UNIVERSITÄT BREMEN
Fachbereich 3 - Studiengang Informatik

Diplomarbeit

Entwicklung einer Client/Server-Anwendung zur Generierung und Auswertung von Online-Prüfungsbögen

AUTOR

Martin Blum
(Mat.-Nr. 1043820, blum@tzi.de)

GUTACHTER

Prof. Dr. Herbert Kubicek, Dr. Berthold Hoffmann

E R K L Ä R U N G

Ich versichere, die Diplomarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche gekennzeichnet.

Bremen, den 20. Dezember 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Der Computer zur Vermittlung von Wissen wird bereits seit geraumer Zeit genutzt. Ein Lerner war bis jetzt dabei jedoch auf sich selbst gestellt und konnte sich im Gegensatz zu einem Kursteilnehmer eines realen Kurses nicht mit anderen austauschen. Durch die rasante Verbreitung des Internets im kommerziellen wie auch im privaten Bereich wurde durch diese Technologie der ideale Mechanismus zur Kommunikation innerhalb solcher Gruppen geschaffen. Ähnlich dem realen Leben können über das Internet Nachrichten direkt oder zeitverzögert ausgetauscht werden, und ebenso ist es möglich, dass Informationen einmal abgelegt werden und dann allen Teilnehmern zur Verfügung stehen. Daher wird im Moment verstärkt darauf hingearbeitet, Kursinhalte online zu stellen und die reine Vermittlung von Wissen um die Aspekte der Kommunikation und Interaktivität zu erweitern. So geht beispielsweise die Volkshochschule Bremen nach eigenen Angaben davon aus, dass für das Jahr 2002 das eLearning-Angebot durch 30% der Teilnehmer (gegenüber 1% in 2001) genutzt wird.

Darüber hinaus gibt die Internettechnologie auch ungeschulten Personen die Möglichkeit, über leicht zu bedienende Oberflächen rechen- oder speicherplatzintensive Arbeitsabläufe von beliebigen Rechnern aus über einen leistungsfähigen Server zu erledigen, um so den Client-PC zu entlasten. Somit kann sowohl Tutoren eine leichte Möglichkeit der Kursgestaltung, als auch räumlich weiter entfernten Lernern ein Mehrwert zum klassischen Kurs gegeben werden.

1.1 Ziel der Arbeit

Der Grundgedanke für diese Arbeit beruht auf der Idee, mittels der T3C-Methode (*Test-Train-Test-Certify*) einen effektiven Lernprozess zu unterstützen. Grundsätzlich geht es in dieser Methode darum, dass die Abfrage von Wissen einem Lerner auf mehrere Arten Nutzen bringen kann. Der Test kann verwendet werden als Eingangstest, als Verifizierungstest, zum Zertifikatserwerb oder zum Trainieren neu angeeigneter Fähigkeiten. Diese Methode wird in Kapitel 2 Grundlagen näher erläutert.

Momentan existierende Softwareprodukte ermöglichen bis jetzt nur eine Generierung von Single- oder Multiple-Choice-Tests. Fast alle Prüfungen bestehen aber nur zu einem Teil aus dieser Form von Fragen. Die Lernwirkung ist somit eher gering, da nicht die tatsächliche Prüfungssituation

vermittelt wird. Sehr oft muss zur Beantwortung der Frage ein freier Text eingegeben werden und der Prüfer entscheidet darüber, ob die Antwort richtig oder falsch ist. Bei einem Multiple-Choice-Test ist die richtige Lösung mit in der Lösungsmenge enthalten, d.h. (im Gegensatz zur richtigen Prüfung) kann der Prüfling seine Antwort notfalls einfach raten. Von der Autorenseite aus gesehen kommt dazu, dass für die Gestaltung der Seiten oft noch HTML-Kenntnisse notwendig sind. Dies sollte nicht als Voraussetzung für die Anwendung der Software gesehen werden.

Ich werde im Rahmen dieser Diplomarbeit ein Softwarepaket erstellen, das einem Tutor ermöglicht, über einen Webbrowser Übungsaufgaben als Ergänzung zu einem Online-Kurs zu erstellen. Er legt den Typ einer Aufgabe fest, erfasst ihren Inhalt und die Lösung und gruppiert sie entsprechend des Themengebiets und der Schwierigkeit. Für die Zertifikatsprüfung legt er fest, in welcher Reihenfolge die Aufgaben absolviert werden müssen. Das kann beispielsweise die möglichst wirklichkeitsgetreue Nachbildung von Prüfungsbögen einer Abschlussprüfung sein. Ein Kursteilnehmer kann die Software dann anwenden, um beispielsweise seinen Wissensstand zu überprüfen, Gelerntes zu wiederholen oder um sich auf eine Prüfung vorzubereiten.

Diese Diplomarbeit ist in Zusammenarbeit mit einem Projekt der Bremer Volkshochschule entstanden, das die Volkshochschulen in ganz Deutschland virtuell unter dem Pseudonym „vhs-virtuell“ vereint und eine Plattform aufgebaut hat, auf der Online-Kurse eingestellt und abgerufen werden können. Ich konnte die vorhandene Infrastruktur nutzen und die Mitarbeiter haben mich mit Tipps und Anregungen unterstützt. Meine Arbeit integriert sich vorerst in den Kurs „Sportbootführerschein See“. Dazu wurden über die erstellte Software zusammen mit einem Tutor die amtlichen Prüfungsbögen erfasst und lernbegleitend eingesetzt.

Um die gesetzten Ziele überprüfen zu können, schloss sich der praktischen Arbeit eine Evaluationsphase an, in der über Fragebögen und Interviews die Akzeptanz der Software diskutiert wurde. Mittlerweile ist sie unter der Adresse `"http://www2.vhs-virtuell.de/gaon/"` für alle (berechtigten) Benutzer zugänglich, die Übungsaufgaben in ihren Kurs einfügen möchten. Zu Testzwecken existiert die Benutzerkennung "`diplom`" mit dem Kennwort "`diplom`".

1.2 Kapitelübersicht

Die Arbeit gliedert sich in 8 Kapitel. Dies ist als *Einleitung* das erste Kapitel, um Motivation und Ziel der Arbeit darzustellen. Das zweite Kapitel *Grundlagen* behandelt lerntheoretische Grundlagen und Vorüberlegungen zum Einsatz von Online-Prüfungen. Kapitel drei vergleicht und bewertet *bestehende Softwareprodukte* und versucht die gefunden Schwächen zusammenzufassen. Im vierten Kapitel *Systemanforderungen* werden die Anforderungen an ein solches Softwaresystem spezifiziert. Der *Lösungsansatz* wird im fünften Kapitel dargestellt, in dem die zentralen Algorithmen näher erläutert werden. Im sechsten Kapitel *Implementierung* gehe ich auf die konkrete Realisierung des Software ein. Kapitel sieben ist eine *Evaluation* des erstellten Produktes anhand der im dritten Kapitel vorgestellten Kriterien und der Bewertung durch die Anwender des Systems. Am Ende schließt das achte Kapitel mit einer *Zusammenfassung und einem Ausblick* auf mögliche Verbesserungen des Systems. Die Anhänge umfassen eine *kurze Einführung in die Sprache XML*, die entworfenen *Fragebögen* mit den erhobenen Daten, die *DTD* für die Maskendefinition, das vollständige *Datenbankdesign*, den *Java Source-Code* ausgewählter Funktionen, das *Literaturverzeichnis* und ein *Glossar*.

Kapitel 2

Grundlagen

Bevor ich mit den Anforderungen an die Software beginne, möchte ich noch kurz einige Vorüberlegungen anstellen, die den Zugang zum Thema Online-Lernen und Online-Prüfungen erleichtern sollen. Gerade sie decken auf, warum der Bedarf solcher Lösungen erst heute eine immense Steigerung erfährt.

2.1 Grundlagen zur Lerntheorie

Zur Theorie des Lernens am Computer existieren mehrere didaktische Modelle, von denen ich hier die wesentlichen (nach [Baumgartner 1994a]) kurz anreißen möchte. Diese didaktischen Modelle sollen verdeutlichen, auf welche Weise an die Erzeugung von Wissen beim Lernenden herangegangen werden kann und wie sich daraus ein entsprechender Lernprozess ableitet. Grundsätzlich unterscheiden sich die Modelle im wesentlichen durch den Freiheitsgrad, der dem Lernenden gegeben wird, d.h. wie viel er auf seinem Weg selbst bestimmen darf und wie viel ihm durch eine übergeordnete Stelle vorgeschrieben wird. Erhält er die Lehrmaterialien z.B. bereits vorstrukturiert oder kann er sich aus einer Fülle von Materialien, die für ihn interessantesten selbst auswählen. Da sich die Modelle grundlegend in der Art unterscheiden, wie sie beim Lernenden das Wissen erzeugen, ist auch die jeweilige Art unterschiedlich, wie das Verständnis des Lehrstoffes bei ihm überprüft wird.

Behaviorismus

Etwa 1920 begann die Blütezeit des *Behaviorismus* [Flechsig 1996]. Behavioristen betrachten das Denken bzw. das menschliche Gehirn als eine „Black Box“, die sich das Wissen aneignet und speichert. Der Vorgang des Lernens ist hierbei das korrekte Ablegen einer „Reiz-Reaktions-Kette“, das heißt auf einen bestimmten Input (oder Reiz) gibt es einen (einzigen) richtigen Output (eine Reaktion). Wissen wird als eine korrekte Ein-/Ausgabe-Relation gesehen, die am Besten durch einen starr vorgegebenen Programmablauf eines autoritären Lehrers vermittelt wird („drill and practice“). Innerhalb dieses Denkmodells sollen die Lernenden möglichst viel Wissen auswendig lernen, um es dann im Bedarfsfall abrufen und anwenden zu können. Eine tiefergehende Motivation zum Zeitpunkt des Lernens ist dabei im Allgemeinen nicht vorhanden, sondern es wird vielmehr aufgrund der Autorität des Lehrers angenommen, dass das vermittelte und auswendig zu lernende

Wissen in irgendeiner Form zu einem späteren Zeitpunkt relevant sein wird. Das Wissen wird dabei abgelegt in der Form "Die Antwort auf Problem a ist die Tatsache b". Hintergründe, Konsequenzen oder komplexe Strukturen werden somit meistens nicht vermittelt, sondern höchstens nur als eine weitere Input-Output-Kette, die nicht oder nur sekundär mit dem ursprünglichen Problem verknüpft ist. Aus dieser Art zu Lernen sind die auch heute noch häufig verwendeten Prüfungen über Fragebögen entstanden, bei denen auf eine größere Anzahl von Fragen im Allgemeinen eine relativ kurze Antwort gegeben wird und durch die sehr einfach die beim Lerner gespeicherten Informationen überprüft werden können. Multiple- oder Single-Choice-Tests haben zusätzlich den großen Vorteil, dass sie sehr schnell auswertbar sind.

Kognitivismus

Ca. 1960 wurde das Paradigma des Behaviorismus durch die Theorie des *Kognitivismus* ergänzt. Kognitivisten sehen das Denken als einen informationsverarbeitenden Prozess, bei dem Wissen verarbeitet und gespeichert wird. Lernen ist hier der Aufbau kognitiver Strukturen, d.h. einen passenden internen Verarbeitungsprozesses zu finden, mit dem eine Aufgabe gelöst werden kann. Dies kann nicht durch einen starren Programmablauf erreicht werden, da mehrere Lösungen oder Lösungswege richtig sein können. Vielmehr soll ein Tutor den Lerner beobachten und helfend eingreifen, um ihn bei der Zielfindung zu unterstützen.

Im Kognitivismus muss auch Basiswissen über ein Problem vorhanden sein (das vorher erst auswendig gelernt wurde). Es wird dem Menschen aber bereits zugestanden, dass die Lösung eines Problems nur dadurch erreicht werden kann, indem dieses Wissen auf unterschiedliche Art und Weise miteinander verknüpft wird. So entstehen durchaus mehrere Lösungswege, die aber trotzdem zum richtigen Ziel führen. Dies lässt den Kapazitäten und Fähigkeiten eines Menschen bereits einen wesentlich größeren Spielraum, so dass hier eine Motivation durch den Ehrgeiz gegeben ist, ein bestimmtes Problem selbständig zu lösen. Bei Prüfungen innerhalb dieser Methode werden zu einer Aufgabenstellung nicht nur die Ergebnisse gefordert, sondern es muss für den Prüfer auch der Lösungsweg erkennbar sein. So kann dann noch ein Teil der Punkte gegeben werden, wenn der Ansatz zwar richtig war, aber dem Prüfling während der Lösung dieser Aufgabe ein Fehler unterlaufen ist.

Konstruktivismus

Seit 1990 wiederum wird das Modell des *Konstruktivismus* diskutiert, das in großem Maß die Entwicklung und Gestaltung neuer Bildungsmedien beeinflusst hat [Bruns 2000]. Für Konstruktivisten ist das menschliche Gehirn bzw. das Denken ein geschlossenes Informationssystem, in dem das Wissen vom Lernenden aktiv in komplexen, realen Lebenssituationen konstruiert und dann gespeichert wird. Somit wird Lernen zum Erwerb von Erfahrungen und dient dazu, mit einer konkreten Situation umgehen zu können. Erreicht wird der Wissensaufbau durch die Konstruktion einer Lösung - z.B. innerhalb einer Simulation oder einer Mikrowelt - in der der Lerner den Programmablauf selbst bestimmt und der Computer nur als kooperierender Berater fungiert. Aus diesem Grund dienen konstruktivistische Ansätze als lerntheoretische Begründung für Hypermedia-Systeme. Hier werden die Informationen nicht linear, sondern komplex miteinander verknüpft dargestellt und lassen den Anwender frei in seiner Vorgehensweise agieren. Diese realitätsgetreuere Darstellung von Informationen soll die Authenzität und Situationsbezogenheit des Lernens fördern.

Dem gegenüber steht jedoch das oftmals beobachtete Phänomen des „*lost in Hyperspace*“, da die weitläufig vernetzten Strukturen des Internets den Anwender durch die Flut an Informationen von den wirklich relevanten Daten fernhalten können.

In der Auseinandersetzung mit dem Modell des Konstruktivismus wurden weitere Theorien entwickelt. Sie unterscheiden sich in den grundlegenden Details aber nicht wesentlich, gehen jedoch mehr auf die aktuellen technischen Möglichkeiten ein (z.B. *Instruktionsdesign 2. Grades, Learning Cycle, 5 Cs of Internet Based Group Learning*). Sie spezifizieren z.B. Software näher, die das Arbeiten in Gruppen vereinfacht (sogenannte Groupware) bzw. speziell auf das Internet und dessen rasante Verbreitung zugeschnitten ist.

Prüfungen zur Leistungskontrolle werden innerhalb dieses Modells sehr schwierig, da oftmals durch die Auseinandersetzung mit der gegebenen Situation beim Lerner bereits soviel Wissen und Erfahrung aufgebaut wird, dass dabei gar nicht mehr entscheidend ist, ob das gestellte Problem tatsächlich gelöst wurde.

Zusammenfassung

Welches Paradigma „richtig“ oder „falsch“ ist, lässt sich nicht sagen, da sowohl die Lernenden als auch der Lernbegriff selbst zu vielfältig sind. Man lernt Fakten, körperliche Bewegungsabläufe (z.B. Laufen), die Anwendung von Regeln, man lernt Menschen kennen oder lernt, sich in komplexen Situationen zurechtzufinden. Es existieren also verschiedene Ebenen des Lernens und das außerdem noch auf verschiedenen Stufen: Anfänger, Fortgeschrittene oder Experten haben andere Bedürfnisse an die Lernmaterie und somit auch einen anderen Lernstil. Allerdings lässt sich immerhin noch ein Anhaltspunkt festmachen: Viele Prüfungen zum Erwerb eines Wissenszertifikats beruhen auf dem Abfragen von Wissen anhand von Fragebögen (oft aufgrund der einfachen Auswertbarkeit). Sie folgen also immer noch dem Grundgedanken des Behaviorismus, dass es auf jede Frage (genau) eine richtige Antwort gibt (Führerscheinprüfungen, Tests in Schulen, Schulungen innerhalb von Unternehmen, EDV-Kurse von Erwachsenenbildungsinstituten etc.).

2.2 Online-Tests nach der Methode T3C (*Test-Train-Test-Certify*)

Die Methode *T3C* wird vermehrt als Möglichkeit zur Gestaltung von Online-Tests diskutiert [Enlight 2001]. Diese Methode geht davon aus, dass ein Test-Tool (unabhängig von der Art der Implementierung) auf verschiedene Arten von den Anwendern genutzt wird. Das Testsystem muss also dem Anwender verschiedene Möglichkeiten bereitstellen, damit er alle seine unterschiedlichen Anforderungen an einen Test mit diesem einen Tool abdecken kann.

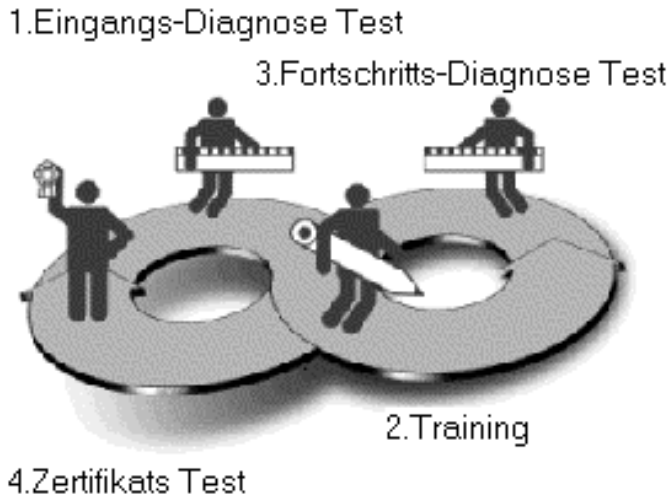

Abbildung 1: Die Methode T3C (Quelle: [Enlight 2001])

Diese unterschiedlichen Arten des Wissenstests können sein:

- Diagnosetest (Eingangstest)
- Verifizierungstest (Überprüfung der Schulungsmethode)
- Fähigkeitstest (Überprüfung der Lernentwicklung und des Lernerfolgs)
- Zertifikatsprüfung
- Einstellungstest
- Assessmenttest

Über einen Diagnosetest sollen eigene Schwachstellen und Wissenslücken erkannt werden, diese dann über Schulungen und Verifizierungstests beseitigt und dann mittels eines Fähigkeitstest der Lernerfolg überprüft werden. Im Anschluss daran kann dann eine Prüfung zum Erwerb eines Zertifikats abgelegt werden.

Eine Testsoftware kann natürlich auch über eine reine Lernumgebung hinaus benutzt werden, um beispielsweise die Fähigkeiten von Bewerbern einzuordnen. Das Testergebnis selber ist in diesem Fall dabei nicht entscheidend, sondern eher die Sortierung der Testkandidaten hinsichtlich gewisser Präferenzen. Assessmenttests werden als ein Hilfsmittel verwendet, um Kosten für Aufgaben oder Stellenbereiche zu analysieren und festzusetzen.

2.3 Warum hat sich das Lernen über Online-Tests bis jetzt noch nicht durchgesetzt

Dass Computer zur Vermittlung von Wissen eindeutige Vorteile besitzen, ist unbestritten. So wird laut dem Bericht der Europäischen Kommission/Eurostat für das Jahr 1999/2000 von sämtlichen europäischen Staaten der Einsatz und die Nutzung von Informations- und Kommunikationstechnologien gefördert und teilweise sogar verpflichtend im Lehrplan des Primarbereichs vorgesehen. Sie

wird dort als Instrument für die Durchführung von Projekten oder zur Vermittlung von Unterrichtsinhalten empfohlen [Eurostat 1999]. Allerdings wird in diesem Bericht ebenfalls angemerkt, dass der Termin für die vollständige Einführung dieser Verfahren häufig erst im Jahr 2000 liegt. Diese Form der Ausbildung wird demnach erst der kommenden Schülergeneration zuteil und somit ist noch vielen Menschen die Nutzung der IKT (Informations- und Kommunikationstechnologien) nicht vertraut. Ebenso kann die Höhe der Investitionen durch den statistischen Bericht nicht beurteilt werden, da die Ausgaben für die Ausstattung nur sehr selten auf zentraler Ebene verwaltet werden. Laut diesem Bericht ist es bemerkenswert, dass zwar der Einsatz und die Beherrschung dieser Instrumente für die Schüler gefördert wird, jedoch die Ausbildung für die Lehrkräfte oft keine Schulung der neuen Technologien vorsieht. Somit ist es den Lehrkräften nicht ohne weiteres möglich, die erforderlichen Kompetenzen zu erwerben, um die Schüler bei der Aneignung und schrittweisen Beherrschung der IKT zu begleiten.

Parallel dazu verändert der Übergang von der Industrie- zur Informations- oder gar Wissenschaftsgesellschaft auch die Arbeitswelt und hat dazu geführt, dass mittlerweile viele Arbeitsplätze mit Informationstechnologien ausgestattet sind. Dabei erfordern neue Softwareversionen und Hardwaregenerationen, ebenso wie moderne Bürokommunikationsmittel und das zunehmende Informationsangebot, eine ständige Anpassung eingespielter Arbeitsabläufe. Das Schlagwort des lebenslangen Lernens wird immer häufiger verwendet. Klassische Weiterbildungsmöglichkeiten vermögen hier kaum noch den Weiterbildungsbedarf zu decken und der Bedarf an berufsbegleitenden Fortbildungen ist unbestritten. Aus diesen Überlegungen heraus wurde bereits Anfang der 80er Jahre in größeren Unternehmen begonnen, technologiebasiertes Lernen oder Training (kurz TBT) als Weiterbildungsmaßnahme zu etablieren. Allerdings ist es erst durch die Verbreitung des Internets in den letzten Jahren zu einer größeren Vernetzung der bestehenden Bildungsangebote gekommen, und die notwendige Aktualität und Informationsbreite des Angebots wurde erreicht. Durch Hypertext- und Hypermediasysteme können die vorhanden Datenquellen neu strukturiert und verbunden werden, um wirksame Informations- und Lernsysteme zu schaffen. So wird z.B. in einem Bericht der Computerwoche [CW 2001] eine mögliche Senkung der betrieblichen Fortbildungskosten für das Jahr 2002 durch den Einsatz web-basierter Lern- und Testumgebungen von etwa 35% prognostiziert.

Ein weiterer Hinderungsgrund beim Einsatz technologiebasierten Lernens war bis vor kurzem der geringe Grad an selbstbestimmten Lernprozessen innerhalb der Lernprogramme. Deren Abläufe sind überwiegend starr vorgegeben und ermöglichen dem Lernenden nicht, selbstorganisiert über Zeit und Ort sowie seinem Qualifizierungsbedarf entsprechend zu lernen. Diese fehlenden Freiheitsgrade innerhalb des Systems schränken die Motivation des Lernenden stark ein, da es ein bestimmtes Verhalten aufzwingt und somit der neugierigen, „explorativen“ Ader des Menschen widerspricht.

Ein letzter Trend, der sich ebenfalls erst durch die globale Vernetzung innerhalb von Lernumgebungen durchsetzen konnte, jedoch zur Steigerung der Lernmotivation und des Lernerfolgs ganz erheblich beiträgt, ist das kommunikative Element, das das Lernen als sozialen Prozess stark prägt [Mandel 1998]. Erst jetzt kann sich eine Kommunikation, gestützt durch eine Lernumgebung der neuesten Generation, nicht nur zwischen Lernendem und Lehrendem entwickeln, sondern zunehmend auch unter Kollegen, unter den Lernenden untereinander oder zu externen Experten.

2.4 Folgerungen für das Projekt "vhs-virtuell"

Für das Projekt „vhs-virtuell“ hat sich aus diesen Details die Notwendigkeit ergeben, dass bestehende Kurse und die damit verbundenen sozialen Gefüge nicht aufgelöst werden dürfen, sondern durch neue Technologien ergänzt werden sollen. Den Kursteilnehmern werden nicht gewohnte und nützliche Aspekte eines realen Kurses genommen, vielmehr erhalten sie durch die Teilnahme an einer Lernplattform einen Mehrwert. Beispielsweise wohnen die Teilnehmer des Kurses „Sportbootführerschein See“ oftmals weit vom Ort des Kurses entfernt. Jetzt sind sie durch die Lernplattform in der Lage, Kursinhalte zu lernen, auch ohne persönlich zu erscheinen. Dabei können sie wie gewohnt auf die Unterstützung durch den Tutor oder durch Mit-Lernende zurückgreifen, die ihnen als reale Personen im Gedächtnis sind und nicht nur als anonyme Login-Kennungen.

Durch die neuen Technologien wird sich jedoch nicht so sehr die Form der Wissenskontrolle ändern, da die Missbrauchs- und „Schummel“-Möglichkeiten an einem Computer noch zu stark vorhanden sind. Momentan kann weder durch Software noch durch Hardware erkannt werden, wer tatsächlich die Eingaben an einem Computer vornimmt. Es werden also weiterhin durch Menschen überwachte Prüfungen stattfinden, um ein Zertifikat über einen Wissensstand zu erwerben. Aufgrund dieser Tatsache sollte während eines Kurses auch die Art der Zertifikatsprüfung vorbereitet werden, um die Stresssituation während einer Prüfung für den Kursteilnehmer möglichst gering zu halten.

2.5 Warum diese Diplomarbeit über Online-Prüfungen?

Im Gespräch mit mehreren Dozenten der Bremer Volkshochschule haben sich einige Ideen herauskristallisiert, die einen traditionellen Kurs durch den Einsatz der Internettechnologien aufwerten würden. Eine Idee dabei war das hier durchgeführte Projekt, da für die Prüfung des "Sportbootführerscheins See" Fragebögen verwendet werden. Bei den Kursteilnehmern wird diese Art zu üben bis jetzt nur dadurch abgedeckt, dass sich jeder Teilnehmer einen Satz Fragebögen kaufen musste. Dazu kommt, dass sie sich bei Fragen nur innerhalb der Präsenzveranstaltungen mit dem Tutor oder den anderen Teilnehmern austauschen können und nur eine manuelle Kontrolle über Lernerfolg und Defizite möglich ist. Darüber hinaus existieren aber noch mannigfache weitere Gründe, die ich hier noch kurz aufführen möchte.

a) Es ist keine Lern-Software vorhanden, die einem Lerner ermöglicht, Fragen aus einem Wissensgebiet zufällig, sortiert nach Themengebiet, nach Prüfungsbogen oder Leistungsdefiziten zu bearbeiten und dabei entweder noch Hilfestellungen zur Aufgabe zu erhalten oder eine Prüfung mit einem Endergebnis zu simulieren.

b) Vorhandene Tools unterstützen nur die Auswertung von vorgegebenen Antwortmöglichkeiten (Multiple-Choice). Allerdings geben vorgegebene Antwortmöglichkeiten bereits einen Hinweis auf die Lösung (da die richtige Lösung auch immer mit vorhanden sein muss), so dass auf amtliche Fragebögen oft ein freier Text verfasst werden muss.

c) Vorhandene Tools unterstützen nur die bekannten HTML-Elemente. Es existieren jedoch auch Fragen, die durch diese Typen nicht abgedeckt werden (zeichnerische Lösungen, Zuordnungsaufgaben etc.) und die sich leichter durch eine interaktive, parametrisierbare Schnittstelle ab-

fragen lassen würden (z.B. Position und Kurs auf einer Karte einzeichnen). Diese Schnittstelle muss so allgemeingültig sein, dass sie alle denkbaren Aufgabenstellungen abfangen kann und ist daher günstigerweise eine Schnittstelle zu einem Java-Applet.

d) Vorhandene Tools haben oft keine Sicherheitsmechanismen gegen „Schummeln“, da HTML- oder JavaScript-Dokumente immer auch vom Bediener im Klartext gelesen werden können. Das ist in einer Prüfungssituation jedoch nicht zulässig.

e) Es ist kein Autorentool vorhanden, das aus Fragebögen HTML-Dokumente generiert. Diese Fragebögen sind nicht einheitlich abgefasst und es liegen noch keine Untersuchungen darüber vor, den Aufbau zu standardisieren (ohne sich dabei vom Inhalt der Frage zu entfernen). Die generierten HTML-Dokumente müssen (möglichst) identisch zu den Vorlagen sein, um den Test so zu üben, wie er in der Realität stattfinden wird.

f) Es existiert noch kein Tool, das die Generierung und Auswertung von Fragebögen zusammenfasst und somit die Belange von Dozenten und Lernern in einer Software vereint.

2.6 Grenzen dieser Diplomarbeit

Eine Diplomarbeit kann natürlich nicht mit einem kommerziellen Produkt konkurrieren, da hier dem Softwarehaus ganz andere (finanzielle und personelle) Mittel zur Verfügung stehen. Die Optik der Oberfläche unterliegt meinen persönlichen grafischen Unzulänglichkeiten und entspricht nicht dem momentanen State-of-the-Art. Des weiteren ist die Interpretation freier Texte nicht trivial und bietet trotz verschiedener Forschungen im Bereich Computer-Linguistik noch ein breites Spektrum an offenen Fragen. Ein Algorithmus zur exakten Interpretation unterliegt den Grenzen des eingesetzten Rechnersystems und eine umfassende Lösung dieser Aufgabenstellung würde wahrscheinlich eine eigene Diplomarbeit ausmachen. Der hier vorgestellte Weg erfüllt allerdings seinen Zweck, da eine Kombination von computergesteuerter Auswertung und menschlicher Nachbewertung eingesetzt wurde.

Kapitel 3

Vergleichende Bewertung bestehender Softwareprodukte

Bei der vergleichenden Bewertung bestehender Test-Tools wurde aus einer Reihe von Anbietern via Internet-Recherche eine Auswahl von 4 Produkten getroffen. Da der Themenbereich noch relativ neu ist, existieren erst sehr wenige Produkte, die dieses spezielle Segment der Lernunterstützung abdecken. Häufig integrieren sich diese Test-Tools dabei in umfangreichere Softwaresysteme, wie z.B. Lernplattformen. Entscheidungskriterien für die nähere Betrachtung waren dabei die gebotene (bzw. angekündigte) Funktionalität, die Position am Markt (Anzahl Installationen, bzw. Referenzen) und die Verfügbarkeit einer kostenlosen Demoversion. Dieses letzte Kriterium hat jedoch nicht zum Ausschluss eines eventuell relevanten Produktes geführt, da sich sämtliche Anbieter bereit erklärt hatten, die Software vor Ort zu präsentieren, wenn eine entsprechende Demoversion nicht verfügbar war. Die genauen Kriterien der Evaluation folgen im nächsten Abschnitt.

Aufgrund der rasanten Veränderung des Softwaremarktes innerhalb dieses Bereiches sind nachträglich noch zwei weitere Lerntools untersucht worden, die im Zeitraum dieser Ausarbeitung veröffentlicht wurden. Sie liegen inhaltlich sehr dicht an dieser Arbeit und sind somit von sehr großem Interesse. Ihre Entwicklung zeigt, dass ein entsprechender Bedarf solcher Software am Markt existiert.

3.1 Vergleichskriterien

Um die verschiedenen Produkte objektiv miteinander vergleichen zu können, habe ich folgenden Kriterienkatalog aufgestellt, der auf die Anforderungen an eine Software zur Generierung und Auswertung von Online-Prüfungsbögen zugeschnitten ist.

überprüftes Kriterium	prozentuale Bewertung
Kommunikationsmechanismen (Chat, Foren, E-Mail)	20 %
Art der Implementierung (Stand-Alone / Applet / Servlet)	10 %
Antwortzeitverhalten	10 %
Verfügbarkeit bzw. Bedienungskomfort eines Autorentools, das keine bzw. wenig HTML-Kenntnisse erfordert	10 %
Bedienungskomfort der Oberfläche für Anwender/Tutor/Autor/Systemverwalter	5 %
verfügbare Aufgabentypen und damit „Realitätsgrad“ der Übungen	30 %
Abhängigkeit vom benutzten System oder benutzter Software	15 %

Erläuterungen zu den Kriterien:

- Jedes Kriterium erhält eine Bewertung von 1 bis 6 (Schulnotensystem) und wird dann entsprechend der angegebenen Bewertung gewichtet.
- Bei der Art der Implementierung wird die Stand-Alone-Variante als die schlechteste angesehen, da hier das Produkt entweder nur auf einem Betriebssystem lauffähig ist oder im Falle von Java vorher eine entsprechende Laufzeitumgebung installiert werden muss. Auch die Applet-Variante erfordert auf der Client-Seite einen gewissen Aufwand (Download des Applets, Integration der JVM in den Browser), so dass hier die Servlet-Variante als die günstigste eingeschätzt wird. Eine genaue Erklärung dieser Überlegungen folgt aber noch im Kapitel 4.2 Grobkonzept.
- Die Gewichtung des letzten Kriteriums (Abhängigkeit vom System oder benutzter Software) fällt relativ hoch aus, da das Internet ein heterogenes Netz ist und daher internetfähige Software entsprechend flexibel auf den durch die Anwender vorgegebenen unterschiedlichen Systemen lauffähig sein muss. Eine Einschränkung auf nur einen Browser oder nur ein Betriebssystem wird daher als gravierender Nachteil gesehen.

