den Studenten wieder zurück zum letzten gelesenen obligatorischen Kapitel.

Die optionalen Kapitel dienen sozusagen dazu, „über den Tellerrand“ zu schauen. Sie enthalten weiterführende und vertiefende Informationen, die nicht zu den unmittelbaren Lernzielen des Studenten gehören, aber für ihn interessant sein könnten. Dem Studenten wird in jedem Kapitel eine Liste der optionalen Unter- und Geschwisterkapitel angezeigt, die noch nicht gelesen oder noch nicht verstanden wurden.

Alle weiteren Kapitel sind für den Studenten zu komplex, zu trivial oder aus anderen Gründen nicht von Interesse. Sie werden als obsolete Kapitel bezeichnet. Es besteht für den Studenten die Möglichkeit, auch diese Bereiche zu betreten. Allerdings wird ihm dies von KOLIBRI nicht empfohlen.

5.1.3.3 Einteilung der Bausteine

KOLIBRI verwaltet auch zwei Relationen zwischen den Bausteinen und den Gruppen eines Kurses. Diese Relationen geben an, ob ein Baustein für eine bestimmte Gruppe *sichtbar* oder *unsichtbar* ist. Die Relationen sind ebenfalls disjunkt, d. h. kein Baustein befindet sich in mehr als einer dieser Relationen. Dafür wird beim Einfügen eines Kurses in das System Sorge getragen. Da sich auch hier eine Relation aus der anderen ergibt, wird nur die Relation der sichtbaren Bausteine tatsächlich in der Datenbank gespeichert.

Beim Aufbau eines Kapitels für einen Studenten werden die Bausteine, die für mindestens eine seiner aktuellen Gruppen in der Sichtbar-Relation enthalten sind, fest in die HTML-Seite eingebaut. Alle anderen Bausteine kann er selbständig ein- und ausblenden. Letzteres wird in der Historie vermerkt. Besucht der Student ein bereits besuchtes Kapitel erneut, sind Bausteine, die er beim letzten Besuch selber eingeblendet hatte, als Vorgabe eingeblendet.

Die Bausteinsichtbarkeit ist nur für Lektionen relevant. Innerhalb von Aufgaben und Projekten wird sie ignoriert und durch das Konzept der Zustände ersetzt (siehe Abschnitte 5.1.2.2 und 5.1.2.3).

5.1.3.4 Implizite Wissensstandkontrolle

Unter impliziter Kontrolle wird innerhalb von KOLIBRI die Analyse des Verhaltens verstanden, das ein Student in bezug auf den dargebotenen Stoff zeigt. KOLIBRI erkennt anhand der besuchten Kapitel die Interessen und möglicherweise das Wissen des Studenten und paßt dessen Gruppenzugehörigkeiten und damit den Kursverlauf entsprechend an.

Wichtig ist in diesem Zusammenhang, daß Aussagen nur auf Basis der besuchten Kapitel gemacht werden. Das Ignorieren oder Überspringen von Kapiteln wird nicht ausgewertet. Zum einen bedeutet dies sowohl bei der Modellierung als auch bei der Implementierung einen erheblichen Aufwand. Zum anderen kann das System nicht eindeutig feststellen, aus welchem Grund der Student ein Kapitel übersprungen hat. Der Stoff könnte zu trivial, zu komplex oder schlicht nicht interessant für ihn sein.

Der Besuch eines Kapitels läßt hingegen immer eindeutig Interesse folgern, wenn man Bedienungsfehler und willkürlich handelnde Studenten bei der Betrachtung ausschließt.

Der Besuch eines obligatorischen Kapitels hat keinen Einfluß auf die Einstufung eines Studenten. Es wird vom Studenten erwartet, daß er dieses Kapitel besucht, da es zum Pflichtteil seines Kurses gehört.

Der Besuch eines optionalen oder obsoleten Kapitels aktiviert die implizite Kontrolle. Dafür existiert zu jedem Kapitel eine Liste mit Tupeln von Einstufungen und Punkten. Für jedes Tupel dieser Liste werden dem Studenten die Punkte auf den Zähler der entsprechenden Einstufung addiert. Die Punkte dürfen auch negativ sein. Bei diesen Operationen auf den Zählern müssen die oben angesprochenen Regeln eingehalten werden (siehe Abschnitt 5.1.3.1).

Die implizite Kontrolle ist nur bei Lektionen relevant. Bei Aufgaben und Projekten wird dieser Mechanismus nicht verwendet.

5.1.3.5 Explizite Wissensstandkontrolle

Zur expliziten Kontrolle des Studenten werden Lernkontrollen verwendet, also Kapitel, die als Aufgabe oder Projekt markiert sind und bewertet werden. Eine bewertete Lernkontrolle kann folgende Auswirkungen haben:

- Die Gruppenzugehörigkeiten des Studenten können sich in Abhängigkeit von seinem Erfolg bei den Teilaufgaben oder -projekten ändern.
- Ein Teil des Kurses kann in Abhängigkeit von seinem Erfolg bei der gesamten Aufgabe oder dem gesamten Projekt als verstanden oder nicht verstanden markiert werden.

Änderung der Gruppenzugehörigkeit Lernkontrollen haben unmittelbaren Einfluß auf die Gruppenzugehörigkeiten eines Studenten. Zu jeder Teilaufgabe existiert eine Liste mit Tupeln von Einstufungen und Punkten. Diese gibt an, welche Einstufungen wie stark vom Erfolg oder Mißerfolg des Studenten beim Lösen der Teilaufgabe betroffen werden. Die verschiedenen Punktzahlen ermöglichen es, die Teilaufgaben einer Lernkontrolle verschieden stark zu gewichten.

Der folgende Algorithmus gibt an, wie Einstufungen durch das Lösen einer Teilaufgabe beeinflußt werden:

- Sei t die Anzahl der Eingabeelemente einer Teilaufgabe.
- Sei g die Anzahl der Punkte, mit denen die Teilaufgabe für eine Einstufung e gewichtet ist.
- Sei r die Anzahl der Eingabeelemente, die vom Studenten richtig ausgefüllt wurden.
- Dann ergibt sich die Anzahl p der Punkte, die auf den Zähler der Einstufung e addiert werden, nach folgender Formel:

$$p = \left\lfloor g \cdot \left(\frac{2r}{t} - 1 \right) \right\rfloor .$$

  Hat der Student die Aufgabe komplett richtig gelöst, werden ihm also g Punkte gutgeschrieben. Bei einer komplett falschen Aufgabe werden g Punkte abgezogen. Ist die Aufgabe zu exakt 50% richtig, dann hat das Ergebnis keinen Einfluß auf die Einstufung e.

- Diese Regel wird für jede in der Lernkontrolle enthaltene Teilaufgabe durchgeführt.

Verstandene Kursteile Der Erfolg einer Lernkontrolle bestimmt außerdem, ob der Teil des Kurses, der von ihr abgedeckt wird, als verstanden oder nicht verstanden markiert wird.

Jede Lernkontrolle deckt einen genau festgelegten Teil des Kurses ab. Im Normalfall erstreckt sich diese Abdeckung im Inhaltsverzeichnis des Kurses über sämtliche vorangehenden Bruderknoten der Lernkontrolle bis hin zum Vaterknoten. Falls ein Kapitel mehrere Lernkontrollen enthält, zählt nur diejenige, die sich im Kurs weiter vorn befindet. Damit ist gewährleistet, daß jedes Kapitel eines Kurses entweder gar nicht oder durch genau eine Lernkontrolle überprüft wird. Somit kann das Verständnis jedes Kapitels nach dem folgenden Algorithmus bestimmt werden:

1. Ein Kapitel gilt für einen Studenten als verstanden, wenn eine abdeckende Lernkontrolle existiert und der Student diese bestanden hat. Andernfalls wird Regel 2 betrachtet.

2. Ein Kapitel gilt für einen Studenten als nicht verstanden, wenn eine abdeckende Lernkontrolle existiert und der Student diese nicht bestanden hat. Andernfalls gilt automatisch Regel 3.

3. Existiert keine das Kapitel abdeckende Lernkontrolle, oder hat der Student diese noch nicht gelöst, so kann keine Aussage über das Verständnis des Kapitels gemacht werden.

Die Lernkontrolle ist mit einem ganzzahligen Wert b markiert, der angibt, welcher Prozentsatz der enthaltenen Teilaufgaben mindestens gelöst werden muß, damit sie als bestanden gilt. Ist kein solcher Wert festgelegt, dann gilt eine Schranke von 50%.

Auch bei der Entscheidung, ob die Lernkontrolle bestanden wurde, spielt die Gewichtung der Teilaufgaben eine Rolle. Der folgende Algorithmus gibt den genauen Zusammenhang an:

- Sei a die Anzahl der Teilaufgaben, die in der Lernkontrolle enthalten sind.
- Sei t_i die Anzahl der Eingabeelemente der i-ten Teilaufgabe.
- Sei r_i die Anzahl der Eingabeelemente der i-ten Teilaufgabe, die vom Studenten richtig ausgefüllt wurden.
- Sei g_i die Summe der Punkte, mit denen die i-te Teilaufgabe für die einzelnen Einstufungen gewichtet ist.
- Dann ergibt sich der Lösungsgrad l der gesamten Lernkontrolle nach folgender Formel:

$$l = \frac{\sum_{i=1}^{a} g_i \cdot \frac{r_i}{t_i}}{\sum_{i=1}^{a} g_i}.$$

- Ist nun l größer oder gleich dem für die Lernkontrolle angegebenen Wert b, dann war die Lernkontrolle erfolgreich, und die von ihr abgedeckten Kapitel werden im Kurs-Browser als verstanden markiert.

5.2 Erstellung von Kursen

Kurse für KOLIBRI werden erstellt, indem die Kursstruktur in einer Textdatei beschrieben wird. Diese Textdatei wird mit Hilfe des Programms `KursAdmin` compiliert und in die KOLIBRI-Datenbank übertragen. Das Programm wird im Abschnitt 5.3 auf Seite 99 ausführlich beschrieben. Anhang B enthält Ausschnitte der Kursdatei des implementierten Beispielkurses.

Die Kursbeschreibung besitzt eine rekursive, blockorientierte Struktur, die einer einfachen kontextfreien Grammatik folgt. Wenn im folgenden der Begriff *Element* erwähnt wird, bezieht sich dies stets auf die terminalen und nichtterminalen Symbole, die an einer bestimmten Stelle der Kursbeschreibung zulässig sind. Jedes terminale Symbol wird nur einmal beschrieben, sofern es nicht an verschiedenen Stellen mit unterschiedlichen Bedeutungen eingesetzt wird.

5.2.1 Token

Der Parser für die Kursbeschreibungen erkennt folgende Arten von Token:

- **Bezeichner**: Ein Bezeichner besteht aus einem Buchstaben (`A` bis `Z` und `a` bis `z`) gefolgt von einer beliebigen Folge von weiteren Buchstaben, Ziffern (`0` bis `9`) oder Unterstrichen (`_`). Groß- und Kleinschreibung wird nicht unterschieden.
- **Schlüsselwort**: Die folgenden Schlüsselwörter sind reserviert und können somit nicht als Bezeichner verwendet werden:

```
ALL       APPLET    AUTHOR    AUTO      BASE
COURSE    END       EVALUATE  EXERCISE  FILE
FORM      GROUP     HEIGHT    HIDDEN    HINT
IMAGE     LESSON    LINK      MANUAL    MOVIE
NORMAL    OPTIONAL  PARAM     PART      PRIVATE
PROJECT   PUBLIC    RATING    REQUIRED  RIGHT
SCORE     SOUND     SUCCESS   TABLE     TEXT
TITLE     USELESS   VISIBLE   WIDTH     WRONG
```

- **String**: Ein String besteht aus einer Folge beliebiger Zeichen, die in doppelte Anführungszeichen (`"`) eingeschlossen ist.
- **Integer**: Ein Integer-Wert besteht aus einem optionalen Vorzeichen (`+` oder `-`) und einer beliebigen, nichtleeren Folge von Ziffern (`0` bis `9`).

Zwei Token müssen durch mindestens ein Leerzeichen, ein Tabulatorzeichen oder einen Zeilenumbruch /-vorschub voneinander getrennt sein, damit sie korrekt erkannt werden.

5.2.2 Kommentare

Innerhalb der Kursdatei sind an beliebigen Stellen, aber nicht innerhalb eines Tokens die in C++ sowie Java üblichen Kommentare erlaubt.

- `//` leitet einen Kommentar ein, der sich bis zum Ende der aktuellen Zeile erstreckt.

- /* leitet einen Kommentar beliebiger Länge ein, der durch */ abgeschlossen werden muß.

Kommentare werden wie Leerzeichen behandelt und wirken sich nicht auf die Übertragung eines Kurses in die KOLIBRI-Datenbank aus.

5.2.3 Beschreibung eines Kurses (COURSE)

Die Beschreibung eines Kurses wird durch das Schlüsselwort `COURSE` eingeleitet. Es folgt ein Bezeichner für den Kurs. Dieser Bezeichner ist der interne Name, unter dem der Kurs von KOLIBRI verwaltet wird. Der Name wird für die Administration von Kursen benötigt, etwa um einen vorhandenen Kurs zu löschen. Die Bezeichner der Kurse, die in der KOLIBRI-Datenbank gespeichert sind, müssen eindeutig sein.

Folgende Elemente sind innerhalb eines Kurses erlaubt:

- `TITLE`
- `AUTHOR`
- `BASE`
- `RATING`
- Lektion
- Aufgabe
- Projekt

Die Beschreibung eines Kurses muß durch das Schlüsselwort `END` abgeschlossen werden.

5.2.3.1 Element TITLE

Das Element `TITLE` legt den Titel eines Kurses fest. Der Titel wird an verschiedenen Stellen von KOLIBRI angezeigt, z. B. bei der Kursauswahl oder im Inhaltsverzeichnis des Kurs-Browsers. `TITLE` ist ein optionales Element. Es wird auch für Kapitel und Bausteine verwendet.