3.2 Ergebnisse des Vergleichs

3.2.1 CBT’s von Delius-Klasing

Lernprogramme auf einem nicht vernetzten Computer (sogenanntes Computer Based Training CBT) stellte den Anfang von computergestütztem Lernen für den Home-Bereich da. Zum Zeitpunkt der Softwareerstellung in den 90er Jahren waren zwar bereits multimediafähige Computer in

Lerneinrichtungen oder privat verfügbar, diese verfügten meistens aber nicht über einen Internet-Anschluss. Aber auch heute noch wird diese Produktlinie weiter entwickelt, da CBT's gegenüber netzbasiertem Lernen (Web Based Training WBT) den entscheidenden Vorteil haben, dass aufgrund der systemnahen Programmierung auf einem speziellen Rechnertyp dessen Ressourcen entsprechend gut ausgenutzt werden können. So besitzen CBT's eine große Bandbreite an Interaktionsmöglichkeiten, haben eine gut ausgereifte und optisch ansprechende grafische Benutzeroberfläche und sind sehr schnell. Nachteilig wirkt sich aber aus, dass sämtliche Kommunikationsmechanismen fehlen und ein Lernen innerhalb einer Gruppe nicht unterstützt wird. Das Softwarehaus Delius und Klasing, dessen CBT „Sportbootführerschein See" in der neuesten Version[1] hier von mir getestet wurde, integriert zwar mittlerweile auch Zugriffe auf das Internet in ihre Software. Das beschränkt sich aber fast ausschließlich auf den Aufruf vordefinierter Internetseiten und einen Mailkontakt zum Verlag und ist somit eher als Zugriff auf Daten zu sehen, die auch ohne die Lernprogramme verfügbar wären.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	**Beschreibung**	**Bewertung**	**gewichtete Bewertung**
Kommunikationsmechanismen	nicht vorhanden	6	1,2
Art der Implementierung	Stand-Alone	6	0,6
Antwortzeitverhalten	sehr gut	1	0,1
Autorentool verfügbar	nein, jeder Kurs muss für jeden Rechner neu vom Verlag gekauft werden	6	0,6
Bedienungskomfort	sehr gut durch grafische Oberfläche	1	0,05
verfügbare Aufgabentypen	Freitexteingabe, Multiple-Choice, Lückentext, interaktive Abläufe	1	0,3
System-/Browsereinschränkungen	läuft nur auf dem Betriebssystem/Rechnertyp, für den die Software gekauft wurde	6	0,9
Endergebnis			**3,75**

[1] *Delius & Klasing.* CBT zum Sportbootführerschein See. DK-Softmedia, 2001. `http://www.deliusklasing.de`

3.2.2 Idea 3.0 von Link&Link

Idea 3.0[2] ist ein Autorensystem, das in seiner Funktionalität stark an einen Wysiwyg-Gui-Builder erinnert. Hier liegt der große Vorteil dieses Systems, das durch seine intuitive Bedienung auch von Laien schnell zu erlernen ist. Neben der Gestaltung einer Lernseite (begrenzt auf eine Bildschirmseite) zur Wissensrepräsentation bietet Idea noch die Möglichkeit einer Lernkontrolle und einer Ergebnisprotokollierung. Bei der Lernkontrolle werden Übungen kreiert, die der Lernende lösen muss und für die er Punkte erhält. Das können beispielsweise Multiple- oder Single-Choice-Aufgaben sein, genauso wie Zuordnungsaufgaben oder reine Texteingaben. Es besteht die Möglichkeit, beliebige Elemente einzubringen, die über eine Script-Sprache abgefragt werden können. Bei der Ergebnisprotokollierung kann beispielsweise abgefragt werden, wie viele Aufgaben ein Lerner bearbeitet, gelöst oder falsch beantwortet hat. Idea ist jedoch momentan noch ein reines CBT-Produkt, da Online-Elemente zwar angekündigt sind, jedoch noch nicht ausgeliefert werden. In Idea existieren daher keine E-Mail-Funktionen, News-Foren oder Chat-Räume, ebensowenig wie eine Tutorenbetreuung und –bewertung mehrerer Kursteilnehmer. Als ein weiterer Nachteil von Idea ist aufzuführen, dass eine plattformabhängige Laufzeitumgebung (Delphi-Runtime) benötigt wird, die momentan nur für Windows erhältlich ist. Eine Integration in bestehende Web-Seiten ist daher nicht möglich.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	**Beschreibung**	**Bewertung**	**gewichtete Bewertung**
Kommunikationsmechanismen	angekündigt, aber noch nicht vorhanden	4	0,8
Art der Implementierung	Stand-Alone	6	0,6
Antwortzeitverhalten	nicht gut, da die Delphi-Laufzeitumgebung relativ langsam ist	4	0,4
Autorentool verfügbar	ja, allerdings mit Schwachpunkten bei der Erstellung von Tests	2	0,2
Bedienungskomfort	gut durch grafische Oberfläche, mit Einschränkungen durch die Laufzeitumgebung	2	0,1
verfügbare Aufgabentypen	Multiple-Choice, Lückentext, Zuordnungen, beliebige Elemente	2	0,6
System-/Browsereinschränkungen	läuft nur unter Windows	6	0,9
Endergebnis			**3,6**

[2] *Link & Link.* Idea 3.0. Version 2000. `http://www.linkundlink.de`

3.2.3 IBT-Server von Time4You

Der IBT-Server der Firma Time 4 You[3] ist eine Software, die auf einem Server die entsprechenden Aufgaben der Lernumgebung ausführt. Das hat neben Vorteilen in der Ablaufperformance auch den Vorteil, dass keine Client-Software installiert werden muss. Der IBT-Server ist kein Autorensystem im eigentlichen Sinn, da er nicht die Erstellung der Seiten unterstützt. Der Server stellt lediglich die Funktionalität einer Seite zur Verfügung, die bei Bedarf vom Designer angefordert werden kann. Zum Beispiel kann ein Formular entsprechend eingegebener Regeln ausgewertet und das Ergebnis für spätere Abfragen bereitgestellt werden. Dazu muss der Autor aber tiefergehende Programmierkenntnisse besitzen, da es keine leicht zu bedienende Oberfläche zur Integration der Serverfunktionen gibt. Ein Zusatzprodukt der Firma Allaire mit dem Namen HomeSite vereinfacht die Bedienung, da es sich hier um einen erweiterten HTML-Editor handelt. Trotzdem sind weiterhin mindestens gute Programmierkenntnisse erforderlich. So müssen zum Beispiel die oben angesprochenen Ergebnisauswertungen teilweise selbst in JavaScript programmiert werden.

Der IBT-Server stellt dem Anwender Community-Funktionen wie Chat, Foren, E-Mail-Versand oder Benutzerprofile zur Verfügung, der Schwachpunkt hierbei ist jedoch die komplizierte Erstellung eines Tests und die Hinterlegung der Testergebnisse in den HTML-Seiten oder Scripts.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	Beschreibung	Bewertung	gewichtete Bewertung
Kommunikationsmechanismen	Chat, Diskussionsforen, E-Mail vorhanden	2	0,4
Art der Implementierung	Server	1	0,1
Antwortzeitverhalten	gut, abhängig von der Serverauslastung und der allgemeinen Netzlast	2	0,2
Autorentool verfügbar	bedingt, selbst mit Tool sind gute Programmiererkenntnisse erforderlich	4	0,4
Bedienungskomfort	für Anwender sehr gut, für Autoren und Systemverwalter sehr schlecht	4	0,2

[3] *Time 4 You.* IBT Server. Version 2000. `http://www.time4you.de`

verfügbare Aufgabentypen	Server unterstützt verschiedene Aufgabentypen, hat aber keine eigenen Routinen zur Verifizierung der Eingaben. Der Programmierer muss die Lösung im Quelltext der Aufgabe hinterlegen und macht sie somit für den Prüfling erkennbar.	5	1,5
System-/Browsereinschränkungen	keine Einschränkungen für Client (jeder handelsübliche Browser) oder Server (läuft unter Java)	1	0,15
Endergebnis			**2,95**

3.2.4 Lerneffekt WBT von AHR

Die Firma AHR vertreibt ihr Produkt "Lerneffekt WBT"[4] als Server-basierte Plattform einer kombinierten Lösung aus Lerninhalt und Online-Test. Die Inhalte und Tests müssen dabei von der Firma AHR selbst programmiert werden, da kein Autorentool verfügbar ist. Sie bestehen aus einer Reihe von HTML-Seiten und JavaScripts für den Kursablauf, sowie einem eigenen Auswertungsserver für die Online-Tests. Bis jetzt existieren nur Tests zur Lernerfolgskontrolle mit den automatisch ausgewerteten Aufgabentypen Single- oder Multiple-Choice, für den zusätzlich vorhandenen Aufgabentyp eines freien Textes wird nur eine Musterlösung präsentiert. Der Anwender muss bei diesem Typ selbst entscheiden, ob die gegebene Antwort richtig war oder nicht. Daher fließt dieser Aufgabentyp auch nicht in die Lernerstatistik ein. Die Tests lassen sich von der Optik nicht anpassen und können somit auch kein gutes Look-and-Feel einer realen Prüfung bieten.

In die Oberfläche sind eine Reihe von Kommunikationsmechanismen integriert. Kontakt kann aufgenommen werden zu einem Tutor oder zu den anderen Kursteilnehmern über einen Chat, über Diskussionsforen oder über E-Mail.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	**Beschreibung**	**Bewertung**	**gewichtete Bewertung**
Kommunikationsmechanismen	Chat, Diskussionsforen, E-Mail vorhanden	2	0,4
Art der Implementierung	Server	1	0,1

[4] *AHR.* Lerneffekt WBT. Version 2000. `http://www.lerneffekt.de`

Antwortzeitverhalten	gut, abhängig von der Serverauslastung und der allgemeinen Netzlast	2	0,2
Autorentool verfügbar	nein, jeder Test muss von AHR implementiert werden	6	0,6
Bedienungskomfort	gut, durch klar strukturierte Oberfläche	2	0,1
verfügbare Aufgabentypen	Single- und Multiple-Choice, Freitextaufgaben ohne automatische Auswertung	4	1,2
System-/Browsereinschränkungen	keine Einschränkungen für den Client (jeder handelsübliche Browser). Server ist eigenprogrammiert in C	1	0,15
Endergebnis			**2,75**

3.2.5 Nachträglich: Teststation der Firma Enlight

Die Firma Enlight hat mit der Teststation-Reihe[5] mehrere Produkte herausgebracht, die in ihrer Zielsetzung sehr nahe an dieser Arbeit liegen. Dieses Produkt wurde nachträglich eingehend untersucht, obwohl zum Erscheinungsdatum die Vergleichsphase bereits abgeschlossen war. Einige Ideen sind bei der Betrachtung dieser Software noch in das Design eingeflossen. Die Firma Enlight arbeitet eng mit der von "vhs-virtuell" eingesetzten Lernplattform zusammen (siehe unten) und deckt dort den Bereich Wissenstest ab. Mithilfe dieses Tools ist es nun möglich, einen Online-Test zu erstellen und diesen dann in einen Kurs zu integrieren. Die Datenbestände werden dabei zentral auf einem Server verwaltet, der über ein Applet die eingegebenen Daten des Anwenders erhält. Als besonders hervorzuheben ist hier die Tatsache, dass dem Anwender neben den primitiven Aufgabentypen Single-/Multiplechoice und „Klicken auf eine bestimmte Bildposition“ auch ein Test direkt an einer Applikation angeboten werden kann. So startet das Applet beispielsweise die Anwendung „Word“ auf dem Rechner des Benutzers, er kann dann die gestellte Aufgabe lösen und das Applet bewertet hierbei die Eingaben. Im Hinblick auf den Lerneffektes ist das natürlich ein ganz erheblicher Vorteil, da nicht einfach theoretische Fragen beantwortet werden müssen, sondern konkret am praktischen Beispiel gelernt und getestet wird. Ob dieser Vorteil jedoch die daraus entstehenden Sicherheitsbedenken wettmachen kann, bleibt dahingestellt. Ein Applet sollte per Definition nur innerhalb der Browser-Sandbox laufen und daher natürlich keinen Zugriff auf Systemressourcen haben. Das Teststation-Applet nutzt hierbei die in Java 2 (ab JDK1.1) neu eingeführte Möglichkeit, Applets, die in signierten JAR-Archiven gepackt sind, Zugriffe auf Systemressourcen zu ermöglichen (siehe auch [Krüger 2000]). Dem Anwender wird beim ersten Aufruf des Applets vom

[5] *Enlight.* Teststation Performer. Version 2001. `http://www.teststation.nl/de`

Browser die Möglichkeit gegeben, dem hinterlegten Schlüsselzertifikat zu vertrauen und die angeforderten Operationen zuzulassen. Vertraut der Anwender also diesem Softwarehaus, so kann er die Vorteile dieser wirklich innovativen Idee nutzen. Ein Autorentool ist vorhanden, war aber nicht im Umfang der Demoversion enthalten. Hierin wird der Lernstoff in Wissensgebiete und diese wiederum in einzelne Messpunkte unterteilt, um mit dieser Struktur dem Ablaufmodul eine Auswahl aus dem Fragenpool zu ermöglichen. Zusätzlich dazu gibt es ein Administrationstool, das die Verwaltung der Tests und der Lernenden übernimmt.

Dieses Produkt arbeitet eng mit der Plattform Corporate Learning[6] der Firma T-Systems zusammen, über die die Kommunikationsmechanismen abgedeckt werden. Diese Plattform ist eine eigene Entwicklung der Telekom, die eine Lernumgebung für interne Schulungen entwickelt hat und mit dieser nun auch auf den offenen Markt geht. Ihre ausgeprägte Mandantenfähigkeit hat dazu geführt, dass dieses Produkt für das Projekt „vhs-virtuell“ ausgewählt wurde.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	**Beschreibung**	**Bewertung**	**gewichtete Bewertung**
Kommunikationsmechanismen	Chat, Diskussionsforen, E-Mail über die Plattform CL vorhanden	2	0,4
Art der Implementierung	Applet	3	0,3
Antwortzeitverhalten	gut aufgrund nur kurzer Phasen der Datenübertragung	2	0,2
Autorentool verfügbar	ja, lag aber nicht zum Test vor	1	0,1
Bedienungskomfort	sehr gut durch grafische Oberfläche	1	0,05
verfügbare Aufgabentypen	Single-/Multiple-Choice, Klicken auf Bildposition, Test innerhalb externer Applikation (dieser Punkt führt zu einer Aufwertung)	1	0,3
System-/Browsereinschränkungen	Client läuft nur unter Internet-Explorer, keine Informationen über den Server verfügbar	4	0,6
Endergebnis			**1,95**

[6] *T-Systems.* Corporate Learning. Version 2.5, 2001. `http://www.global-learning.de`

3.2.6 Nachträglich: Net-Coach von Orbis Communications

Net-Coach ist eine Lernplattform inkl. Autorentool der Firma Orbis Communications[7] und von zwei Psychologen entwickelt worden. Bei der Entwicklung ist besonderer Wert auf das didaktische Design eines Kurses sowie auf die Punkte Interaktivität, Multimedia, Adaptierbarkeit und Feedback gelegt worden, um eine gute Lernerunterstützung und somit ein effektives Lernen zu ermöglichen. Das System läuft auf einem speziellen Server (CL-FRAU, entwickelt vom MIT) und kommuniziert über jeden handelsüblichen Browser mit dem Anwender. Somit muss auf der Client-Seite weder für Autorentool noch für die Lernerumgebung eine spezielle Software installiert werden und die Funktionalität kann sofort nach Freischalten des Zugangs genutzt werden. Als Kommunikationsmechanismen werden automatisch Chat-Räume, Diskussionsforen und E-Mail-Verknüpfung durch das System erzeugt und entlasten so den Online-Autor. Ebenso werden eine Guided-Tour und ein Suchindex automatisch generiert. Einzig die Installation des Servers muss auf einem speziellen Betriebssystem erfolgen (Windows NT oder Mac OS) und setzt das Vorhandensein des CL-FRAU-Servers (Common-Lisp) voraus.

Beachtlich ist hierbei der große Umfang dieses Pakets, da im Allgemeinen von Softwarehäusern entweder nur die Lernplattform, in seltenen Fällen ein Autorentool und häufig kein integriertes Testtool geliefert wird. Net-Coach ist dabei in der Lage, seine Tests sowohl in den Kurs (als Übung) zu integrieren, wie auch den Test als Abschlussbewertung an das Ende des Kurses zu stellen. Als Aufgabentypen stehen hierbei Single- und Multiple-Choice-Fragen, Lückentexte oder Freitexte zur Verfügung, wobei allerdings bei den Freitexten nur eine Musterlösung angezeigt wird und somit der Anwender selbst entscheiden muss, ob seine eingegebene Lösung richtig oder falsch ist. Für einen registrierten Lerner wird während der Nutzung ein Benutzermodell angelegt, in dem gespeichert wird, welche Seiten bereits besucht wurden, welche Fragen beantwortet wurden und mit welchem Ergebnis. Somit kann jeder Lerner frei im Kurs navigieren (er kann natürlich auch dem vorgeschlagenen Weg folgen) und über das Benutzermodell seinen aktuellen Wissensstand überprüfen. Nachteilig erscheint in diesem Produkt, dass sich das Design der Tests nicht anpassen lässt und sich somit vom Aussehen einer realen Prüfung unterscheiden wird. Außerdem ist jeder Test in einen Kurs integriert und somit nicht direkt als Prüfung zu verwenden. Ist bereits eine Lernplattform vorhanden, lassen sich deren Funktionen nicht mehr ohne weiteres benutzen (da Net-Coach hier ja eigene Funktionen anbietet). Was den Lerneffekt angeht ist dieses System jedoch sehr beeindruckend und vereint nahezu sämtliche Anforderungen an eine Online-Lernumgebung in einem Produkt.

[7] *Orbis Communications.* Net-Coach. Version 2001. `http://www.net-coach.de`

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	Beschreibung	Bewertung	gewichtete Bewertung
Kommunikationsmechanismen	Chat, Diskussionsforen, E-Mail vorhanden	1	0,2
Art der Implementierung	Server	1	0,1
Antwortzeitverhalten	gut, abhängig von Serverauslastung und allgemeiner Netzlast	2	0,2
Autorentool verfügbar	ja, integriert in die Umgebung, erstellt automatisch Guided Tour, Suchindex, Glossar und Kommunikationsmechanismen	1	0,1
Bedienungskomfort	sehr guter didaktischer Aufbau von Kurs und Test, für alle Benutzergruppen leicht zu bedienen über Browserinterface	1	0,05
verfügbare Aufgabentypen	Freitexteingabe (ohne Verifizierung), Single-/Multiple-Choice, Lückentext	3	0,9
System-/Browsereinschränkungen	keine Einschränkungen auf der Client-Seite, Server benötigt spezielle Umgebung	1	0,15
Endergebnis			**1,7**

3.3 Zusammenfassung

Nach Sichtung der bestehenden Softwareprodukte lässt sich erkennen, dass zwar Teile jedes Produktes Vorteile haben, die Anforderungen sich dabei aber auf verschiedene Produkte verteilen, die sich nicht miteinander kombinieren lassen. Erst jetzt ist mit dem Produkt Net-Coach eine Software vorhanden, die den gewünschten Anforderungen schon sehr nahe kommt. Ausschlaggebend für die Entscheidung für eine bestimmte Lernplattform war für das Projekt „vhs-virtuell“ die ausgeprägte Mandantenfähigkeit und gute Benutzerverwaltung und –führung der Lernplattform "Corporate-Learning" von T-Systems. Die ausstehenden Anforderungen an eine Testsoftware können mit dieser Software aber nicht abgedeckt werden. Mittlerweile bietet das Softwarehaus Enlight sein Tool Teststation an, das sich in diese Umgebung integrieren lässt, für das aber eigene Lizenzen erworben werden müssen. Da aber auch dieses Tool nicht alle Anforderungen abdeckt, entsteht die Not-

wendigkeit, durch genaue Analyse der Systemanforderungen eine Software zu entwickeln, die den geforderten Spezifikationen entspricht. Über diesen Prozess handelt das nächste Kapitel.

Kapitel 4

Systemanforderungen

4.1 Problemanalyse (Ist-Zustand)

In diesem Kapitel wird der Ist-Zustand beschrieben, das System kurz mit seinen Bestandteilen vorgestellt und die Integration in die Umgebung von „vhs-virtuell“ skizziert.

Als ersten Punkt wird die finanzielle, personelle und technische Ausstattung beschrieben. Das Budget des Projektes „vhs-virtuell“ sieht ausschließlich die Anschaffung der technischen Mittel und der dafür nötigen Software vor, also zum Beispiel Web-Server, Medien-Server, Video-Encoding-Arbeitsplätze und die dafür nötige Software. Es wurden keine Gelder für Inhalte eingeplant, so dass die Beschaffung von Kursen und Lehrmaterialien nicht möglich ist. Des weiteren sind keine Mittel für personelle Maßnahmen vorgesehen, beispielsweise die Einstellung von Programmierern oder Entwicklern, die entsprechende Kurse im Hause erstellen könnten. Aus diesen Randbedingungen lassen sich bereits folgende Eckdaten für die Software herleiten:

- Es sind leistungsfähige Server im Hause, die mit Software bestückt werden können. Über die Client-Seite kann keine Aussage getroffen werden, da zwar auch ein gut ausgestattetes Selbst-Lernlabor existiert, jedoch gerade das Lernen von zu Hause oder dem Arbeitsplatz aus den großen Mehrwert von „vhs-virtuell“ darstellt. Diese Clients sind also vermutlich im Home-Bereich eher mit langsamer Datenübertragung und älterer Hardware ausgestattet, im Business-Bereich wahrscheinlich eher in ein schnelles Netzwerk integriert und mit aktueller Hardware ausgestattet. Gerade dort sind sie jedoch oftmals durch entsprechende Sicherheitsmechanismen geschützt (z.B. Firewalls).
- Die Software muss neben den Anwendern auch durch die Tutoren eines Kurses bedient werden können, die sich eventuell nicht mit der Erstellung von HTML-Seiten auskennen.
- Des weiteren kann die Software nur durch quasi kostenlose Mitarbeiter erstellt werden.

Der nächste Punkt ist eine kurze Skizzierung des Gesamtprojektes. Die zu erstellende Software gliedert sich auf in 3 Bereiche.

1. Der erste Bereich betrifft die Systemverwalter von „vhs-virtuell“, die Benutzer anlegen, entfernen und verändern müssen, Abrechnungsdaten aus dem System ziehen wollen und bei Systemfehlern in die Struktur der Daten eingreifen müssen.

2. Der zweite Bereich betrifft die Autoren eines Kurses, die über das System die Aufgaben der Fragebögen erfassen und entsprechend ihrer Vorstellungen strukturieren wollen.
3. Der dritte Bereich betrifft die Anwender der hinterlegten Daten. Das sind zum einen die Autoren selbst, die zu Testzwecken eine Prüfung ablaufen lassen wollen, aber hauptsächlich natürlich die Tutoren und Lernenden eines Online-Kurses, die die hinterlegten Daten als Wissensüberprüfung, als Fortschrittstest oder als Zertifikatsprüfung benutzen möchten. Tutoren haben dabei die Funktion, eventuelle Fragen der Teilnehmer zu beantworten sowie bei Prüfungen eine (menschliche) Kontrolle der maschinellen Auswertung eines Tests vorzunehmen. Bei den Lernenden gibt es wiederum die Untergruppe der Prüflinge, die ausschließlich Zertifikatsprüfungen ablegen dürfen und somit von den vorhandenen Hilfestellungen des Systems abgeschnitten sind.

Zeitlich ist diese Software durch die Dauer der Diplomarbeit auf 6 Mann-Monate begrenzt, so dass die Phase der Wartung und Verbesserung über diesen Zeitraum hinaus nur im Rahmen von nachträglich abgeschlossenen Vereinbarungen möglich ist.

4.2 Spezifikationen (Anforderungsdefinitionen)

Das zu entwickelnde System soll eine ausschließlich online zu bedienende Umgebung sein. Es soll verhindert werden, dass Materialien auf externen Medien leicht vervielfältigt werden können. Außerdem gelten die Nutzungsrechte nur für die Dauer des gebuchten Kurses und der Anwender darf aus lizenzrechtlichen Gründen nach Ablauf dieser Zeitspanne nicht mehr auf die Datenbestände zugreifen.

Das zu entwickelnde System soll die primitiven Aufgabentypen (Single- und Multiple-Choice sowie Feldeingaben) und darüber hinaus Freitext-Eingaben (bei denen auf sinngemäße Richtigkeit überprüft wird) sowie Aufgaben mit komplizierteren Abläufen behandeln können (z.B. Kurs und Position auf einer Karte eintragen, Kreuzpeilungen ausführen etc.). Da die Software so universell wie möglich eingesetzt werden soll, müssen diese komplizierteren Aufgabentypen über eine standardisierte Schnittstelle eingebunden werden. Da diese Schnittstelle sowohl Werte aus dem eingebundenen Objekt ausliest (um die Benutzereingabe über das Formular an das Servlet zu senden) als auch Werte in das eingebundene Objekt importiert (für die wiederholte Anzeige einer bereits beantworteten Aufgabe) müssen diese Objekte die dafür nötigen Methoden zur Verfügung stellen. Das ist momentan nur bei einem Applet gegeben, obwohl mittlerweile auch Flash-Objekte über die Export-Kommandos „GetURL“ und „FSCommand“ Ansätze in diese Richtung vorweisen (aber noch nicht dazu in der Lage sind, siehe dazu auch [Franklin 2001]). Das Layout der Fragebögen muss dabei dem Originallayout möglichst ähnlich sein. Sicherheitsüberlegungen müssen ebenso in die Entwicklung einfließen wie die Qualitätsanforderungen Korrektheit, Zuverlässigkeit, Benutzerfreundlichkeit, Wartungsfreundlichkeit, Effizienz und Portabilität. Aktuelle Standards wie z.B. XML müssen daher in das Produkt integriert werden.

4.2.1 Korrektheit

Die Anforderung Korrektheit entspringt dem Wunsch, Fehler in einer Software zu vermeiden bzw. sie zu erkennen und zu beseitigen. Ein Fehler definiert sich durch eine Abweichung zwischen Spezifikation und der tatsächlichen Eigenschaft des Programmsystems. Korrektheit kann gewährleistet werden durch eindeutige Spezifikationen sowie durch Testverfahren wie „Top-Down-Test“, „Bottom-Up-Test“, „Black-Box-Test“ oder „White-Box-Test“. Im Bereich Testautomatisierung sind mittlerweile in der Forschung Fortschritte erzielt worden, die eine automatische Überprüfung eines Programms gegen die Spezifikationen über eine gemeinsame Meta-Sprache (CSP) ermöglichen. Um dieser Anforderung zu genügen, haben sich die unter Kapitel 6.3 dokumentierten Testläufe an die Softwareentwicklung angeschlossen. In diesem Kapitel werden dabei auch die angewendeten Testverfahren näher erläutert. Ebenso wird in Kapitel 4.2.4 Wartungsfreundlichkeit bereits näher auf den Begriff der Testbarkeit eingegangen.

4.2.2 Zuverlässigkeit

Der Begriff der Zuverlässigkeit ist eine Kombination aus Korrektheit eines Programmsystems und dessen Verfügbarkeit. D.h. für die Zuverlässigkeit ist ein Zeitintervall entscheidend, innerhalb dessen bei häufigen Eingaben die Wahrscheinlichkeit eines Fehlers möglichst gering ist. Auch diese Anforderung wird durch die umfangreichen Tests innerhalb des Kurses „Sportbootführerschein See“ abgedeckt.

4.2.3 Benutzerfreundlichkeit

Der Begriff der Benutzerfreundlichkeit teilt sich nach [Pomberger 1996] in die drei untergeordneten Begriffe *Adäquatheit*, *Erlernbarkeit* und die *Robustheit* eines Systems auf. Dabei geht es in erster Linie um die Schnittstelle zum Anwender eines Systems, der ein Softwareprodukt intuitiv bedienen können sollte, dabei allerdings auch bei Fehleingaben keine irreparablen Schäden hervorrufen darf. Wie bei den anderen Softwarequalitätsmerkmalen existieren auch für die Benutzerfreundlichkeit keine Methoden oder Maße, mit der die Güte gemessen werden könnte. Somit muss es Ziel der Programmierung sein, bei Planung, Durchführung und Kontrolle der Software-Herstellung die Definition dieses Qualitätsmerkmals im Auge zu behalten, da dieser Punkt mit entscheidend für die Akzeptanz des Produktes sein wird.

Der Punkt *Adäquatheit* (oder Angemessenheit) bezieht sich

a) auf die vom Benutzer verlangten Eingaben, die sich auf das Notwendige beschränken, einer Plausibilitätsprüfung unterzogen werden und in ihren Möglichkeiten flexibel gestaltet sein sollten. Die Benutzerführung sollte dabei einheitlich, klar und einfach sein.

b) auf die vom Programmsystem angebotene Leistung, die den Wünschen des Benutzers angepasst und in ihrer Funktionalität auf die Spezifikationen beschränkt sein sollte.

c) auf die produzierten Ergebnisse, die übersichtlich, gut strukturiert und einfach zu interpretieren sein sollten. Ergebnisse ebenso wie Fehlermeldungen sollten für den Benutzer flexibel

bezüglich ihres Umfangs, ihres Detailliertheitsgrades und der Art der Präsentation zu gestalten sein.

Für die *Erlernbarkeit* eines Programmsystems ist neben der Gestaltung der Benutzerschnittstelle, die die abgerufenen Informationen realitätskonform präsentieren und die eine effiziente Nutzung der Funktionen unterstützen sollte, auch die Klarheit und Einfachheit des Benutzerhandbuchs entscheidend. Es soll frei von unnötigem Ballast dem Benutzer erklären, was das Programmsystem insgesamt zu leisten vermag, wie die einzelnen Funktionen zusammenhängen und wie sie im Detail anzuwenden sind. Das Benutzerhandbuch sollte darauf eingehen, welche Ausnahmesituationen eintreten und wie sie behoben werden können. Als Nachschlagewerk sollte es geeignet sein, auf Fragen schnell und bequem eine Antwort zu finden.

Die *Robustheit* eines Programmsystems definiert sich durch die Eigenschaft, mit der häufig zu erwartende Fehler in der Bedienung, der Eingabe oder der Hardware nur minimale Folgen nach sich ziehen. Weniger häufig auftretende Fehler (wie z.B. ein Stromausfall) benötigen nicht diese besondere Umsicht, dürfen aber trotzdem keine irreparablen Schäden verursachen.

4.2.4 Wartungsfreundlichkeit

Anforderungen an die Wartungsfreundlichkeit sollen die Möglichkeiten zur Lokalisierung und Korrektur von Fehlern verbessern und die Eignung zur Veränderung und Erweiterung des Programmsystems gewährleisten. Dazu ist es natürlich unter anderem unbedingt erforderlich, dass der Programmcode auch von fremden Programmierern verstanden werden kann bzw. der Programmierer auch noch nach längerer Zeit die Funktionsweise und die Bedeutung der einzelnen Routinen begreift. Dieser Zustand wird mit dem Begriff der Lesbarkeit eines Programmsystems beschrieben, zu dem (neben einer Dokumentation in möglichst guter Qualität) auch die Implementierungssprache selbst sowie der persönliche Programmierstil des Autors zählt.

Der nächste Begriff im Zusammenhang mit der Wartbarkeit eines Programmsystems ist die Erweiterbarkeit, unter der die Möglichkeit verstanden wird, Änderungen und Verbesserungen am Programmcode vorzunehmen ohne dabei unerwünschte Nebenwirkungen einzufügen. Dies ist natürlich auch stark von der Lesbarkeit der Quellcodes abhängig, jedoch ist auch die Strukturierung der einzelnen Programmmodule ein wichtiger Faktor bzw., ob die gewählte Implementierungssprache eine solche Strukturierung überhaupt zulässt.

Ebenso fällt der Begriff Testbarkeit unter diese Überschrift, da hierunter die Eignung für die Verfolgung des Programmablaufs verstanden wird. D.h., ob durch die Sprache selbst oder durch den Programmierer z.B. Möglichkeiten gegeben sind, während des Programmablaufs oder im Fehlerfall etwas über den internen Zustand des Systems zu erfahren und so Fehler leichter lokalisieren zu können (genaue Beschreibung der Exception, Zeile im Programmcode, Inhalte von Variablen etc.). Ein ebenso übliches Verfahren ist der Einsatz von Tools für automatisierte Testläufe, die über das Generieren aller möglichen Zustände und die Aufzeichnung der daraus resultierenden Ergebnisse einen systematischen Gesamttest durchführen können. Förderlich ist dafür ein modulares, gut strukturiertes Programmsystem, da die Qualität der Ergebnisse solcher automatisierten Tests bei unstrukturierten Systemen stark abnimmt. Begründet ist dies durch die Art und Weise, wie solche Tools arbeiten. Sie benötigen eine Schnittstelle für die Eingabe von Testdaten an das zu testende System und für das Auslesen der Ergebnisse, ebenso wie eine Spezifikation der Anforderungen an

das Programmsystem, um zu überprüfen, ob das gelieferte Ergebnis als gültig oder als fehlerhaft zu werten ist. Daraus folgt, dass

a) die Anzahl der zu testenden Fälle endlich sein muss (sonst könnte es passieren, dass Fehlerfälle erst in unendlicher Zeit entdeckt würden)

b) die Spezifikationen in einer Metasprache wie z.B. CSP korrekt formuliert wurden [Peleska 1996].

Je größer und unstrukturierter das zu testende System aber ist, umso geringer wird die Wahrscheinlichkeit der korrekten Formulierung in CSP.

4.2.5 Datenschutz und Datensicherheit

Softwaresysteme, die personenbezogene Daten verarbeiten, unterliegen mehreren gesetzlichen Regelungen, die zum Schutz der betroffenen Personen festlegen, welche Rechte und Pflichten die Datenverarbeiter und die von der Datenverarbeitung betroffenen Personen inne haben. Das sind neben dem Bundesdatenschutzgesetz (BDSG) und den Landesdatenschutzgesetzen (LSDG) auch das Teledienstdatenschutzgesetz (TDDSG) sowie Betriebsverfassungsvorschriften und eine Reihe von speziellen Rechtsvorschriften. Sie regeln die rechtlichen Schutzmaßnahmen vor Problemen und Folgen einer erhöhten Programmflexibilität. Für diese Programmflexibilität werden die Daten benötigt, aber durch ihre Erhebung wird eventuell in die Rechte der Betroffenen eingegriffen. Dies ist insbesondere die Würde des Menschen und das Recht auf freie (d.h. auch unkontrollierte und unregistrierte) Entfaltung der Persönlichkeit, wie sie in Artikel 1 und 2 des Grundgesetzes garantiert wird. Gesetzliche Erlaubnisse zur Datenverarbeitung gibt es nur sehr wenige: z.B. Nutzungs- und Abrechnungsdaten, die die Nutzung (Zugangskennung) oder Abrechnung (Nutzungsdauer) von Telediensten ermöglichen (§ 6 TDDSG). Nutzungsdaten müssen allerdings unmittelbar nach Ende der jeweiligen Nutzung, Abrechnungsdaten spätestens 80 Tage nach Rechnungsversand gelöscht werden.

Umgesetzt werden können diese Datenschutzbestimmungen über verschiedene organisatorische und technische Mittel, bei einer Erfassung von Betriebsdaten wäre das beispielsweise erreichbar durch die Reduktion der erfassten Daten auf Stichproben oder eine Anonymisierung der erhobenen Daten. Ferner sehen die gesetzlichen Regelungen vor, dass der Betroffene darüber informiert und sein Einverständnis eingeholt wird, dass Daten gespeichert werden und welche. Ebenso sehen sie vor, dass nicht korrekt gespeicherte Daten korrigiert werden können (siehe dazu auch [Thome 1990]). Die Software ist darauf zu prüfen, ob sie laut dieser gesetzlichen Definition personenbezogene Daten verarbeitet und ob sie eventuell sogar über den gesetzlichen Rahmen hinaus entsprechende Schutzmaßnahmen trifft.

Neben dem Datenschutz gibt es noch die Datensicherung, die die erhobenen Daten vor Verlust, Manipulation oder Missbrauch schützen soll. Das sind neben den bereits beschriebenen Mechanismen z.B. Sicherungen auf externe Datenträger, gespiegelte Festplatten oder Prüfsummenverfahren während der Speicherung der Daten. Eine räumlichen Zugangskontrolle zu den Datenbankrechnern und ein durch Passwort gesicherter Zugang zum Datenerfassungssystem schützen die Daten vor einer direkten Einsicht oder Manipulation durch unberechtigte Dritte, wobei eine Kombination z.B. mit einem Chipkarten-Lesegerät eine noch bessere Authentifizierung des Anwenders

ermöglicht. Die Datenübertragung zwischen Client und Server muss durch Verschlüsselungsalgorithmen wie PGP oder SSL vor unberechtigtem Zugriff gesichert werden.

4.2.6 Anforderungen an die Funktionalität zur Generierung und Auswertung von Online-Prüfungen

Nach diesen allgemeinen Anforderungen an die Software soll im folgenden Kapitel konkret auf die einzelnen Anforderungen an die Funktionalität eingegangen werden.

4.2.6.1 Programmiersprache und Systemarchitektur

Einer der essentiellen Punkte eines solchen Systems ist die Überlegung, auf welche Art das System implementiert wird. Wie bereits in Kapitel 3.1 Vergleichskriterien angesprochen, existieren mehrere grundsätzliche Arten, ein solches System zu steuern, die nun eingehender untersucht werden. Da sich durch die Anforderungen bereits ergibt, dass in jedem Fall der Zugriff auf eine Datenbank nötig ist, muss das System in einer Programmiersprache verfasst werden, die solche Datenbankzugriffe erlaubt. Daher folgen als erstes Überlegungen zur verwendeten Programmiersprache.

Mittlerweile existieren bereits mehrere Skript-Sprachen (wie z.B. PHP oder ActiveServerPages), die dem Programmierer Zugriffe auf Datenbanken erleichtern sollen. Der Versuch, über eine solche Skriptsprache (benutzt wurden die JavaServerPages von SunSoft [Sun 2001]) rudimentäre Funktionen des Systems zu implementieren, ist daran gescheitert, dass die hinterlegten Skripte sehr schnell unübersichtlich wurden und damit kaum noch wartbar waren. Einige Funktionen ließen sich überhaupt nicht in dieses System bringen, da die Skriptsprache dafür keine entsprechenden Sprachelemente vorgesehen hatte (insbesondere die Kommunikation zur Applet-Schnittstelle war ein nur schwer zu lösendes Problem). Als drittes Manko wurde dabei die langsame Verarbeitungsgeschwindigkeit deutlich, da die Skripte (einmalig beim ersten Aufruf) jeweils neu übersetzt wurden und nach der Übersetzung als eine Reihe von eigenständigen Objekten existiert haben, die nur ungenügend aufeinander abgestimmt waren.

Somit entfiel die Möglichkeit der Skriptsprachen und es musste entschieden werden, in welcher höheren Programmiersprache das System verfasst werden sollte. Hierbei spielte die größte Rolle, in welcher Sprache sich am Besten die Konzepte von Portierbarkeit, Wiederverwendbarkeit, Sicherheit und Kommunikation entwickeln lassen. So fiel die Wahl recht schnell auf Java, wobei im Hinblick auf die Verarbeitungsgeschwindigkeit ebenso die Sprachen C oder C++ interessant waren. Ausschlaggebend war jedoch die Überlegung, ein möglichst plattformunabhängiges System zu verfassen, und die neuen Erkenntnisse und Technologien der sich immer noch weiterentwickelnden Sprache Java nutzen zu können. Die Zugriffe auf Ressourcen des Internets werden im Moment durch keine andere Sprache so gut unterstützt, wie eben durch Java.

Die nächsten Überlegungen sollen herauskristallisieren, welche Systemarchitektur für das Online-Programmsystem unter Java genutzt wird. Es existieren folgende 3 Möglichkeiten:

1. Zwei Stand-Alone-Applikationen kommunizieren über eine TCP/IP-Verbindung miteinander
2. Ein Applet baut eine Kommunikationsverbindung zu einem Server auf (TCP/IP oder HTTP)

3. Der Server erhält die Daten eines Benutzers über die Formularschnittstelle aus HTML-Dokumenten

Diese 3 Möglichkeiten sind mit unterschiedlichen Vor- und Nachteilen behaftet, die je nach Art der Aufgabe unterschiedlich bewertet werden müssen.

4.2.6.1.1 Variante 1: Stand-Alone-Applikationen

Als Stand-Alone-Applikation wird in Java jede Klasse bezeichnet, die eine eigene „Main“-Methode enthält. Das bedeutet, dass dieser Programmteil nicht innerhalb eines Browsers laufen muss, sondern innerhalb einer eigenen Run-Time-Umgebung. Somit entfallen sämtliche Restriktionen der Browser-Sandbox und die Applikation hat vollen Zugriff auf sämtliche Rechner-Ressourcen. Es können also beliebige Dateien des Client-Computers gelesen und verändert werden, ebenso kann zu jedem beliebigen Rechner eine Verbindung aufgebaut werden. Allerdings übernehmen in Unternehmen Sicherheitsmechanismen wie z.B. Firewalls oder Zugriffsbeschränkungen oftmals die Rolle der Browser-Sandbox.

Die benötigte Run-Time-Umgebung ist ein separates Produkt, das vor Installation des eigentlichen Programms erst auf jedem Client-Rechner installiert werden muss. Mittlerweile existieren jedoch bereits Setup-Programme, die diese Installation vereinfachen.

Vorteile: Keine Beschränkungen durch Browser-Sandbox, Client-Oberfläche und Funktionalität können nach eigenen Vorstellungen gestaltet werden (gute Möglichkeiten für Datenverschlüsselung und –transport).

Nachteile: Aufwendige Installation vor erster Nutzung, Anwender muss dem System „vertrauen“, Firewalls blockieren teilweise die Verbindung zum Server.

4.2.6.1.2 Variante 2: Applet kommuniziert mit Server

In dieser Variante läuft innerhalb eines Browsers das Programm in einer klar definierten Sandbox, d.h. Zugriffe auf Rechnerressourcen und Kommunikationspartner werden limitiert. Das Applet muss bei jedem Browserstart neu über das Netz geladen werden (außer es liegt noch im Cache), dafür entfallen jedoch jegliche Installationsmaßnahmen. Durch das neue Sicherheitsmodell von Java 2 vermischt sich diese Variante mit der im vorigen Kapitel beschriebenen Stand-Alone-Variante, da durch das Signieren von Applet-Archiven die Sicherheitsbeschränkungen der Browser-Sandbox umgangen werden können, wenn der Anwender dem zustimmt.

Vorteile: Höheres Vertrauen durch Einschränkung des Zugriffs auf Ressourcen bzw. durch eigenes Freigeben vertrauenswürdiger Zertifikate, Client-Oberfläche und Funktionalität kann nach eigenen Vorstellungen gestaltet werden (gute Möglichkeiten für Datenverschlüsselung und –transport). Die Überprüfung der Daten kann bereits vor dem Senden auf dem Client geschehen.

Nachteile: Applet wird wiederholt über das Netz geladen, Applets unterliegen der Verarbeitungsgeschwindigkeit des Client-Rechners und sind mitunter sehr langsam. Applets laufen nicht ohne Probleme in Browsern, so muss z.B. ein zur Java-Version des Applets passendes Browser-Plugin installiert sein.

4.2.6.1.3 Variante 3: formularbasierte Übertragung der Daten an ein Servlet

In dieser Variante existiert nicht tatsächlich ein Programm auf der Client-Seite, sondern der Browser an sich fungiert als Client. So werden für die Kommunikation zum Anwender eine Reihe von HTML-Seiten bereitgestellt, die über ein Formular die Daten sammeln und dann an das angegebene Servlet senden. Somit ist auf der Client-Seite keine Installation und kein Download nötig, da sämtliche HTML-Versionen bereits Formulare unterstützen und der Anwender somit sofort „loslegen“ kann. Die Server-Seite wird in dieser Methode etwas aufwendiger, da nur HTML-Seiten an den Client zurückgesendet werden können. Allerdings ist die Leistungsfähigkeit des Servers im Allgemeinen kein Problem, da hier nur ein Rechner mit einer guten Ausstattung zu versehen ist (und nicht sämtliche Clients, wie bei den beiden anderen Varianten). Des Weiteren tauchen keine Übertragungsprobleme auf, da reines HTML für die Kommunikationsverbindung genutzt wird, das nicht durch Firewalls beschränkt ist. Durch den Einsatz einer Verschlüsselungstechnologie wie z.B. SSL kann eine ausreichende Sicherheit gegen Missbrauch der Daten und Fehler in der Datenübermittlung gewährleistet werden. Eine weitere Möglichkeit ist eine nachgeschaltete Sichtprüfung der übertragenen Daten, so dass Eingabe- oder Übermittlungsfehler vom Anwender erkannt und dadurch verhindert werden können.

Vorteile: Keine Client-Software nötig, daher bessere Wartbarkeit, geringerer Installationsaufwand, weniger Fehlerquellen, höhere Akzeptanz. Bessere Integration von Fremdsoftware durch Browser-Plugins (z.B. Flash-Animationen, Applets, Video-Streams etc.)

Nachteile: Langsamere Verarbeitung und höhere Übertragungskosten, da jedes Zwischenergebnis übertragen werden muss und nicht mehrere Verarbeitungsschritte zu einem Datensatz zusammengefasst werden können. Außerdem wird zu allen Daten immer der HTML-Rahmen mitgesendet. Das Servlet ist abhängig von der eingesetzten WebServer-Software.

4.2.6.1.4 Abschätzung der 3 Varianten

Die Auswahl aus einer der 3 Varianten fällt sehr schwer, da es in allen 3 Möglichkeiten entscheidende Vorteile und gravierende Nachteile gibt. Die beiden ersten Varianten unterscheiden sich mittlerweile nahezu nur noch durch die Installation. Soll also eine Anwendung wiederholt immer auf dem gleichen Rechner laufen, ist im Vergleich dieser beiden Varianten die erste vorzuziehen. Im Vergleich zur dritten Variante unterscheiden sie sich durch die Abhängigkeit von der Verarbeitungsgeschwindigkeit des Clients. Des Weiteren ist ein Test über Objekte, die nur auf dem Client-PC existieren (z.B. ein Test innerhalb einer Office-Applikation wie beim Produkt Teststation) nur durch ein Applet zu realisieren. Trotzdem erscheint die dritten Variante als die günstigste, da der Vorteil der besseren Wartbarkeit und der geringeren Fehleranfälligkeit als der entscheidende Faktor ins Gewicht fällt. Ebenso wird die Kommunikation von Applets durch firmeninterne Firewalls erschwert, so dass das System universeller einsetzbar ist, wenn es die Kommunikation zwischen Client und Server über den Browser abwickelt. Tests über Elemente, die nicht zum HTML-Standard gehören, treten im untersuchten Fall „Sportbootführerschein See“ relativ selten auf (pro Fragebogen max. einmal) und können dann über die Appletschnittstelle realisiert werden.

Um diese Einschätzung zu unterstützen, wurde innerhalb des Testkurses „Sportbootführerschein See“ ein Fragebogen an die Teilnehmer ausgegeben (siehe auch Anhang B.1 Fragebogen Grundlagen), der unter anderem Fragen enthielt, um die Akzeptanz einer der drei Varianten abzuschätzen. Als Ergebnis stellte sich heraus, dass von den 18 Teilnehmern des Kurses etwa 50% eine entsprechende Software nicht einsetzen würden, wenn sie vorher eine Installation ausführen müssten, und

33%, denen die Begriffe Java und Applet nichts sagen. Dieser Prozentsatz wäre bei der Installation des Applets eventuell auf Hilfe angewiesen, bzw. würde die Software eher nicht verwenden.

Weitere Ergebnisse dieser Umfrage waren:

- 89% halten eine solche Software für sinnvoll, ein Teil davon (11% der Teilnehmer) allerdings nur auf Rechnern der VHS, da entweder kein eigener Rechner vorhanden ist oder die laufenden Online-Kosten zu hoch erscheinen. Ein anderer Teil (33% der Teilnehmer) hält sie nur auf dem eigenen PC für sinnvoll, da auf diese Art ein Üben losgelöst vom Kurs erfolgen kann.
- Die meisten Anwender haben gutes bis sehr gutes Anwenderwissen in Bezug auf die gängigen Browser, so dass ein servletbasiertes System ohne Schulungsaufwand bedienbar wäre.
- Sicherheitsüberlegungen wären für 11% der Teilnehmer ein Grund diese Software in keiner der 3 Varianten zu benutzen. Der Rest steht dem Schutz der Daten eher unkritisch gegenüber. Dieser hohe Prozentsatz unkritischer Anwender zeigt auf, dass bei vielen Anwendern noch eine entsprechende Sensibilität für den Schutz ihrer Daten fehlt.

Aus diesen Überlegungen folgert die Entscheidung, das System als formularbasiertes Servlet-System (Variante 3) zu implementieren und Aufgabentypen, die sich nicht mittels der Standard-HTML-Elemente darstellen lassen, über eine Applet-Schnittstelle zu realisieren. Da die Formularschnittstelle auch die Kommunikation der Applets übernimmt, gibt es dabei z.B. keine Probleme mit Firewalls. Bei der Installation und Inbetriebnahme der Appletschnittstelle könnten die Anwender zwar auch auf Schwierigkeiten stoßen, diese betreffen dann aber nur einzelne Aufgaben und nicht das gesamte System.

4.2.6.2 Systemverwaltung

Unter dem Begriff Systemverwaltung vereinigen sich sämtliche Funktionen, die notwendig sind, ein solches Softwaresystem zu installieren und zu betreuen.

Das zu erstellende Produkt muss eine Benutzerverwaltung besitzen, um das bereits angesprochene Zugriffssystem zu implementieren. Benutzer müssen mit den erforderlichen Daten angelegt und gepflegt werden, sie können innerhalb eines Zeitrahmens für mehrere unterschiedliche Kurse zugelassen sein und sie haben innerhalb des Systems eine bestimmte Berechtigungsebene. Das System muss Zeitpunkt und Dauer einer Anmeldung protokollieren, da aus Abrechnungsgründen diese Daten innerhalb eines frei wählbaren Abrechnungszeitraums an ein Fremdsystem exportiert werden müssen (vorzugsweise unter Benutzung von XML, siehe auch Anhang A: kleiner Exkurs in XML).

Die einzelnen Menüpunkte und Bildschirmmasken sollen durch ein leicht zu erweiterndes System angesteuert werden. Dies geschieht am Besten durch die Entwicklung einer Beschreibungssprache auf der Basis von XHTML, erweitert mit den Funktionen zur Steuerung des Systemablaufs. Aus Gründen der besseren Wartbarkeit empfiehlt sich dabei ebenso die Benutzung des Standards XML, der die Logik eines solchen Dokuments geeignet darstellen kann.

Zugriffe auf den Datenbestand sollten über eine einheitliche Schnittstelle erfolgen, um Programmfehler besser lokalisieren zu können. Eine Datenbank-Schnittstelle sorgt für alle Lese- und Schreibzugriffe und abstrahiert somit das Kernprogramm vom eingesetzten Datenbankmodell.

Es existieren mehrere projektübergreifende Parameter, die ebenso durch die Systemverwalter gepflegt werden müssen.

4.2.6.3 Autorenwerkzeug zur Erstellung eines Tests

Um ein Autorenwerkzeug für Tests zu erstellen, ist es nötig, erst zu analysieren, aus welchen Bestandteilen ein solcher Test besteht. Hierfür wurden die Fragebögen des „Sportbootführerschein See“ exemplarisch untersucht, eine Querüberprüfung gegen die Tests des PKW-Führerscheins und einer Schulung im Bereich Office 2000 hat dabei gezeigt, dass dieses Modell universell genug ist, um beliebige Arten von Prüfungen darzustellen.

Ein Projekt ist hierbei die oberste Hierarchie, unter der sich alle Datenbestände zusammenfassen. Jedes Projekt teilt sich auf in mehrere Themengebiete. Jeder Fragebogen benutzt Aufgaben aus einem Aufgabenpool, so dass eine Frage auf mehreren Fragebögen erscheinen kann. Die Fragebögen sind dabei sortiert nach einer vorgegebenen Logik (oftmals nach Themengebiet), die über eine Positionsnummer auf dem Fragebogen erreicht wird. Eine Aufgabe unterteilt sich jetzt wiederum in mehrere Positionen, wobei jede Position wiederum in Abhängigkeit des Antworttyps eventuell mehrere Lösungselemente haben kann.

Dabei sind die einzelnen Antworttypen Abbildungen zwischen Element und Lösung nach folgender Tabelle:

• checkbox:	$n \mapsto m$ (n Elemente, von denen m richtig sein können)
• radio / select-one:	$n \mapsto 1$ (n Elemente, von denen nur eines richtig ist)
• Field:	$1 \mapsto 1$ (ein Eingabefeld mit unendlichen Eingabemöglichkeiten, wobei nur eine Eingabe richtig ist)
• List:	$n \times (1 \mapsto 1)$ (Eine Liste mit n Elementen, von denen jedes einzelne ein Eingabefeld mit unendlichen Eingabemöglichkeiten ist, wobei nur eine Eingabe richtig ist). Die Reihenfolge der Eingabe ist im Gegensatz zu einer Aufzählung von Elementen des Typs "Field" unerheblich. Oftmals muss auch nur eine geringere Anzahl von richtigen Lösungen gegeben werden. Klassisches Beispiel ist die Frage: "Nennen Sie 4 der 7 Möglichkeiten, einen Wetterbericht zu erhalten."
• Text:	*n* (eingegebener Text) $\mapsto$ *Funktion*(Parameter: Daten zur Interpretation der Eingabe, Beispieltext einer richtigen Lösung)
• Applet:	*n* (Eingabemöglichkeiten des Applets)$\mapsto$ *Funktion*(Parameter: Wertlisten je nach Art des Applets)

Das Autorentool muss also entsprechend der aufgeführten Struktur die benötigten Masken und Funktionen zur Verfügung stellen, so dass die erforderlichen Daten vom Anwender erfasst und geändert werden können.

Aus den hinterlegten Daten muss das System die entsprechenden Objekte erstellen und auf dem Server ablegen. Diese Objekte dürfen durch den Autor nachbearbeitet werden, wenn das erzeugte Layout nicht den Wünschen des Autors entspricht.

4.2.6.4 Der Ablauf eines Tests bzw. einer Prüfung

Da über das Autorentool die Objekte mit den Fragen bereits erstellt wurden, müssen sie während des Ablaufs eines Tests nur in der gewünschten Reihenfolge dem Anwender angezeigt werden. Dieser soll mittels einer automatisch eingefügten Navigation die Möglichkeit erhalten, innerhalb der festgelegten Reihenfolge nach vorne und nach hinten zu blättern, um so z.B. erst alle leichten Fragen zu beantworten und dann zu den schweren zurückzukehren. Zur Erleichterung gibt es je Richtung die nächste unbeantwortete oder die nächsten Frage, unabhängig ob beantwortet oder nicht. Die Antwort wird erst durch die Betätigung eines separaten Buttons als gültig abgespeichert, im Testmodus wird dann die Korrektheit überprüft und das Ergebnis angezeigt. Außerdem sind noch Buttons nötig, um einen Test vorzeitig zu beenden (ohne dass alle Aufgaben beantwortet sein müssen) und einen Button, um die gegebene Antwort wieder zu löschen. Ein Hilfe-Button kann innerhalb des Tests das entsprechende Kapitel des Online-Kurses aufschlagen. Außerdem ist eine Anzeige erforderlich, welche Aufgabe von wie vielen gerade bearbeitet wird.

Im Unterschied zu einem Test wird bei einer Prüfung nicht schon nach jeder Aufgabe die Bewertung der Antwort angezeigt, sondern erst am Ende der Prüfung, nachdem sie durch einen Tutor nachkontrolliert wurde. Dem Tutor wird in diesem Fall zwar bereits der Lösungsvorschlag angezeigt, er hat aber die Möglichkeit, diese mit seiner eigenen Bewertung zu überschreiben. Durch dieses System werden die Schwächen der Freitext-Auswertung ausgeglichen. Außerdem erhöht sich die Akzeptanz des Ergebnisses, da nicht "der doofe Computer" über "mich als Mensch" urteilt, sondern eben ein zweiter Mensch.

Außerdem ist die Hilfe-Funktion während einer Prüfung deaktiviert, da natürlich nicht geschummelt werden soll. Das ist aber verständlicherweise noch kein ausreichender Schutz gegen Betrug, da ein Online-Kurs trotzdem parallel in einer zweiten Browser-Instanz aufgerufen werden kann. Hier ist dann während einer Prüfung wiederum der Tutor gefordert, der auch bei einer realen Prüfung darauf achten muss, dass Täuschungsversuche erkannt werden. Aber auch technische Maßnahmen, wie z.B. das Abtrennen des Subnetzes vom Internet kann solche Täuschungsversuche erschweren.

4.2.6.5 Statistische Daten zur Lernerbewertung

Während der Ausführung eines Tests oder einer Prüfung sammelt die Software Daten über Erfolg und Misserfolg bei der Beantwortung der verschiedenen Aufgaben. Da diese Aufgaben bestimmten Wissensbereichen in einer festgelegten Schwierigkeitsstufe zugeordnet sind, wird so für den Benutzer ein Profil erstellt, das ihn beim weiteren Lernen unterstützen soll. Gerade in den Bereichen, in denen ein Defizit im Wissensstand herrscht, sollte weiter gelernt werden und daher kann dem Benutzer auf Wunsch automatisch eine Frage aus diesem Bereich vorgeschlagen werden.

Da dafür jedoch eine Zuordnung zwischen der Benutzerkennung des Lerners und der erhobenen Daten notwendig ist, fallen diese Erhebungen unter das Datenschutzrecht. Der Benutzer muss auf die Art der Erhebung hingewiesen werden und ihr zustimmen. Er muss die Möglichkeit haben, die erhobenen Daten einzusehen und sie bei Wunsch löschen zu können.

Kapitel 5

Lösungsansätze

5.1 Grobkonzept

Da die Anwendung wie in Kapitel 4.2.6.1 Programmiersprache und Systemarchitektur beschrieben eine servlet-basierte Implementierung sein wird, teilt sich das Grobkonzept in 5 Teilbereiche auf. Es soll durch die folgende Abbildung veranschaulicht werden.

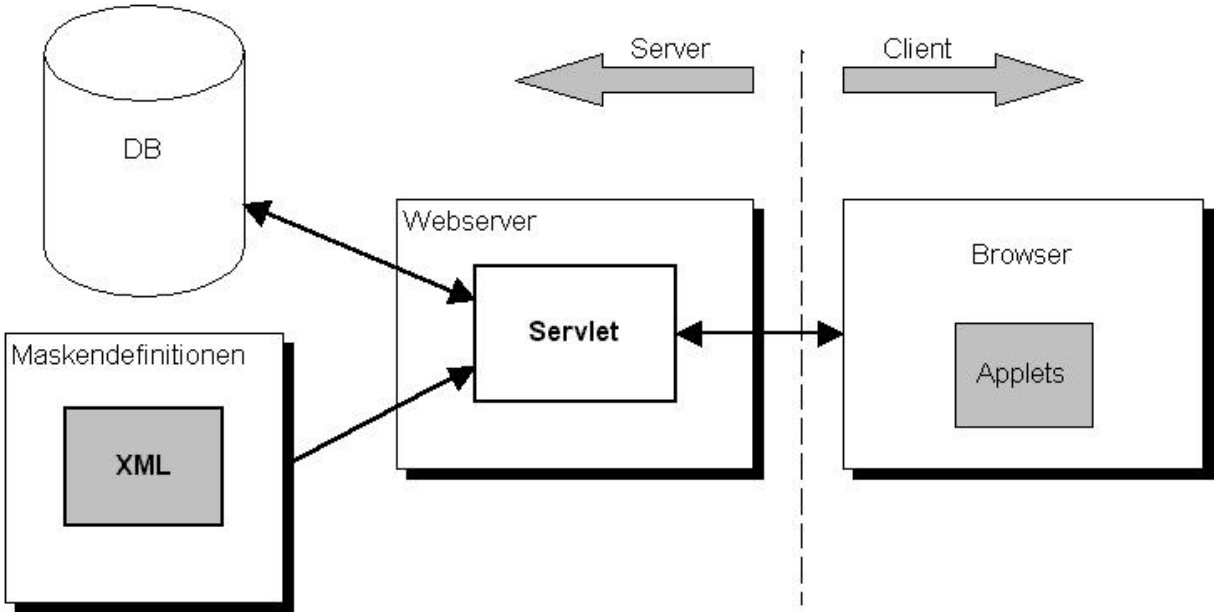

Abbildung 2: Makroarchitektur

Auf dem Client (Teilbereich 1) wird jede Verarbeitung über den Browser ausgeführt. Das sind zum einen die Standard HTML-Elemente ebenso wie kleine Applets, um zusätzlich beliebige weitere Elemente zuzulassen (jedoch ohne dabei direkt mit dem Server zu kommunizieren). Der zweite Teilbereich behandelt die Kommunikation zwischen Servlet und Browser. Sie läuft (wie innerhalb eines solchen Konzepts üblich) über das Einlesen und Übertragen von Benutzereingaben über die

Formularschnittstelle von HTML und die Generierung von Antwort-Seiten durch das Servlet. Der dritte Teilbereich betrifft die Kommunikation zu einem beliebigen Datenbanksystem in dem die Zustände der Software persistent abgelegt werden. Der vierte Teilbereich ist die Definition der Ein- und Ausgabemasken durch XML-Dateien. So können Änderung am Layout vorgenommen werden, ohne in den Source-Code eingreifen zu müssen. Ebenso können Verbesserungen der Funktionalität einfacher durchgeführt werden, da der HTML-Syntax nicht umständlich über Java nachgebildet wird. Der letzte Teilbereich ist die gewünschte Funktionalität selbst, das heißt die Generierung, der Ablauf und die Auswertung eines Tests ebenso wie die Verwaltung der Systemfunktionen und die Bereitstellung von zusätzlichen Kommunikationsmechanismen (Diskussionsforen, Messageboards, E-Mail).

Entscheidende Vorteile dieses Konzeptes sind zum einen die strikte Aufteilung der Aufgaben auf voneinander unabhängige Programmteile, die über wohldefinierte Schnittstellen miteinander agieren. Zum anderen kann durch die Verlagerung möglichst vieler Aufgaben auf den Server und zeitlich gesehen vor die Ausführung eines Tests, den Anwendern des Systems - möglichst unabhängig von der eingesetzten Hardware auf der Client-Seite - ein konstant gutes Antwortzeitverhalten geboten werden.

5.2 Prototyp und Feinkonzept

Prototypen und Feinkonzept der Programmfunktionalität gliedern sich in die 5 eben genannten Teilbereiche auf. Ein Prototyp ist insbesondere für die Schnittstelle zum Anwender, für die Kommunikation zur Datenbank, für das Generieren von HTML-Seiten, für die Appletschnittstelle und für das allgemeine Konzept eines Servlets sinnvoll. Diese Prototypen waren bereits nach kurzer Zeit in der Lage zu zeigen, dass das gewählte Verfahren die Anforderungen abdecken kann. Da bei der Erstellung der Prototypen bereits auf ihre Wiederverwendbarkeit geachtet wurde, sind diese Prototypen direkt in das Gesamtsystems eingeflossen, da nur noch die fehlenden Programmteile eingefügt werden mussten.

5.2.1 Client

Da die Client-Seite des Systems browserbasiert ist, müssen als Prototyp für die Schnittstelle zum Anwender nur eine Reihe von in sich verlinkten HTML-Seiten erzeugt werden, um dem Anwender bereits einen guten Eindruck der Funktionsweise zu verschaffen. Nach [Oestereich 1998] ist der Austausch zwischen Entwickler und Anwender wesentlich einfacher, wenn er auf einem konkreten Bildschirmdialog basiert. Die eigentliche Arbeit wird später von einem Server erledigt, dessen Aufruf anstelle der Links in die so verfasste Oberfläche eingesetzt wird. Da der Server jedoch auch wieder HTML-Seiten zurücksendet, ist diese Methode bereits sehr dicht an der späteren Arbeitsweise des Systems und zudem kann dieser Prototyp hochgradig für die eigentliche Entwicklung wiederverwendet werden.

Die verschiedenen Funktionen werden über eine einfache Baumstruktur von Links innerhalb der HTML-Seiten verwirklicht. Berechtigungssteuerung und Funktionalität liegen dabei ebenso in der

Definitionsdatei als Quasi-HTML (XML mit einer erweiterten HTML-Definitionsdatei) vor und werden bei Abruf durch den Server ausgewertet.

Die Erfassung von Aufgaben (ebenso Projektdaten, Loginkennungen, Fragenbogenzuordnungen etc.) erfolgt formularbasiert über die Standardelemente von HTML. Der entsprechende Verarbeitungsschritt wird dabei durch das Anwählen des jeweiligen Buttons initiiert.

Während eines Testlaufs werden dem Anwender vorgenerierte Fragedokumente gesendet, die eine automatisch eingefügte Navigation enthalten. So kann der Anwender neben der Beantwortung der Frage auch (innerhalb festgelegter Regeln) eigenständig den Ablauf des Tests mitbestimmen.

Ein Test des Prototyps mit Anwendern des Systems hat gezeigt, dass dieses Modell hinsichtlich der Funktionalität und der Bedienbarkeit ausreichend flexibel erscheint (obwohl an der Optik der fehlende Status-Quo grafischer Oberflächen bemängelt wurde).

Die Appletschnittstelle soll über ein kleines JavaScript-Modul die Eingaben an das Formular übertragen und ebenso in die umgekehrte Richtung arbeiten können. D.h. es sind Methoden erforderlich, die

- die Anzahl einstellbarer Werte liefern
- die den Inhalt jedes der einstellbaren Werte auslesen
- die den Inhalt jedes der einstellbaren Werte setzen
- und die eine Veränderung der eingestellten Werte durch den Anwender erlauben und verhindern können

Für die Datensicherheit ist eine Verschlüsselung der Kommunikation erforderlich, die günstigerweise bei den handelsüblichen Browsern und Webservern bereits durch die Benutzung von SSL mitgeliefert wird. SSL übernimmt dabei gleich zwei Aufgaben und verschlüsselt die Daten des Formulars und die Antworten des Servers über ein symmetrisches Verfahren mit einem Schlüssel von mittlerweile bis zu 128 bit, wobei die Aushandlung dieses Session-Keys vorher über ein asymmetrisches Verfahren mit einer Schlüssellänge von i.A. 1024 bit erfolgt. Des Weiteren werden die übertragenen Daten auf Vollständigkeit und Unverändertheit überprüft und im Falle einer Manipulation zurückgewiesen [Netscape 1999]. Dieses Verfahren wird allgemein als sicher angesehen (und ist in dieser Form auch z.B. in die Sprache Java (Version 1.3) integriert [Krüger 2000]).

5.2.2 Kommunikation zwischen Servlet und Client

Die Kommunikation vom Servlet zum Client wird durch die Übertragung programmgenerierter HTML-Seiten erreicht. Die Syntax von HTML sollte dabei durch eine eigenständige Klasse überwacht werden, über die alle Schlüsselwörter sowie der korrekte Aufbau einer Seite erzeugt wird. Dadurch wird die Sprache HTML nicht mehr eine Ansammlung von String-Konkatinationen, sondern abstrahiert sich für den Programmierer zu einer Reihe von Methoden-Aufrufen. Das wiederum erhöht die Wartbarkeit des Programmsystems.