5.2.3.2 Element AUTHOR

Das Element `AUTHOR` legt den Autor eines Kurses fest. Nach `AUTHOR` wird ein String erwartet, der den Namen und die E-Mail-Adresse des Autors in der Form `Name <User@Domain>` enthält. `AUTHOR` ist ein optionales Element, das auch für Kapitel und Bausteine verwendet wird.

5.2.3.3 Element BASE

Das Element **BASE** gibt an, unter welchem Pfad der Kurs aus der Sicht des WWW-Servers zu finden ist. Dieser Pfad unterscheidet sich meist von dem Pfad, unter dem der Kurs im Dateisystem zu finden ist. Das Element ist optional. Wenn es nicht angegeben wird, dann muß sich der Kurs im Basis-Dokumentverzeichnis des WWW-Servers befinden, um zur Laufzeit korrekt angezeigt zu werden.

5.2.3.4 Element RATING

Das Element **RATING** fügt dem Kurs eine neue Einstufung hinzu. Es folgt ein Bezeichner, der den Namen der Einstufung angibt. Der Name wird an zahlreichen Stellen verwendet, zum Beispiel um Ausdrücke über Einstufungen zu bilden. Die Bezeichner der Einstufungen eines Kurses müssen eindeutig sein. Die Beschreibung einer Einstufung muß durch das Schlüsselwort **END** abgeschlossen werden.

Innerhalb einer Einstufung sind zwei Elemente erlaubt. Das erste beginnt mit **TITLE** und legt einen Titel für die Einstufung fest. Anders als der Name sollte der Titel so gewählt werden, daß er sich zur Anzeige eignet.

Das zweite erlaubte Element beginnt mit dem Schlüsselwort **GROUP** und fügt einer Einstufung eine neue Gruppe hinzu.

Nach **GROUP** wird ein Bezeichner erwartet, der den Namen der neuen Gruppe angibt. Der Name wird an zahlreichen Stellen verwendet, um Ausdrücke über Gruppen zu bilden. Die Namen aller Gruppen eines Kurses müssen eindeutig sein.

Im Anschluß an den Namen kann ein String angegeben werden, der einen Titel für die Gruppe festlegt. Anders als der Name sollte der Titel so gewählt werden, daß er sich zur Anzeige eignet.

Es folgen ein oder zwei Integer-Werte, die die Grenzen der Gruppe definieren. Beide Werte sind positiv. Der erste Wert gibt die Anzahl der Punkte an, die ein Benutzer verlieren muß, um in die nächsttiefere Gruppe zu wechseln. Der zweite Wert gibt die Anzahl der Punkte an, die nötig sind, um in die nächsthöhere Gruppe zu gelangen. Wird nur ein Wert angegeben, gilt dieser für beide Grenzen.

Eine Einstufung muß mindestens zwei Gruppen enthalten. Die Reihenfolge der Gruppen ist signifikant: Die erste Gruppe ist automatisch der niedrigste Wert, den eine Einstufung annehmen kann. Die weiteren Gruppen folgen in aufsteigender Reihenfolge.

5.2.4 Beschreibung einer Lektion (LESSON)

Das Schlüsselwort **LESSON** leitet die Beschreibung einer Lektion ein. Es folgt ein optionaler Bezeichner für die Lektion. Dieser Bezeichner wird verwendet, um das Ziel eines Verweis-Bausteins zu spezifizieren.

Folgende Elemente sind innerhalb von Lektionen erlaubt:

- `TITLE`
- `AUTHOR`
- `REQUIRED`
- `OPTIONAL`
- `USELESS`
- `EVALUATE`
- `SCORE`
- Baustein
- Lektion
- Aufgabe
- Projekt

Die Beschreibung einer Lektion wird durch das Schlüsselwort `END` abgeschlossen.

5.2.4.1 Element `REQUIRED`

Das Element `REQUIRED` legt fest, für welche Gruppen von Studenten eine Lektion obligatorisch ist. Nach `REQUIRED` wird eine Liste von Gruppen oder das reservierte Wort `ALL` erwartet. `REQUIRED` kann mehrfach angegeben werden. Effektiv nimmt das Aufführen einer Gruppe an dieser Stelle die Gruppe aus den Relationen der optionalen und obsoleten Kapitel heraus und fügt sie der Relation der obligatorischen Kapitel hinzu. Die Reihenfolge von `REQUIRED`, `OPTIONAL` und `USELESS` ist damit signifikant.

5.2.4.2 Element `OPTIONAL`

Das Element `OPTIONAL` legt fest, für welche Gruppen von Studenten eine Lektion optional ist. Nach `OPTIONAL` wird eine Liste von Gruppen oder das reservierte Wort `ALL` erwartet. `OPTIONAL` kann mehrfach angegeben werden. Effektiv nimmt das Aufführen einer Gruppe an dieser Stelle die Gruppe aus den Relationen der obligatorischen und obsoleten Kapitel heraus und fügt sie der Relation der optionalenn Kapitel hinzu. Die Reihenfolge von `REQUIRED`, `OPTIONAL` und `USELESS` ist damit signifikant.

5.2.4.3 Element `USELESS`

Das Element `USELESS` legt fest, für welche Gruppen von Studenten eine Lektion obsolet ist. Nach `USELESS` wird eine Liste von Gruppen oder das reservierte Wort `ALL` erwartet. `USELESS` kann mehrfach angegeben werden. Effektiv nimmt das Aufführen einer Gruppe an dieser Stelle die Gruppe aus den Relationen der obligatorischen und optionalen Kapitel heraus und fügt sie der Relation der obsoleten Kapitel hinzu. Die Reihenfolge von `REQUIRED`, `OPTIONAL` und `USELESS` ist damit signifikant.

5.2.4.4 Element EVALUATE

Das Element EVALUATE legt fest, welcher Mechanismus zur Auswertung der Eingaben einer Lektion verwendet wird. Nach EVALUATE wird ein String erwartet, der den Namen einer nachladbaren Klasse angibt, welche die Eingaben bearbeiten oder bewerten soll. Alternativ können die reservierten Worte AUTO oder MANUAL angegeben werden, um einen Standardmechanismus zur automatischen oder manuellen Auswertung zu verwenden.

5.2.4.5 Element SCORE

Das Element SCORE legt fest, auf welche Einstufungen sich das Anzeigen einer Lektion auswirkt und wie stark die Auswirkung sein soll (implizite Wissensstandkontrolle). Nach SCORE kann durch einen Integer-Wert die Anzahl der Punkte festlegt werden, mit denen die Lektion bewertet ist. Anschließend kann optional eine Liste von Einstufungen angegeben werden, für die diese Punktzahl gilt. Fehlt die Angabe der Punktzahl, wird eine Vorgabe von einem Punkt angenommen. Fehlt die Liste der Einstufungen, gilt diese Punktzahl für alle Einstufungen. SCORE kann mehrfach pro Lektion angegeben werden, um verschiedene Punktzahlen für die unterschiedlichen Einstufungen eines Kurses zu vergeben.

5.2.5 Beschreibung einer Aufgabe (EXERCISE)

Das Schlüsselwort EXERCISE leitet die Beschreibung einer Aufgabe ein. Es folgt ein optionaler Bezeichner für die Aufgabe. Wie bei Lektionen wird dieser Bezeichner verwendet, um das Ziel eines Verweis-Bausteins zu spezifizieren.

Folgende Elemente sind innerhalb von Lektionen erlaubt:

- TITLE
- AUTHOR
- REQUIRED
- OPTIONAL
- USELESS
- EVALUATE
- SUCCESS
- Baustein
- Teilaufgabe

Die Beschreibung einer Aufgabe wird durch das Schlüsselwort END abgeschlossen.

5.2.5.1 Element SUCCESS

Das Element SUCCESS legt fest, bei welchem Prozentsatz erreichter Punkte eine Aufgabe als bestanden gilt. Dies ist wichtig für die Entscheidung, welche Kursteile nach dem

Bewerten einer Aufgabe als verstanden oder nicht verstanden markiert werden. Nach SUCCESS wird ein Integer-Wert erwartet, der angibt, wieviel Prozent der erreichbaren Punkte der Student benötigt, um den Aufgabenabschnitt zu bestehen. SUCCESS ist ein optionales Element. Wird es nicht angegeben, dann gilt eine Vorgabe von 50 %.

5.2.6 Beschreibung einer Teilaufgabe (PART)

Das Schlüsselwort PART innerhalb einer Aufgabe leitet die Beschreibung einer Teilaufgabe ein. Es folgt ein optionaler Bezeichner für die Teilaufgabe, der als Zielangabe eines Verweis-Bausteins verwendet werden kann.

Folgende Elemente sind innerhalb von Teilaufgaben erlaubt:

- TITLE
- AUTHOR
- EVALUATE
- SCORE
- Baustein

Die Beschreibung einer Teilaufgabe wird durch das Schlüsselwort END abgeschlossen.

5.2.6.1 Element SCORE

Das Element SCORE legt fest, auf welche Einstufungen sich das Bewerten einer Teilaufgabe auswirkt und wie stark die Auswirkung sein soll (explizite Wissensstandkontrolle). Nach SCORE kann durch einen Integer-Wert die Anzahl der Punkte festlegt werden, mit denen die Teilaufgabe bewertet ist. Anschließend kann optional eine Liste von Einstufungen angegeben werden, für die diese Punktzahl gilt. Fehlt die Angabe der Punktzahl, wird eine Vorgabe von einem Punkt angenommen. Fehlt die Liste der Einstufungen, gilt diese Punktzahl für alle Einstufungen. SCORE kann mehrfach pro Teilaufgabe angegeben werden, um verschiedene Punktzahlen für die unterschiedlichen Einstufungen eines Kurses zu vergeben.

5.2.7 Beschreibung eines Projektes (PROJECT)

Das Schlüsselwort PROJECT leitet die Beschreibung eines Projektes ein. Es folgt ein optionaler Bezeichner für das Projekt. Dieser Bezeichner wird verwendet, um das Ziel eines Verweis-Bausteins zu spezifizieren.

Folgende Elemente sind innerhalb von Projekten erlaubt:

- TITLE
- AUTHOR

- REQUIRED
- OPTIONAL
- USELESS
- EVALUATE
- SUCCESS
- Baustein
- Teilprojekt

Die Beschreibung eines Projektes wird durch das Schlüsselwort END abgeschlossen.

5.2.8 Beschreibung eines Teilprojektes (PART)

Die Schlüsselwortfolgen PRIVATE PART und PUBLIC PART innerhalb eines Projektes leiten die Beschreibung eines privaten bzw. öffentlichen Teilprojektes ein. Statt PRIVATE PART kann auch einfach PART verwendet werden. Es folgt ein optionaler Bezeichner für das Teilprojekt, der als Zielangabe eines Verweis-Bausteins verwendet werden kann.

Folgende Elemente sind innerhalb von Teilprojekten erlaubt:

- TITLE
- AUTHOR
- EVALUATE
- SCORE
- Baustein

Die Beschreibung eines Teilprojektes wird durch das Schlüsselwort END abgeschlossen.

5.2.8.1 Element SCORE

Das Element SCORE legt fest, auf welche Einstufungen sich das Bewerten eines Teilprojektes auswirkt und wie stark die Auswirkung sein soll (explizite Wissensstandkontrolle). Innerhalb der Beschreibung eines Teilprojektes hat SCORE die gleiche Bedeutung wie innerhalb von Teilaufgaben.

5.2.9 Beschreibung eines Bausteins

Der Definition eines Bausteins kann eine beliebige Kombination der folgenden vier Schlüsselwörter vorangehen, die den Verwendungszweck des Bausteins festlegen:

- **NORMAL**: Das Schlüsselwort NORMAL legt fest, daß der Baustein normale Informationen enthält. In Aufgaben und Projekten sowie deren Teilen werden solche Bausteine angezeigt, wenn die Aufgabenstellung präsentiert wird.

- **HINT**: Das Schlüsselwort `HINT` legt fest, daß der Baustein sichtbar ist, wenn der Student Lösungshinweise zu einer Teilaufgabe oder einem Teilprojekt anfordert.
- **RIGHT**: Das Schlüsselwort `RIGHT` legt fest, daß der Baustein sichtbar ist, wenn dem Studenten die Kommentare zu einer korrekt gelösten (Teil-)Aufgabe oder einem korrekt gelösten (Teil-)Projekt angezeigt werden.
- **WRONG**: Das Schlüsselwort `WRONG` legt fest, daß der Baustein sichtbar ist, wenn dem Studenten die Kommentare zu einer falsch gelösten (Teil-)Aufgabe oder einem falsch gelösten (Teil-)Projekt angezeigt werden.

Wird keines der vier Elemente angegeben, dann ist die Kombination aller vier Elemente die Vorgabe, mit dem Effekt, daß der Baustein in jedem Zustand sichtbar ist.

Ein Baustein besitzt zudem einen Typ, der Aufschluß über die Art der enthaltenen Information gibt. Folgende Bausteintypen werden derzeit unterstützt:

- **TEXT**: Mit dem Schlüsselwort `TEXT` wird die Definition eines Bausteins eingeleitet, der Fließtext enthält.
- **IMAGE**: Mit dem Schlüsselwort `IMAGE` wird die Definition eines Bausteins eingeleitet, der eine Grafik enthält.
- **SOUND**: Mit dem Schlüsselwort `SOUND` wird die Definition eines Bausteins eingeleitet, der digitale Audiodaten enthält.
- **MOVIE**: Mit dem Schlüsselwort `MOVIE` wird die Definition eines Bausteins eingeleitet, der digitale Videodaten enthält.
- **TABLE**: Mit dem Schlüsselwort `TABLE` wird die Definition eines Bausteins eingeleitet, der eine Tabelle enthält.
- **FORM**: Mit dem Schlüsselwort `FORM` wird die Definition eines Bausteins eingeleitet, der ein beliebig komplexes Eingabeformular für den Studenten bereitstellt.
- **APPLET**: Mit dem Schlüsselwort `APPLET` wird die Definition eines Bausteins eingeleitet, der ein Applet enthält.
- **LINK**: Mit dem Schlüsselwort `LINK` wird die Definition eines Bausteins eingeleitet, der einen Verweis enthält.

Auf das Schlüsselwort, das den Typ des Bausteins festlegt, folgt ein optionaler Bezeichner für den Baustein. Dieser Bezeicher wird verwendet, um das Ziel eines Verweis-Bausteins zu spezifizieren.

Folgende Elemente sind innerhalb von Bausteinen erlaubt:

- `TITLE`
- `AUTHOR`
- `FILE`
- `VISIBLE`
- `HIDDEN`
- `WIDTH`
- `HEIGHT`
- `PARAM`

Die Beschreibung jedes Bausteins wird durch das Schlüsselwort `END` abgeschlossen.

5.2.9.1 Element `FILE`

Das Element `FILE` legt den Dateinamen eines Bausteins fest. Nach `FILE` wird ein String erwartet, der den Dateinamen angibt. Für jeden Baustein muß ein Dateiname angegeben werden. Bei Verweis-Bausteinen gibt `FILE` das Ziel des Verweises an. Die Datei sollte existieren, während der Kurs in die Datenbank übertragen wird. Andernfalls wird eine Warnung ausgegeben. Die Anforderungen an den Inhalt der Datei sind im Abschnitt 5.1.1 auf Seite 64 bei der Beschreibung der einzelnen Bausteintypen aufgelistet.

5.2.9.2 Element `VISIBLE`

Das Element `VISIBLE` legt fest, für welche Gruppen von Studenten ein Baustein beim ersten Betreten eines Kapitels sichtbar ist. Nach `VISIBLE` wird eine Liste von Gruppen des Kurses oder das reservierte Wort `ALL` erwartet. `VISIBLE` kann mehrfach angegeben werden. Effektiv nimmt das Aufführen einer Gruppe an dieser Stelle die Gruppe aus der Relation der unsichtbaren Bausteine heraus und fügt sie der Relation der sichtbaren Bausteine hinzu. Die Reihenfolge von `VISIBLE` und `HIDDEN` ist damit signifikant.

5.2.9.3 Element `HIDDEN`

Das Element `HIDDEN` legt fest, für welche Gruppen von Studenten ein Baustein beim ersten Betreten eines Kapitels unsichtbar ist. Nach `HIDDEN` wird eine Liste von Gruppen des Kurses oder das reservierte Wort `ALL` erwartet. `HIDDEN` kann mehrfach angegeben werden. Effektiv nimmt das Aufführen einer Gruppe an dieser Stelle die Gruppe aus der Relation der sichtbaren Bausteine heraus und fügt sie der Relation der unsichtbaren Bausteine hinzu. Die Reihenfolge von `VISIBLE` und `HIDDEN` ist damit signifikant.

5.2.9.4 Element `WIDTH`

Das Element `WIDTH` gibt die Breite an, mit der ein Baustein in die erzeugte HTML-Seite eingebettet werden soll. Nach `WIDTH` wird ein Integer erwartet, der die Breite in Punkten spezifiziert.

5.2.9.5 Element HEIGHT

Das Element HEIGHT gibt die Höhe an, mit der ein Baustein in die erzeugte HTML-Seite eingebettet werden soll. Nach HEIGHT wird ein Integer erwartet, der die Höhe in Punkten spezifiziert.

5.2.9.6 Element PARAM

Das Element PARAM gibt einen beliebigen Parameter für den Baustein an. Diese Parameter werden für den Bausteintyp Applet verwendet (siehe Abschnitt 5.1.1.7 auf Seite 68). Nach PARAM wird ein Bezeichner für den Parameter und anschließend ein String mit dem Wert des Parameters erwartet. PARAM kann beliebig oft angegeben werden.

5.2.10 Syntaxdiagramme

Die Syntaxdiagramme auf den folgenden Seiten geben die Grammatik der Kursdatei grafisch wieder.

5.2.11 Verzeichnisstruktur

Bei der Aufteilung der Dateien eines Kurses auf Verzeichnisse und den Verweisen auf Bausteine innerhalb der Kursbeschreibung sind einige Punkte zu beachten. Anderenfalls kann es zur Laufzeit des Systems zu Problemen kommen, die sich im wesentlichen darin äußern, daß entweder KOLIBRI oder der WWW-Client des Studenten einen Baustein nicht findet.

- Das Verzeichnis, in dem sich die Kursbeschreibung befindet, wird von KOLIBRI als das *Basisverzeichnis* des Kurses betrachtet. Es ist möglich, in der Kursbeschreibung ein anderes Basisverzeichnis anzugeben. Das Basisverzeichnis teilt KOLIBRI mit, wo es die Daten eines Kurses findet.
- Alle Bausteine, die zum Kurs gehören, sollten sich im Dateisystem *unterhalb* des Basisverzeichnisses befinden. Die Struktur kann zur leichteren Verwaltung auf beliebig viele Unterverzeichnisse aufgeteilt werden.
- Verweise auf Bausteine müssen in der Kursbeschreibung einer der beiden folgenden Varianten entsprechen:
  - **Relativ:** Im Normalfall werden Verweise relativ zum Basisverzeichnis des Kurses bzw. zur Kursdatei angegeben. Verweise dieser Art werden von KOLIBRI erkannt und zur Laufzeit automatisch in absolute Verweise umgewandelt. Nur so ist gewährleistet, daß die Bausteine vom WWW-Client des Studenten wirklich gefunden werden.

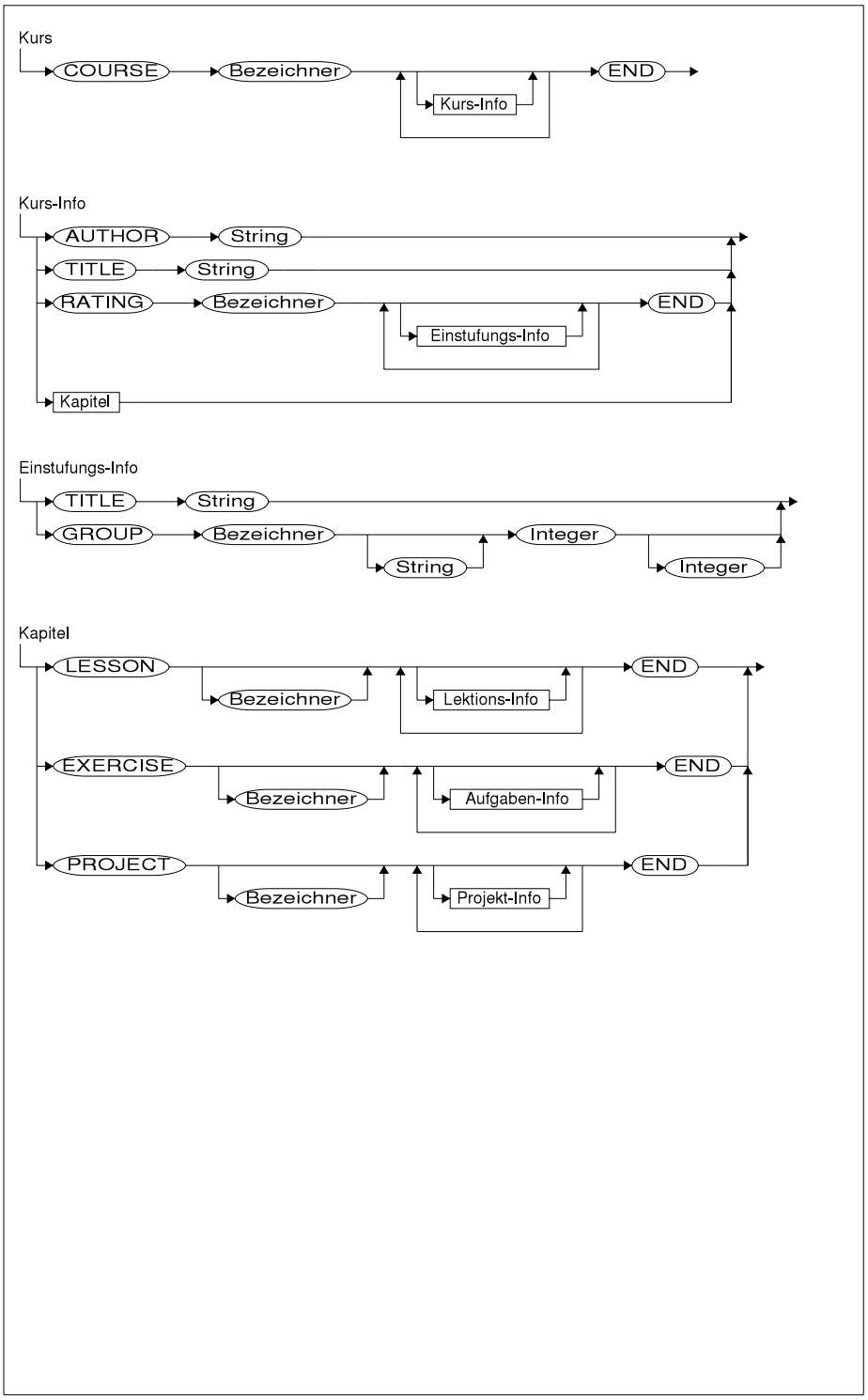

Abbildung 5.8: Syntaxdiagramm für Kursdateien (Teil 1 von 3)

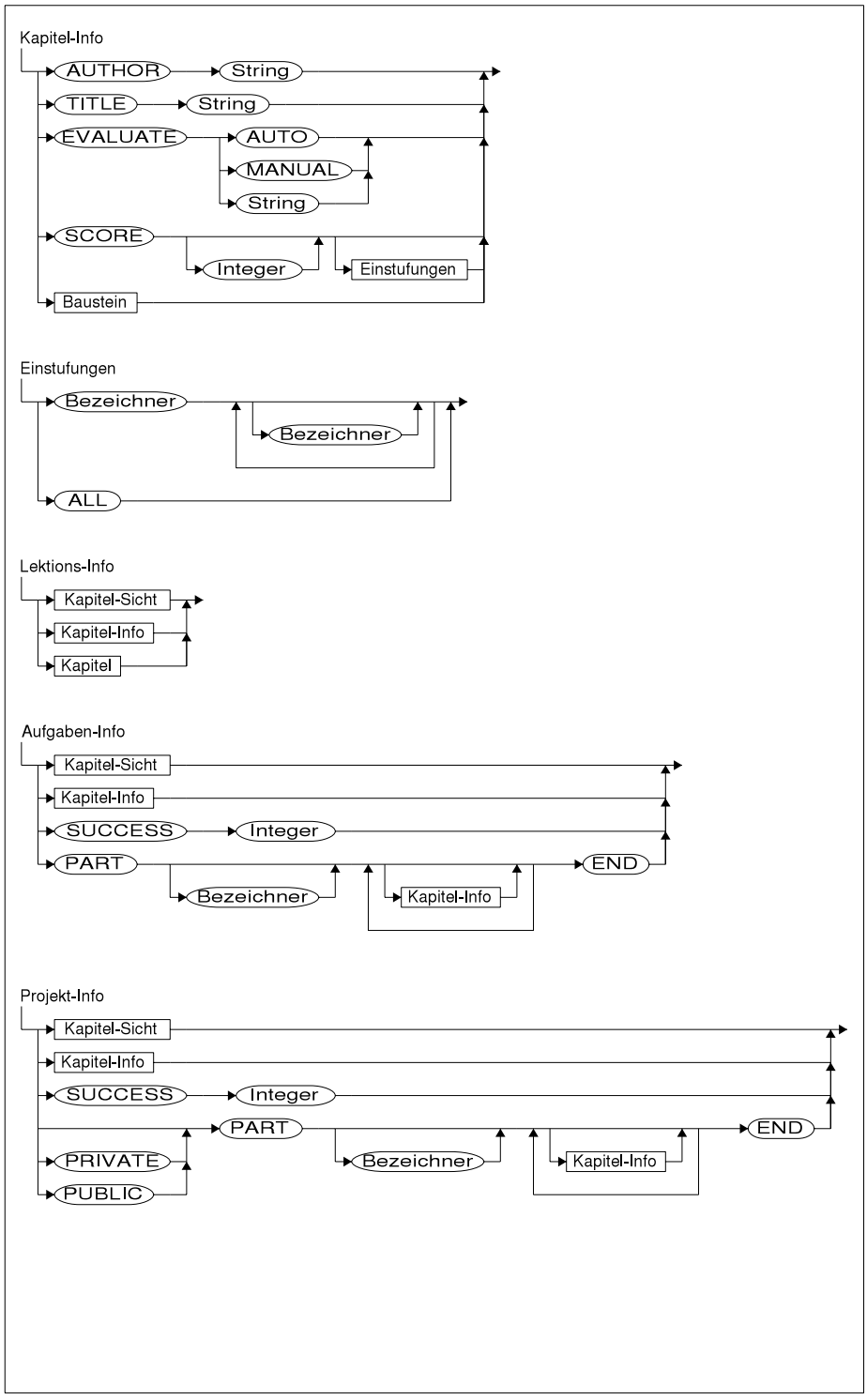

Abbildung 5.9: Syntaxdiagramm für Kursdateien (Teil 2 von 3)