Die Kommunikation vom Client zum Servlet wird über die Formularschnittstelle abgewickelt, die über das Java-Servlet-Paket abgefragt wird. Aus Gründen der Nebenläufigkeit zu anderen Anwendern müssen diese Variableninhalte innerhalb der Session des Benutzers abgelegt werden. Hier

sollen separate Methoden den Programmierer von dieser Aufgabe entbinden, damit ein einheitlicher Zugriff auf die Variablen des Formulars gewährleistet ist. Ein weiterer Aspekt der Nebenläufigkeit ist, dass das Gesamtsystem nicht blockieren darf, wenn ein Anwender eine aufwendige Operation angefordert hat. Damit sich aber nicht die internen Zustände der einzelnen Benutzer überschneiden, muss das System eine vordefinierte Anzahl Threads vorsehen, die somit eine quasi-parallele Verarbeitung ermöglichen. Erhält der Server vom Client eine Anfrage, so muss er den nächsten freien Thread suchen und ihm diese Anfrage zur Verarbeitung übergeben. Es sollte nicht mit einer unendlichen Anzahl von Threads gearbeitet werden (theoretisch denkbar), da die Prozessorauslastung eines Servers ab einem bestimmten Zeitpunkt an ihre Grenzen stößt und ein weiteres Generieren von Threads dann keinen Sinn mehr macht. In diesem Fall ist es günstiger, den Anwender über eine Fehlermeldung auf den Zustand der Serverüberlastung hinzuweisen.

5.2.3 Definition der Masken über XML

Über die vordefinierten Masken soll nicht nur das Aussehen der Oberfläche gesteuert, sondern auch bereits ein Teil der Funktionalität abgedeckt werden. So sind Lese-Zugriffe auf die Datenbank oftmals definiert über entsprechende Eingabefelder in der Maske. Die dazugehörige Ausgabe ist häufig eine standardisierte Tabelle, die nur noch in der Optik definiert werden muss. Des weiteren sollen Ergebnisse des Servers dargestellt werden: Variableninhalte, von Datenbanken abhängige Auswahllisten oder durch Bedingungen gesteuerte HTML-Passagen. Ebenso soll eine Berechtigungssteuerung darstellen, ab welcher Stufe diese Maske angezeigt werden darf. Es muss also eine Definitionssprache eingeführt werden, die neben dem normalen HTML-Syntax ebenso diese zusätzlichen Funktionen abdecken kann. Da sich diese Funktionen vorteilhafterweise in die Baumstruktur des HTML-Dokumentes einfügen sollten, bietet sich eine allgemeine Baumstruktur an, wie sie durch XML am Besten geboten wird. Es muss also ein XML-Interpreter die Maskendefinitionen lesen, die Knoten mit den Server-Erweiterungen umsetzen und das Ergebnis als HTML-Code wieder in das Dokument reintegrieren. Danach sollte das Dokument nur noch aus HTML-Code bestehen, der dann an den Client gesendet werden kann.

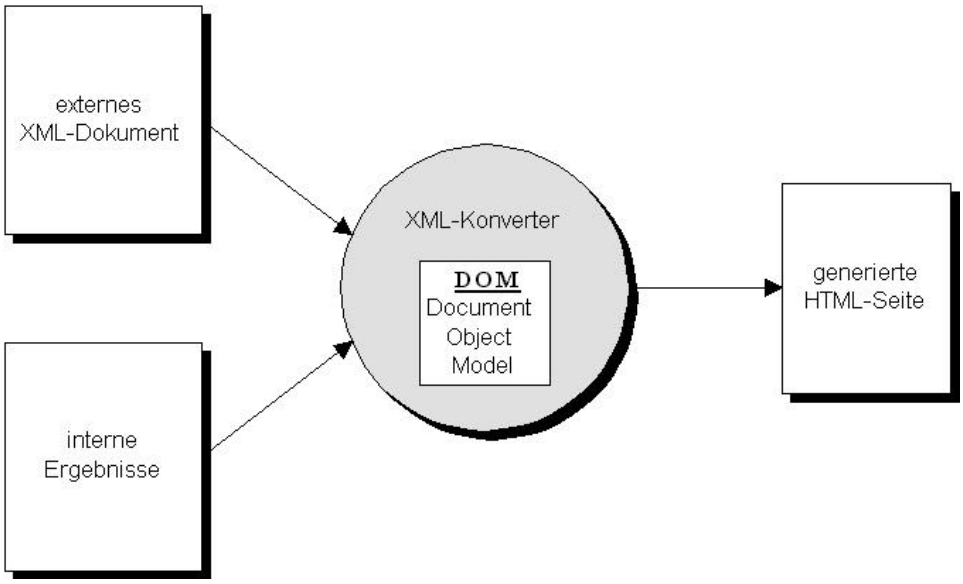

Abbildung 3: XML-Konverter

Dieses Verfahren erweitert das Konzept der Server-Side-Includes, wie es zum Beispiel durch die Java-Server-Pages von SunSoft [Sun 2001] oder die Active-Server-Pages von Microsoft [Microsoft 2001] verwendet wird. Dort werden Scripte (Javascript, VisualBasic, PHP oder ähnliches) innerhalb einer HTML-Datei vor Übertragung an den Client ausgewertet und durch die angegebenen HTML-Elemene ersetzt. Die von mir vorgeschlagene Ersetzung der Script-Elemente durch einen wohldefinierten Satz von XML-Elementen hat den entscheidenden Vorteil, dass die Definitionen auf Korrektheit validiert werden können. Das wiederum hat eine schnellere Entwicklungszeit und eine bessere Wartbarkeit zur Folge.

Für die Interpretation und Validierung von XML-Dokumenten bestehen bereits eine Reihe von frei verfügbaren Klassen (SAX- und DOM-Parser), die die Dokumentbearbeitung standardisieren und dementsprechend vereinfachen. Sie bauen dabei aufeinander auf und stellen dem Programmierer Methoden und Schnittstellen zur Verarbeitung von Ereignissen zur Verfügung.

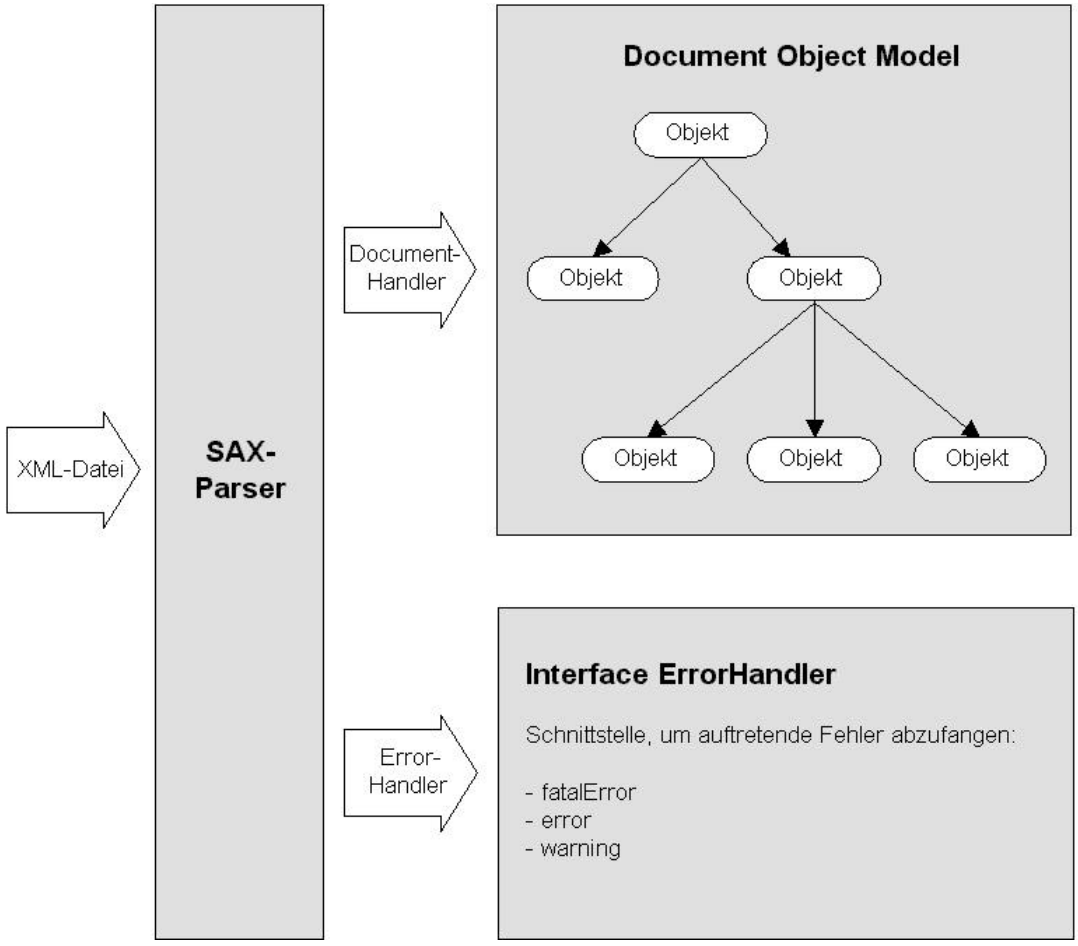

Abbildung 4: Aufbau eines DOM auf Basis eines SAX-Parsers

Dabei ist allerdings zu berücksichtigen, dass die Verarbeitungsgeschwindigkeit durch den Aufbau des Dokumentenmodells im Hauptspeicher sinkt. Ein entsprechender Vergleich zu einem proprietären Format ähnlich den Server-Side-Includes sollte Bestandteil der Entwicklung sein.

Das Document Object Model ließe auch das Übertragen eines Mixes aus HTML und XML an den Client zu, da der XML-Anteil auch dort über eine Kombination aus JavaScript und einem ActiveX-Control (oder ein JScript mit der Anweisung `document.all.tags`) wieder zu HTML

zurückgewandelt werden kann [Seeboerger 2000]. Das unterliegt allerdings einigen Einschränkungen, die dazu geführt haben, diese Idee nicht weiter zu verfolgen. Zum einen sind ActiveX-Controls bzw. JScript nur unter dem Internet Explorer 5.0 verfügbar (keine Browserunabhängigkeit mehr). Darüber hinaus wird wieder ein Teil der Verarbeitung dem Client übergeben. Das sollte aber aus Performance- und Stabilitätsgründen vermieden werden. Es erscheint daher in diesem Zusammenhang sinnvoller, bereits auf dem Server die Daten der Datenbank über ein in XML deklariertes Schema in ein lesbares Format umzuwandeln und nicht den Zwischenschritt über ein XML-Client-Datenformat zu gehen. Dieses Format würde nur dann Sinn machen, wenn (z.B. im Falle eines Exports von Katalog-Daten) der Anwender eine Möglichkeit erhalten soll, die Datenbestände in seinem Eigensystem weiterzuverarbeiten.

5.2.4 Kommunikation zur Datenbank

Die Kommunikation zur Datenbank soll einen eigenen Bereich des Gesamtsystems darstellen, da beim Zugriff von Java auf Datenbanken über die vom JDBC (Java-Database-Connectivity) bereitgestellten Klassen je nach Betriebssystem und eingesetztem Datenbanktreiber gravierende Unterschiede im Laufzeitverhalten auftreten. So sind die von Microsoft ausgelieferten ODBC-Treiber scheinbar nicht vollständig kompatibel (zumindest nicht ausreichend fehlertolerant) zu den von JDBC-ODBC-Treibern geforderten Vorraussetzungen, so dass sich syntaktische oder semantische Fehler beim Zugriff auf die Datenbestände (in diesem Fall eine Microsoft-Access-Datenbank) schnell als nicht mehr nachvollziehbar entpuppt haben. Erst die Zusammenfassung der Datenbankzugriffe in abgeschotteten Klassen lässt eine Abstraktion zwischen dem Datenbestand und den technischen Voraussetzungen, auf ihn zuzugreifen, zu. Durch eine strikte Trennung von den Schnittstellen des JDBC bleibt das Gesamtsystem transparent genug, um eingeschlichene Fehler zu lokalisieren und zu eliminieren.

Die Datenbank-Schnittstelle profitiert von den Erfahrungen einer anderen Lehrveranstaltung (das an der Bremer Uni übliche zweijährige Softwareprojekt mit dem Namen "Bali" [Hoffmann 1999]). Hier wurde bereits eine ähnliche standardisierte Schnittstelle entwickelt, die als Prototyp für das Projekt Gaon eingesetzt und um die noch fehlenden Funktionen erweitern werden kann. Ähnlich dem vom JDBC eingesetzten Modell sollen die Ergebnisse einer Abfrage in ResultSets abgelegt werden, allerdings erweitert um die Möglichkeit, die gesamte Anfrage als Ergebnisvektor zurückzusenden. Daher muss das im JDBC verwendete ResultSet umkodiert werden in eine eigenständige Klasse, um so die Daten eines Ergebnissatzes unabhängig vom vorherigen oder nächsten Satz verfügbar zu halten. Außerdem können so Methoden in diese Klasse gekapselt werden, die für den Import und Export von Daten in ein vorgegebenes XML-Format die nötigen Konvertierungen vornehmen. Des Weiteren soll die Datenbankschnittstelle dem Programmierer Routinearbeiten abnehmen, die für die Verbindung zu einer Datenbank vollzogen werden müssen (Treiber laden, Statements verwalten etc.). Es kann allerdings davon ausgegangen werden, dass entweder die Datenbank zusammen mit dem Servlet auf dem gleichen Rechner läuft (also ausschließlich lokal) oder dass eine Datenbank verwendet wird, die einen Netzwerk-Treiber bereitstellt. Es wäre zwar jetzt durch das modulare Konzept auch möglich, eine Netzwerk-Funktionalität in die Datenbankschnittstelle zu integrieren (tatsächlich ist das im Projekt "Bali" auch geschehen), allerdings würde der Aufwand für Übertragungsperformance, Sicherheitsüberlegungen und Datentransparenz den Rahmen dieser Arbeit sprengen und könnte nicht mit bereits frei erhältlichen Produkten konkurrieren

(zum Beispiel mit dem jetzt von mir eingesetzten voll netzwerktauglichen MySQL-Treiber von Mark Matthews [Matthews 1999]).

Eine der Hauptaufgaben der Datenbankschnittstelle ist, neben der Transparenz und Vereinfachung der Zugriffe auf eine Datenbank, die Umsetzung von Lese- und Schreibanfragen aus und in ein XML-Format, um somit einen Import oder Export von Daten zu erreichen. Das Klassendesign der Datenbankschnittstelle soll durch das folgende OOD-Diagramm nach [Booch 1994] veranschaulicht werden:

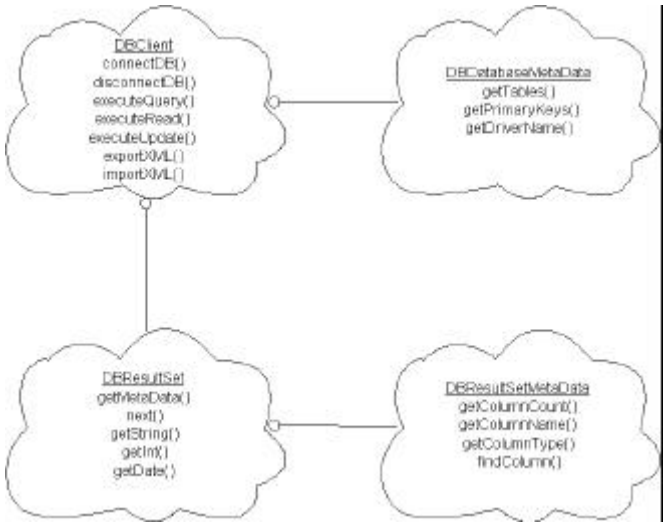

Abbildung 5: Klassenbeziehungen des Datenbank-Interfaces

Die Klasse `DBClient` enthält dabei alle Methoden, um die Kommunikation zur Datenbank abzuwickeln. Bei jeder Anfrage wird ein `DBResultSet` zurückgeliefert, das auf die von SQL bereitgestellte Ergebnismenge verweist. Innerhalb dieser Menge kann über die Methode `next()` auf das nächste `ResultSet` vorgeblättert werden. Dabei enthält jedes `ResultSet` einen Verweis auf die dazugehörigen Metadaten `DBResultSetMetaData`, das Daten wie die Spaltenüberschriften oder die Spaltentypen enthält. Durch den Verbindungsaufbau zur Datenbank wird vom `DBClient` auch ein `DBDatabaseMetaData`-Objekt bereitgestellt, über das die Metainformationen der Datenbank abgefragt werden können.

5.2.5 Funktionalität für das Generieren und Auswerten von Online-Prüfungen

Der erste Schritt für die Generierung von Tests ist das Erstellen der Aufgaben innerhalb des Autorensystems. Dazu muss zu einem Projekt (mit den dazugehörigen Parametern, Themengebieten und Schwierigkeitsgraden) ein Pool von Aufgaben angelegt werden. Zu jeder Aufgabe aus diesem Pool gehören wiederum Teilaufgaben und Lösungspositionen, die in die Datenbank erfasst werden müssen. Nach der Erfassung kann aus diesen Datensätzen dann eine HTML-Datei erzeugt werden, die aus folgenden Bestandteilen besteht:

1. Aufgabenheader, der die Navigation innerhalb des Tests ermöglicht

2. Aufgabenbeschreibung
3. Zu jeder Teilaufgabe die gespeicherten Fragen und je nach Aufgabentyp die entsprechenden Lösungselemente. Bei einer Multiple-Choice-Aufgabe also eine Tabelle mit der entsprechenden Anzahl von Checkboxen oder bei einer Freitextaufgabe ein ausreichend großes Textarea.

Das Klassendesign von Aufgabe, Positionen und Lösungsmenge soll durch das folgende OOD-Diagramm nach [Booch 1994] veranschaulicht werden:

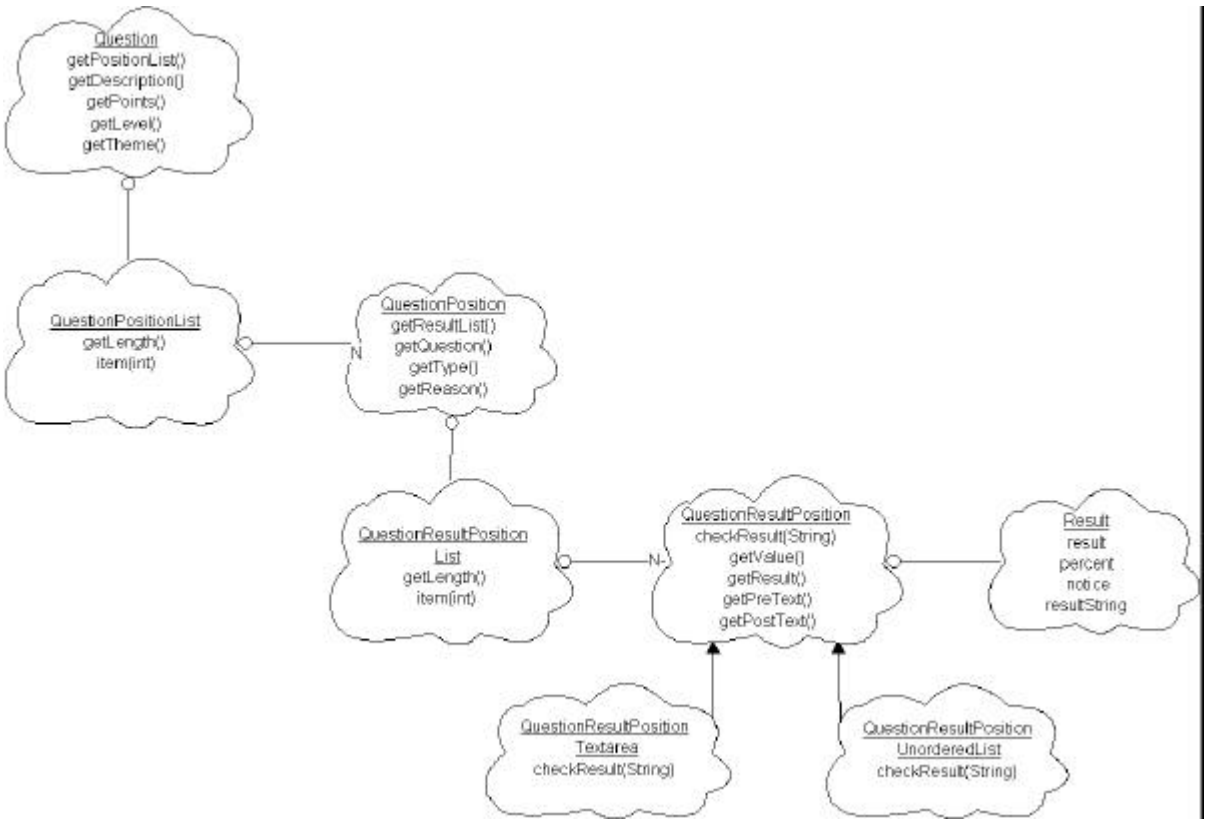

Abbildung 6: Klassendesign der Aufgabenklassen

Die Klasse `Question` hält dabei alle Daten, die sich direkt auf eine Aufgabe beziehen, sowie eine Liste aller zugehörigen Aufgabenpositionen. Um einen einfacheren Zugriff auf jede Position dieser Liste zu erhalten, ist sie in einer eigenständigen `QuestionPositionList`-Klasse gekapselt. Neben den Daten, die sich auf eine Aufgabenposition beziehen, enthält die Klasse `QuestionPosition` wiederum eine Liste aller zugehörigen Lösungspositionen, die zur Vereinfachung des Zugriffs wiederum in einer eigenständigen Klasse `QuestionResultPositionList` gekapselt ist. Diese Liste von `QuestionResultPositions` enthält zum Beispiel bei einer Aufgabenposition vom Typ Checkbox oder Radiobutton alle ankreuzbaren Einträge. Ob ein angekreuzter Eintrag nun vom Ergebnis her richtig oder falsch ist, beurteilt dabei die Methode `checkResult()`. Sie liefert als Ergebnis eine Klasse `Result` zurück, über die der prozentual richtige Anteil der Antwort sowie weitere Informationen zum Auswertungsergebnis übertragen werden. Bei einer Checkbox kann das Ergebnis nur richtig oder falsch sein, jedoch gibt es auch Aufgabentypen – wie zum Beispiel die Freitextauswertung – bei denen nur ein Teil der Lösung korrekt beantwortet sein

könnte. Diese unterschiedliche Art der Auswertung wird durch eine entsprechende Vererbung auf die darunter liegenden Klassen dieses Typs erreicht.

Für die Auswertung wird jede Eingabe gegen die in der Lösungsdatenbank hinterlegten Antwort verglichen und nach einem Verteilungsverfahren für die einzelnen Aufgabenposition eine Punktzahl errechnet. Dieser Vergleich ist bei "Ankreuz"-Aufgaben (Single- oder Multiple-Choice) noch recht trivial, aber bereits bei Lückentexten erscheint es nötig, eine Liste möglicher richtiger Antworten vorzusehen, da es im Allgemeinen mehrere Möglichkeiten gibt, eine Frage zu beantworten. Außerdem gibt es manchmal Ungenauigkeiten z.B. bei Zahlen (3.6 ist genauso richtig wie 3,6 und bei 3,5 würde der Tutor auch noch mal ein Auge zudrücken), für die eine Lösung gefunden werden muss. Um die Datenbank nicht allzu kompliziert zu machen wird zur Speicherung ein reines Textfeld benutzt, in das der Autor eine Liste möglicher Werte, getrennt durch ein Sonderzeichen, eingeben kann. Das hat zwar zum einen den Nachteil, dass die Eingabe unkomfortabler wird (da die Daten auf den ersten Blick etwas verwirrend erscheinen) und dass das Sonderzeichen innerhalb der Lösungsmenge entwertet werden muss. Es bringt aber auf der anderen Seite den Vorteil einer höheren Flexibilität, da auf eine Art alle verschiedenen Aufgabentypen ausgewertet werden können. Für die Erfassung wäre es relativ leicht, in einer späteren Version der Software bessere Eingabemasken bereitzustellen, so dass dieser Nachteil wieder aufgehoben wäre.

Für die Auswertung der Freitextaufgaben wurde dieses Konzept noch etwas verfeinert und festgelegt, dass zu jeder Frage eine Reihe von Schlagwörtern in der Antwort erscheinen müssen. Diese Liste von Schlagwörtern (die dann jeweils wieder verschiedene Alternativen haben können) wird ebenso in ein reines Textfeld erfasst und durch ein Sonderzeichen getrennt. Jetzt muss noch verhindert werden, dass ein Anwender einfach alle denkbaren Schlagworte eingibt und somit die Lösung doch nur errät. Dazu wird ein drittes Sonderzeichen eingeführt, das eine Liste von Negativ-Wörtern trennt. Über diese Mechanismen ist es nun möglich, die Eingabe des Anwenders auf Richtigkeit zu überprüfen und ein prozentuales Ergebnis bezogen auf die in der Aufgabe festgelegte Punktzahl zu ermitteln. Dieses Ergebnis wird in einer Datenbanktabelle abgelegt und dient dann später der Ermittlung der erreichten Gesamtpunktzahl.

Die Aufgaben des Aufgabenpools werden den entsprechenden Fragebögen zugeordnet, damit bei einer Prüfung oder bei einem Test nach Fragebogen die korrekte Reihenfolge angeboten werden kann. Grundsätzlich wird vor Ausführung eines Testlaufs entschieden, ob es sich um einen Test oder um eine Prüfung handeln soll. Dabei unterscheidet sich der Ablauf in einigen Punkten. Der entscheidende Unterschied ist, dass bei einem Test bereits nach jeder Aufgabe die Auswertung der Eingabe angezeigt wird und dann nochmals zur Eingabe zurückgekehrt werden kann, während eine Prüfung erst vollständig beantwortet werden muss. Die Prüfung wird dem Anwender nochmals zur Kontrolle vorgelegt und dann durch einen Tutor gegenbewertet. Erst danach wird dem Probanden das Ergebnis präsentiert. Außerdem ist während einer Prüfung die Hilfe-Funktion deaktiviert.

Als ersten Schritt eines Testlaufs muss sich das System die Reihenfolge zusammenbauen (die bei einem Test auch nach Themengebiet, eine gezielte Auswahl, eine Zufallsauswahl oder eine Defizitanalyse sein kann) und sie in einer entsprechenden Datenbanktabelle ablegen. Dem Anwender wird die nächste unbeantwortete Aufgabe gesendet und auf die Beantwortung gewartet. Nach der Beantwortung werden die Antwortfelder der Reihe nach eingelesen und in einer weiteren Datenbanktabelle abgelegt (um dem Anwender während einer Prüfung eine Eingabe-Kontrolle zu ermöglichen). Bereits zu diesem Zeitpunkt wird sowohl bei einem Test als auch bei einer Prüfung die Antwort ausgewertet und diese Auswertung in einer weiteren Datenbanktabelle abgelegt. Bei einem

Testlauf wird dem Anwender nach jeder Frage eine Auswertung gegeben und so ist es für eine Prüfung dann nur noch nötig, die Übertragung der Antwort zu unterdrücken.

Nach Beantwortung aller Fragen werden in einer Prüfungssituation jetzt noch folgende zwei Teilschritte zwischengeschaltet. Zum einen wird der komplette Test gesammelt mit einer Kurzform der Aufgabe (keine Navigation, geringere Zeilenabstände, keine Trenner zwischen den Teilaufgaben) und der eingegebenen Antwort (ohne die Möglichkeit der Veränderung) dem Anwender zur Überprüfung angezeigt, so dass für ihn die Möglichkeit besteht, zum Test zurückzukehren und dort nochmals Korrekturen vorzunehmen. Zum anderen muss vor Anzeige des Testergebnisses erst ein Tutor den Test gegenbewerten, um sicherzustellen, dass das Lösungsmodul die Antworten richtig bewertet hat. Der Tutor erhält ebenso die Kurzdarstellung des gesamten Tests und kann zu jeder Aufgabe eine eigene, zweite (prozentuale) Bewertung der Eingabe abgeben. Aus dieser Nachbewertung wird dann das Endergebnis ermittelt. Aus den Projektparametern kann dann ersehen werden, ob der Test als bestanden anzusehen ist und es kann ein entsprechendes Zertifikat erzeugt werden.

5.3 Design der Datenbank

Um die Daten persistent zu speichern, müssen sie in einer Datenbank abgelegt werden. Das ist nach jedem Teilschritt notwendig, da die Session eines Benutzers immer nur für den Zeitraum, der für das Durchlaufen der Routine `doPost()` benötigt wird, zur Verfügung steht. Nach Ablauf dieser Zeitspanne kann auf die Daten erst wieder beim nächsten `doPost()` zugegriffen werden, das aber– durch Zeitüberschreitung, Aufruf anderer Seiten oder Schließen des Browsers – eventuell nicht mehr erfolgt. Das hätte den Verlust dieser Daten zur Folge.

Um dabei Redundanzen zu vermeiden, bestehen eine Reihe von Abhängigkeiten zwischen den einzelnen Datenbanktabellen, die mittels des folgenden UML-Diagramms nach [Oestereich 1998] veranschaulicht werden sollen.

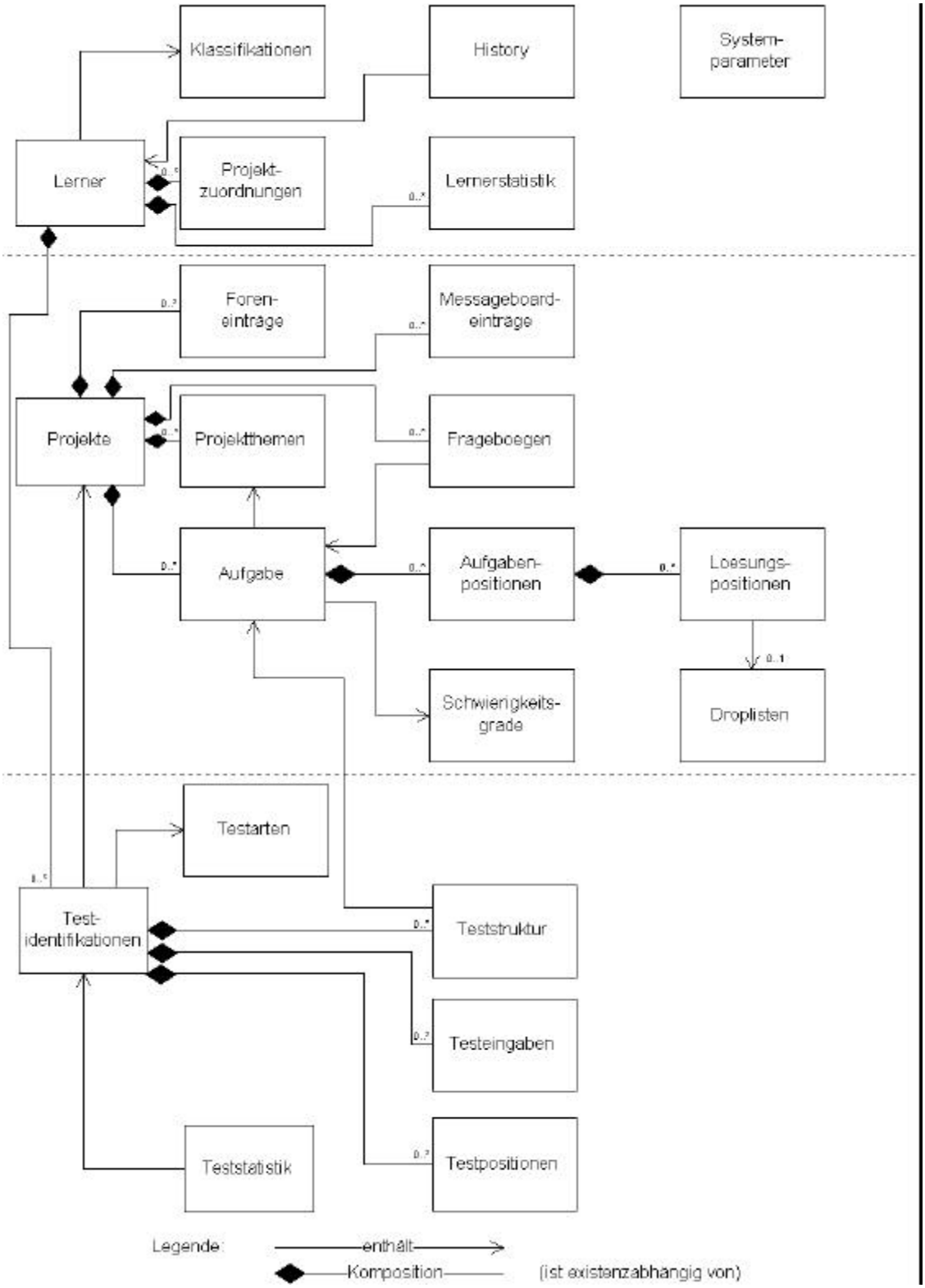

Abbildung 7: Datenbankdesign

Der obere Teil des Diagramms enthält alle Tabellen, die für die Verwaltung und Steuerung eines Lerners oder des Systems notwendig sind. So zum Beispiel die Zuordnungen, für welche Projekte ein Lerner zugelassen ist, seinen Namen und seine Klassifizierung sowie Informationen, die für die Abrechnung der genutzten Leistungen notwendig sind. Im mittleren Teil finden sich alle Tabellen,

die den Aufgabenpool darstellen. Das sind die angelegten Projekte, die Aufgabenelemente mit den dazugehörigen Lösungspositionen sowie die Zuordnung zu Schwierigkeitsgraden und Themengebieten. Der untere Teil des Diagramms enthält alle Tabellen zum Ablauf und zur Bewertung eines Testlaufs, also Informationen zum Start und zum Ende, seine Struktur, die zugehörigen Eingaben des Benutzers sowie die statistischen Details.

Kapitel 6

Implementierung

Aus dem Wunsch heraus, mit dieser Diplomarbeit eine Software zu erstellen, die neben den Anforderungen an die Funktionalität auch die mindestens ebenso wichtigen Aspekte der Softwarequalität mit berücksichtigt, soll hier vorweg kurz auf die verschiedenen Möglichkeiten eingegangen werden, eine Software zu planen und einzuführen. Zu diesem Zweck werden in der Literatur verschiedene Modelle diskutiert, aus denen versucht werden muss, ein für die Anforderungen am Besten geeignetes zu entwickeln.

6.1 Modell zur Softwareentwicklung

Die Realisierung eines komplexen Softwareprojektes wird durch die Schaffung einer Systematik vereinfacht, die dem Entwickler eine schrittweise Lösung des Gesamtproblems an die Hand geben soll. Dadurch teilt sich das Projekt in mehrere überschaubare Phasen auf, die entsprechend leichter zu planen, zu organisieren und zu kontrollieren sind. Die ersten solcher Phasenmodelle sahen eine klar sequentielle Durchführung dieser Phasen vor, so dass eine neue Phase erst in Angriff genommen werden soll, wenn die vorhergehende Phase abgeschlossen ist und das klar definierte Ergebnis vorliegt. Diese strikte Beschränkung würde jedoch dazu führen, dass der Entwicklungsprozess sehr starr vorgegeben ist und auf Benutzerwünsche, neue Anforderungen oder nicht vorhergesehene Schwierigkeiten während der Ausführung einer Phase nur noch schlecht eingegangen werden kann. Daher ist in dem hier verwendeten und im nächsten Absatz beschriebenen Modell dieses streng sequentielle Vorgehen aufgeweicht. Die Phasen selbst sind dabei weiterhin vorhanden, beeinflussen sich jedoch auf unterschiedliche Weise untereinander. Konkret existieren im Wesentlichen folgende Phasen des Software-Entwicklungsprozesses:

- Problemanalyse und Planung (Definition des Ist-Zustands),
- Systemspezifikation (Definition der Anforderungen),
- Grob- und Feinentwurf von System und Komponenten
- Implementierung und Test der einzelnen Komponenten
- Systemtest

- Betrieb und Wartung

Ein prototypingorientiertes Life-Cycle-Modell, wie es in Abbildung 8 skizziert ist, vereint die Vorteile verschiedener anderer Modelle. Da es im Gegensatz zu klassischem Phasenmodellen nicht linear sondern iterativ ist, kann es in den aufeinander aufbauenden Durchläufen jeweils durch die Betrachtung des Prototypen durch Anwender und Systementwickler wesentlich einfacher auf Fehler und Funktionalität überprüft werden [Oestereich 1998]. Das Risiko einer falschen Spezifikation oder eines unvollständigen Entwurfs wird somit vermindert und lässt entsprechend Raum, um zu einer verbesserten Systemspezifikation zu gelangen (siehe auch Abbildung 9). Des Weiteren unterscheidet sich dieses Modell vorteilhaft von klassischen Phasenmodellen durch das starke zeitliche Überlappen der Aktivitäten Problemanalyse und Spezifikation sowie Entwurf, Implementierung und Test, die somit besser ineinander verschmelzen.

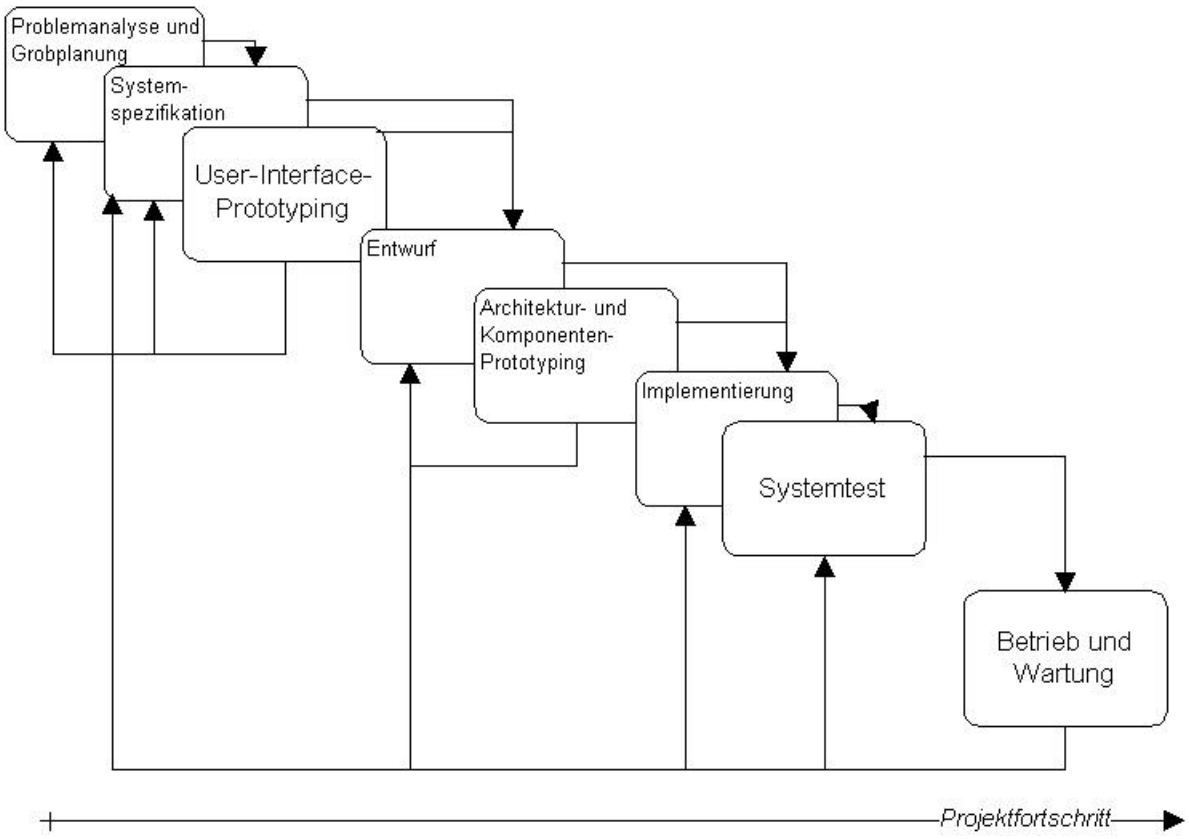

Abbildung 8: Prototypingorientierter Software-Life-Cycle (Quelle: [Pomberger 1996])

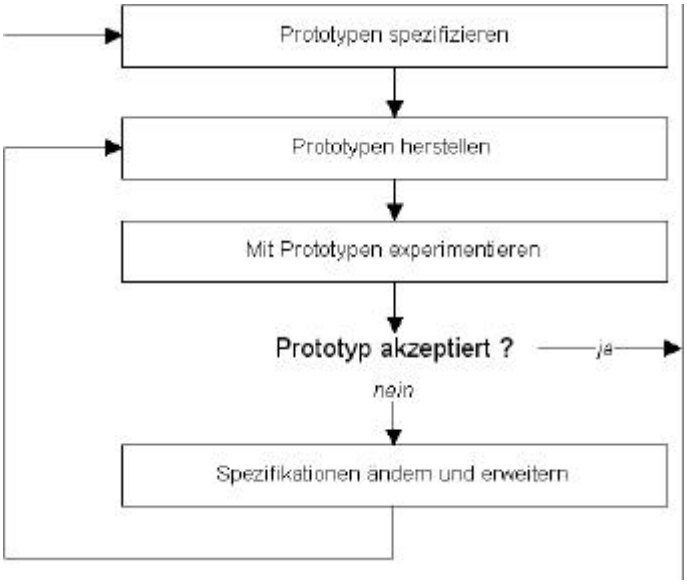

Abbildung 9: Prototyping-Aktivitäten (Quelle: [Pomberger 1996])

Der große Vorteil der gewählten Methode ist, dass im Gegensatz zur klassischen life-cycleorientierten Entwicklungsmethode so früh wie möglich implementiert wird. Die Praxis hat dabei gezeigt, dass gerade im Gespräch mit den Benutzern eines Systems das Verständnis für die Funktionsweise erheblich besser ist, wenn das dynamische Verhalten eines Systems anhand eines Prototypen dargestellt werden kann, als wenn es in bloßer Beschreibung eines Pflichtenheftes vorliegt. Somit wird das Risiko einer Fehleinschätzung vermindert und „die Effekte, die mit den Zwischenergebnissen durch Experimente erzielt werden können, erleichtern die Qualitätssicherung“ ([Pomberger 1996], Seite 26).

Bei der Entwicklung der Prototypen habe ich versucht, auf deren Wiederverwendbarkeit zu achten, da es aufgrund des Umfangs der einzelnen Teilaufgaben nötig erschien, die Softwareerstellung so rationell wie möglich zu halten. Daher sind diese Prototypen nicht mit einem speziellen Werkzeug erstellt, sondern gliedern sich möglichst sofort in das Gesamtsystem ein.

6.2 Implementierung

In diesem Kapitel wird kurz die tatsächliche Implementierung des Entwurfes skizziert und dabei nochmals näher auf das Klassendesign eingegangen. Der Sourcecode ist über die Webadresse "`http://www2.vhs-virtuell.de/gaon/source`" abrufbar, da er mehrere tausend Zeilen Code enthält und daher ein Ausdruck nicht sinnvoll erscheint. Im Anhang sind allerdings die wichtigsten Routinen und Algorithmen abgebildet, da sie für das Gesamtverständnis der Funktionsweise förderlich sind. Zuerst jedoch ein kurzer Überblick über die Verzahnung der Klassen untereinander.

Jedes Absenden eines Formulars auf der Client-Seite ruft die Methoden `doGet` oder `doPost` des Servlets auf (wobei allerdings jeder `doGet` nur auf `doPost` umgeleitet wird und daher nicht weiter interessant ist). Dabei werden entsprechende Handler mit übergeben, die das Auslesen der Formularinhalte und das Rücksenden der Antwortseite an den Client ermöglichen. Das Master-Servlet sucht sich den nächsten freien Thread aus seinem Pool und übergibt ihm die Anfrage. Da

ein Thread immer nur bis zum Ende seiner Ausführung existiert, ist in den Thread die Verarbeitung in einer eigenständigen Worker-Klasse gekapselt. Von dort aus wird die Methode `processTask` der Verarbeitungsklasse (heißt bei mir Gaon, als Synonym für Generierung und Auswertung von Online-Prüfungsbögen) aufgerufen, die wiederum zuerst alle benötigten Variableninhalte des Formulars ausliest. Dann wird die Verarbeitung ausgeführt, die für die angeforderte Task erforderlich ist, und die Ergebnisse wiederum dem Thread zur Verfügung gestellt. Als letztes wandelt der Thread die bereitgestellten Ergebnisse zusammen mit der angegebenen XML-Maske in eine HTML-Antwort um und sendet diese dem Client. Die folgende Abbildung soll diese Vorgehensweise nochmals etwas anschaulicher dokumentieren.

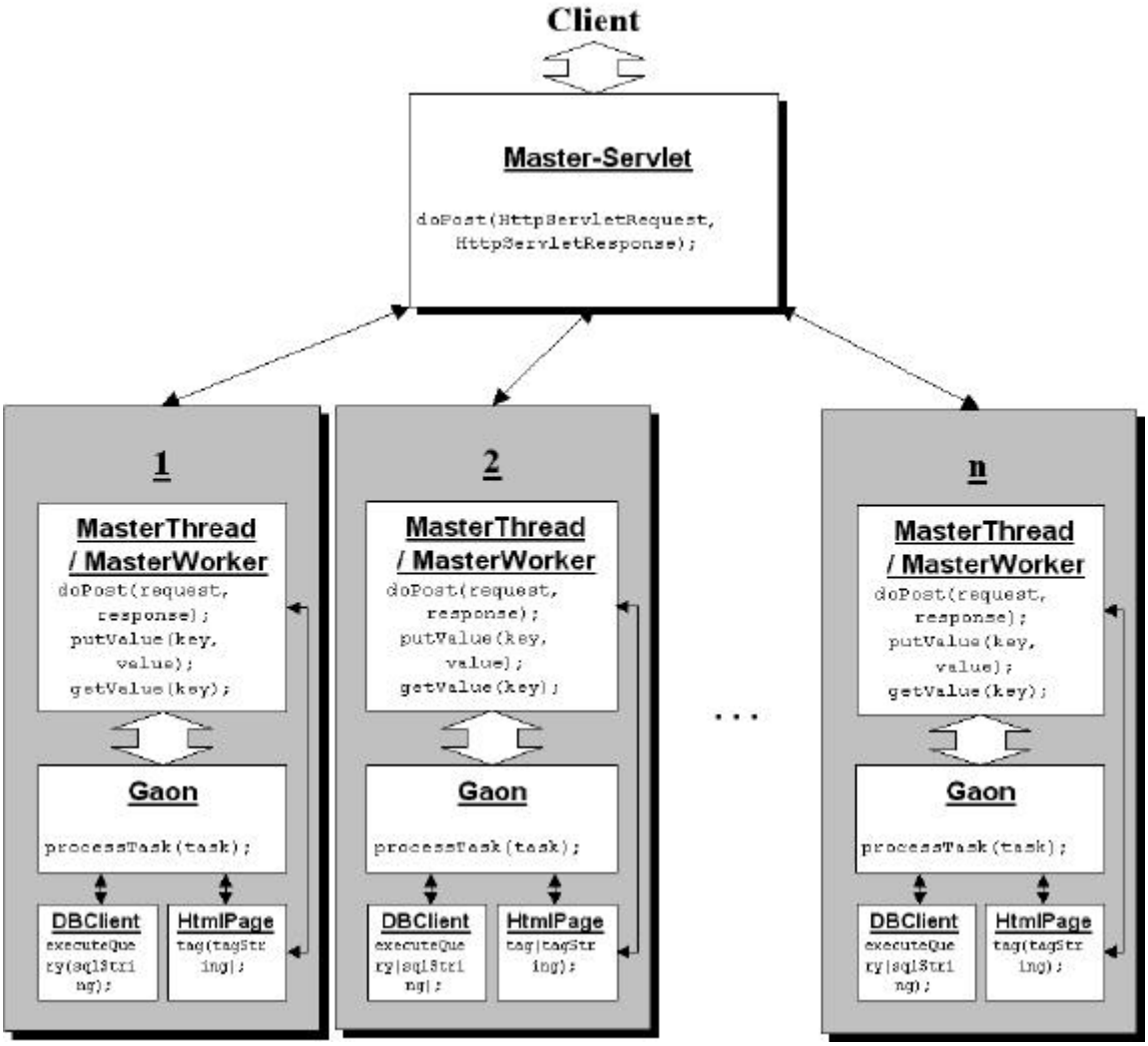

Abbildung 10: Implementierung

Hier wird aus dem Pool der Größe n der nächste freie MasterThread gesucht, der mit den Klassen Gaon, DBClient und HtmlPage kommuniziert, darüber die Antwort an den Client erzeugt und dann an ihn zurück überträgt.

Als Veranschaulichung der Interaktionen der einzelnen Programmparts auf dem Server soll das folgende Interaktionsdiagramm dienen:

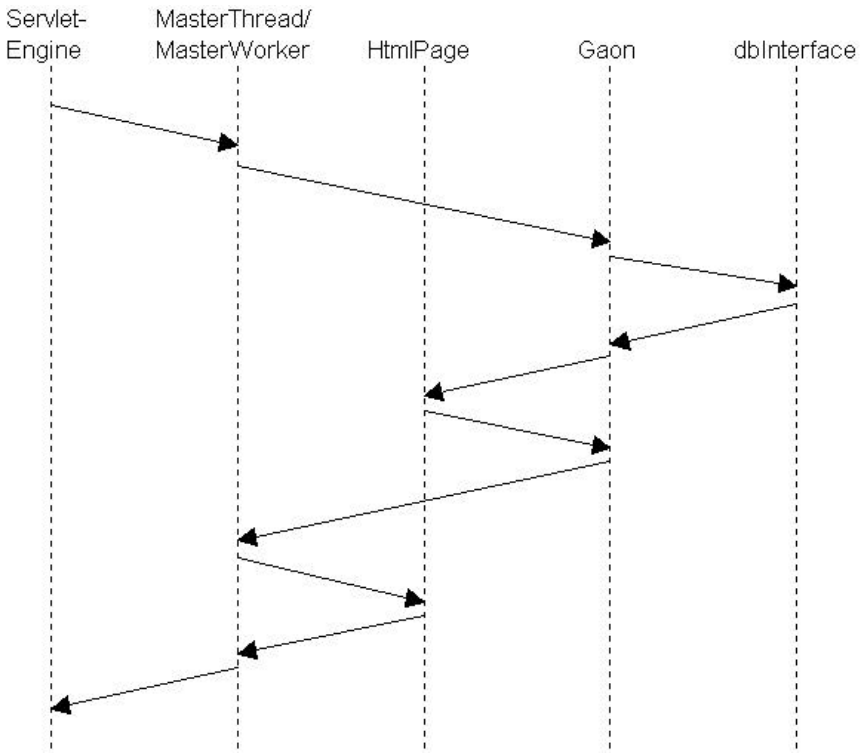

Abbildung 11: Interaktionsdiagramm

6.2.1 Client

Die Auswertung eines Applets verlangt ein standardisiertes Interface, das implementiert werden muss, damit die Ergebnisabfrage aus der Verarbeitungsklasse heraus funktionieren kann. Dieses Interface muss die folgenden Funktionen beinhalten:

```
public interface AppletFrame {
      public int getNumberOfPositions();
      public String getValue(int pos);
      public void setValue(int pos, String value);
      public void setChange(boolean change);
} // end of interface
```

Abbildung 12: Interface der Appletschnittstelle

Über die Funktion `getNumberOfPositions` wird abgefragt, wie viele Lösungspositionen das Applet liefern wird, `getValue` liest den Wert einer speziellen Lösungsposition, `setValue` setzt den Wert dieser Lösungsposition (für die Rückübertragung nötig) und `setChange` legt fest, ob noch Änderungen am Ergebnis vorgenommen werden dürfen.

Über die Objekthierarchie von Javascript kann auf jedes Applet eines Dokumentes zugegriffen werden (für den Source-Code siehe auch Anhang E.1 Appletschnittstelle). Die Methoden des Applets werden dadurch wie Methoden eines normalen Javascript-Objekts benutzbar (natürlich nur, wenn sie als `public` definiert sind). Als kleiner Kunstgriff ist es aber nötig, die Variablen des Applets im Formular nach einem vorgegebenen Schema zu benennen, um sie automatisiert abfragen und verändern zu können (bei mir heißen die Appletvariablen alle `applet`*`I`*`Var`*`N`*, wobei I die Nummer des Applets ist und N die Nummer der Variablen dieses Applets). Ohne diesen Kunstgriff ließe sich im Script nicht entscheiden, ob es sich um ein reguläres Eingabefeld oder um eine Appletvariable handelt. Da jede Aufgabe aber automatisch durch das Autorentool erzeugt wird, braucht sich der Anwender nicht um die Benennung der Variablen zu kümmern.

Die Clientoberfläche ist eine Reihe von in sich verlinkten HTML-Seiten, die vom Server generiert und an den Client übertragen werden. Jede Funktion des Anwenders erhält dabei eine Task-Nummer, damit das Servlet entscheiden kann, welche Wahl der Anwender getroffen hat. Ein typischer Menü-Link sieht dabei dann folgendermaßen aus:

```
<a href="/gaon/servlet/Master?task=A010&reset=YES">Verwaltung Login-
Kennungen</a>
```

Abbildung 13: Menülink

Hier wird das Master-Servlet angewiesen, alle internen Variablen zurückzusetzen und dann Task "A010" auszuführen. Innerhalb der Projektklasse ist dann aufgeschlüsselt, welche Anweisungen für diese Task nötig sind, um beispielsweise wie hier die Verwaltung der Login-Kennungen zu beginnen.

Der üblichere Fall wird aber die Übertragung eines Formulars sein, in das der Anwender bestimmte Werte eingetragen hat. Für das Servlet unterscheidet sich die Übertragung der Daten allerdings nicht, da im obigen Beispiel die URL-kodierte Übertragung von Feldinhalten dargestellt ist, die auch von der HTML-Formular-Schnittstelle benutzt wird. Das Formular zur Auswahl einer bestimmten Login-Kennung nach Kennung, Name oder E-Mail-Adresse würde beispielsweise so aussehen:

```
<p><b>Bitte spezifizieren sie ihre Auswahl:</b>
<form action="/gaon/servlet/Master" method="post">
<input type=hidden name="task" value="A011">
<input type=hidden name="reset" value="">
<table>
<tr><td>Login Kennung</td><td><input type=text name="queryLogin" size=30
maxlength=30></td></tr>
<tr><td>Nachname</td><td><input type=text name="queryLastname" size=30
maxlength=60></td></tr>
<tr><td>E-Mail Adresse</td><td><input type=text name="queryMail" size=30
maxlength=60></td></tr>
</table>
<input type=submit value="Anfrage senden">
<input type=reset value="Zurücksetzen">
<input type=submit name="newButton" value="neuen Eintrag hinzufügen">
</form>
```

Abbildung 14: Formularschnittstelle zum Servlet

An das Servlet werden bei Betätigung eines der als "submit" gekennzeichneten Buttons die Daten übertragen, die sich in den zwei versteckten Feldern befinden (`task` und `reset`) und die sich in den 3 sichtbaren Textfeldern befinden (`queryLogin`, `queryLastname` und `queryMail`). Zusätzlich wird noch die Beschriftung des gedrückten Buttons übertragen, wenn dieser einen Namen hat. So kann überprüft werden, ob der Button für "Anfrage senden" oder für "neuen Eintrag hinzufügen" gedrückt worden ist, da im 2. Fall der Inhalt der Variablen `newButton` ungleich `null` ist.

Für die Verschlüsselung mittels SSL ist auf der Client-Seite nichts zu tun (nur das Zertifikat muss vor Beginn der Sitzung angenommen werden), da bereits beim Start durch eine Adresse die mit dem Protokoll "`https://`" beginnt, gewährleistet ist, dass das Protokoll SSL aktiviert ist. Auf der Server-Seite muss mittels einer geeigneten Certification-Authority ein Schlüsselpaar hinterlegt werden. Dieses zeigt dem Client an, dass der Server tatsächlich derjenige ist, für den er sich ausgibt und dient dann als Grundlage für die Verschlüsselung des Nachrichtenaustauschs zwischen Client und Server. Für diese Aufgabe existieren eine Reihe von Trust-Centern, die solche Schlüsselpaare kostenpflichtig erstellen, ebenso wie es einige Softwarepakete gibt, die diese Aufgabe kostenlos übernehmen, allerdings dabei nicht die Vertrauenswürdigkeit eines anerkannten Trust-Centers bieten.

6.2.2 Kommunikation zwischen Servlet und Client

Die Kommunikation zwischen dem Client und dem Servlet geschieht über die Methode `doPost` oder `doGet`, die vom Servlet-Code überschrieben werden. Dabei muss das Servlet von der Klasse `HttpServlet` erben, um die entsprechende Basisfunktionalität bereitgestellt zu bekommen. An diese beiden Methoden werden zwei Handler übergeben, über die Daten vom Client gelesen und an den Client gesendet werden können.

Jeder `GET` eines Clients wird umgeleitet auf `POST`, da das Servlet nicht unterscheiden muss, welche der beiden Funktionen aufgerufen wurde. Wenn das Servlet allerdings über einen Link aufgerufen wird (also z.B. durch das Menüsystem über `<a href="/gaon/servlet/Master usw.">`) sendet der Browser die Anforderung automatisch als `GET`, während jedes Eingabeformular üblicherweise als `POST` gesendet wird. Daher müssen in jedem Servlet immer beide Methoden implementiert sein.

Die Methode `doPost` kann auf `synchronized` gesetzt sein, um einen nebenläufigen Zugriff zu serialisieren, d.h. zu verhindern, dass mehrere Clients gleichzeitig versuchen auf die internen Variablen des Servlets zuzugreifen. Das würde die unterschiedlichen Anforderungen miteinander vermischen und zu unerwünschten Ergebnissen führen. Die Benutzung von synchronisierten Methoden bietet dabei eine recht simple Möglichkeit, konkurrierende Zugriffe zu ordnen, allerdings gibt es dabei das Problem der "galoppierenden Horden", wenn die Methode wieder freigegeben wird und dann mehrere Clients gleichzeitig versuchen, sich die Ressourcen zu sichern. Dabei ist es reiner Zufall, welche Anforderung als nächstes bearbeitet wird, so dass es bei größeren Projekten aus Fairnessgründen sinnvoll erscheint, dieses Problem anders zu lösen (siehe unten).

Üblicherweise erfolgt innerhalb der Methode `doPost` folgende Verarbeitung: über die Methode `req.getParamterValue("name")` kann vom Client eine Variable eingelesen werden, die z.B. im Formular den Namen "name" trägt (und den Namen des Anwenders enthalten könnte). Danach wird festgelegt, dass als Antwort ein HTML-Dokument zurückgesendet wird und der Writer, der die Anwort an den Client umsetzt, erzeugt. Über diesen Writer wird jetzt der HTML-Code ausgegeben, der auf dem Client die gewünschte Antwort darstellt. Danach kann der Ausgabekanal geschlossen werden und die Arbeit ist erledigt.

Ein einfaches Synchronisieren reicht allerdings für die Belange dieser Software nicht aus (wie bereits im Entwurf beschrieben und im obigen Ansatz angedeutet), da aus Fairnessgründen versucht werden soll, eine festgesetzte Anzahl von Anforderungen parallel abzuarbeiten. Über diese Anzahl hinaus soll dem Client ein Fehlerhinweis auf Überlastung des Servers gesendet werden, um ihn nicht zu lange zu blockieren. Innerhalb des `doPost` wird aus einem endlichen Pool von Verarbeitungsthreads der nächste freie Thread gesucht. Zur Verdeutlichung soll das folgende Diagramm dienen.

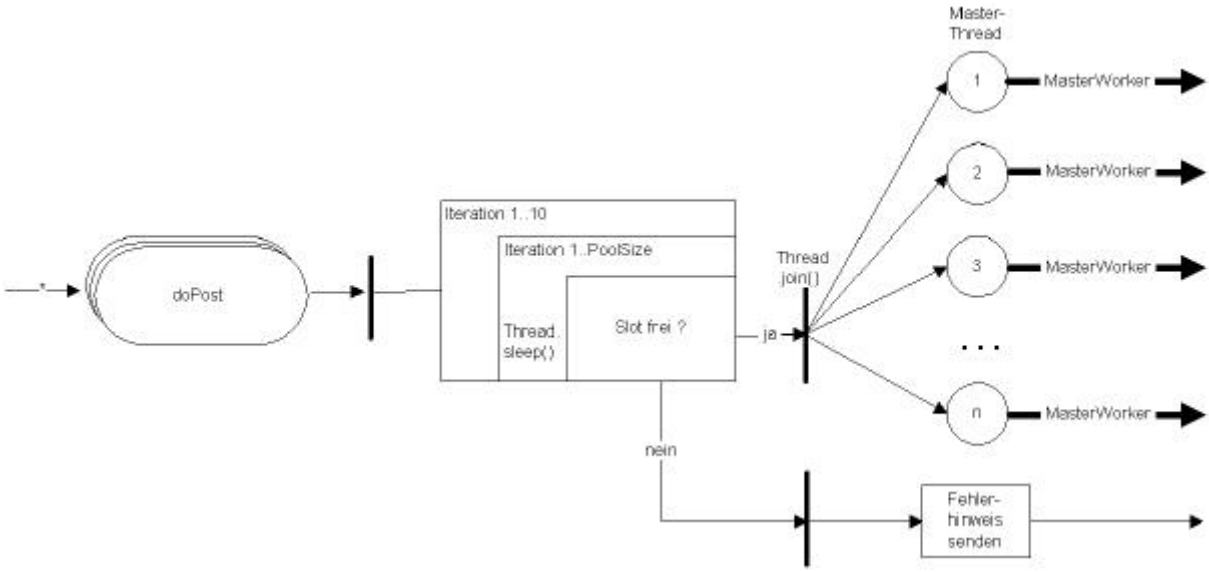

Abbildung 15: Auswahl eines Verarbeitungsthreads aus dem Pool

Es wird für jede Anfrage ein neuer Thread gestartet und dann über die Methode `Thread.join()` auf seine Beendigung gewartet. Dieser kleine Kunstgriff ist notwendig, da der Servletkontext verloren geht, wenn die Methode `doPost` beendet ist. Da die Auswahl des nächsten freien Threads aber synchronisiert erfolgt und innerhalb von `doPost` keine internen Variablen verwendet werden, kann es hierbei nicht zu Überschneidungen kommen.

Für die Auswahl des MasterThreads wird 10 mal versucht, den nächsten freien Slot zu finden, ansonsten wird ein "Server overload"-Fehler ausgegeben, da augenscheinlich die Anfrage des Clients nicht in akzeptabler Zeit beantwortet werden kann (hier willkürlich mit einer halben Sekunde angenommen). Innerhalb dieser Schleife werden alle Plätze des Pools nach dem nächsten freien Thread abgesucht und dann die Verarbeitung an ihn übergeben. Der ausgewählte MasterWorker reicht die Anforderung des Clients an die Verarbeitungsklasse weiter, die sich um die Funktionalität des Systems kümmert, die Antwortfelder bereitstellt und dann die Verarbeitung wie-

der an den MasterWorker zurückgibt. Hier werden die Antwortfelder mit der XML-Maske kombiniert und an den Client zurückgesendet (wie in Abbildung 3: XML-Konverter dargestellt).

Auch das Speichern der Feldinhalte wird durch den MasterWorker erledigt, um den Programmierer von der Servlet-Syntax zu entbinden, die benötigt wird, um Inhalte über den Request-Handler auszulesen. Die Werte werden dabei in einer Hash-Tabelle abgelegt, die der Session des Benutzers zugeordnet ist, so dass das Servlet die Werte der Variablen pro Browser-Instanz getrennt speichern kann. Jeder Thread kann nicht vor Beendigung durch eine neue Anforderung unterbrochen werden, daher kann ohne Probleme am Anfang der Verarbeitung die Session dieser Instanz geladen werden (über die Methode `req.getSession();`). Jedes Ablegen eines Inhaltes geschieht über eine der drei put-Methoden. Da im Allgemeinen eine Zuordnung Schlüssel-Wert besteht, legt die Methode `putValue()` ein String-Objekt zu einem Schlüssel ab. Die Variablen einer Webseite werden dabei jedoch immer als Array von Strings gesendet, da ein Name innerhalb eines Formulars mehrfach auftreten darf. Das zweite Auftreten wird dann im nächsten Feld des Arrays abgelegt (Methode `putValues()`). Da es allerdings auch nötig war beliebige Objekte zwischenzuspeichern (z.B. ein Stack oder eine weitere Hash-Table), ist noch eine entsprechende dritte Methode hinzugekommen (Methode `putObject()`). Der Source-Code findet sich in Anhang E.2.

Das Lesen aus der UserSession geschieht auf eine ganz ähnliche Weise über drei mögliche Methoden, je nachdem, ob der Programmierer gerade genau einen Wert als String, mehrere Werte als String-Array oder beliebige Objekte zwischengespeichert hat und nun wieder benutzen möchte.

Das Aufbereiten der Antwort an den Client ist sehr eng an die Maskendefinition über XML gekoppelt und wird im nächsten Abschnitt erläutert.

6.2.3 Definition der Masken über XML

Einer der großen Punkte dieser Diplomarbeit war die Definition der Ein- und Ausgabemasken des Systems über XML. Die Möglichkeiten von HTML sollten um Funktionen erweitert werden, die eine Abfrage an eine Datenbank definieren und die Optik des Ergebnisses dieser Abfrage als Template festlegen. Da HTML und XML sehr stark miteinander verwandt sind, erscheint hierfür XML die geeignete Wahl. Es ist auch ein proprietäres eigenes Format denkbar (ähnlich den bereits angesprochenen Server-Side-Includes), das den Vorteil hätte

a) besser an die Anforderungen angepasst zu sein (da durch XML zwangsläufig ein Overhead für die Definition der DTD und der Baumstruktur produziert wird).

b) schneller zu definieren gewesen wäre (da der HTML-Anteil nicht noch zusätzlich in die Definition mit aufgenommen werden muss).

c) Dieser eigene Standard wäre bei einer Veränderung von HTML immer noch gültig (wiederum, da der HTML-Anteil nicht noch zusätzlich mit aufgenommen werden muss).

Aber da bereits DTD's für (striktes) HTML existieren, die auch im Falle einer neuen HTML-Version relativ schnell integriert werden können, und der zusätzliche Overhead dafür aber den Vorteil bringt, dass die Masken z.B. auf Fehler in der Struktur überprüft werden können, wurde für die Masken eine DTD für folgende Funktionen entwickelt:

```
// Authorisation
<PAGECLASS level="ADM"> </PAGECLASS>

// Benutzung von Variablen
<VARIABLE name="queryLogin" />, <VARIABLE name="queryName.resultVar1" />

// Datenbank-Abfrage
<QUERY name="q1"><SQL>select Lerner, Vorname, Nachname, Mail, Klasse from
Lerner where Lerner like '<VARIABLE name="queryLogin" />' and Nachname like
'<VARIABLE name="queryLastname" />' and Mail like '<VARIABLE
name="queryMail" />' order by Nachname</SQL><RESULTROW><VARIABLE
name="q1.Mail" /></RESULTROW></QUERY>

// Datenbank-Abfrage als selektierbare Liste
<GAONSELECT name="s1" size="1"><OPTIONS><SQL>select Klassifikation,
Klassifikation from Klassifikationen order by Klassifikation</SQL><SELECTED
field="Klassifikation" cond="eq"><VARIABLE name="user.Klasse"
/></SELECTED></OPTIONS></GAONSELECT>

// Datenbank-Abfrage als mehrfach selektierte Liste verknüpft mit einer
zweiten Abfrage
<GAONSELECT name="s2" size="4" multiple="multiple"><OPTIONS><SQL>select
Projekt, Projekttitel from Projekte order by Projekt</SQL><JOINSELECTED
joinfield="Projekt" cond="eq">select Projekt from Projektzuordnungen where
Lerner='<VARIABLE name="keyVar1" />'</JOINSELECTED></OPTIONS></GAONSELECT>

// Checkbox, angekreuzt je nach Bedingung
<GAONINPUT type="checkbox" name="resultVar7" value="1<CHECKED
var1="resultVar2" cond="eq"><VARIABLE name="resultVar3"
/></GAONINPUT>"><VARIABLE name="resultVar1" />
```

Abbildung 16: Erweiterungen für die GAON-Funktionalität

Bei einem Teil der Erweiterungen reichte es aus, neue Tags zu definieren (z.B. die Abfrage einer Datenbank über das Schlüsselwort "QUERY"). Allerdings ist es für gültigen HTML-Syntax notwendig, bestimmte Attribute innerhalb eines Tags zu benutzen (z.B. CHECKED bei Checkboxen und SELECTED bei Auswahllisten). Da diese Schlüsselwörter jedoch als Attribute innerhalb des dazugehörigen HTML-Tags erscheinen müssen, kam ich nicht umhin auch in den regulären Syntax von HTML einzugreifen. Dabei musste ich aus Gründen der Verschachtelungsmöglichkeit neue Tags definieren, die dann zu den bestehenden Tags mit gesonderten Attributen umkodiert werden. So soll beispielsweise das Attribut "CHECKED" bei dem Tag `<INPUT type="checkbox">` eingefügt werden, wenn eine angegebene Bedingung erfüllt ist, für die der Inhalt einer Variablen ausgewertet werden muss. Da es aber in XML (ebenso wie in HTML) nicht zulässig ist, innerhalb eines Tags ein weiteres Tag anzugeben, muss der Umweg über ein neues Tag benutzt werden.