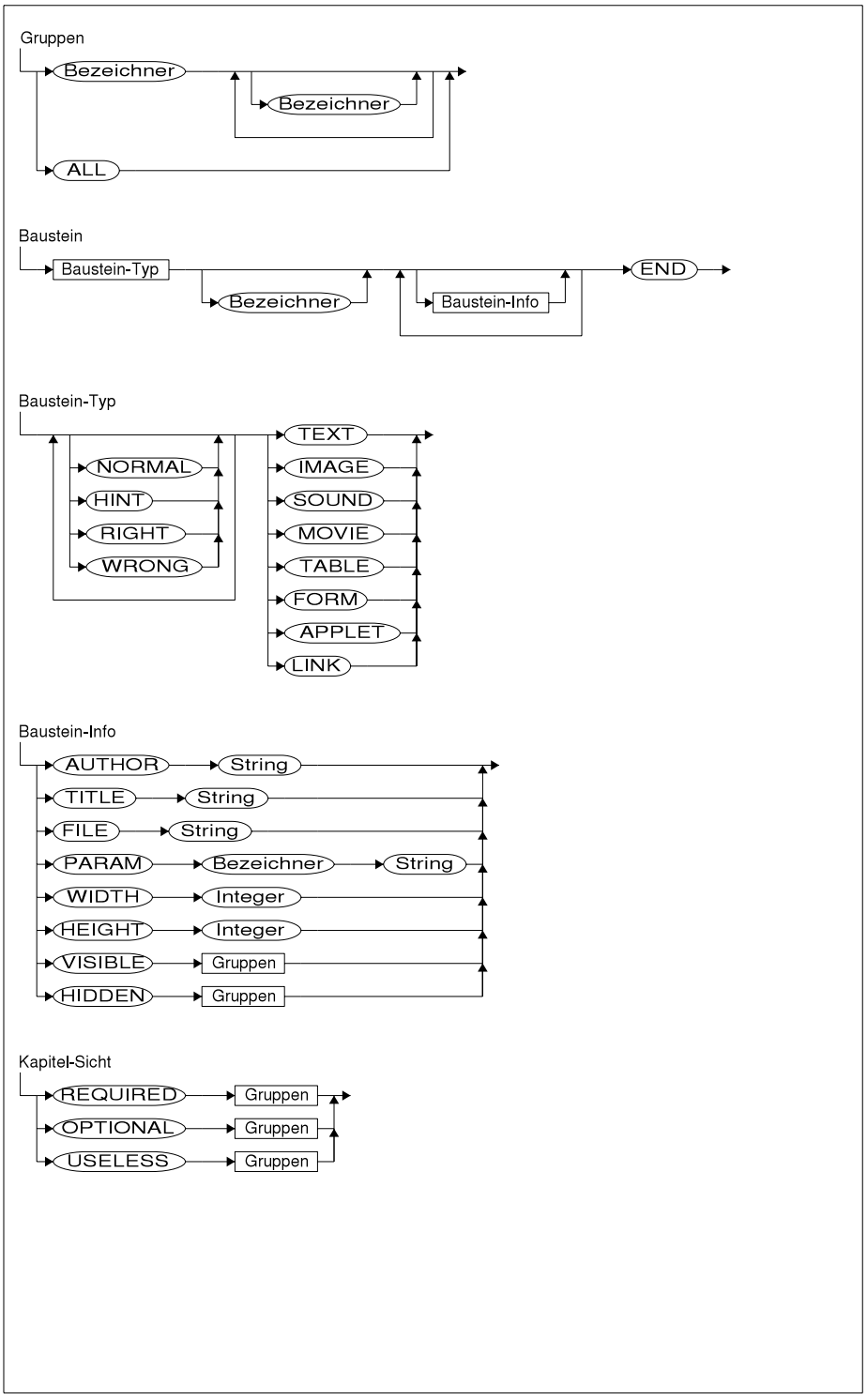

Abbildung 5.10: Syntaxdiagramm für Kursdateien (Teil 3 von 3)

- **Absolut:** Falls ein Baustein verwendet wird, der von einem anderen Server stammt, muß der komplette URL des Bausteins – inklusive Protokoll – angegeben werden. Verweise dieser Art werden von KOLIBRI nicht geändert, sondern unverändert an den WWW-Client des Benutzers übertragen.

Absolute Verweise innerhalb des lokalen Dateisystems sollten nicht verwendet werden, da Pfade, die mit einem Schrägstrich (/) beginnen, im Dateisystem und im WWW-Server eine unterschiedliche Bedeutung haben. Dies wird zur Laufzeit zu Problemen führen, weil Dateien nicht gefunden werden können.

Die folgende beispielhafte Verzeichnisstruktur für KOLIBRI-Kurse sollte eingehalten werden, soweit das möglich ist:

`kurs/`	Basisverzeichnis für Kurs
`kurs/kurs.txt`	Datei mit Kursbeschreibung
`kurs/kapitel-1/`	Verzeichnis für Kapitel 1
`kurs/kapitel-1/baustein-1.html`	Ein Baustein
`kurs/kapitel-1/baustein-2.html`	Ein weiterer Baustein
`...`	...
`kurs/kapitel-1/bilder`	Abbildungen für Kapitel 1
`kurs/kapitel-2`	Verzeichnis für Kapitel 2
`...`	...
`kurs/kapitel-n/`	Verzeichnis für Kapitel n
`kurs/bilder`	Abbildungen für alle Kapitel

Tabelle 5.1: Empfohlene Verzeichnisstruktur für Kurse

Es wird meist nötig sein, den Kurs entweder unterhalb des Basis-Dokumentpfads des WWW-Servers abzulegen oder dort einen symbolischen Link auf das Kursverzeichnis anzulegen.

5.3 Verwaltung von Kursen

Das Hilfsprogramm `KursAdmin` dient zur Administration von KOLIBRI-Kursen. Es bildet die Schnittstelle zwischen dem Autor eines Kurses und der KOLIBRI-Datenbank.

`KursAdmin` ist ein Java-Programm, das im einzelnen folgende Aufgaben erfüllt:

- Compilieren einer Kursbeschreibung und Einfügen des neuen Kurses in die KOLIBRI-Datenbank.
- Auflisten der Kurse, die in der KOLIBRI-Datenbank vorhanden sind.

- Entfernen eines Kurses und aller abhängigen Daten aus der KOLIBRI-Datenbank.

Der Aufruf von `KursAdmin` ohne Parameter zeigt eine kurze Übersicht der möglichen Kommandos an.

5.3.1 Einfügen von Kursen

Um eine Kursbeschreibung zu compilieren und den compilierten Kurs in die KOLIBRI-Datenbank einzufügen, ist folgender Aufruf von `KursAdmin` erforderlich:

```
java kolibri.kursadmin.KursAdmin <Datenbank> insert <Datei>
```

Der Parameter `<Datenbank>` gibt den Uniform Resource Locator der KOLIBRI-Datenbank an. Für den im Rahmen der Projektgruppe installierten Prototyp ist dies der folgende URL:

```
msql://delta.informatik.uni-dortmund.de:12348/kolibri
```

Der Parameter `<Datei>` gibt den Pfad der Datei an, in welcher die Beschreibung des neuen Kurses enthalten ist. Der Kurs wird nur dann in die Datenbank übernommen, wenn er syntaktisch und – soweit es KOLIBRI betrifft – semantisch korrekt ist. Die Datenbank verbleibt in jedem Fall in einem konsistenten Zustand.

5.3.2 Anzeigen von Kursen

Um eine Liste aller Kurse zu erhalten, die derzeit in der KOLIBRI-Datenbank gespeichert sind, ist folgender Aufruf von `KursAdmin` erforderlich:

```
java kolibri.kursadmin.KursAdmin <Datenbank>
```

Der Parameter `<Datenbank>` gibt den Uniform Resource Locator der KOLIBRI-Datenbank an. Es werden die Titel, Bezeichner und Autoren sämtlicher Kurse angezeigt, die in der angegebenen KOLIBRI-Datenbank enthalten sind.

5.3.3 Entfernen von Kursen

Um einen Kurs wieder aus der KOLIBRI-Datenbank zu entfernen, ist folgender Aufruf von `KursAdmin` erforderlich:

```
java kolibri.kursadmin.KursAdmin <Datenbank> delete <Kurs>
```

Der Parameter `<Datenbank>` gibt den Uniform Resource Locator der KOLIBRI-Datenbank an.

Der Parameter `<Kurs>` gibt den internen Bezeichner des Kurses an, der entfernt werden soll. Der Bezeichner eines Kurses wird unmittelbar hinter dem Schlüsselwort `COURSE` festgelegt, das die Beschreibung eines Kurses einleitet (siehe Abschnitt 5.2.3).

5.4 Implementierter Beispielkurs

Als konkrete Anwendung von KOLIBRI wurde innerhalb der Projektgruppe ein Kurs mit dem Titel „Einführung in die Fuzzy-Logik“ implementiert. Zu diesem Thema war am Lehrstuhl bereits Material in digitaler Form verfügbar, das verwendet werden konnte. Der Stoff wurde ergänzt durch verschiedene Aufgaben, die den Wissensstand und Lernfortschritt eines einzelnen Studenten prüfen. Abgerundet wurde der gesamte Kurs durch ein Projekt, in dem zwei Studenten mit Hilfe von KOLIBRI kooperativ einen unscharfen Regler zur Regulierung des Wasserstands eines Beckens entwerfen.

5.4.1 Übersicht über den Kursinhalt

Der Beispielkurs „Einführung in die Fuzzy-Logik“ verwendet Material aus dem gleichnamigen Skript von Prof. Dr. Helmut Thiele, das der Projektgruppe freundlicherweise als LaTeX-Quellcode zur Verfügung gestellt wurde (siehe dazu [Thie96]).

Das gesamte Dokument wurde mit Hilfe von `latex2html` (siehe dazu [Drak96]) in HTML-Dateien umgewandelt. Da derzeit kein WWW-Client das Anzeigen mathematischer Formeln unterstützt, wurde die Entscheidung getroffen, die Formeln des Skripts automatisch in eingebettete GIF-Grafiken zu konvertieren.

Das Ergebnis der Konvertierung in HTML-Dateien wurde manuell nachbearbeitet und in Bausteine zerlegt, die von KOLIBRI verarbeitet werden können. Daraus wurde ein Kurs erstellt, der die Möglichkeiten von KOLIBRI demonstriert.

5.4.1.1 Einstufungen und Gruppen

Der Beispielkurs verwendet folgende Einstufungen und Gruppen:

- **Wissen**: Diese Einstufung setzt sich aus den drei Gruppen *Niedrig (Anfänger)*, *Mittel (Fortgeschrittener)* und *Hoch (Experte)* zusammen. Sie wird dazu verwendet, den allgemeinen Wissensstand bzw. Fortschritt des Studenten zu messen.
- **Theorie**: Diese Einstufung setzt sich aus den zwei Gruppen *Nontheorie (Student ist Nicht-Theoretiker)* und *Theorie (Student ist Theoretiker)* zusammen. Sie wird dazu verwendet, das Interesse des Studenten an im Kurs enthaltenen theoretischen Informationen zu messen.

- **Praxis:** Diese Einstufung setzt sich aus den zwei Gruppen *Nonpraxis (Student ist Nicht-Praktiker)* und *Praxis (Student ist Praktiker)* zusammen. Analog zur Einstufung *Theorie* wird sie dazu verwendet, das Interesse des Studenten an im Kurs enthaltenen praktischen Informationen zu messen.

5.4.1.2 Kapitelstruktur

Der Beispielkurs besitzt die im folgenden dargestellte Kapitelstruktur:

1. Einleitung

   Dieses Kapitel begrüßt den Studenten zum Kurs und gibt einige Hinweise zur Bedienung des Systems. Der Stoff des Kapitels ist größtenteils allgemeiner Natur und dementsprechend für alle Gruppen von Interesse.

   Das Kapitel enthält folgende Unterkapitel:

   1.1. Quellen

   1.2. Probleme der klassischen Logik

   1.3. Geschichte der unscharfen Logik

   1.4. Anwendungen im Überblick

2. Unscharfe Mengen und Operationen

   Dieses Kapitel enthält Grundlagenstoff zu scharfen und unscharfen Mengen sowie den Operationen darauf. Insbesondere werden T- und S-Normen behandelt. Da der Grundlagenstoff für alle Studenten wichtig ist, ist das Kapitel für alle Gruppen obligatorisch. Es umfaßt unter anderem zwei Lernkontrollen. Die zweite Hälfte des Kapitels wurde als fortgeschrittener Stoff klassifiziert. Sie ist anfangs für den Studenten obsolet, wird aber optional oder sogar obligatorisch, wenn er – etwa durch die erste Lernkontrolle – innerhalb der Einstufung *Wissen* aufsteigt.

   Das Kapitel enthält folgende Unterkapitel:

   2.1. Scharfe Mengen

   2.2. Unscharfe Mengen

   2.3. Operationen auf scharfen Mengen

   2.4. Łukasiewiczsche Funktionen

   2.5. Standard-Operationen auf Fuzzy-Mengen

   2.6. Lernkontrolle

   2.7. T-Normen

   2.8. S-Normen

   2.9. Grafische Darstellung von Operatoren

2.10. Lernkontrolle

3. Approximatives Schließen

   Dieses Kapitel enthält Informationen zum approximativen Schließen. Die Informationen des Kapitels wurden für die meisten Gruppen als optional eingestuft. Einzig für Theoretiker ist dieses Kapitel obligatorisch. Es wurde verwendet, um das Funktionieren der impliziten Kontrolle zu testen. Besucht ein Student mehrere Unterkapitel dieses Kapitels, so wird er als theoretisch interessiert eingestuft. Damit wird das gesamte Kapitel für ihn obligatorisch.

   Das Kapitel enthält folgende Unterkapitel:

   3.1. Approximatives Schließen und generalisierter Modus Ponens

   3.2. Kompositionsregel der Inferenz

   3.3. Unscharfe Relationen

4. Unscharfe Regelung

   Dieses Kapitel enthält Informationen zur unscharfen Regelung. Die Informationen des Kapitels wurden für die meisten Gruppen als optional eingestuft. Einzig für Praktiker ist dieses Kapitel obligatorisch. Auch dieses Kapitel wurde verwendet, um das Funktionieren der impliziten Überwachung zu testen. Der Besuch mehrerer Unterkapitel dieses Kapitels stuft einen Studenten als praktisch interessiert ein. Damit wird das gesamte Kapitel für ihn obligatorisch.

   Das Kapitel enthält folgende Unterkapitel:

   4.1. Probleme der klassischen Regelung

   4.2. Unscharfe Regler

   4.3. Neuronale Netze

   4.4. Anwendungen im Detail

5. Entwurf eines unscharfen Reglers

   Dieses Kapitel stellt die Abschlußprüfung des Kurses dar: Der Student entwickelt zusammen mit einem weiteren Studenten kooperativ einen eigenen unscharfen Regler. Dieser Regler soll den Pegel eines mit Wasser gefüllten Beckens möglichst konstant halten. Das Kapitel wurde als anspruchsvoller Stoff gekennzeichnet. Es ist anfangs für den Studenten obsolet. Steigt er innerhalb der Einstufung *Wissen* auf – entweder durch das Besuchen vieler Kapitel oder durch das Lösen von Lernkontrollen –, dann wird es für ihn optional oder sogar obligatorisch.

   Das Kapitel enthält folgende Unterkapitel:

   5.1. Vereinbaren gemeinsamer Parameter

   5.2. Bearbeiten der linguistischen Variablen

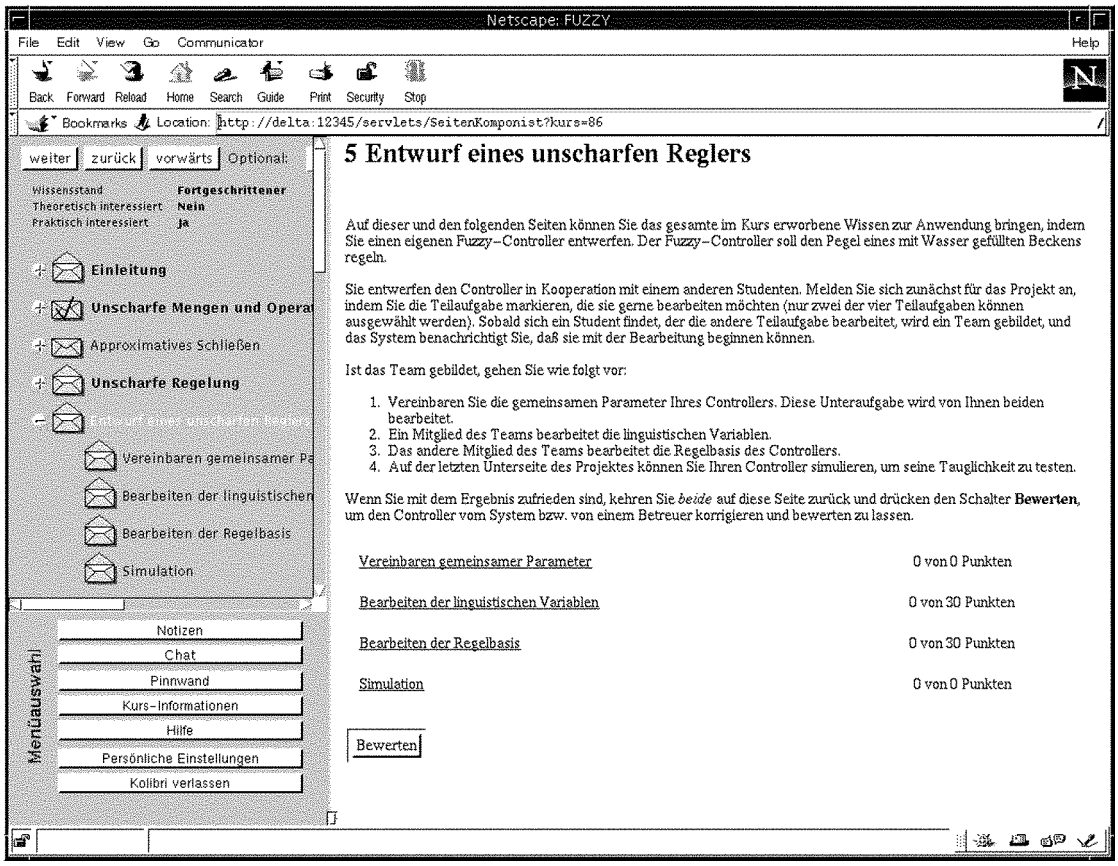

Abbildung 5.11: Entwurf eines unscharfen Reglers

5.3. Bearbeiten der Regelbasis

5.4. Simulation

Die folgenden Abschnitte beschäftigen sich mit der Beschreibung der Aufgabenstellung der kooperativen Aufgabe und mit den Applets, die den Benutzer bei der Bearbeitung unterstützen.

5.4.2 Entwurf eines unscharfen Reglers

Am Ende des implementierten Beispielkurses befindet sich ein mehrteiliges Projekt, das von zwei Studenten in Kooperation zu bearbeiten ist. Das Projekt zerfällt in folgende Teilprojekte:

- Vereinbaren gemeinsamer Parameter (öffentliches Teilprojekt).

- Bearbeiten der linguistischen Variablen (privates Teilprojekt).
- Bearbeiten der Regelbasis (privates Teilprojekt).
- Simulation (öffentliches Teilprojekt).

5.4.2.1 Aufgabenstellung

Ziel des Projektes ist die unscharfe Regelung eines physikalischen Systems, das aus einem Wasserbecken mit verschiedenen Zu- und Abflüssen besteht. Der Wasserstand des Beckens soll etwa auf der Hälfte des zur Verfügung stehenden Volumens gehalten werden. Abbildung 5.12 verdeutlicht den Aufbau des Wasserbeckens. Die folgenden Abschnitte beschreiben das zu regelnde System detaillierter.

Der Schwimmer und die Ventile Der Wasserstand drückt sich durch den jeweils aktuellen Winkel aus, der zu einem Zeitpunkt zwischen dem Schwimmer und der gedachten waagerechten Achse besteht: 0° bedeutet hierbei den Optimalzustand des Systems, während 90° (Becken ist voll) und -90° (Becken ist leer) kritische Fälle darstellen, bei denen die Regelung als gescheitert angesehen werden kann.

Um Dynamik in das System zu bringen, existieren drei Eingänge und drei Ausgänge, durch die Wasser in das Becken einfließt bzw. aus dem Becken herausfließt. Der Durchsatz

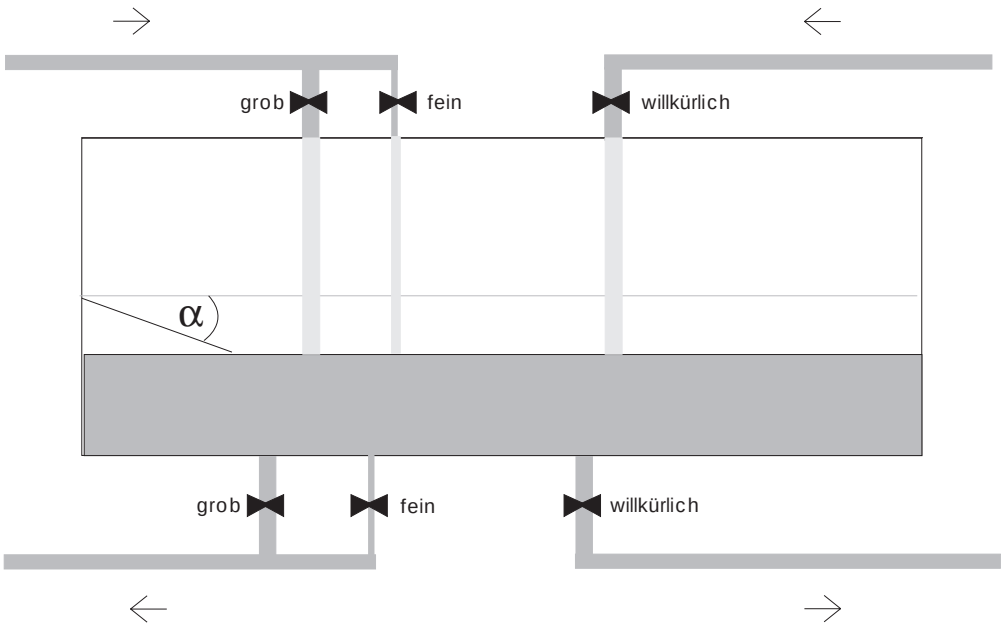

Abbildung 5.12: Aufbau des Wasserbeckens

der Eingänge ist dabei proportional zu der aktuellen Ventilöffnung. Der Durchsatz bei den Ausgängen ist zusätzlich abhängig vom aktuellen Wasserstand.

Jeweils zwei Ventile der Ein- bzw. Ausgänge sind steuerbar, eines ist nicht steuerbar, wird also willkürlich geändert. „Willkürlich“ bedeutet dabei folgendes: Mit einer Wahrscheinlichkeit von etwa 20 % wird die jeweilige Ventilöffnung auf einen zufälligen Wert zwischen 0%-iger und 100%-iger Öffnung neu gesetzt.

Die steuerbaren Ventile unterscheiden sich darin, wieviel Volumen Wasser pro Zeiteinheit maximal durch sie fließen kann (durch das *grobe* Ventil z. B. neunmal soviel wie durch das *feine*). Gleichzeitig kann der Öffnungsgrad eines Ventils nur in bestimmten Schritten erhöht werden (z. B. 10 Prozentpunkte), wodurch sich die Begriffe *grob* und *fein* erklären. Um beispielsweise 7 % des maximal durch die beiden steuerbaren Eingänge möglichen Volumens an Wasser pro Zeit einfließen zu lassen, muß bei den oben gewählten Werten das Ventil des groben Eingangs auf 0%-ige und das Ventil des feinen Eingangs auf 70%-ige Öffnung gestellt werden.

Meß- und Steuergrößen (und deren Wertebereiche) Zum Entwurf des Reglers ist es wichtig zu wissen, welche Größen gemessen werden und welche Größen zu steuern sind.

Steuergrößen des Systems sind:

- Öffnungsgrade der vier steuerbaren Ventile. Wertebereich $[0, 1]$.

Meßgrößen des Systems sind:

- Winkel des Schwimmers. Wertebereich $[-90, 90]$.
- Winkeländerung pro Zeiteinheit. Wertebereich $[-90, 90]$.
- Öffnungsgrade der beiden willkürlichen Ventile. Wertebereich $[0, 1]$.
- Gesamtabfluß der beiden steuerbaren Ausgänge pro Zeiteinheit. Wertebereich $[0, x]$, wobei $x = \sqrt{2gh} \cdot A_{\mathrm{MaxS}}$ problematischerweise abhängig ist von den Eigenschaften des Wasserbeckens. Als Lösung wird x einfach festgesetzt (z. B. auf 100). Das ist keine große Einschränkung, da der Gesamtabfluß wahrscheinlich sowieso keine relevante Größe darstellt.
- Temperatur des Wassers. Wertebereich $[10, 70]$.

Physikalische Grundlagen Das Volumen des Wassers im Becken ist gleich dem Produkt von Wasserstand, Länge und Breite des Beckens:

$$V_{H_2O} = h_{H_2O} \cdot l \cdot b.$$

Das Volumen des Wassers ändert sich (innerhalb einer Zeiteinheit) durch das einfließende und abfließende Wasser:

$$\dot{V}_{H_2O} = \dot{V}_{\text{Eingang}} - \dot{V}_{\text{Ausgang}}.$$

Sowohl das einfließende als auch das abfließende Wasser bestehen aus dem steuerbaren und dem willkürlichen Anteil:

$$\dot{V}_{\text{Eingang}} = \dot{V}_{\text{E\_steuerbar}} + \dot{V}_{\text{E\_willkürlich}}$$

und

$$\dot{V}_{\text{Ausgang}} = \dot{V}_{\text{A\_steuerbar}} + \dot{V}_{\text{A\_willkürlich}}.$$

Da der Druck auf den Eingängen konstant bleibt, kann das Volumen pro Zeit an den Eingängen über den maximal möglichen Durchsatz berechnet werden. Die Öffnungsgrade o können Werte zwischen 0 und 1 annehmen. Wenn das Verhältnis c_{GrobFein} der maximal möglichen Durchsätze von groben und feinen Ventil festgelegt ist, dann ergibt sich mit den Hilfskonstanten

$$c_{\text{grob}} = \frac{c_{\text{GrobFein}}}{c_{\text{GrobFein}} + 1}$$

und

$$c_{\text{fein}} = \frac{1}{c_{\text{GrobFein}} + 1}$$

die allgemeinere Gleichung:

$$\dot{V}_{\text{E\_steuerbar}} = \dot{V}_{\text{ES\_max}}(o_{\text{ES\_grob}} \cdot c_{\text{grob}} + o_{\text{ES\_fein}} \cdot c_{\text{fein}}).$$

Für den willkürlichen Eingang gilt analog:

$$\dot{V}_{\text{E\_willkürlich}} = \dot{V}_{\text{EW\_max}} \cdot o_{\text{EW}}.$$

Da der Druck auf den Ausgängen abhängig vom Wasserstand ist, gilt für das Volumen pro Zeit an den Ausgängen (wobei $g \approx 9,81 m/s^2$ die Gravitationskonstante und h der Wasserstand ist):

$$\dot{V}_{\text{A\_steuerbar}} = v_{\text{AS}} \cdot A_{\text{AS}} = \sqrt{2gh} \cdot A_{\text{MaxS}} \cdot (o_{\text{AS\_grob}} \cdot c_{\text{grob}} + o_{\text{AS\_fein}} \cdot c_{\text{fein}})$$

und

$$\dot{V}_{\text{A\_willkürlich}} = v_{\text{AW}} \cdot A_{\text{AW}} = \sqrt{2gh} \cdot A_{\text{MaxW}} \cdot o_{\text{AW}}.$$