Die gesamte DTD ist dieser Arbeit als Anhang C: DTD für die Maskendefinition beigefügt, an dieser Stelle sollen nur kurz zwei Beispiele dargestellt werden.

`<VARIABLE name="message" />` liefert den Variableninhalt der Server-Variablen mit dem Namen "message" und ist deklariert über folgende XML-Definition:

```
<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE
  name        CDATA          #REQUIRED
>
```

Abbildung 17: DTD-Auszug für Benutzung einer Variablen innerhalb einer Maske

und

`<QUERY name="q1"><SQL>select * from Lerner</SQL> <RESULTROW> ... </RESULTROW></QUERY>` liefert alle Ergebnissätze einer Abfrage und bereitet diese entsprechend der Definition "Ergebnissatz" einzeln auf. Diese Definition wird dabei üblicherweise ein Tabellensatz sein, in dem die einzelnen Felder des Ergebnis-Sets dargestellt werden. Dabei kann auf die Felder entweder durchnummeriert von resultVar1 bis resultVar*N* zugegriffen werden oder über den Namen des Feldes innerhalb der Datenbank. In beiden Fällen kann der Variablen noch der Name der Abfrage vorangestellt ist, um Abfragen ineinander verschachteln zu können. Beispielsweise sieht die Abfrage auf alle Lerner des Systems innerhalb der Maske folgendermaßen aus:

```
<TABLE><TR><TH>Login-Kennung</TH><TH>Name</TH></TR>
<QUERY name="q1"<SQL>select * from Lerner order by Lerner</SQL>
<RESULTROW><TR><TD><VARIABLE name="q1.resultVar1" /></TD><TD><VARIABLE
name="q1.resultVar2" /> <VARIABLE name="q1.resultVar3"
/></TD></TR></RESULTROW>
</QUERY>
</TABLE>
```

Abbildung 18: Maskendefinition mit Datenbankabfrage

Und die dazugehörigen DTD-Bestimmungen sehen so aus:

```
<!ELEMENT QUERY (SQL, RESULTROW)>
<!ATTLIST QUERY
  name        NMTOKEN        #IMPLIED
  >
<!ELEMENT SQL (#PCDATA | VARIABLE)*>
<!ELEMENT RESULTROW (#PCDATA | %block; | form | %misc; | %inline; | tr |
td)*>
```

Abbildung 19: DTD-Auszug für Datenbankabfrage innerhalb einer Maske

Die angegebenen Entities stammen dabei aus der DTD für striktes HTML und bilden die Blockstruktur von HTML-Elementen ab, die auch innerhalb des Gaon-Knotens vorkommen dürfen.

Umgesetzt in Java wird diese Definition über standardisierte Klassen zur Verarbeitung von XML innerhalb von Java, beispielsweise über das kostenfrei erhältliche Produkt "Xerces 1.3.1", das bis 1999 von der Firma IBM und seit dem von der Apache Software Foundation entwickelt wird. Diese Klassen implementieren die Industriestandards SAX version 1-API und DOM Level 1, sowie darüber hinaus die W3C-Empfehlungen zum XML 1.0-Standard sowie SAX version 2 und DOM Level 2 version 1.0. Sie entbinden somit den Programmierer von der Umsetzung eines ASCII-Textfiles in ein baumartiges XML Objekt [Apache 2000]. Des Weiteren ist dieses Produkt in der

Lage, die XML-Daten nicht nur auf Wohlgeformtheit zu überprüfen, sondern sie auch auf Wunsch zu validieren [Harold 2000b].

Für die Umsetzung der Maskendefinition wird ein XML-Baum traversiert und an jedem Knoten entschieden, ob es sich um eine Gaon-Erweiterung handelt. Wenn nicht, wird dieser Knoten als XHTML-Element in den Ergebnisbaum eingehangen. Handelt es sich um eine Gaon-Erweiterung, wird der Teilbaum an die entsprechende Methode zur Verarbeitung übergeben und dort in einen XHTML-Teilbaum konvertiert (z.B. indem eine Abfrage ausgeführt wird und dann alle ResultSets auf die Ergebnisdefinition angewendet werden oder indem eine Bedingung überprüft wird und dann die angeforderte Operation ausgeführt wird etc.). Da dies auch rekursiv geschehen kann, ist eine beliebige Verschachtelungstiefe innerhalb der Maskendefinition möglich.

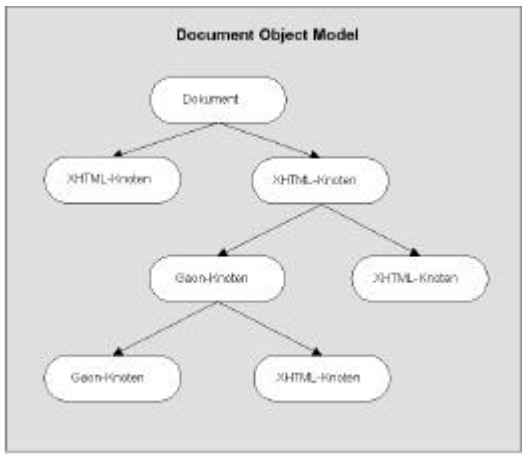

Abbildung 20: Aufbau des Gaon-DOM's

Im Anhang E.3 findet sich exemplarisch der Source-Code für den Aufruf des XML-Parsers und die Traversierung des DOM-Objekts. Innerhalb der Element-Knoten findet dabei die Auswertung und Generierung der Maskendaten statt und um das zu veranschaulichen, ist in diesem Beispiel die Auswertung des Query-Tags mit aufgeführt. Der komplette Java-Code zur Auswertung der XML Masken findet sich im Source-Code des Master-Workers und die vollständige DTD sowie eine komplette Beispieldefinition für eine Maske findet sich in Anhang C: DTD für die Maskendefinition.

Aus Performance-Gründen ist in die Implementierung ebenfalls die schnellere Auswertung durch Skriptelemente aufgenommen worden, wie sie in Kapitel 5.2.3 als Server-Side-Include beschrieben worden ist. Da hier bei der Auswertung kein DOM aufgebaut wird, läuft sie etwa 3 bis 5 mal schneller (140 ms gegenüber 600 ms). Wird im XML-Parser noch zusätzlich das Feature zum Validieren des Dokumentes angesetzt, verlängert sich die durchschnittliche Verarbeitungszeit nochmals um das 2-3-fache (von 600 ms auf 1210 ms). Teilweise hat sie sogar (aus unbekannten Gründen) bis zu 8800 ms gedauert.

Der Programmierer der Maskendefinitionsdatei kann selbst entscheiden, ob er auf die Vorteile der besseren Wartbarkeit und erhöhten Funktionalität verzichten, bis die Entwicklung im Bereich XML-Parser neuere und insbesondere schnellere Generationen hervorbringt.

6.2.4 Kommunikation zur Datenbank

Die Datenbankschnittstelle soll dem Programmierer den Verbindungsaufbau zur Datenbank (mit dem dazugehörigen Laden der entsprechenden Treiber) erleichtern, das Lesen aller Ergebnis-Elemente einer Abfrage in einen Vektor ermöglichen, sowie ihn von allen unnötigen Instruktionen des JDBC befreien, die nicht direkt mit der Manipulation der Daten zu tun haben. Dazu sind mehrere neue Klassen nötig, die die bestehenden Klassen des JDBC ummanteln (zum Klassendesign siehe auch Kapitel 5.2.4 Kommunikation zur Datenbank). Die zentrale Klasse `DBClient` enthält dabei die entsprechenden Methoden als Schnittstelle zum Programmierer. Sie liefert dabei die Elemente oder Vektoren mit den Elementen `DBResultSet`, `DBResultSetMetaData` und `DBDatabaseMetaData` an die aufrufenden Methoden zurück, aus denen dann die Daten wieder extrahiert werden können. Eine Klasse `dbServer` kann anstelle eines Netzwerktreibers die Kommunikationsmechanismen zum Client bereitstellen (z.B. bei der Benutzung einer ODBC-Datenbank), ist aber in dieser Diplomarbeit nicht bis ins Detail ausprogrammiert (insbesondere fehlen hier noch Sicherheitsüberlegungen). Da allerdings eine MySQL-Datenbank im realen Einsatz ist, die über einen nativen, netzwerkfähigen Treiber angesprochen wird, besteht in meinem konkreten Beispiel dazu auch keine Notwendigkeit. Hilfreich ist jedoch auf jeden Fall die Zentralisierung der JDBC-Aufrufe in wenige Methoden separater Klassen, so dass bei fehlerhaftem Verhalten leicht Debug-Punkte aufgenommen werden können und die Fehlersuche somit vereinfacht wird.

Zwei Anwendungsbeispiele sollen hier die gewählte Implementierung verdeutlichen. Zum einen soll der Verbindungsaufbau zur Datenbank über den Methodenaufruf `dbClient.connectDB(dbName, dbDriver, dbUser, dbPassword);` erfolgen, bei dem insbesondere der Parameter dbDriver nicht versorgt zu werden braucht, wenn der hinterlegte Standardtreiber benutzt werden soll. Der Verbindungsaufbau zur MySQL-Datenbank sieht dementsprechend bei mir folgendermaßen aus:

```
dbClient.connectDB("jdbc:mysql://172.16.200.234:3306/Gaon", null,
"GaonServlet", "Password");
```

Abbildung 21: Verbindungsaufbau zur Datenbank über das Datenbank-Interface

Die IP-Adresse der Datenbank wird dabei vom Web-Server über einen Startparameter übergeben und ist natürlich nicht fest im Code hinterlegt. Über das angegebene Subprotokoll ("`jdbc:mysql:`") wird erkannt, dass der MySQL-Treiber geladen werden muss, da kein abweichender Treiber angegeben ist (Parameter 2 ist `null` oder leer).

Das zweite Anwendungsbeispiel soll das Lesen eines Ergebnisvektors sein, in diesem Fall sollen sämtliche Lösungspositionen zu einer Aufgabenposition gelesen werden. Dazu wird folgender Aufruf benutzt:

```
String sql2="select Nummer, Wert, Loesung, TextVor, TextNach from
Loesungspositionen where Projekt='"+projekt+"' and Aufgabe="+aufgabe+" and
Position="+position+" order by Nummer";
try {
```

```
      Vector rsv2=dbClient.executeRead(sql2);
      for(int I2=0; I2<rsv2.size(); I2++) {
            DBResultSet rs2=(DBResultSet)rsv2.elementAt(I2);
            int nummer=rs2.getInt("Nummer");
            String wert=rs2.getString("Wert");
            String loesung=rs2.getString("Loesung");
            String textVor=rs2.getString("TextVor");
            String textNach=rs2.getString("TextNach");
} catch(Exception e) { // do some error-handling }
```

Abbildung 22: Einlesen eines Ergebnisvektors über das Datenbank-Interface

Der SQL-String wird an die Methode `executeRead` übergeben, die einen Vector mit Ergebnis-Sets zurückliefert. Diese Ergebnis-Sets werden der Reihe nach aus dem Vector ausgelesen und somit können die Felder aus dem Set benutzt werden. Der Vorteil dieser Vorgehensweise ist, dass sich der Programmierer nicht mehr um die Verwaltung der JDBC-Statements kümmern muss, wenn ineinandergeschachtelte Abfragen erforderlich sind (in diesem Beispiel wurden zuerst alle beantworteten Aufgaben, zu jeder Aufgabe alle Aufgabenpositionen und dann zu jeder Aufgabenposition alle Lösungspositionen gelesen). Bei der Benutzung des JDBC steht im Gegensatz dazu eine Abfolge von Result-Sets nur solange zur Verfügung, bis das Statement durch eine andere Abfrage benutzt wird. Es können zwar auch mehrere Statements parallel instatiiert werden, die Anzahl ist jedoch teilweise durch den Treiber begrenzt (beim Standard-ODBC-Treiber auf etwa 60). Das bedeutet zwar, dass theoretisch etwa 60 Querys ineinander verschachtelt werden können, allerdings muss sich der Programmierer dann um ihre korrekte Verwaltung bemühen.

Dabei wird die Klasse DBResultSet als Zwischenspeicher benutzt, um einen Satz des Ergebnisses der Abfrage abzulegen, da ein "normales" ResultSet des JDBC durch die nächsten Lese-Operation überschrieben wird.

Die Datenbankschnittstelle stellt ebenso Funktionen zum Import und Export von Daten der Datenbank in ein XML-Format zur Verfügung. Die entsprechenden Methoden folgen dabei den 3 Hauptanforderungen:

1. eine Möglichkeit zur Datensicherung einzelner Tabellen zu schaffen
2. Daten in Fremdsysteme exportieren zu können (z.B. Abrechnungsdaten)
3. Daten aus Fremdsystemen importieren zu können (z.B. vorgefertigte Kurse)

Während das Datenformat für die Anforderungen 2 und 3 nur in einer DTD spezifiziert und entsprechend generiert werden muss, war bei der Anforderung einer Datensicherung zu berücksichtigen, dass die Ausgaben aus der Export-Schnittstelle direkt wieder als Eingabe der Import-Schnittstelle dienen können. Daher folgen die implementierten Methoden sowohl für den Import als auch den Export folgender DTD:

```
<!ELEMENT GaonData (Query*)>
<!ELEMENT Query (ResultSet*, Exception?)>
<!ATTLIST Query name NMTOKEN #REQUIRED>
<!ATTLIST Query tables NMTOKENS #IMPLIED>
<!ATTLIST Query clearBeforeOnImport (true|false) #IMPLIED>
<!ATTLIST Query updateExistingOnImport (true|false) #IMPLIED>
```

```
<!ELEMENT ResultSet (Column*)>
<!ATTLIST ResultSet row CDATA #IMPLIED>
<!ELEMENT Column (#PCDATA)><!ATTLIST Column name NMTOKEN #REQUIRED>
<!ELEMENT Exception (#PCDATA)>
<!ATTLIST Exception name NMTOKEN #REQUIRED>
```

Abbildung 23: DTD für den Datenbank Import und Export

Wenn ein Export für eine genauer spezifizierte Aufgabe vorgenommen werden soll, so muss er nach den gewünschten Regeln neu implementiert werden (wie z.B. in Anhang A im XML-Beispiel geschildert). Die bereitgestellten Methoden sind jedoch allgemein genug, um alle Daten, die mittels einer SQL-Anfrage selektiert werden können, in dieser Form zu exportieren.

6.2.5 Funktionalität zur Generierung und Auswertung von Online-Prüfungen

Die Klasse `Gaon` ist die Klasse, die die eigentliche Arbeit des Systems übernimmt. Sie ist eingebunden in die offene Struktur des Master-Servlets als Implementierung der Schnittstelle `ProcessorClass`, da das Master-Servlet Methoden voraussetzt, deren Ergebnisse für die Bereitstellung der Funktionalität benötigt werden. Dieses Interface ist folgendermaßen definiert:

```
import javax.servlet.http.*;
public interface ProcessorClass {
      public void destroy(); // free resources
      public String getCodeDirectory(); // find XML-Documents
      public DBClient getDBClient(); // get access to database
      public void login(HttpServletRequest req); // do additional login
stuff
      public String processTask(HttpServletRequest req, String task); //
process the user-request
} // end of interface
```

Abbildung 24: Interface für die Processor-Klasse

So sollen mittels der Methode `destroy()` alle Ressourcen freigegeben werden, die von dieser Verarbeitungsinstanz noch gebunden sind. Insbesondere die Verbindung zur Datenbank sollte gelöst werden, da sonst je nach Qualität des Treibers immer noch Objekte gesperrt bleiben. Die Methode `getCodeDirectory()` liefert dem MasterWorker das Verzeichnis, in dem nach den XML-Dokumenten gesucht wird und ist natürlich je nach Verarbeitungsklasse unterschiedlich. `getDBClient()` gibt die Referenz auf die Datenbankschnittstelle zurück, damit das MasterServlet nicht unnötigerweise eine eigene Verbindung zur Datenbank aufbauen muss. Der Login-Vorgang und die Berechtigungssteuerung ist eigentlich Aufgabe des Master-Servlets, allerdings kann es sein, dass auch die Verarbeitungsklasse selbst eine eigene Prozedur oder Initialisierung beim Login-Vorgang vornehmen will. Daher muss jede dieser Klassen eine `login()`-Methode besitzen. Die Methode `processTask()` erledigt die eigentliche Arbeit.

Neben diesen Schnittstellenmethoden existieren natürlich noch eine ganze Reihe weiterer Methoden, die nötig sind, um die gewünschte Funktionalität zur Verfügung zu stellen. Die wesentlichen davon werden nun in den folgenden Kapiteln vorgestellt, um die Grundidee dieser Implementierung zu verdeutlichen.

6.2.5.1 Systemsteuerung und Systemverwaltung

Zur Steuerung des Systems liefert der Client in seinem Request einen Parameter (mit dem Namen "Task"), über den ermittelt werden kann, welche Aktion vom Anwender angefordert wurde. Je nach Task werden dann entsprechende Routinen aufgerufen und die Antwort aufbereitet. Die Tasknummern sind dabei im Programm hart mit der entsprechenden Funktionalität verdrahtet, da im Allgemeinen zu jeder Task eine gesonderte Programmierung erforderlich ist. Es existieren zwar auch eine Reihe von Aufgaben, die standardisiert werden könnten, allerdings würde der deutlich erhöhte Arbeitsaufwand für die Erzeugung eines durch den Anwender definierbaren Menüsystems nicht die Arbeitsersparnis bei einer Erweiterung des Systems um neue Funktionen rechtfertigen (und ist auch nicht Bestandteil der Spezifikation). Ein Teil der Tasks wird ohne eine weitere Verarbeitung einfach auf eine neue Seite umgelenkt (z.B. die Task "`A010`" auf die Seite "`QueryLogin.xml`"), da entweder keine Verarbeitung nötig ist, oder die Verarbeitung in der Maske bereits kodiert ist (zum Beispiel eine Datenbankabfrage). Bei anderen Tasks werden vor der Umlenkung bestimmte Eingabefelder vorbesetzt (z.B. bei der Sammelanzeige aller Login-Kennungen wird ein Sternchen vorbesetzt, wenn keine Selektion getroffen wurde, um alle Einträge anzuzeigen). Und wiederum bei anderen Tasks werden erst weitere Methoden aufgerufen und eventuell aufgrund des Returncodes entschieden, welche Seite als nächstes angezeigt werden soll (z.B. wird nach dem Speichern von Änderungen die Sammelanzeige angesteuert, wenn bei der Eingabe oder beim Speichern der Daten kein Fehler festgestellt wurde. Ansonsten wird die Maske mit Hinweis auf den Fehler erneut angezeigt).

Der Source-Code für das Einlesen der Maske und das Anlegen eines neuen Benutzers mit den angegebenen Werten wird in Anhang E.5 Verarbeitung der Benutzereingaben exemplarisch für die Maskeninhalte aller verschiedenen Tasks aufgeführt.

Eine kleine Erweiterung des Feinkonzeptes soll an dieser Stelle noch erwähnt werden. Während der Softwareentwicklung war es häufiger nötig, die Datenbankinhalte und das Datenbankdesign manuell zu manipulieren, da durch fehlerhafte Programmmodule inkonsistente Datenbestände erzeugt wurden oder sich noch nachträgliche Änderungen am Datenbankdesign ergeben haben. Um zu vermeiden, dass dafür jede Änderung ein persönliches Erscheinen am Datenbank-Rechner erforderlich ist, ist der Bereich Systemverwaltung um eine Online-SQL-Schnittstelle zur Datenbank erweitert worden. Diese Schnittstelle ermöglicht (nach einer weiteren Authentifizierung gegenüber der Datenbank) das Absetzen eines beliebigen SQL-Kommandos an jedem Client-Rechner. Die Antworten aus der Datenbank werden dabei generisch als HTML-Tabelle zurückgeliefert. Somit ist es möglich, sich Informationen über Feldnamen, Tabelleneigenschaften oder Datenbankinhalte zu beschaffen und diese zu manipulieren. Die Zugriffe auf die Datenbank werden dabei wiederum über die Datenbankschnittstelle abgewickelt, so dass das SQL-Interface als eine Art Ein-/Ausgabe-Pipeline verstanden werden kann, über die Ein- und Ausgaben der Datenbankschnittstelle aufbereitet und weitergeleitet werden.

6.2.5.2 Autorenwerkzeug zur Erstellung eines Tests

Das Autorenwerkzeug ist eine Gruppe von Tasks, die sich um die Verwaltung der entsprechenden Datenbank-Tabellen kümmern. Diese Verwaltung von Datenbankinhalten ist vom Prinzip her ebenso wie in der Benutzerverwaltung bereits exemplarisch dargestellt, daher sei an dieser Stelle nur auf das obige Kapitel verwiesen. Es werden Tabellen gepflegt, die die Projektparameter enthalten (Name, Beschreibung, Themengebiete), der Pool von Aufgaben kann verwaltet werden (Aufgabenbeschreibung, Aufgabenposition mit Typ und Fragestellung, Lösungspositionen mit den benötigten Werten) und die Aufgaben können einem oder mehreren Fragebögen zugewiesen werden. Für jede dieser Datenbanktabellen existieren eine oder mehrere Masken, die nach dem beschriebenen Prinzip Datenbankinhalte auslesen, anzeigen oder verändern.

Eine der Anforderungen war, dass der Autor keine oder nur wenig HTML-Kenntnisse benötigen sollte, um einen Fragebogen zu erstellen. Ob diese Forderung nun erfüllt ist, ist Auslegungssache, da zwar bei reinen Textdarstellungen einer Aufgabensituation keine weiteren Kenntnisse nötig sind, jedoch ein etwas schickeres Layout oder die Einbindung externer Materialien nur möglich sind, wenn entweder direkt der HTML-Code in die Formularfelder eingetragen wird, oder die erstellte Aufgabe mit einem beliebigen XHTML-Editor nachbearbeitet wird. Durch die Möglichkeit der Nachbearbeitung ist das System offen genug, um die Forderung als erfüllt anzusehen. Verbesserungen des Konzeptes sind aber möglich und werden in Kapitel 8.3 Ausblick und Verbesserungen noch einmal näher behandelt.

Die Methodik der Generierung von Fragedokumenten folgt dabei folgendem Schema:

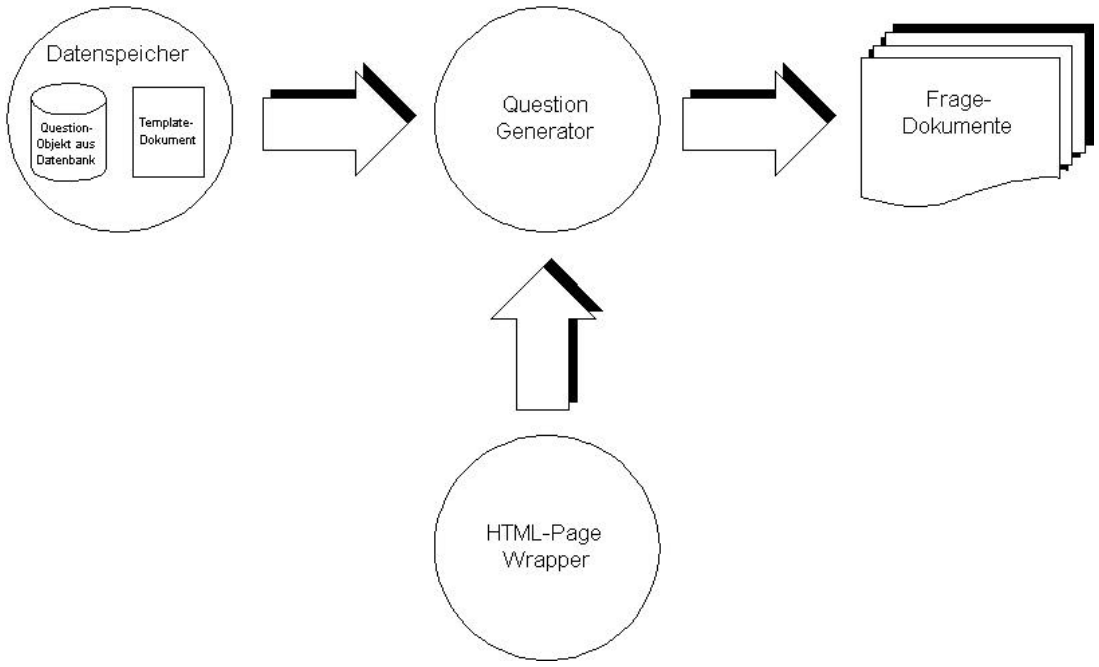

Abbildung 25: Generieren von HTML-Dokumenten

In den Ausgabestrom der angeforderten Aufgabe wird zuerst das hinterlegte Template eingefügt, das die Definitionen zur Optik und für die Navigation enthält. Dann werden alle Datenbankinfor-

mationen gelesen und je nach hinterlegtem Aufgabentyp der entsprechende HTML-Code über die Wrapper-Klasse erzeugt (es wird also beim Aufgabentyp "textarea" der HTML-Code "`<textarea rows=x cols=y></textarea>`" eingefügt). Der erzeugte Dokumentenname wird in die Datenbanktabelle der Aufgabe eingetragen, damit beim Ablauf eines Tests darauf zugegriffen werden kann.

6.2.5.3 Der Ablauf eines Tests bzw. einer Prüfung

Der Ablauf eines Tests bzw. einer Prüfung folgt immer dem gleichen Schema. Zu Beginn wird in einer Datenbanktabelle ein eindeutiger Schlüssel für jeden Test generiert, über den die Reihenfolge und der Bearbeitungsstatus des Tests abgelegt wird. Diese Reihenfolge kann nach Zufall, Schwierigkeitsgrad oder Themengebiet gebildet werden, dabei werden per Zufall eine durch den Anwender festgelegte Anzahl von Aufgaben aus dem entsprechenden Gebiet ausgewählt. Es kann ein bestimmter Fragenbogen gestartet werden, dessen Reihenfolge bereits durch den Autor des Projekts in einer eigenen Datenbanktabelle hinterlegt ist, und es kann eine Auswahl aufgrund der bereits beantworteten Fragen erfolgen, über die das System ermitteln kann, bei welchem Themengebiet und welcher Schwierigkeitsstufe noch Defizite beim Lerner vorliegen. In dieser Datenbanktabelle, in der die Reihenfolge abgelegt ist, wird auch gespeichert, ob eine Frage bereits beantwortet ist (da sie ja auch ohne Eingabe einer Lösungsmenge durch den Anwender als beantwortet gekennzeichnet werden kann). So erkennt das Servlet bei Steuerung des Tests über die Navigationsleiste, welche die nächste beantwortete oder unbeantwortete Aufgabe ist. Etwas abgewandelt vom Prototypen sieht diese Navigation jetzt folgendermaßen aus:

Abbildung 26: Implementierung des Testlaufs

Die Steuerung erfolgt über die Tasten

- "?<" (vorherige unbeantwortete Aufgabe)
- "<" (vorherige Aufgabe)
- ">" (nächste Aufgabe)
- ">?" (nächste unbeantwortete Aufgabe).

Zusätzlich wird in der Navigationsleiste die Positionsnummer und die Gesamtanzahl der Aufgaben angezeigt. Die Tasten

- "X" (Aufgabe unbeantwortet lassen)
- "C" (alle Eingaben löschen)
- "Antwort" (Antwort ablegen und bewerten)
- "Fertig" (Test vorzeitig beenden)
- "Hilfe" und "erweiterte Hilfe" (zum Online-Kurs verzweigen)
- sowie "i" (Informationen zur Aufgabe anzeigen, also z.B. Themengebiet, Schwierigkeitsgrad, erreichbare Punkte etc.)

dienen der weiteren Steuerung des Tests und lassen dem Anwender viel Spielraum bei der individuellen Bearbeitung eines Tests.

Jeder dieser Tests kann nach dem Start an jeder Stelle unterbrochen werden (z.B. durch Beenden des Browser oder durch Anwahl einer neuen Seite), um in der nächsten Sitzung mit ihm fortzufahren. Nach der Beendigung des Testlaufs wird die Gesamtbewertung grafisch angezeigt.

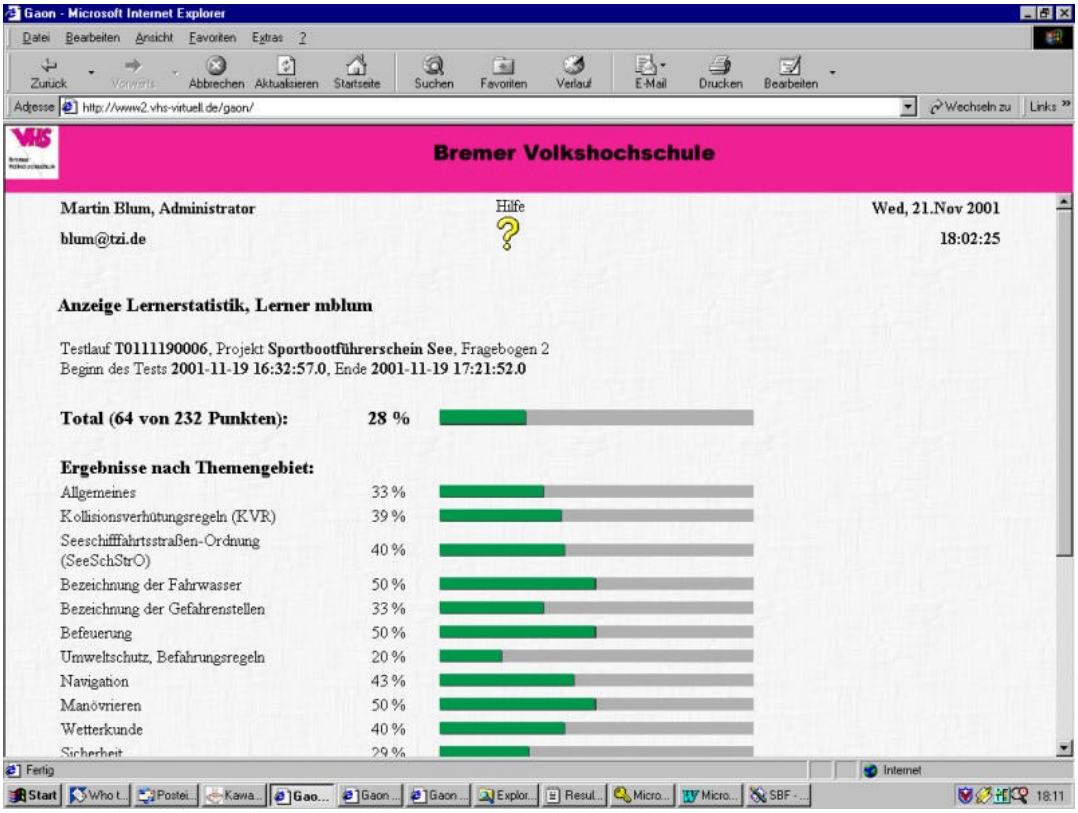

Abbildung 27: Anzeige statistischer Daten eines Tests

6.2.5.3.1 Das Grundkonzept bei der Überprüfung von Aufgaben gegen Lösungen

Vom Prinzip her sehr simpel werden aus dem ausgefüllten Fragedokument die Eingabefelder ausgelesen, indem zu jeder Aufgabenposition alle Inputfelder als Array übertragen werden. Bei der Generierung erhalten alle Felder einer Aufgabenposition den selben Namen ("`input-Var`"+Aufgabenposition), so dass dieses Array automatisch vom Client gelesen werden kann. Jede Position des Arrays muss damit eine zugehörige Position in der Lösungspositionentabelle haben, die dann miteinander verglichen werden.

Zwei Dinge verkomplizieren diesen Vergleich etwas: zum einen muss die Eingabe des Benutzers je nach Aufgabentyp nochmals umgesetzt werden, da das Array zum Beispiel bei Radiobuttons oder Checkboxen (Single- oder Multiple-Choice) anders aufgebaut ist als bei Textfeldern oder Applets. Und zum anderen ist der Vergleich von Checkboxen nur auf richtig oder falsch erforderlich, während z.B. bei Textfeldern auch Alternativlösungen überprüft werden müssen.