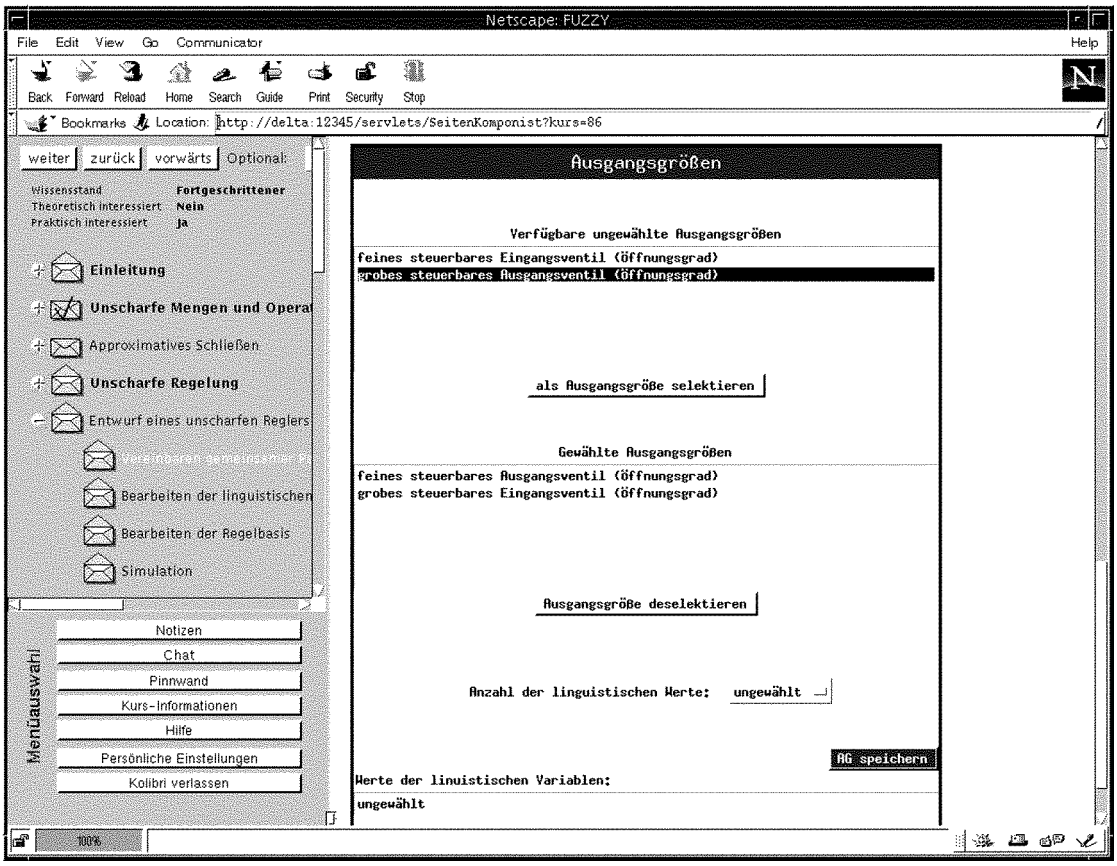

Abbildung 5.13: Vereinbaren gemeinsamer Parameter

5.4.2.2 Vereinbaren gemeinsamer Parameter

Das erste Teilprojekt ist öffentlich; es kann und soll also von beiden Studenten bearbeitet werden. Das Teilprojekt dient zum Vereinbaren der Parameter, auf denen die beiden folgenden privaten Teilprojekte aufbauen.

- **Wahl der Eingangsgrößen:** Aus der Gesamtmenge der vorhandenen Eingangsgrößen müssen diejenigen gewählt werden, auf denen der unscharfe Regler aufbauen soll. In der obigen Aufgabenbeschreibung werden diese als Meßgrößen bezeichnet.
- **Wahl der Ausgangsgrößen:** Aus der Gesamtmenge der vorhandenen Ausgangsgrößen müssen diejenigen gewählt werden, die der unscharfe Regler ansteuern soll. In der obigen Aufgabenbeschreibung werden diese als Steuergrößen bezeichnet.
- **Wahl linguistischer Variablen:** Für jede der Eingangs- und Ausgangsgrößen muß eine linguistische Variable definiert werden, im einzelnen also die gewünschte Anzahl linguistischer Werte und die Namen dieser Werte (etwa *klein*, *mittel*, *groß*).

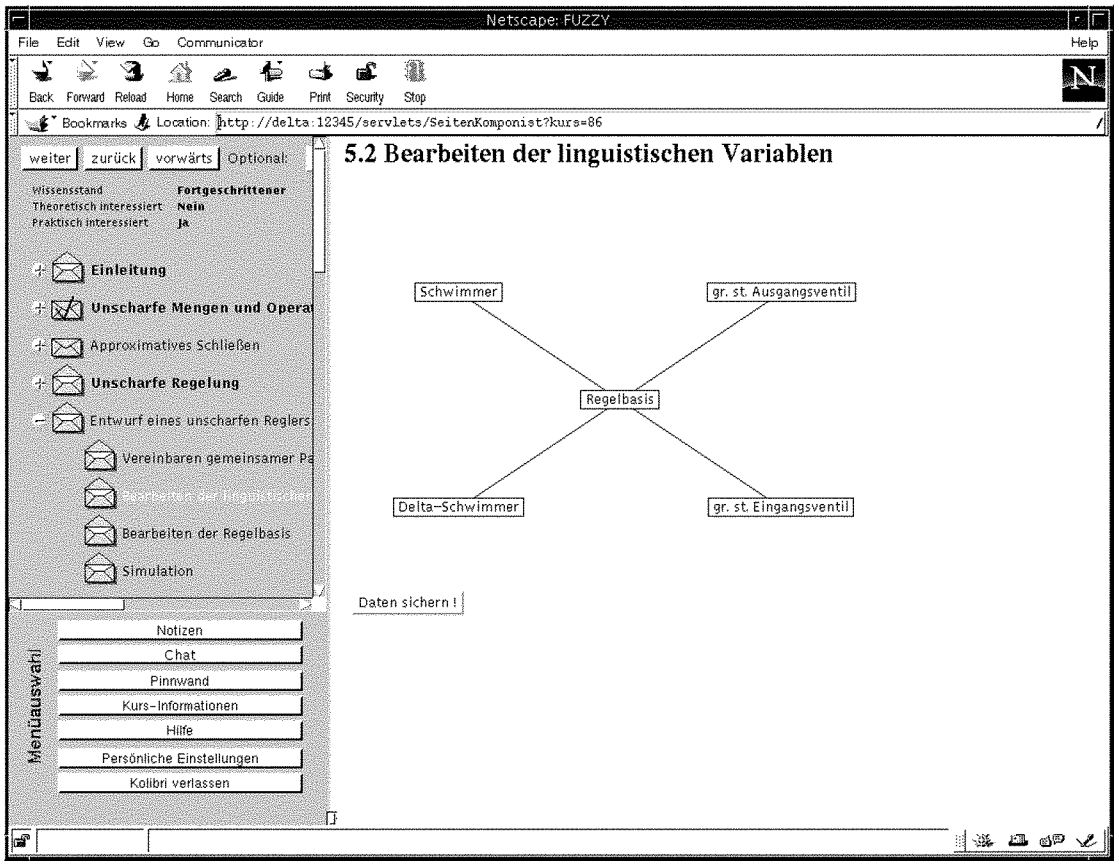

Abbildung 5.14: Übersicht über die Meß- und Regelgrößen

Da die Festlegungen, die in diesem Teilprojekt getroffen werden, für die beiden anderen Teilprojekte von entscheidender Bedeutung sind, muß das Teilprojekt vor den anderen bearbeitet werden. Das System sorgt dafür, daß diese Reihenfolge eingehalten wird, in dem es im Fehlerfall entsprechende Meldungen in den anderen Teilprojekten anzeigt. Abbildung 5.13 auf der vorherigen Seite zeigt diesen Teil der kooperativen Aufgabe.

5.4.2.3 Bearbeiten der linguistischen Variablen

Zu jedem linguistischen Wert wird eine zugehörige unscharfe Menge definiert. Unscharfe Mengen werden der Einfachheit halber aus Punkten definiert, wobei der Zugehörigkeitswert jeweils durch lineare Interpolation berechnet wird. Die Punkte können einzeln gesetzt, bewegt und wieder gelöscht werden. Ein unterstützendes Gitter kann hinzugeschaltet werden, um die Editierfeinheit zu bestimmen. Zusätzlich kann auch auf vorgegebene unscharfe Mengen zurückgegriffen werden, die dann auch editierbar sind. Die Defuzzifizierungsmethode kann für jede Ausgangsgröße einzeln gewählt werden.

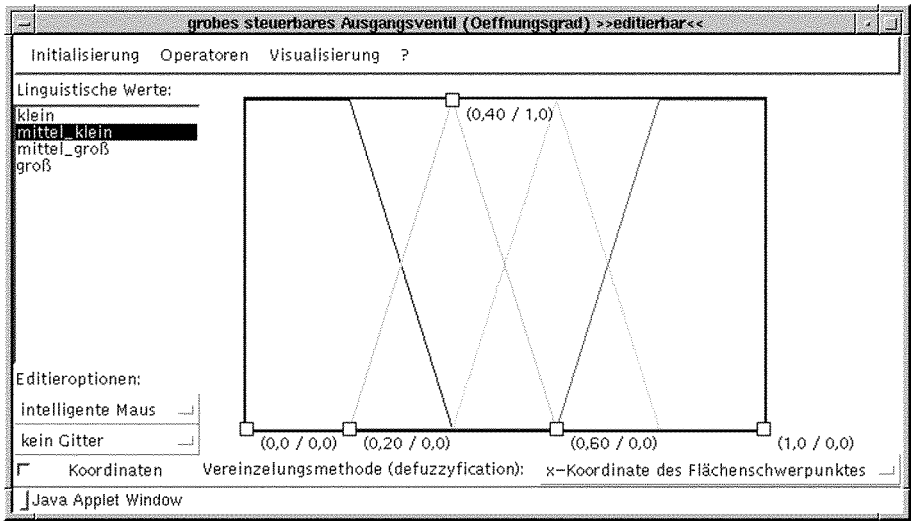

Abbildung 5.15: Bearbeiten der linguistischen Variablen

Alle Operationen, die in der Inferenz angewendet werden, können bereits in dieser Editierphase ausprobiert werden. Daher läßt sich im voraus feststellen, welche Operatoren besser für den individuellen Regler geeignet sind. Die Abbildungen 5.14 und 5.15 zeigen diesen Teil der kooperativen Aufgabe.

5.4.2.4 Bearbeiten der Regelbasis

Zusätzlich zur Bearbeitung der einzelnen Regeln werden hier die gewünschten Interpretationen der Kombination, Implikation und Aggregation gewählt. Hierbei ist die Möglichkeit gegeben, den Regler bereits punktweise zu testen, d. h. nach Spezifizierung der Meßgrößen durch scharfe Werte wird die Inferenz durchgeführt und visualisiert. Diese Funktionalität unterstützt die richtige Wahl der einzelnen Parameter des Reglers. Die Abbildungen 5.16 und 5.17 auf der nächsten Seite zeigen diesen Teil der kooperativen Aufgabe.

5.4.2.5 Simulation

Der erstellte Regler kann hier auf seine Funktionsfähigkeit getestet werden, bevor er über den Schalter [Bewerten] der Hauptseite an einen Kursleiter gesendet wird.

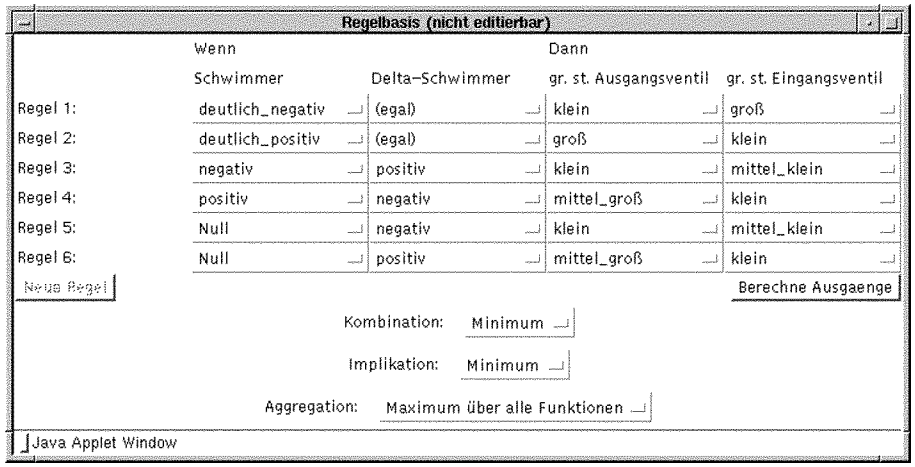

Abbildung 5.16: Bearbeiten der Regelbasis

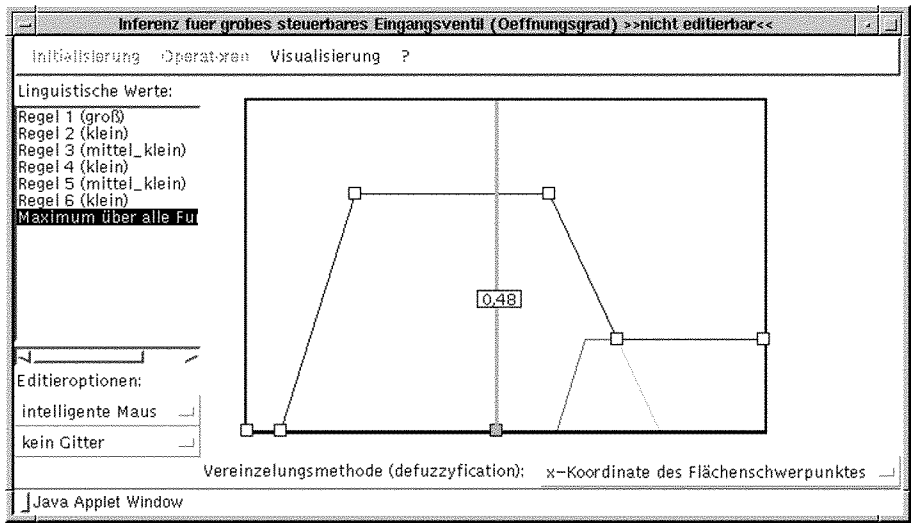

Abbildung 5.17: Visualisierung der Inferenz

6 Bewertung und Ausblick

6.1 Soll-Ist-Vergleich

Nach einjähriger Arbeit ist es angebracht zu fragen, wieviel von dem ursprünglich geplanten Umfang eines Projektes implementiert werden konnte und welche Teile dem Zeitdruck oder unvorhergesehenen technischen Problemen zum Opfer gefallen sind. Die folgenden Abschnitte enthalten einen Vergleich des Anforderungskataloges der Projektgruppe (siehe Abschnitt 2) mit dem Prototypen, der am Ende des zweiten Semesters im Rahmen des Fachgesprachs öffentlich vorgestellt wurde.

6.1.1 Multimediale Darstellung von Lerninhalten

KOLIBRI unterstützt – wie gefordert – verschiedene Medien in einem Kurs. Der implementierte Beispielkurs enthält Text, Grafiken, Formulare, Applets und ein digitalisiertes Video. Tabellen und digitalisierte Audiodaten werden ebenfalls unterstützt, wurden jedoch im Beispielkurs nicht genutzt. Das Bausteinkonzept garantiert Wiederverwertbarkeit und ermöglicht es sogar, Kapitelinhalte dezentral zu speichern, so daß Teile eines Kurses von entfernten Servern bezogen werden können.

6.1.2 Information/Teaching on Demand

KOLIBRI unterstützt – wie gefordert – *Information* bzw. *Teaching on Demand*. Der Student hat zu jedem Zeitpunkt über das Inhaltsverzeichnis Zugriff auf den gesamten Inhalt eines Kurses. Zu jedem Zeitpunkt werden ihm optionale Kapitel als Vertiefung oder Ergänzung der aktuellen Seite angeboten. Während der Bearbeitung einer Aufgabe oder eines Projektes kann der Student Hinweise anfordern, die ihn bei der Lösung unterstützen. Diese Hinweise können auch Verweise auf Teile des Kurses sein. Nach dem erfolglosen Bearbeiten einer Aufgabe werden die zugehörigen Kapitel im Inhaltsverzeichnis als nicht verstanden markiert, so daß der Student sie erneut besuchen kann. Zudem können, abhängig vom Ergebnis der Aufgabe, explizit Verweise auf Teile des Kurses angezeigt werden.

6.1.3 Interaktives Arbeiten mit dem System

KOLIBRI unterstützt – wie gefordert – interaktives Arbeiten. Der Student kann sich frei im System bewegen und zu jedem Zeitpunkt auf beliebige Lerninhalte zugreifen. Durch in den Kurs eingebettete Applets ist ein hochgradig interaktives Arbeiten möglich. Der Entwurf des unscharfen Reglers zeigt das auf eindrucksvolle Weise.

6.1.4 Wissensstandkontrolle

KOLIBRI unterstützt – wie gefordert – sowohl implizite als auch explizite Wissensstandkontrolle. Der Kontrollmechanismus, der durch frei definierbare Einstufungen und Gruppen beliebig fein an einen Kurs angepaßt werden kann, arbeitet auf Kapitel- und auf Bausteinebene. Er kann nicht nur zur Überwachung des Wissens bzw. Fortschritts genutzt werden, sondern auch, um spezielle Interessen des Studenten an Teilbereichen des Kursinhaltes zu erkennen und ihm verwandte Themen bevorzugt anzubieten.

6.1.5 Unterstützung von kooperativem Arbeiten

KOLIBRI unterstützt – wie gefordert – kooperatives Arbeiten. Mit den Projekten existiert ein spezieller Aufgabentyp, der sowohl zur synchronen als auch zur asynchronen Lösung von mehrteiligen Problemstellungen verwendet werden kann. Das System kümmert sich weitgehend darum, daß jeder Student nur Änderungen an dem Teilprojekt vornehmen kann, das ihm zugeteilt wurde. Die üblichen Mechanismen zur Kommunikation – etwa E-Mail und Chat – können und sollen selbstverständlich zusätzlich benutzt werden, um die Arbeit zu kooordinieren.

6.1.6 Validierung des Systems durch Verhaltensanalyse

KOLIBRI unterstützt prinzipiell die Validierung des Systems durch Verhaltensanalyse, jedoch sind die entsprechenden Auswertungsmechanismen derzeit nicht implementiert. Das System ist in der Lage, den Autor jedes Kurses, Kapitels oder Bausteins zu speichern. Damit ist es eine triviale Aufgabe, Verweise in erzeugte HTML-Seiten einzubetten, über die ein Student unmittelbar Kritik oder Fragen absenden kann. Außerdem speichert das System über die Historie den Weg, den ein Student durch den Kurs genommen hat. Es ist möglich, die auf diesem Weg gewonnenen Daten auszuwerten und daraus Rückschlüsse – etwa auf unverständliche Teile des Kurses – zu ziehen.

6.1.7 Persönliche Mappe/Arbeitsbereiche des Lernenden

KOLIBRI unterstützt – wie gefordert – einen eigenen Arbeitsbereich für jeden Studenten, auch wenn dieser für ihn nicht unmittelbar als Mappe oder ähnliches sichtbar ist.

Sämtliche Notizen des Studenten zu Teilen des Kurses sowie seine Lösungen zu Aufgaben werden in der Datenbank gespeichert. Betritt ein Student eine Seite, die er bereits bearbeitet hat, so findet er die Seite stets in dem Zustand vor, in dem er sie verlassen hat. Insofern ist der gesamte Kurs der Arbeitsbereich des Studenten, ähnlich einem Buch, an dessen Rand man eigene Bemerkungen notieren kann.

6.2 Mögliche Erweiterungen

Auch wenn das von der Projektgruppe entwickelte System die geforderte Funktionalität bereitstellt und einen Beispielkurs zu Demonstrationszwecken enthält, so muß man ehrlicherweise doch zugeben, daß es eher den Status eines Prototyps als den eines verkaufsfähigen Produkts besitzt – nichts anderes sollte das Ziel der Projektgruppe sein. Die folgende Liste zeigt einige Bereiche auf, in denen die Entwicklung von KOLIBRI weitergetrieben werden könnte, um letztlich ein leistungsfähiges und professionell einsetzbares System zu gewinnen.

- Einige Teile von KOLIBRI, die nicht zur Kernfunktionalität gehören, sind derzeit nur rudimentär implementiert. Dazu zählt zum Beispiel die Auswertung protokollierter Daten und die anschließende Erstellung von Statistiken zur Validierung des Systems. Auch eine kontextsensitive Hilfefunktion würde sowohl Studenten als auch Kursleitern den Umgang mit KOLIBRI erleichtern.

- Der Entwurf von Kursen ist derzeit eine von viel Handarbeit geprägte Tätigkeit. Die notwendige Kenntnis von Details zu KOLIBRI-Kursdateien und zu HTML lenkt den Autor eines Kurses zudem von seiner eigentlichen Aufgabe ab: dem Erstellen eines Kurses. Es wäre sicherlich wünschenswert, über ein komfortables grafisches Autorenwerkzeug zu verfügen, das auf der Schnittstelle aufsetzt, die von *KursAdmin* bereitgestellt wird.

- KOLIBRI-Kurse besitzen eine Grundstruktur, die einem Buch ähnlich ist. Daher sollte es mit geringem Aufwand möglich sein, bestehendes, in digitaler Form vorliegendes Material zu einem Online-Kurs umzuarbeiten. Es wäre ohne Zweifel eine große Hilfe für den Autor eines solchen Kurses, wenn er Werkzeuge zur Verfügung hätte, die eine automatische Konvertierung – etwa von *Word*- oder LaTeX-Dateien – unterstützen würden.

- Die Geschwindigkeit des Systems ist wesentlich von den zahlreichen Datenbankzugriffen und der Übertragung großer Datenmengen über das Internet geprägt. Während dies für einen Prototypen kein bedenkliches Problem darstellt, dürften lange Antwortzeiten im realen Einsatz von einem Studenten eher als lästig empfunden werden. Es wäre sicherlich sinnvoll, sowohl die Geschwindigkeit als auch die Stabilität des Systems in einigen Teilbereichen zu optimieren.

- Es wäre zu überlegen, ob nicht die Fähigkeiten von KOLIBRI in den Bereichen Multimedia oder Kommunikation ausgeweitet werden könnten. Angesichts wachsender Netzbandbreite und der Verfügbarkeit kleiner digitaler Kameras für wenige hundert Mark könnte man etwa die Möglichkeit von Videokonferenzen in das System integrieren. Dies könnte etwa als optionale Erweiterung des Chat-Bereiches betrachtet werden. Neben dem praktischen Nutzen bei kooperativen Aufgaben würde es die soziale Komponente des Systems verstärken, da mit einem anderen Studenten dann nicht nur ein Name und eine Matrikelnummer verbunden wäre, sondern auch ein Gesicht.

Die Projektgruppe wünscht sich natürlich, daß ihre Arbeit nicht umsonst war, und hofft, daß das System KOLIBRI insgesamt oder spezielle Teile davon weiterentwickelt werden.

6.3 Schlußwort

Das eine Jahr, in dem die Projektgruppe KOLIBRI gearbeitet hat, war sicherlich mit viel Arbeit, aber auch mit viel Spaß verbunden – wir hoffen, daß man beides dem entstandenen Prototyp anmerkt. Die Zeit war zudem von einigen interessanten Erfahrungen in bezug auf die Arbeit in einem größeren Team geprägt. Die Projektgruppe hat sich also ohne Zweifel gelohnt.

Ein besonderes Dankeschön an dieser Stelle an unsere Betreuer, die zwar darauf geachtet haben, daß die Arbeit vorankommt, uns aber sonst viel Freiraum zur Entwicklung und Umsetzung eigener Ideen gelassen haben.

A Installation

A.1 WWW-Server Jigsaw

Die Projektgruppe verwendete die Version 1.0 Beta 2 des WWW-Servers Jigsaw. Die Installation erfolgt unter dem Account des Benutzers `westbomk` auf dem Rechner `delta.informatik.uni-dortmund.de`. Die Programm- und Konfigurationsdateien werden getrennt, um eine nachträgliche Installation einer neueren Version zu erleichtern.

Zuerst wird die gesamte Distribution in `Jigsaw-1.0beta2` entpackt. Die Verzeichnisse `config` und `configadm` sowie die beiden Installationsprogramme `Install.*` werden nach `~/wwwserver` verschoben.

Es sind einige zusätzliche Bibliotheken nötig, die mit in den `CLASSPATH` aufgenommen werden müssen:

- **Java Servlet Development Kit (JSDK):**
  `/home/pg301/westbomk/lib/JSDK1.0.1/lib/classes.zip`
- **Java Database Connection (JDBC):**
  `/home/pg301/share/classes/imaginary.zip`
- **Java Mail API:**
  `/home/pg301/westbomk/lib/javamail-0.1/mail.jar`
  `/home/pg301/westbomk/lib/javamail-0.1/activation.jar`
- **Jigsaw:**
  `/home/pg301/westbomk/Jigsaw-1.0b2/Jigsaw/classes/jigsaw.zip`

Genauere Installationshinweise sind den entsprechenden Dokumentationen der einzelnen Pakete zu entnehmen.

Dann wird das Installationsprogramm `Install.class` aufgerufen. Dieses paßt die Ressourcen des Jigsaw an die gegebene Pfadstruktur an.

Mit Hilfe des Administrationstools `jigadm` wird der WWW-Server nun folgendermaßen eingerichtet:

Die *Portnummer* ist 12345.

Die *Realms pg301* (für Projektgruppen-interne Seiten), *Root* (als Zugriffsschutz für das Servlet *Kursleiterbuch*) und *admin* (für den Administrationsbereich des WWW-Servers) werden eingerichtet.

Die *Servlets* befinden sich im Verzeichnis `~/wwwserver/WWW/servlets`. Dabei werden in diesem Verzeichnis nur Links auf den jeweiligen Java-Bytecode in `/home/pg301/share/classes` angelegt. Die Einrichtung der Servlets für den Server erfolgt über das `jigadm`-Tool: Die Ressource für das `servlets`-Verzeichnis muß zuerst gelöscht, und dann als `ServletDirectory`-Ressource wieder angelegt werden. Unterhalb dieses Verzeichnisses werden nun die Ressourcen (`ServletWrapper`) für die Servlets angelegt. Es gibt folgende Servlets:

- `Klassenbuch` (M)
- `Kursinfo` (M)
- `KursleiterAnmeldung`
- `Kursleiterbuch` (R)
- `NeueKursleiterBearbeiten` (R)
- `NeueStudentenBearbeiten` (M)
- `NewsInit` (M)
- `SeitenKomponist` (M)
- `StudentenAnmeldung`
- `Willkommen`

Wichtig ist, in den Attributen jedes Servlets noch das Feld *servlet-class* auszufüllen.

Die mit (M) gekennzeichneten Servlets werden durch den Filter `MsqlAuthFilter` geschützt und die mit (R) gekennzeichneten durch den Realm *Root*.

Die *Applets* befinden sich im Verzeichnis `~/wwwserver/WWW/applets`. In diesem Verzeichnis existiert ein Link auf `/home/pg301/share/classes/kolibri`, so daß ein Applet durch `code=kolibri.xyz.AppletName` im Applet-Tag des HTML-Quelltextes angegeben werden kann. Als `codebase` muß dann nur `/applets/` eingetragen werden.