Die Umsetzung der Benutzereingaben ist beispielsweise für Checkboxen erforderlich, da der Wert einer Checkbox nur übertragen wird, wenn die Box angekreuzt ist. Somit müssen alle nicht angekreuzten Boxen generiert werden, damit die Reihenfolge der Lösungspositionen wieder mit dem Eingabearray übereinstimmt. Außerdem muss dabei entschieden werden, welche der Boxen angekreuzt wurde. Daher bekommt jede Box im HTML-Code der Aufgabe einen anderen Wert zugewiesen, der zum Wert "1" zurücktransformiert wird, wenn er den Zweck der Positionsnummer erfüllt hat (siehe auch Anhang E.7 Überprüfung von Eingaben gegen die Lösungsmenge).

Die Überprüfung, ob die Eingabe richtig oder falsch war, erledigt die Methode der dazugehörigen Lösungsposition. Für Checkboxen oder Radiobuttons ist das sehr simpel und läuft auf einen reinen Stringvergleich hinaus.

Die Überprüfung der anderen Aufgabentypen ist etwas komplizierter und soll am aufwendigsten Beispiel, der Freitexterkennung, im folgenden Kapitel erläutert werden.

6.2.5.3.2 Das Lösungsmodul zur Erkennung von Freitext

Die Interpretation der menschlichen Sprache ist nicht trivial und findet im Bereich der Computer-Linguistik bereits ein breites Spektrum an Lösungsansätzen. Um halbwegs den Sinn eines Satzes erkennen zu können, muss bereits soviel Vorarbeit geleistet werden, dass das den Rahmen dieser Diplomarbeit gesprengt hätte. So müsste für jedes Wort eines Satzes seine Anwendungsform gefunden werden (d.h. ist es das Subjekt, Prädikat oder Objekt; ist es Plural oder Singular; ist es im Akkusativ, Dativ oder Genetiv; ist es männlich, weiblich oder neutral usw.). Erst dann könnte der Wortstamm in einen Strukturbaum des Satzes eingehangen werden, über den wiederum die Bedeutung ermittelt werden könnte. Dabei sind aber immer noch eine Reihe von Doppeldeutigkeiten möglich, die sich nur über das Umfeld (den Kontext) des Satzes eliminieren lassen würden [Kunze 2001]. Um nun die Antwortsätze gegen die Fragestellung zu vergleichen, wäre wiederum eine Reihe von Möglichkeiten zu hinterlegen, mit welchen Wörtern oder Satzgebilden die Frage beantwortet werden kann (zum Beispiel kann ein Boot gesteuert, gelenkt oder geführt werden. Das sind zwar unterschiedliche Wörter, sie beschreiben jedoch den gleichen Sachverhalt).

Da die Antworten des Anwenders stark mit dem jeweiligen Projektthema verknüpft sind, ist ein so großer Aufwand nicht notwendig. Wenn über umfangreiche Algorithmen nicht gewährleistet werden kann, dass die Bewertung des Computers korrekt erfolgt, erscheint es sinnvoll, durch das System eine Vorbewertung zu ermitteln, die dann nochmals durch einen Menschen geprüft und notfalls korrigiert wird. Auch diese Rückübertragung an einen Tutor mit der Möglichkeit zur Neubewertung ist ein Teil meiner Routine zur Ergebnisermittlung. Allerdings tritt sie nur im Falle einer Prüfung in Kraft, da bei einem Test normalerweise der Anwender für sich lernt und eventuell gerade kein Tutor online ist. Außerdem kann der Anwender zusätzlich zur automatischen Bewertung selbst entscheiden, ob seine Lösung korrekt war, da ihm bei einem Test eine Musterlösung mit einer entsprechenden Begründung angezeigt wird.

Der automatisierte Teil der Freitexterkennung wird in ähnlicher Form auch bei den anderen Aufgabentypen angewendet. Dabei wird zuerst die eingegebene Antwort in eine Reihe von Wörtern zerlegt (was über eine Liste von Sonderzeichen erkannt wird, also Leerschritt, Satzpunktuation, Klammern usw.). Danach wird die Lösungsmenge aufgeteilt in zwei Listen, zum einen in alle Schlagwörter, die in der Lösung vorkommen müssen (getrennt über einen ":"), und zum anderen in alle Negativwörter, von denen keines in der Lösungsmenge vorkommen darf (getrennt über ein "!"). Jedes der Schlagwörter ist nun wiederum eine Liste von alternativen Möglichkeiten (getrennt über ein "/"). Dieses System ist für das benutzte Anwendungsbeispiel ausreichend, obwohl von einer Testperson noch weitere Möglichkeiten gewünscht wurden, die Lösungsmenge etwas umfangreicher ablegen zu können.

Die Überprüfung der Eingabe gegen die Lösung ist ein Durchgehen durch die beiden Listen, wobei ein Schlagwort als erkannt gilt, sowie eine der Alternativen in der Menge der Eingabewörter gefunden wurde. Wird nur ein Teil der Schlagwörter in der Eingabe erkannt, so wird eine prozentuale

Bewertung zurückgegeben. Wird in der Eingabemenge eines der Negativwörter entdeckt, gilt die Eingabe als vollständig falsch beantwortet.

6.2.5.3.3 Statistische Daten zur Lernerbewertung

Wie bereits im obigen Kapitel zu ersehen ist, wird bereits bei der Überprüfung der Aufgabe auch die prozentuale Bewertung dieser Aufgabenposition ermittelt. Über die vom Autor festgelegte Anzahl von Punkten für diese Aufgabe und die Anzahl der angelegten Aufgabenpositionen kann die erreichte Punktzahl dieser Aufgabe ermittelt werden. Diese Punktzahl wird nun in mehreren Datenbanktabellen abgelegt, um daraus ein Profil für den Lerner erstellen zu können. Zum einen wird die Punktzahl innerhalb des Tests gespeichert, um den Lerner direkt anzuzeigen, wie seine eingegebene Antwort bewertet wurde. Zum anderen wird der Punktestand für alle versuchten Lösungen dieser Aufgabennummer festgehalten, um eine durchschnittliche Bewertung je Aufgabe erhalten zu können. Und zum dritten wird die Punktzahl in der Gesamtsumme des Lerners, der Summe für das Themengebiet der Aufgabe und für den Schwierigkeitsgrad der Aufgabe abgelegt. Sinn dieser Datenerhebung ist - neben dem direkten Feedback zu einem Test und somit der Möglichkeit, den Lernfortschritt beurteilen zu können - ein Profil für den Lerner zu erstellen, über das Defizite im Wissensgebiet erkannt werden können. Diese Bereiche können dann gezielt trainiert werden. Die statistischen Daten können ebenso angezeigt werden, wie die Auswertung eines Tests.

6.3 Test der Implementierung

Um den Anforderung an die Korrektheit dieser Software zu genügen, sollen eine Reihe von Testläufen die einzelnen Module überprüfen und validieren. Die angewendeten Testverfahren werden in den nächsten Kapiteln kurz umrissen, die je nach Anwendbarkeit bei den verschieden Einzelmodulen genutzt wurden.

6.3.1 Black-Box-Test

Bei dieser Art von Test wird, wie der Name schon vermuten lässt, das Softwaresystem als eine Black-Box angesehen, d.h. als ein System, deren innere Struktur für den Test unbekannt ist. Die Ausgaben aus dem System – bei vorgegebener Eingabe - werden verglichen mit den aus den Spezifikationen ersichtlichen Erwartungen. Ein Black-Box-Test ist erfolgreich, wenn eine Anzahl an Tests (im Allgemeinen die gängigen Funktionen des Systems) die erwarteten Ergebnisse liefern. Das bedeutet zwar nicht, dass das System fehlerfrei ist, da eventuell ein bestimmter, fehlerhaft umgesetzter Sonderfall, nicht getestet wurde. Allerdings kann so gewährleistet werden, dass die täglichen Arbeiten am System funktionieren werden und somit bereits ein Großteil des Funktionsumfangs fehlerfrei umgesetzt wurde. Außerdem ist dies ein sehr schneller Test, da das Ergebnis sofort ersichtlich ist.

Getestet wurde das Softwaresystem GAON anhand des Kurses "Sportbootführerschein See". Dafür mussten Lerner angelegt, die Aufgaben erfasst und die Tests auf Vollständigkeit und Korrektheit überprüft werden. Einen Teil dieser Arbeiten wurde dabei von mir übernommen, so dass ich sehr

schnell selbst gemerkt habe, wo der Programmablauf von den Anforderungen abgewichen ist. Ein Teil der Datenerfassung wurde durch eine externe Kraft erledigt, die nicht nur das Ergebnis betrachtet hat, sondern auch neutraler (als ich) die Oberfläche und Bedienbarkeit bewertet hat. Die Tests selbst wurden natürlich auch von den Kursteilnehmern absolviert, die dann durch die Vielfältigkeit von unterschiedlichen Eingaben einen Test sehr nahe an der späteren realen Anwendungssituation durchgeführt haben (siehe auch Kapitel 7 Evaluation).

6.3.2 White-Box-Test

Der White-Box-Test wählt im Gegensatz zum Black-Box-Test die Testfälle in Kenntnis der inneren Struktur des Softwaresystems aus. Die Testfälle sollten so gestaltet sein, dass jeder Zweig des Programms einmal durchlaufen wird. Danach werden dann das Ausgabeverhalten und die Datenmanipulation gegenüber den Spezifikationen geprüft. So kann jeder Sonderfall getestet und damit ein fehlerfreies System erstellt werden (zumindest in der Theorie). In der Praxis ist diese Art von Test jedoch mit dem Anspruch der Vollständigkeit kaum noch durchführbar, da die Anzahl der zu testenden Fälle mit der Größe des Gesamtsystems exponentiell steigt. Aber da für die Kenntnisse der inneren Struktur des Systems auf jeden Fall die Programmierer am Testlauf beteiligt werden müssen, können die Testfälle auf die entscheidenden und "interessanten" Verzweigungen beschränkt werden. Dadurch kann zwar nicht mehr eine vollständige Korrektheit gewährleistet werden, es werden aber zumindest auch Fehler in Sonderfällen oder nicht so häufig genutzten Routinen entdeckt.

White-Box-Tests konnte ich natürlich nur selbst ausführen, da ansonsten keiner die Spezifikationen und die interne Struktur des Systems kennt. Für diese Art von Test ist allerdings auch kein Massentest erforderlich, da es ja ausreicht, wenn ein bestimmter Programmzweig einmalig auf Korrektheit geprüft wurde.

6.3.3 Topdown-/Bottomup-Test

Ähnlich der Topdown- oder Bottomup-Entwicklung von Softwaresystemen existieren auch entsprechende Testverfahren. Dabei wird während der Implementierung bereits geprüft, ob Methoden und Module bereits das richtige Ergebnis liefern, auch wenn abhängige Programmteile noch nicht fertiggestellt sind. Die Ergebnisse dieser nicht fertiggestellten Programmteile werden dabei simuliert, indem das richtige Ergebnis geliefert (ge*fake*t) wird. So kann bereits in einem sehr frühen Stadium begonnen werden zu testen und somit die Größe des zu testenden Systems eingeschränkt werden. Dabei unterscheidet sich der Bottumup-Test vom Topdown-Test durch die Reihenfolge, welche Verfeinerungsstufe getestet wird. Beim Bottumup-Test geht man von innen nach außen vor, d.h. es wird mit dem Testen der elementaren Bausteine und Algorithmen begonnen, danach werden einzelne Methoden, dann größere Module und zum Schluss die Integration getestet. Beim Topdown-Test läuft es genau umgekehrt, hier wird zuerst die Hauptsteuerung implementiert und getestet, danach der Ansprung der einzelnen Module usw.

Diese Art zu Testen hat einige Vorteile. Da bereits während der Implementierung getestet wird, fallen Designfehler zu einem frühestmöglichen Zeitpunkt auf und können dann eventuell direkt

behoben werden. Der Entwicklungsstand kann relativ einfach erkannt und somit auch die Akzeptanz durch die Benutzer überprüft werden. Das Bereitstellen von Testumgebungen kann oftmals dabei entfallen.

6.3.4 Code-Inspektion

Eine relativ simple, allerdings sehr nützliche und effektive Methode zum Auffinden von Entwurfs- und Implementierungsfehlern ist die Code-Inspektion. Dabei wird der Source-Code des Programmsystems Schritt für Schritt vom Autor (und sinnvollerweise von weiteren Mitarbeitern des Projekts) betrachtet und in seiner Logik bewertet. Da die Schritte nachvollziehbar sein sollten (ansonsten ist der Code zu komplex programmiert worden), müsste sich der Code wie ein gutes Buch lesen lassen und Fehler in der Logik oder im Design dem Testteam während der Betrachtung auffallen.

Diese Art zu testen ist allerdings bei großen Programmsystem sehr mühsam, da die Aufmerksamkeit des Testteams nach einiger Zeit schwinden wird und somit Fehler eventuell übersehen werden. Außerdem fällt es schwer, sich alle internen Zustände innerhalb eines Moduls zu merken, anders als der Computer, der beim Ablauf mit der Speicherung solcher internen Zustände keine Schwierigkeiten haben wird.

Ich habe versucht, bei diesem Softwareprojekt eine Code-Inspektion durchzuführen und sie dabei aber auf die wesentlichen Funktionen bzw. auf aktuell bearbeitete Routinen beschränkt. Es standen keine weiteren Mitarbeiter zur Verfügung, denen der Programmablauf hätte erklärt werden können, so dass hier nur ein Teil des Tests absolviert wurde.

6.3.5 Leistungstest

Bei einem Leistungstest werden die nicht-funktionalen Anforderungen und die Stabilität des Softwaresystems überprüft. Da Anforderungen an Rechnerauslastung und Antwortzeitverhalten ebenso wie das Verhalten über einen längeren Zeitraum im Allgemeinen nicht näher spezifiziert werden, können die Ergebnisse dieses Tests nur mit dem persönlichen Eindruck oder dem allgemein akzeptierten Standard verglichen werden (wobei hier die Stabilität von windows-basierten Betriebssystemen nicht als allgemein akzeptierter Standard gelten soll). Diese Leistungstests werden oftmals durch externe Tools durchgeführt, die z.B. die Rechnerauslastung vor und nach der Installation des Softwaresystems messen. Ob ein Antwortzeitverhalten als gut oder als schlecht zu bewerten ist, ist ein subjektiver Eindruck. Allerdings hängt das auch von der Art des Einsatzes der Software ab, da eine reine Datenerfassung irgendwann durch den Anwender blind erfolgen wird und sich Verzögerungen dann sofort negativ bemerkbar machen, während im vorliegenden Fall ein Anwender erst über die Lösung nachdenken wird, bevor er zur Beantwortung schreitet und dabei durch den Anwender bedingte Pausenzeiten eintreten werden.

Ein guter Leistungstest kann erst erstellt werden, wenn sich das System im realen Einsatz befindet und von einer größeren Anzahl Anwender benutzt wird. Im Moment arbeitet das System innerhalb akzeptabler Grenzen, da die Antworten vom Server in einer Geschwindigkeit geliefert werden, die

sich nicht merkbar von Antworten mit fest abgelegten Seiten unterscheidet (solange nicht mit einer XML-Validierung gearbeitet wird).

6.4 Integration in die Umgebung von vhs-virtuell

Das folgende Kapitel soll kurz abhandeln, welche Schritte nötig waren, um mit der implementierten Software tatsächlich arbeiten zu können. Dazu musste das System vom Entwicklungsrechner in die bestehende Systemumgebung der Volkshochschule Bremen integriert werden, was eine Reihe von nicht vorhergesehenen Schwierigkeiten verursacht hatte. Diese Schwierigkeiten treten vermutlich bei jedem größeren Softwareprojekt auf, so dass die getroffenen Maßnahmen noch als Teil der Projektphase Implementierung anzusehen sind.

Die zur Entwicklung eingesetzte Datenbank "Microsoft Access" bietet durch die komfortable grafische Oberfläche eine gute Basis für das Design einer Datenbank, hat aber für den realen Einsatz eines Softwareproduktes einige gravierende Nachteile. Die Zugriffe sind auf die Datenbank nur über die JDBC-ODBC-Bridge möglich, die alle Anfragen aus dem JDBC an den ODBC-Treiber weiterleitet. Durch diesen Zwischenschritt leidet die Performance bei Zugriffen auf die Datenbank, die bereits unter dem relativ langsamen ODBC-Treiber nicht besonders gut war. Ein von mir durchgeführter Benchmarktest mit 1 bis 100 Lese- und 1 bis 100 Schreibzugriffen hat zwischen der jetzt im Einsatz befindlichen mySQL-Datenbank und der MS-Access-Datenbank einen Geschwindigkeitsvorteil ca. um den Faktor 2 herum für das Schreiben und Lesen von Sätzen ergeben. Beim Lesen von Datensätzen mit dem mySQL-Treiber war der durchschnittliche Zugriff pro Satz bei etwa 0,4ms. Demgegenüber lag der Zugriff auf die MS-Access-Datenbank über den ODBC-Treiber bei einer durchschnittlichen Zugriffszeit zwischen 0,80-0,96ms. Neben diesen besseren Zugriffszeiten steht aber zusätzlich die höhere Ausfallsicherheit und die bessere Funktionalität von mySQL als Argument, im realen Einsatz auf diese Datenbank zu wechseln.

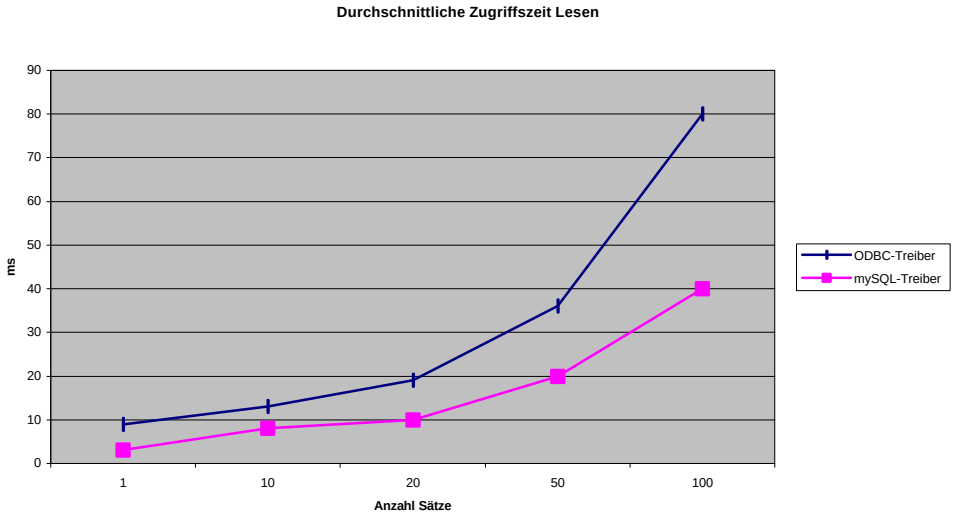

Abbildung 28: Vergleich Zugriffszeiten zwischen ODBC und mySQL

Da bereits während der Entwicklung Inhalte in die Datenbank eingeflossen sind, stellte sich die Frage der Migration zwischen den beiden Datenbanken. Vorteilhafterweise sind beides recht populäre Datenbanken, die von vielen Entwicklern benutzt werden, so dass es hierfür einige frei verfügbare Softwareprodukte gibt. In meinem Fall habe ich das Produkt "MS-Access to mySQL" der Firma Cynergi benutzt, das über die Webseite von mySQL kostenfrei vertrieben wird [Freire 1998]. Da der Entwurf der Datenbankschnittstelle auch eine Export- und Importfunktion vorsieht, stellt sich die Frage, warum nicht diese Funktionen dafür benutzt wurden. Nun, die Beantwortung ist dabei relativ einfach. Es sollte so früh wie möglich erkannt werden, ob das System innerhalb der Umgebung der Volkshochschule lauffähig ist und zu diesem Zeitpunkt gab es die Import- / Export-Funktionalität noch nicht (auch nicht im Entwurf). Erst durch die Notwendigkeit der Migration ist aufgefallen, dass eine entsprechende Funktionalität in der Software fehlt und erst dann in den Entwurf eingefügt worden.

Die zweite große Teilaufgabe war die Einbindung der Servlets in den Webserver der VHS (WebSite von Allaire + Jrun Servlet-Engine) im Gegensatz zum Sun Webserver auf dem Entwicklungsrechner. Der Sun Webserver hat den Vorteil einer sehr geringen Parametrisierung und ist bereits von Haus aus auf die Entwicklung von Servlets ausgelegt. Dadurch können sehr schnell Servlets zum Laufen gebracht werden, da Fehler zwangsläufig im eigenen Code zu suchen sind und nicht durch falsche Einstellung der Servlet-Engine verursacht werden (Minimierung der Fehlerquellen). Der Webserver "WebSite" wiederum bietet die Möglichkeit, verschiedene Domains oder Mappings zu verwalten, was dazu führte, das eine Reihe von Parametern gefunden werden mussten, die nötig waren, um

a) die Servlet-Engine zu aktivieren.

b) bei Eingabe einer bestimmten Adresse den Anwender zu den richtigen Klassen des Servlets umzuleiten.

Außerdem hatte der Webserver eine andere Verzeichnisstruktur als der Entwicklungsrechner, so dass der Suchpfad für lokale Daten über einen Startparameter gesetzt werden musste.

Die dritte größere Hürde bei der Integration in die bestehende Umgebung der Bremer Volkshochschule war die Tatsache, dass die Servlet-Engine von Jrun nur in einer veralteten Version installiert war und keine Wartungsverträge vorhanden waren, die ein kostenloses Upgrade ermöglicht hätten. So unterstützt die eingesetzte Jrun-Version nur Java 1.1 (JDK1.1.5), während die Software mit der aktuellen Java-Version 2.0 (JDK1.3.0_02) entwickelt wurde. Für den realen Einsatz war also ein Downgrade auf die entsprechende Version nötig, was eine Umprogrammierung einzelner Funktionen nötig machte (da ein Teil der Java-Methoden in dieser Version noch nicht vorhanden war). Diese Umprogrammierungen waren allerdings noch relativ überschaubar, da die relevanten Änderungen dieser Version (außer im Bereich Security) oft nur Vereinfachungen für den Programmierer oder Korrekturen von Fehlern waren.

Kapitel 7

Evaluation

Zur Überprüfung der gesteckten Ziele schloss sich der praktischen Arbeit an der Software eine Evaluationsphase an, in der über Interviews und Fragebögen die Akzeptanz des neuen Softwaresystems mit den Kursteilnehmern diskutiert wurde. Abschließend wurde die Software anhand der in Kapitel 3 aufgeführten Kriterien bewertet.

7.1 Die benutzten Fragebögen

Die Software sollte von den drei hauptsächlichen Benutzergruppen Lerner, Tutor und Autor hinsichtlich der Punkte ergonomisches Design, Performance, Bedienbarkeit und Effekt auf das Lernverhalten hin untersucht werden. Die eigenständige Benutzergruppe der Administratoren ist hier nicht mehr gesondert befragt worden, da sich die Oberfläche von Autoren und Administratoren sehr stark ähnelt. Für diese Befragung wurden die in Anhang B vorgestellten Fragebögen entworfen. So konnte ein Feedback zu allen Funktionen der Software erreicht werden, wobei die Anzahl der abgegebenen Fragebögen bei Tutoren und Autoren für ein repräsentatives Ergebnis eigentlich zu klein ist. Ausgefüllt und abgegeben wurde der Bogen durch 15 der 20 Teilnehmer, den 2 Tutoren und dem Autorenteam.

7.2 Auswertung der Fragebögen

Die abgegebenen Fragebögen lassen aus der Sicht eines **Lerners** folgende Rückschlüsse zu:

- 66 % der Lerner haben die Software nicht genutzt. Dieser doch unerwartet hohe Anteil hatte mehrere Ursachen, die insofern spannend waren, da es anscheinend grundlegend an Akzeptanz für die Nutzung einer solchen Software fehlte. Zum einen wurde die Entscheidung zur Entwicklung der Software mit einem Tutor getroffen, der relativ früh aus dem Team ausgeschieden ist, da ihm der Arbeitsaufwand zu hoch war. Der dann folgende Tutor hat die Software etwas blockiert, da er sich in seiner freien Kursgestaltung eingeschränkt gesehen hat. Trotzdem hat die Leitung der VHS die Softwareentwicklung auch über seinen Kopf hinweg weiter voran-

getrieben. Von diesen rein menschlichen Problemen einmal abgesehen, wurden von den Teilnehmern aber noch weitere technische und organisatorische Gründe genannt.

1. Die meisten Teilnehmer hatten bereits einen Satz Fragebögen auf Papier erworben. Dadurch ergab sich keine Notwendigkeit, über das Internet zu lernen, da ja auch die tatsächliche Prüfung auf dem Papier stattfindet. Die Vorteile dieses neuen Lernverfahrens wurden nicht so hoch bewertet, dass es für ein Ausprobieren gereicht hätte.
2. Die Kosten für die Internetnutzung, die während des Tests anfallen, sind stark bemängelt worden, da es erforderlich ist, für den kompletten Zeitraum online zu sein.
3. Das Lernen am Computer ist als schwieriger und komplizierter eingeordnet worden, als das Lernen auf Papier. Zum Teil wurde die Bedienung des Computers an sich kritisiert (Tippaufwand gegenüber Schreiben), zum Teil die Störanfälligkeit von Hardware und Telefonleitung und zum Teil die Zeitspanne bis das Lernen begonnen werden konnte (Computer hochfahren, Verbindung aufbauen, langsames Modem).

- Von den Teilnehmern, die die Software genutzt haben, wurde Optik und Antwortzeitverhalten als gut beschrieben, allerdings gab es Verbesserungsvorschläge für die Bedienung der Oberfläche.
- Die Korrektheit der Auswertung erschien den meisten Anwendern akzeptabel, allerdings fehlten noch Daten zur besseren Erkennung der verschiedenen Eingabemöglichkeiten.
- Hinsichtlich der Kommunikation über das Internet zur Lösung von Problemen herrschte eine geteilte Auffassung. 60 % empfanden die angebotenen Mechanismen ausreichend, während der Rest der Lerner das Internet an sich nicht als geeignet für solche Diskussionen ansah.
- 60 % der Kursteilnehmer, die das System getestet haben, empfanden das System als gute Ergänzung zum Kurs, würden allerdings nicht auf die realen Fragebögen verzichten wollen.

Die zwei **Tutoren**-Fragebögen lassen folgende Rückschlüsse zu:

- Es ist notwendig, dass der Tutor hinter der Software steht, da er ihre Vorteile sonst sehr leicht aushebeln kann.
  1. Die Kursteilnehmer müssen von ihm geschult und motiviert werden
  2. Die gestellten Fragen innerhalb der Foren müssen regelmäßig von ihm gesichtet und beantwortet werden
  3. Fragebögen als Prüfung müssen regelmäßig – zumindest aber häufiger, als die stattfindenden Kursabende – von ihm kontrolliert werden.
- Auch die Tutoren bewerteten die Oberfläche als übersichtlich und das Antwortzeitverhalten als gut.
- Die Korrektheit der Testauswertung ergab ein ähnliches Bild, wie bei den Lernern.
- Die Gegenbewertung der abgegebenen Lösungen erschien beiden Tutoren nicht einfacher als die Bewertung realer Fragebögen, aber doch zumindest praktikabel.
- Die Kommunikationsmechanismen wurden auch von den Tutoren ambivalent beurteilt.

Das **Autorenteam** hat folgende Bewertung abgegeben:

- Sie bewerteten die Oberfläche als zweckmäßig, aber verbesserungsfähig und das Antwortzeitverhalten als gut.
- Für die Erfassung der Aufgaben ist Grundwissen in HTML hilfreich, die eingesetzte Lösung ist aber praktikabel. Die Nachbearbeitung der generierten Fragen ist nur mit Grundwissen in HTML möglich.
- Als Verbesserung der Software sollte eine FTP-Schnittstelle das Hochladen der Aufgabenmaterialien vereinfachen.
- Die Anzahl der verfügbaren Aufgabentypen war durch die parametrisierbare Applet-Schnittstelle ausreichend.
- Der Import und Export von Daten des Systems verringert die Entwicklungszeit von Onlinekursen, da die Daten aus Fremdsystemen übernommen werden können. Ebenso ist durch diese Schnittstelle ein verteiltes Arbeiten innerhalb der Gruppe möglich, da über die Import-Funktion die Datenbestände zusammengemischt werden können.

7.3 Interviews

Die geführten Interviews und Diskussionen ergaben ein ähnliches Bild wie die Auswertung der Fragebögen, hier ergaben sich noch einige Anregungen und Verbesserungsvorschläge, die sich im nächsten Kapitel wiederfinden. Insgesamt konnte ich feststellen, dass viele Kursteilnehmer erst einmal offen neuen Möglichkeiten gegenüberstehen, jedoch die Benutzung und die daraus resultierende Akzeptanz eines neuen Softwareproduktes von vielen weiteren Faktoren abhängt. Insbesondere Schulung und Motivation ist ein Aufgabenfeld, dass sehr stark den Erfolg oder Misserfolg eines solchen Projektes mit trägt. Eine durchdachte Marketingstrategie würde dementsprechend gut helfen. Ebenso wurde genannt, dass eventuell mehr Zeit nötig ist, um den Bekanntheitsgrad zu steigern. Ein Wissen um eine solche Software hätte einen Teil der Teilnehmer eventuell dazu bewogen, auf den Kauf der Fragebögen zu verzichten.

Als einen entscheidenden Vorteil gegenüber bestehenden Softwarelösungen wurde im erstellten Produkt die Freitextauswertung hervorgehoben, da sie den Realitätsgrad des Tests erhöht. Laut vieler Teilnehmer macht es keinen großen Sinn anhand von Multiple-Choice-Aufgaben zu lernen, wenn in der realen Prüfung gefordert ist, einen zusammenhängenden Text zu schreiben.

Eine Änderung des Gebührenmodells für den Zugriff auf das Internet und das Fortschreiten der Entwicklung im Hardwarebereich wird dazu beitragen, dass die Nutzung dieser Software für die nächsten Kursteilnehmer immer attraktiver werden wird.

7.4 Abschließende Bewertung

Abschließend wurde die Software nun noch einmal anhand des in Kapitel 3 aufgeführten Kriterienkatalogs überprüft und sie somit bezüglich der Anforderungskriterien mit den bestehenden Produkten verglichen.

Die Bewertung der einzelnen Kriterien sah im Detail folgendermaßen aus:

Kriterium	**Beschreibung**	**Bewertung**	**gewichtete Bewertung**
Kommunikationsmechanismen	E-Mail, Messageboard, Newsforen vorhanden	2	0,4
Art der Implementierung	Servlet	1	0,1
Antwortzeitverhalten	gut, abhängig von Serverauslastung und allgemeiner Netzlast	2	0,2
Autorentool verfügbar	ja, HTML-Kenntnisse sind aber hilfreich	3	0,3
Bedienungskomfort	gut durch grafische Oberfläche, aber verbesserungsfähig	3	0,15
verfügbare Aufgabentypen	Single-/Multiple-Choice, Lückentext, Freitext, beliebige Aufgabentypen durch Appletschnittstelle	1	0,3
System-/Browsereinschränkungen	Client läuft auf jedem handelsüblichen Browser, Server auf jedem Webbrowser, der Servlets verarbeitet	1	0,15
Endergebnis			**1,6**

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurde ein überaus komplexes System implementiert, das den Anwendern eine sehr breite Funktionalität bietet. Aufgrund der zeitlichen Einschränkungen sind diese Funktionen allerdings nicht immer bis ins letzte Detail ausprogrammiert. Ein etwas enger gesteckter Aufgabenrahmen für diese Diplomarbeit hätte es ermöglicht, den Fokus nur auf bestimmte Funktionen zu lenken. Trotzdem hat die Software den gewünschten Effekt im Einsatz. Die noch etwas schleppende Akzeptanz der Anwender habe ich versucht zu begründen, und ich denke, dass sie mit der Zeit wesentlich steigen wird. Eine Diplomarbeit soll in die Zukunft schauen und neue Ansätze vorantreiben. Gerade die Entwicklung im Bereich der Gebühren für DSL-Anschlüsse und Flatrate zeigen, dass es nur noch eine Frage der Zeit ist, bis sich die eingesetzte Technologie auch für die Kursteilnehmer eines Volkshochschulkurses lohnen wird.

8.2 Bewertung

Die Vorgehensweise in dieser Arbeit und die dabei angewendeten Methoden haben sich insgesamt gut bewährt. Mit Hilfe fundierter Grundlagen ist es gelungen, die wesentlichen Probleme der Aufgabenstellung zu lösen. Die Software wurde auf die Bedürfnisse der Bremer Volkshochschule bzw. dem Projekt "vhs-virtuell" zugeschnitten, aber bei der Entwicklung darauf geachtet, einen lebendigen Softwarezyklus zu ermöglichen und das System durch die Schnittstellen und die Benutzung des XML-Standards auch für spätere Entwicklungen offen zu gestalten.

8.3 Ausblick und Verbesserungen

Wie bereits oben angedeutet, wird die zukünftige technische Entwicklung eine höhere Akzeptanz bei den Anwendern hervorrufen. Um die Software für sie noch interessanter zu gestalten, sind aber auch noch eine Reihe von Verbesserungen denkbar.