Zum vereinfachten *Starten* und *Stoppen* von Jigsaw befinden sich unter `~/wwwserver/bin` die Skripten `jigsaw` und `jigkill`. Vor dem Aufruf von `jigkill` sollte aber nach Möglichkeit der Server erst ordnungsgemäß heruntergefahren werden.

A.2 Datenbank

A.2.1 mSQL

Es folgt eine Beschreibung des Installationsvorgangs des mSQL-Servers.

Zunächst muß die Datei `msql-2.0.3.tar.gz` in ein leeres Verzeichnis kopiert werden (im weiteren `/home/pg301/share/dbinstall` genannt).

```
md /home/pg301/share/dbinstall
cd /home/pg301/share/software
cp msql-2.0.3.tar.gz ../dbinstall
```

Nun muß diese Datei in diesem Verzeichnis ausgepackt werden:

```
cd /home/pg301/share/dbinstall
gunzip msql-2.0.3.tar.gz
tar -xf msql-2.0.3.tar
```

Nun in das erzeugte Verzeichnis wechseln:

```
cd msql-2.0.3
```

und

```
make target
```

aufrufen.

In das erzeugte Verzeichnis wechseln (plattformabhängig):

```
cd targets/Solaris-2.5.1-Sparc
```

Spätestens hier den GNU-C-Compiler zugänglich machen:

```
module add gcc
```

Jetzt das Setup-Programm von mSQL aufrufen:

```
./setup
```

In der Datei `site.mm` mit einem Editor die Zeile mit dem Eintrag `INST_DIR` anpassen (z. B. `INST_DIR = /home/pg301/werbeck/dbserver`).

Nun die Kompilierung starten:

```
make all
make install
```

In das Verzeichnis wechseln, das unter `INST_DIR` angegeben wurde:

```
cd /home/pg301/werbeck/dbserver
```

Die Datei `msql.conf` editieren und die folgenden Zeilen anpassen (hier beispielhaft):

```
mSQL-User = werbeck
Admin_User = werbeck
TCP_Port = 12347
```

Die angegebene Portnummer muß in der Klasse `kolibri.CONST` angepaßt werden.

Nun kann der mSQL-Server gestartet werden:

```
cd bin
./msql2d
```

Als nächstes die KOLIBRI-Datenbank erzeugen mit:

```
./msqladmin create kolibri
```

Und zum Schluß die Tabellenstrukturen in diese Datenbank importieren:

```
./msql kolibri < /home/pg301/share/software/makeTables
```

Die Datenbank ist nun komplett eingerichtet und kann durch den *KursAdmin* mit entsprechenden Einträgen gefüllt werden (siehe Abschnitt 5.3).

A.2.2 JDBC

Es folgt eine Beschreibung des Installationsvorgangs der JDBC-Treiber.

Zunächst muß die Datei `mSQL-JDBC.tar` in ein leeres Verzeichnis kopiert werden (im weiteren `/home/pg301/share/jdbcinstall` genannt).

```
cd /home/pg301/share
md jdbcinstall
cp software/mSQL-JDBC.tar jdbcinstall
```

Danach folgt das Auspacken der Datei:

```
cd jdbcinstall
tar -xf mSQL-JDBC.tar
```

Nun wechselt man in das erzeugte Verzeichnis

```
cd mSQL-JDBC_1.0b3
```

und kopiert die Datei `imaginary.zip` in ein Verzeichnis innerhalb des `CLASSPATH`. Die Umgebungsvariable `CLASSPATH` kann mit:

```
echo $CLASSPATH
```

angezeigt werden. Der Pfad `/home/pg301/share/classes` ist im `CLASSPATH` enthalten. Daher kann die Datei beispielsweise dorthin kopiert werden mit:

```
cp imaginary.zip /home/pg301/share/classes
```

Nun ist der JDBC-Treiber für KOLIBRI zugänglich.

A.3 Java-Bytecode

Die zu Bytecode compilierten Java-Klassen werden im Verzeichnis `/home/pg301/share/classes` abgelegt. Der Server greift immer auf dieses Verzeichnis zu und ist somit direkt nach dem Ablegen der Files in diesem Verzeichnis auf dem neuesten Stand.

B Beispiel einer Kursdatei

Dieser Abschnitt zeigt die Kursdatei des implementierten Fuzzy-Kurses. Aufgrund des Umfangs wurde die Datei an vielen Stellen gekürzt, was durch `[...]` kenntlich gemacht wurde. Zur besseren Lesbarkeit wurden außerdem sehr lange Zeichenketten auf mehrere Zeilen umgebrochen, was vom Hilfsprogramm *KursAdmin* eigentlich nicht akzeptiert wird.

```
COURSE fuzzy

  TITLE  "Einf&uuml;hrung in die Fuzzy-Logik"
  AUTHOR "PG 301"

  BASE   "/kurs/"

  RATING wissen
    TITLE "Wissensstand"

    GROUP niedrig "Anfänger"           0  10
    GROUP mittel  "Fortgeschrittener"  10 10
    GROUP hoch    "Experte"            10  0
  END

  RATING theorie
    TITLE "Theoretisch interessiert"

    GROUP nontheorie "Nein"  0 5
    GROUP theorie    "Ja"    5 0
  END

  RATING praxis
    TITLE "Praktisch interessiert"

    GROUP nonpraxis "Nein"  0 5
    GROUP praxis    "Ja"    5 0
  END
```

```
LESSON einleitung

  TITLE "Einleitung"

  REQUIRED ALL

  IMAGE
    FILE "http://ls1-www.informatik.uni-dortmund.de/
          Bilder/LS1-Logo.gif"
  END

  TEXT
    FILE "einleitung/willkommen.html"
  END

  IMAGE
    FILE "http://ls1-www.informatik.uni-dortmund.de/
          Bilder/line_col.gif"
  END

  LINK
    TITLE "In Kapitel 1 erhalten Sie einen kurzen Überblick
           über die Fuzzy-Logik. Des weiteren wird motiviert,
           warum es sich überhaupt lohnt, sich mit Fuzzy-Logik
           zu beschäftigen."

    FILE  "probleme"
  END

  LINK
    TITLE "In Kapitel 2 erwerben Sie Grundlagenwissen zu
           unscharfen Mengen (allgemeiner Stoff)."

    FILE  "mengen"
  END

  LINK
    TITLE "In Kapitel 3 erfahren Sie mehr über approximatives
           Schließen (theoretischer Stoff)."

    FILE  "schliessen"
  END
```

```
LINK
  TITLE "In Kapitel 4 finden Sie Informationen zu unscharfer
         Regelung (praktischer Stoff)."

  FILE  "regelung"
END

LINK
  TITLE "In Kapitel 5 könnnen Sie zusammen mit einem anderen
         Studenten einen eigenen Fuzzy-Controller entwerfen."

  FILE  "entwurf"
END

LINK
  TITLE "Auf dieser Seite können Sie sich über das Kolibri-
         Projekt informieren."

  FILE  "http://delta.informatik.uni-dortmund.de:12345/"
END

LESSON
  TITLE   "Quellen"

  USELESS ALL

  TEXT
    FILE "einleitung/quellen.html"
  END

  LINK
    TITLE "Der Kursinhalt selbst stammt aus dem Skript
           <I>Einführung in die Fuzzy-Logik</I> von
           Prof. Dr. Helmut Thiele."

    FILE  "http://ls1-www.informatik.uni-dortmund.de/
           Thiele.html"
  END

  LINK
    TITLE "Ein Teil der Aufgaben wurde uns von Sascha
           Dierkes zur Verfügung gestellt."

    FILE  "http://ls1-www.informatik.uni-dortmund.de/
```

```
            ~dierkes"
    END

    LINK
      TITLE "Das Video über Prof. Zadeh hat der Mac von
             Jörg Westbomke für uns digitalisiert."

      FILE  "http://ls1-www.informatik.uni-dortmund.de/~westbomk"
    END

    LINK
      TITLE "Die schicken Grafiken zu t- und s-Normen
             wurden von Lars Hildebrand erstellt."

      FILE  "mailto:hildebra@ls1.informatik.uni-dortmund.de"
    END
  END

  [...]

END

LESSON mengen

  TITLE "Unscharfe Mengen und Operationen"

  REQUIRED ALL

  TEXT
    FILE "mengen/mengen.html"
  END

  [...]

  LESSON

    TITLE "Standard-Operationen mit Fuzzy-Mengen"

    TEXT
      FILE "mengen/fuzzyops.operationen.html"
    END

    TEXT
      FILE "mengen/fuzzyops.merke.html"
```

```
  END

  TEXT
    FILE "mengen/fuzzyops.definition.html"
  END

  TEXT
    FILE "mengen/fuzzyops.anschaulich.html"
  END

  TEXT
    FILE "mengen/fuzzyops.grauwert.html"
  END

  TEXT
    TITLE   "Hier sehen Sie eine Veranschaulichung der
             drei Standard-Operationen."

    HIDDEN  ALL

    VISIBLE praxis

    FILE    "mengen/fuzzyops.grafiken.html"
  END

END

EXERCISE

  TITLE "Lernkontrolle"

  TEXT
    FILE "aufgaben/einleitung.html"
  END

  PART

    TITLE    "Frage zur Mengenlehre"

    SCORE    10 wissen

    EVALUATE AUTO

    FORM
```

```
        FILE "aufgaben/schoepfer.frage.html"
      END

      HINT TEXT
        FILE "aufgaben/schoepfer.hinweis.html"
      END

      RIGHT WRONG TEXT
        FILE "aufgaben/schoepfer.antwort.html"
      END

    END

    PART

      TITLE    "Frage zu scharfen Mengen"

      SCORE    10 wissen

      EVALUATE AUTO

      NORMAL HINT FORM
        FILE "aufgaben/scharf.frage.html"
      END

      RIGHT WRONG FORM
        FILE "aufgaben/scharf.antwort.html"
      END

      WRONG LINK
        TITLE "Da Sie diese Aufgabe nicht korrekt gelöst
               haben, sollten Sie vielleicht das Kapitel
               über scharfe Mengen erneut durcharbeiten."

        FILE  "scharf"
      END

    END

  END

  [...]

END
```

```
[...]

PROJECT entwurf

  TITLE "Entwurf eines unscharfen Reglers"

  USELESS  ALL

  OPTIONAL mittel

  REQUIRED hoch

  TEXT
    FILE "entwurf/einleitung.html"
  END

  PUBLIC PART vereinbarung

    TITLE "Vereinbaren gemeinsamer Parameter"

    TEXT
      FILE "entwurf/vereinbarung.html"
    END

    APPLET
      FILE   "/applets/kolibri.fuzzy
              .CoopExamFuzzyArrangeApplet.class"

      WIDTH  550

      HEIGHT 1600

      PARAM  bgcolor "000000"
    END

  END

  PRIVATE PART variablen

    TITLE "Bearbeiten der linguistischen Variablen"

    SCORE 10
```

```
      APPLET
        FILE   "/applets/kolibri.fuzzy.FuzzyEditor.class"

        WIDTH  500

        HEIGHT 400

        PARAM  AUFGABENTEIL "LV"
      END

    END

    PRIVATE PART regeln

      TITLE "Bearbeiten der Regelbasis"

      SCORE 10

      APPLET
        FILE   "/applets/kolibri.fuzzy.FuzzyEditor.class"

        WIDTH  500

        HEIGHT 400

        PARAM  AUFGABENTEIL "RB"
      END

    END

    PUBLIC PART simulation
      TITLE "Simulation"
    END

  END

END
```

Literaturverzeichnis

[Bahn97a] Bahnes, Tim et. al.: *Projektgruppe 301 – Kolibri – Zwischenbericht*, Interner Bericht des Fachbereichs Informatik der Universität Dortmund (1997)

[Bahn97b] Bahnes, Tim et. al.: *Projektgruppe 301 – Kolibri – Konzept*, Interner Bericht der Projektgruppe 301 (1997)

[Bern95] Berners-Lee, Tim et. al: *Hypertext Markup Language (HTML) 2.0*, Internet Diskussionspapier (1995), im Internet unter `ftp://nic.mil/rfc/rfc1866.txt` (zuletzt gesichtet am 16.03.1998)

[Drak96] Drakos, Nikos: *The LaTeX2HTML Translator 98.1*, im Internet unter `http://www-dsed.llnl.gov/files/programs/unix/latex2html/manual/` (zuletzt gesichtet am 16.03.1998)

[Fiel97] Fielding, R. et. al.: *Hypertext Transfer Protocol (HTTP) 1.1*, Internet Diskussionspapier (1997), im Internet unter `ftp://nic.mil/rfc/rfc2068.txt` (zuletzt gesichtet am 16.03.1998)

[Hugh97] Hughes, David: *Mini SQL (mSQL) 2.0.3*, im Internet unter `http://www.hughes.com.au/` (zuletzt gesichtet am 16.03.1998)

[Imag97] The Center for Imaginary Environments: *The Imaginary JDBC Driver for mSQL*, im Internet unter `http://www.imaginary.com/Java/` (zuletzt gesichtet am 16.03.1999)

[Sun96] Sun Microsystems: *The Java Language Specification* (1996), im Internet unter `http://www.javasoft.com/docs/books/jls/html/` (zuletzt gesichtet am 16.03.1998)

[Sun97] Sun Microsystems: *Java Development Kit (JDK) 1.1.5 Documentation* (1997), im Internet unter `http://java.sun.com/products/jdk/1.1/docs/` (zuletzt gesichtet am 16.03.1998)

[Thie96] Thiele, Helmut: *Einführung in die Fuzzy-Logik*, Spezialvorlesung am Fachbereich Informatik der Universität Dortmund (1996)

[W3C98] World Wide Web Consortium: *Jigsaw Overview*, im Internet unter `http://www.w3.org/jigsaw/` (zuletzt gesichtet am 16.03.1998)