- Bei Prüfungen ist oft ein zeitlicher Rahmen gesteckt, der durch das System überwacht werden könnte. So wäre zum Beispiel eine Anzeige der verbleibenden Restzeit und ein automatisches Beenden des Tests nach Ablauf denkbar.
- Eine Reihe von Shortcuts und eine Verminderung der nötigen Benutzereingaben würde die Eingabegeschwindigkeit weiter verbessern. So fehlt beispielsweise bei der Rückübertragung der gemachten Eingaben ein Link, um wieder direkt zur Aufgabe zurückzuspringen.
- Die Erfassung der Lösungsmenge basiert noch auf einem einzelnen Eingabefeld. So muss sich der Autor mit den Sonderzeichen auseinandersetzen, die für die Trennung der Schlagwortlisten nötig sind. Hier wäre eine komfortablere Eingabemaske denkbar.
- Es fehlt ein schneller Trainingsmodus (z.B. durch reine Single-Choice-Aufgaben), der zwar nicht die Prüfung simuliert, aber zum Testen des Wissens einen entscheidenden Geschwindigkeitsvorteil für den Anwender bringt.
- Neue Einträge im Newsforum sollten eine E-Mail an den Tutor auslösen, damit dieser schneller bemerkt, wenn Fragen zu klären sind.
- Die Auswertung von Freitextaufgaben kann durch einen verfeinerten Algorithmus bessere Ergebnisse liefern. Im Moment ist es nötig, in der Eingabe alle Schlagworte zu nennen (in einer der hinterlegten Alternativen). Manchmal ist es aber sinnvoll einen Teil der Schlagworte zu Gruppen zusammenzufassen und Alternativen auf Ebene dieser Gruppen zu ermöglichen.
- Innerhalb des Autorentools könnte auf die Benutzung eines externen Editors eher verzichtet werden, wenn eine Reihe der klassischen externen Materialien (Bilder, Audio- oder Videodateien) über entsprechende Funktionen eingebunden werden können.
- Es sind noch weitere Aufgabentypen denkbar (z.B. Zuordnungsaufgaben oder Image-Flip-Buttons), die zwar nur eine andere Darstellung bestehender Aufgabentypen sind, aber sich multimedial wesentlich besser aufbereiten lassen.
- Bei der Gegenbewertung durch den Tutor könnte die Software erkennen, wenn die hinterlegten Daten der Lösungsmenge inkorrekt sind und dann einen entsprechenden Hinweis an den Autor senden. Ein eigenständiger Lernmodus wäre dabei die Krönung, aber vermutlich recht schwierig zu implementieren.

Anhang A: kleiner Exkurs in XML

<XML>

Die Extensible Markup Language XML ist eine Sprache zur digitalen Abbildung von Dokumenten mit dem Hintergedanken diese Dokumente in einer definierten Form automatisiert weiterverarbeiten zu können. Das kann z.B. das Speichern, das Übertragen, das Durchsuchen, das Drucken oder Anzeigen dieser Dokumente sein. Damit der Computer sinnvoll mit diesen Dokumenten arbeiten kann, benötigt er Informationen darüber, welche Struktur das Dokument hat. XML ist daher eine Metasprache, um Dokumentenbeschreibungen zu definieren (also Grammatiken für eine Beschreibungssprache zu notieren). Sie ermöglicht dadurch ein einheitliches, universelles Datenformat, das insbesondere im Hinblick auf das World Wide Web den Datenaustausch stark vereinfacht.

In dieser Arbeit wurde an einigen Stellen auf XML zurückgegriffen und daher soll hier nochmals intensiver auf diesen Standard eingegangen werden. Gerade zum Datenaustausch in heterogenen Netzen zeigt sich die breite Anwendungspalette von XML und so integriert sie sich z.B. in den Bereichen Import und Export in die Datenbank und Definition von logischen Strukturen innerhalb der Masken in das implementierte System.

Dieser Anhang soll klären, was XML ist, und anschließend einen Überblick über die wichtigsten Bestandteile von XML und deren Einsatz verschaffen. Es soll jedoch keine umfassende Einführung in XML darstellen und erhebt keinen Anspruch auf Vollständigkeit. Detaillierte Informationen finden sich in der einschlägigen Literatur ([Goldfarb 1999], [Seeboerger 2000], [Behme 1998], [W3C 2000b]).

A.1 Die Charakteristika von XML

Im wesentlichen zeichnet sich XML durch folgende Vorteile aus:

a) XML unterstützt eine *allgemeine* Dokumentenabbildung, da aufgrund der im ASCII-Code formulierten Markup's die Dokumente plattformunabhängig, programmiersprachenunabhängig und protokollunabhängig sind. ASCII ist hierbei der kleinste Nenner zwischen verschiedenen Computersystemen und ermöglicht darüber hinaus ebenso ein gutes Verständnis für menschliche Anwender, da XML-Dokumente im Klartext lesbar und selbstbeschreibend sind.

b) XML kann einen *spezifischen* Bezug zu Dokumenteninhalten schaffen. Diese Eigenschaft ist die Erweiterbarkeit bzw. eben das X in XML. Es existieren also keine vordefinierten Tags für die Dokumentenstruktur, sondern der Anwender beschreibt diese über XML (daher der Begriff Meta-Sprache). Dabei wird in XML zwischen Inhalt und Format der Informationen unterschieden und Formatierungsvorschriften (also die Darstellungsattribute

bzw. das Layout der Datenansicht) in Stylesheets verfasst, die dann mit dem XML-Dokument verknüpft werden können. Durch diese Abstraktion können Daten leicht in verschiedenen Darstellungen präsentiert werden.

c) XML erlaubt ein *regelgebundenes* Markup, so dass Computersysteme die Integrität von Dokumenten überprüfen können. Diese Regeln und die Struktur des Dokumentes werden in einer sogenannten Dokumententyp-Definition (DTD) hinterlegt und können dann von Parsern überprüft werden. Diese Parser müssen nicht selbst programmiert werden, da die XML-Dokumente zwar individuell verfasst werden, sie aber auf einem Regelwerk beruhen, das durch die Vorgaben des W3C-Konsortiums wohldefiniert ist. Mittlerweile existieren solche Parser frei verfügbar auf verschiedenen Plattformen und in verschiedenen Programmiersprachen, so dass an dieser Stelle Kosten und Entwicklungszeit eingespart werden kann.

Doch XML hat nicht nur Vorteile, es muss auch der entsprechende Overhead betrachtet werden, durch den die oben genannten Vorteile ermöglicht werden. So muss

a) zu jedem Dokument immer auch die Dokumenttyp-Definition übertragen werden, was natürlich höhere Übertragungskosten und eine längere Übertragungsdauer nach sich zieht.

b) Das ASCII-Format ist nicht sehr effizient in seiner Darstellung, da nur ein Teil der 256 Möglichkeiten eines Bytes ausgenutzt werden. Zahlen werden nicht binär gepackt, genauso wie Texte mit häufig wiederholenden Zeichen nicht komprimiert werden.

c) Hinzu kommt, dass die Markups sich für jeden Anfang eines Elementes und für jedes Ende eines Elementes wiederholen. Stellt man sich die Darstellung einer Datenbanktabelle beispielsweise vor, so werden bei einem Datensatz mit bekannten Satzaufbau ausschließlich die Daten übertragen, während in einem XML-Dokument immer wiederkehrend alle Tags für jedes Feld und jeden Satz mit übertragen werden.

Die Sprache XML hat den ernormen Schub der letzten Jahre allerdings gerade durch ihre Unabhängigkeit von proprietären Binärformaten erreicht. Die höheren Kosten für die Übertragung fallen durch die immer weiter fortschreitende Entwicklung in der Datenübertragungsgeschwindigkeit dabei kaum ins Gewicht.

A.2 Die Sprachelemente von XML

Die wichtigsten Sprachelemente von XML teilen sich in die folgenden Gruppen auf.

A.2.1 Dokumenttypen

Die Menge aller relevanten Dokumente teilt sich in unterschiedliche Kategorien auf, die sich durch ihre Elemente definieren. So unterscheiden sich beispielsweise Sachbücher, Romane und Telefonbücher voneinander, wobei das charakteristische Merkmal von Telefonbüchern wahrscheinlich die Auflistung von Namen mit dazugehörigen Telefonnummern sein wird, unabhängig vom Einband oder vom Titel. Diese Kategorien werden in XML formalisiert in den Dokumenttyp-Definitionen, die eine Reihe von Definitionen für Elemente, Attribute und Entities enthalten. Des weiteren ent-

halten sie die Regeln, welche davon innerhalb des Dokuments an welchen Stellen zulässig sind. Innerhalb eines Dokumentes wird dann deklariert, welche Dokumenttyp-Definition zu verwenden ist.

Dokumente werden ihrerseits unterschieden nach Wohlgeformtheit und nach Gültigkeit. Ein Dokument ist dabei wohlgeformt, wenn es ein gültiges Markup enthält, also zum Beispiel jedes Tag auch wieder beendet wird. Gültig ist ein Dokument, wenn es anhand einer DTD validiert werden kann, das heißt z.B., wenn jeder benutzte Tag auch als Element in der DTD definiert ist und die Struktur des Dokuments konform zur DTD ist.

A.2.2 Elemente

Elemente beschreiben die logische Struktur eines Dokuments und bilden dabei jede logische Komponente ab. Durch die Verwendung eines Wurzelelementes und der Möglichkeit der Verschachtelung der Elemente untereinander kann somit das Dokument in eine baumartige Struktur gebracht werden. Ein Buch teilt sich beispielsweise auf in Kapitel und Artikel, wobei ein Artikel wiederum aus Kapiteln, Artikeln, Absätzen oder Abbildungen bestehen könnten. Natürlich kann ein Element auch ausschließlich aus Zeichendaten bestehen, in diesem Fall spricht man nicht von Zweigen, sondern von Blättern.

Ein Element wird in einer DTD durch die Zeichen `"<!ELEMENT"` eingeleitet. Danach folgen die darunter angeordneten Zweige und die Festlegung, in welcher Reihenfolge und Anzahl das geschehen darf. Dabei wird zwischen folgenden Möglichkeiten unterschieden: Für die Reihenfolge gibt es die Sequenz (`Zweig1, Zweig2`) und die Alternative (`Zweig1|Zweig2`). Für die Anzahl gibt es die Möglichkeit genau einmal (ohne Anhängsel), einmal oder keinmal (? angehängt), mehrfach 0..n (* angehängt) und mehrfach 1..n (+ angehängt).

A.2.3 Attribute

Attribute werden in einer DTD definiert, um die Elemente näher zu spezifizieren. Beispielsweise könnte über ein Attribut der Name einer Export-Anfrage hinterlegt werden oder bei einer Adresse zwischen privat und geschäftlich unterschieden werden, ohne dass alle gemeinsamen Elemente (wie z.B. Ort, Strasse, Telefon) doppelt mit unterschiedlichen Namen aufgeführt werden müssen.

Ein Attributdefinition hat den syntaktischen Aufbau:

```
<!ATTLIST Tagname Attributname (Attributwerte) Option>
```

Mehrere alternative Attributwerte werden durch ein "|" getrennt, wobei ein beliebiger Wert durch den Begriff "CDATA" deklariert wird. Über die Option wird festgelegt, ob dieses Attribut optional ist (`#IMPLIED`), oder nicht (`#REQUIRED`).

A.2.4 Entities

Entities werden in der DTD definiert und sind Abkürzung für wiederkehrende Inhalte, so dass zum einem der Schreibaufwand geringer wird und zum anderen aber auch vermieden wird, dass feststehende Begriffe eventuell falsch wiedergegeben werden. Beispiele für Entities sind die deutschen Umlaute (ü für "ü"), bei der die Entity durch den ensprechenden Wert des Unicodes ersetzt

wird, aber auch die Namen von Teilnehmern eines Projektes, so dass sich der Protokollant z.B. nicht mehr an die Schreibweise eines komplizierten Nachnamens erinnern muss.

Die Definition eines Entitys in einer DTD hat folgenden Aufbau:

```
<!ENTITY Entityname Entitywert>
```

Innerhalb des XML-Dokumentes wird dieses Entity dann über die Form `&Entityname;` angesprochen.

Es können Entitys auch als Abkürzung innerhalb einer DTD verwendet werden (heißen dann Parameter-Entitys), dabei ändert sich der Syntax für die Definition auf

```
<!ENTITY % Entityname Entitywert>
```

und die Anwendung auf `%Entityname;`. Als weitere Vereinfachung können diese Entitys auch extern abgelegt werden, um beispielsweise alle Sonderzeichen eines Themengebietes zu gruppieren.

A.2.5 Markup und Text

Das Markup ist die Darstellung der logischen Dokumentenstruktur innerhalb des XML-Dokumentes und ermöglicht erst die Überprüfung von Gültigkeit und Wohlgeformtheit eines Dokumentes. Ein XML-Dokument besteht ausschließlich aus Markup und Zeichendaten (die den Inhalt darstellen). Das Markup ist entweder ein Tag (wenn es zwischen dem Kleiner-als und dem Größer-als-Zeichen steht oder ein Entity (wenn es zwischen einem & und einem ;-Zeichen steht). Es gibt noch weitere Begrenzer für Markup, diese 4 sind jedoch die gebräuchlichsten [Goldfarb 1999]. Der Text enthält die Daten des XML-Dokuments und ist alles das, was kein Markup ist. Wenn der Text Zeichen enthält, die eigentlich das Markup kennzeichnen (also zum Beispiel das Größer-als-Zeichen), so muss er mit der Anweisung `<![CDATA[...]]>` umschlossen werden, um diese Zeichen nicht als XML-Anweisungen fehlzuinterpretieren.

A.2.6 Stylesheets

Um in XML die Daten von ihrer Darstellung zu abstrahieren hat das W3C die Sprache XSL erarbeitet, über die Fontgrößen, Farben oder Schritarten mit den Daten des XML-Dokumentes verknüpft werden können. Der Großteil des XSL-Codes sieht nach normalem XML aus und lässt sich daher einfach erlernen und bedienen. Ich möchte an dieser Stelle allerdings nicht weiter auf XSL eingehen, da es für mein Projekt nicht relevant ist.

A.3 Ein Beispiel für die Anwendung von XML

Als Beispiel für die Anwendung von XML habe ich den Import und Export von Daten der Datenbank gewählt. Eine der Anforderungen an das System ist die Extraktion von Billing-Daten, d.h. zu Abrechnungszwecken benötigte Daten über Zeitpunkt und Dauer von Anmeldungen innerhalb einer Abrechnungsperiode. Grundsätzlich handelt es sich hier aber nur um einen Auszug aus einer Datenbanktabelle, sie ist jedoch in ihren Anforderungen klar umrissen und weicht daher etwas vom Format der Export-Funktion des Datenbankinterfaces ab.

Anfrage an das Datenbanksystem:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE GaonExport [
      <!ELEMENT GaonExport (Query)>
      <!ATTLIST GaonExport
            name        CDATA #REQUIRED
      >
      <!ELEMENT Query (#PCDATA)>
]>
<GaonExport name='Billing'>
      <Query><![CDATA[select userId, LoginDate, LogoutDate from History
where LoginDate>='2001-09-01' and LoginDate<'2001-10-01']]></Query>
</GaonExport>
```

Abbildung 29: Gaon-Export Anfrage mit dazugehöriger DTD

So kann die Anfrage auf entsprechende Gültigkeit überprüft werden, da über die interne DTD festgelegt ist, dass jede Export-Anfrage einen Namen und genau eine Query-Anweisung haben muss. Die Gültigkeit der Query-Anweisung wird allerdings nicht weiter überprüft, da dafür der gesamte ANSI-SQL-Standard als DTD definiert werden müsste.

Eine exemplarische Antwort aus dem Datenbanksystem könnte so aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE Billing [
      <!ELEMENT Billing (Data*,Exception?)>
      <!ELEMENT Data (Lerner, LoginDate, LogoutDate)>
      <!ELEMENT Lerner (#PCDATA)>
      <!ELEMENT LoginDate (#PCDATA)>
      <!ELEMENT LogoutDate (#PCDATA)>
      <!ELEMENT Exception (#PCDATA)>
      <!ATTLIST Exception Name (CDATA) #REQUIRED>
]>
<Billing>
      <Data>
            <Lerner>mblum</Lerner>
            <LoginDate>2001-09-02 8:12.0</LoginDate>
            <LogoutDate>2001-09-02 16:42.0</LogoutDate>
      </Data>
      <Data>
            <Lerner>mbartosch</Lerner>
            <LoginDate>2001-09-18 19:08.0</LoginDate>
            <LogoutDate>2001-09-18 20:30.0</LogoutDate>
      </Data>
</Billing>
```

Abbildung 30: Gaon-Export Antwort des Systems

Die Antwort des Datenbanksystems ist natürlich abhängig von der Anfrage und somit wird die jeweils benötigte DTD immer zusammen mit den Daten zurückgesendet. Jede Antwort könnte also eine beliebige Anzahl von Datensätzen und ebenso eventuell eine Fehler-Exception enthalten (beispielsweise ein Dezimaldatenfehler im 154. Satz o.ä.).

</XML>

Anhang B: Die benutzten Fragebögen

B.1 Fragebogen Grundlagen

Es geht um die Entwicklung einer Software, die sie online bei der Vorbereitung auf Prüfungen (z.B. Sportbootführerschein See, Autoführerschein etc.) unterstützt. Bitte kreuzen Sie die jeweils am ehesten zutreffende Antwort an (pro Frage jeweils nur eine Antwort). Vielen Dank !

1. Halten sie ein derartiges Lernprogramm für sinnvoll und würden Sie es gerne einsetzen ?
   a) Ja
   b) Ja, aber nur auf PC's in der VHS im Rahmen eines Kurses
   c) Ja, aber nur zum Lernen vor dem eigenen PC zu Hause
   d) nein

2. Besitzen Sie zu Hause einen PC mit Internetzugang ?
   a) Ja, sogar mit einem recht schnellem Zugang (z.B. ISDN)
   b) Ja, allerdings ist mein Modem eher langsam (z.B. 33.6K)
   c) nein

3. Wenn ja, würden Sie ein solches Lernprogramm auch einsetzen, wenn Sie es vorher erst auf Ihrem PC installieren müssten ?
   a) nein, da es im Internet bereits Lernprogramme gibt, die Lernen ermöglichen, ohne dabei Software installieren zu müssen
   b) Eigentlich würde ich lieber auf ein Installieren verzichten, da sich installierte Software oft schlecht wieder vom PC entfernen issb
   c) Na ja, ich würde sie wohl installieren, weiß aber dabei nicht so genau, was auf meinem PC passiert
   d) Klar, Software installiere ich jeden Tag

4. Wie schätzen Sie Ihre Kenntnisse bei der Bedienung eines gängigen Browsers (Internet Explorer oder Netscape) ein ?
   a) keine Kentnisse
   b) Kenntnisse der Grundfunktionen
   c) gutes Anwenderwissen
   d) Expertenwissen

5. Sagen Ihnen die Begriffe Java oder Applet etwas ?
   a) nein, noch nie gehört
   b) schon mal gehört, bin mir aber nicht ganz sicher, was das ist

iss irgend jemand meine Daten sieht und darüber meine Leistung beurteilen kann, aber eine Verschlüsselung der Daten auf dem Übertragungsweg halte ich für einen ausreichenden Schutz

d) Passwörter sind als Schutz bei diesen Daten nicht ausreichend und jeder Benutzer sollte sich auch noch z.B. durch eine Chipkarte ausweisen.
e) Es gibt keinen ausreichenden Schutz vor dem issbrauch meiner Daten, daher würde ich eher auf den Einsatz dieser Software verzichten

7. Sind sie
   a) weiblich
   b) männlich

8. Ist ihr Alter
   a) unter 50 Jahren
   b) über 50 Jahre

Auswertung der Fragebögen:

Nr	Frage 1	Frage 2	Frage 3	Frage 4	Frage 5	Frage 6	Frage 7	Frage 8
1	a	a	d	d	c	b	b	b
2	a	a	a	d	c	b	b	a
3	b	c		c	c	c	b	a
4	a	a	b	d	c	c	b	a
5	a	b	b	b	b	b	b	b
6	c	a	d	c	b	b	b	a
7	a	a	d	d	c	b	b	a
8	a	a	d	c	b	a	b	a
9	b	a	d	b	c	e	b	b
10	c	b	d	c	c	b	b	a
11	a	a	d	d	c	a	b	a
12	a	a	d	c	c	a	b	a
13	c	b	c	a	b	b	a	a
14	c	b	b	c	b	b	b	a
15	d	c	a	b	c	c	b	a
16	d	a	a	c	c	e	b	a
17	c	a	d	d	c	b	b	a
18	c	b	b	b	a	a	b	a
häufigste Antwort	**a (44%)**	**a (61%)**	**d (50%)**	**c (38%)**	**c (66%)**	**b (50%)**	**b (94%)**	**a (83%)**

B.2 Fragebogen Softwarebewertung (Lerner)

1. Haben Sie die Möglichkeit genutzt, sich über das Softwaresystem Gaon auf ihre theoretische Prüfung zum Sportbootführerschein See vorzubereiten?
    a. Ja
    b. Nein, weil
        i. das Lernen am Computer zeitaufwendiger ist als das Lernen mittels des Fragebogensatzes auf Papier (Tippaufwand, langsame Internetverbindung etc.)
        ii. die Kosten für die Online-Zeit mich davon abgehalten haben
        iii. das Arbeiten am Bildschirm unangenehmer ist, als das Arbeiten auf Papier
        iv. keine Notwendigkeit dafür bestand, da der Fragebogensatz bereits auf Papier vorhanden war
        v. ich keine Möglichkeit habe, an einem Computer zu lernen
        vi. ich keine Zeit dafür gefunden habe

2. Wie würden Sie die Oberfläche der Software beurteilen?
    a. übersichtlich, funktional und selbsterklärend
    b. erfüllt ihren Zweck, ist aber von der Optik her verbesserungsfähig
    c. erfüllt ihren Zweck, ist aber von der Bedienung her verbesserungsfähig
    d. die Orientierung fällt schwer und das System sollte umgestellt werden.
    Verbesserungsvorschläge:

3. Wie würden Sie die Geschwindigkeit (Antwortzeitverhalten) der Software beurteilen?
    a. gut
    b. nicht besonders schnell, aber akzeptabel
    c. schlecht

4. Wie würden Sie die Korrektheit der Auswertung von Fragen beurteilen?
    a. größtenteils korrekt
    b. vom System her ausreichend, aber noch nicht mit genügend Daten versorgt
    c. manchmal richtig, aber insgesamt eher unzureichend
    d. größtenteils falsch
    Verbesserungsvorschläge:

5. Hat Ihnen die Software genügend Möglichkeiten zur Kommunikation mit ihrem Tutor und den übrigen Kursteilnehmern gegeben (E-Mail, Messageboard, Diskussions-Foren)?
    a. Ja
    b. Nein, ich hätte mir noch weitere gewünscht (Chat, Videokonferenz etc.)
    c. Die Kommunikation über das Internet ist nicht für die Lösung der Probleme der Kursteilnehmer geeignet
    Verbesserungsvorschläge:

6. Hatten sie das Gefühl, dass das Lernen mittels der Online-Fragebögen Sie gut auf die Prüfung vorbereitet?
    a. Ja, man kann auf die realen Fragebögen verzichten, wenn Online gelernt wird

b. Es war eine weitere Möglichkeit das eigene Wissen zu testen, aber ich würde nicht auf das Lernen mittels der realen Fragebögen verzichten
c. Nein, die Unterschiede zwischen Online-Fragebögen und realen Fragebögen waren mir zu groß
Verbesserungsvorschläge:

Auswertung der Fragebögen:

Nr	Frage 1	Frage 2	Frage 3	Frage 4	Frage 5	Frage 6
1	b(i)	-	-	-	-	-
2	a	a	a	c	a	b
3	a	a	a	b	a	b
4	b(vi)	-	-	-	-	-
5	b(iv)	-	-	-	-	-
6	b(iv)	-	-	-	-	-
7	b(vi)	-	-	-	-	-
8	b(iv)	-	-	-	-	-
9	b(iv)	-	-	-	-	-
10	b(vi)	-	-	-	-	-
11	a	b	a	b	c	b
12	a	a	d	c	c	a
13	b(v)	-	-	-	-	-
14	b(ii)	-	-	-	-	-
15	a	c	a	b	a	a
häufigste Antwort	**b (66%)**	**a (60%)**	**a (80%)**	**b (60%)**	**a (60%)**	**b (60%)**

B.3 Fragebogen Softwarebewertung (Tutor)

1. Haben Sie das Softwaresystem Gaon in ihrem Kurs eingesetzt?
    a. Ja
    b. Nein, weil
        i. mir die Betreuung des Kurses an einem Computer zu zeitaufwendig ist
        ii. die Kosten für die Online-Zeit mich davon abgehalten haben
        iii. mir das Arbeiten am Bildschirm unangenehm ist
        iv. ich keine Möglichkeit habe, an einem Computer zu arbeiten
        v. folgender andere Grund:

2. Wie würden Sie die Oberfläche der Software beurteilen?
    a. übersichtlich, funktional und selbsterklärend
    b. erfüllt ihren Zweck, ist aber von der Optik her verbesserungsfähig
    c. erfüllt ihren Zweck, ist aber von der Bedienung her verbesserungsfähig
    d. die Orientierung fällt schwer und das System sollte umgestellt werden.
    Verbesserungsvorschläge:

3. Wie würden Sie die Geschwindigkeit (Antwortzeitverhalten) der Software beurteilen?
    a. gut
    b. nicht besonders schnell, aber akzeptabel
    c. schlecht

4. Wie würden Sie die Korrektheit der Auswertung von Fragen beurteilen?
    a. größtenteils korrekt
    b. vom System her ausreichend, aber noch nicht mit genügend Daten versorgt
    c. manchmal richtig, aber insgesamt eher unzureichend
    d. größtenteils falsch
    Verbesserungsvorschläge:

5. Wie empfanden Sie die Gegenbewertung bzw. die Korrektur der Fragebögen?
    a. einfacher als bei realen Fragebögen
    b. anders als bei realen Fragebögen, aber praktikabel
    c. zu kompliziert
    Verbesserungsvorschläge:

6. Hat Ihnen die Software genügend Möglichkeiten zur Kommunikation mit ihren Kursteilnehmern gegeben (E-Mail, Messageboard, Diskussions-Foren)?
    a. Ja
    b. Nein, ich hätte mir noch weitere gewünscht (Chat, Videokonferenz etc.)
    c. Die Kommunikation über das Internet ist nicht für die Lösung der Probleme der Kursteilnehmer geeignet
    Verbesserungsvorschläge:

7. Wie oft haben Sie nach neuen Einträgen im Diskussionsforum oder nach abgegebenen Prüfungen nachgesehen und diese bearbeitet?
    a. täglich
    b. 2-3 mal wöchentlich
    c. 1 mal wöchentlich
    d. weniger oder gar nicht

Auswertung der Fragebögen:

Nr	Frage 1	Frage 2	Frage 3	Frage 4	Frage 5	Frage 6	Frage 7
1	a	a	a	b	b	c	a
2	a	a	a	a	b	a	c
häufigste Antwort	**a**	**a**	**a**	**a / b**	**b**	**a / c**	**a / c**

B.4 Fragebogen Softwarebewertung (Autor)

1. Wie würden Sie die Oberfläche der Software beurteilen?

a. übersichtlich, funktional und selbsterklärend
b. erfüllt ihren Zweck, ist aber von der Optik her verbesserungsfähig
c. erfüllt ihren Zweck, ist aber von der Bedienung her verbesserungsfähig
d. die Orientierung fällt schwer und das System sollte umgestellt werden.
Verbesserungsvorschläge:

2. Wie würden Sie die Geschwindigkeit (Antwortzeitverhalten) der Software beurteilen?
   a. gut
   b. nicht besonders schnell, aber akzeptabel
   c. schlecht

3. Wie würden Sie die Erfassung der Aufgaben und der dazugehörigen Lösungen beurteilen?
   a. einfach und selbsterklärend
   b. aufwendig, aber praktikabel, wenn Grundwissen in HTML vorhanden ist
   c. zu aufwendig und verbesserungsfähig
   Verbesserungsvorschläge:

4. Hätten Sie sich mehr Komfort beim Einspielen externer Materialien (Bilder, Multimedia-Elemente etc.) gewünscht?
   a. Nein, der FTP-Zugang zum Server war ausreichend
   b. Ja, als Autor möchte ich mich nicht mit den Details eines FTP-Zugangs auseinandersetzen
   Verbesserungsvorschläge:

5. Wie beurteilen Sie die Anpassung der generierten Fragebögen?
   a. im allgemeinen nicht nötig, aber wenn doch, relativ einfach
   b. aufwendig, aber praktikabel, wenn Grundwissen in HTML vorhanden ist
   c. zu aufwendig und verbesserungsfähig
   Verbesserungsvorschläge:

6. War die Anzahl der verfügbaren Aufgabentypen ausreichend?
   a. Ja
   b. Nein, ich hätte mir noch weitere gewünscht
   Verbesserungsvorschläge:

7. Wie beurteilen Sie die Import bzw. Export der Daten aus bzw. in Fremdsysteme
   a. Hilfreich, da bereits erstellte Kurse auf diese Art schnell in das System integriert werden können (z.B. Kauf vorgefertigter Kurse oder Offline-Arbeit an externen Computern)
   b. Nur für Datensicherung relevant
   c. unrelevant

Auswertung des Fragebogens:

Nr	Frage 1	Frage 2	Frage 3	Frage 4	Frage 5	Frage 6	Frage 7
1	c	a	b	b	b	a	a

Anhang C: DTD für die Maskendefinition

Dies ist die vollständige DTD für die Maskendefinitionen:

```
<!ELEMENT GAONINPUT (#PCDATA | VARIABLE | CHECKED)*>
<!ATTLIST GAONINPUT
  %attrs;
  type        %InputType;    "text"
  name        CDATA          #IMPLIED
  value       CDATA          #IMPLIED
  disabled    (disabled)     #IMPLIED
  size        CDATA          #IMPLIED
  maxlength   %Number;       #IMPLIED
  onchange    %Script;       #IMPLIED
  >
<!ELEMENT CHECKED (#PCDATA | VARIABLE)*>
<!ATTLIST CHECKED
  var1        CDATA         #REQUIRED
  cond        (eq|ne|le|ge|lt|gt) "eq"
  >

<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE name NMTOKEN #REQUIRED>

<!ELEMENT PAGECLASS (#PCDATA | %block; | form | %misc; | %inline; |
AUTHORIZATIONERROR)* >
<!ATTLIST PAGECLASS
  level        CDATA         #REQUIRED
  >
<!ELEMENT AUTHORIZATIONERROR %Flow;>

<!ELEMENT GAONLINK (LINKPARAM*, LINKTEXT)>
<!ATTLIST GAONLINK
  href        %URI;          #IMPLIED
  >
<!ELEMENT LINKPARAM (#PCDATA | VARIABLE)*>
<!ATTLIST LINKPARAM
  name        NMTOKEN        #IMPLIED
  >
<!ELEMENT LINKTEXT (#PCDATA | VARIABLE)*>

<!ELEMENT QUERY (SQL, RESULTROW)>
<!ATTLIST QUERY
  name        NMTOKEN        #IMPLIED
  >
<!ELEMENT SQL (#PCDATA | VARIABLE)*>
```

```
<!ELEMENT RESULTROW (#PCDATA | %block; | form | %misc; | %inline; | tr |
td)*>

<!ELEMENT GAONIMG (#PCDATA | VARIABLE)*>
<!ATTLIST GAONIMG
  %attrs;
  alt         %Text;         #IMPLIED
  name        NMTOKEN        #IMPLIED
  height      %Length;       #IMPLIED
  width       %Length;       #IMPLIED
  align       %ImgAlign;     #IMPLIED
  border      %Length;       #IMPLIED
  >

<!ELEMENT GAONSELECT (OPTIONS)>
<!ATTLIST GAONSELECT
  %attrs;
  name        CDATA          #IMPLIED
  size        %Number;       #IMPLIED
  multiple    (multiple)     #IMPLIED
  disabled    (disabled)     #IMPLIED
  onchange    %Script;       #IMPLIED
  >
<!ELEMENT OPTIONS (SQL, (SELECTED|JOINSELECTED)?)>
<!ELEMENT SELECTED (#PCDATA | VARIABLE)*>
<!ATTLIST SELECTED
  field       CDATA          #REQUIRED
  cond        (eq|ne|le|ge|lt|gt) "eq"
  >
<!ELEMENT JOINSELECTED (#PCDATA | VARIABLE)*>
<!ATTLIST JOINSELECTED
  joinfield   CDATA          #REQUIRED
  cond        (eq|ne|le|ge|lt|gt) "eq"
  >

<!ELEMENT IF (#PCDATA | VARIABLE)*>
<!ATTLIST IF
  var1        CDATA         #REQUIRED
  cond        (eq|ne|le|ge|lt|gt) "eq"
  >
<!ELEMENT THEN %Flow;>
<!ELEMENT ELSE %Flow;>
```

Abbildung 31: vollständige DTD für XML-Maskendefinition

Damit diese Definitionen zusätzlich zur strikten HTML-Notierung XHTML gelesen werden, ist es nötig, sie als externes Entity in die XHTML-DTD einzubinden. Da innerhalb der Notierung auch allgemeingültige Entitys gelesen werden, muss der Import am Ende der DTD erfolgen. Zusätzlich ist es nötig, die HTML-Bestimmungen an den korrekten Stellen um die GAON-Funktionalität zu erweitern. Das geschieht zum einen durch die Definition eines entsprechenden Entitys und dessen Einbindung in den HTML-Syntax.

```
<!--================ new gaon functionality ========================→
```

```
<!ENTITY % Gaon "GAONINPUT | VARIABLE | PAGECLASS | GAONSELECT | GAONIMG |
QUERY | GAONLINK | IF">

<!--================ use Gaon as inline ==========================→

<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;
| %Gaon;">

<!--================ import Gaon-DTD ==========================→

<!ENTITY % gaon SYSTEM "gaon.dtd">
%gaon;
```

Abbildung 32: Einbindung der Gaon-DTD in die XHTML-DTD

Als Beispiel für ihre Anwendung soll hier die Sammelanzeige aller Login-Kennungen dienen, die in ähnlicher Form (etwas ausführlicher) Bestandteil der Benutzerverwaltung des Gaon-Systems ist:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html SYSTEM "gaon/code/gaon-xhtml.dtd">
<html>
<head>
   <meta name="Author" content="Martin Blum" />
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
   <title>Verwaltung Login-Kennungen</title>
   <link rel="stylesheet" type="text/css" href="../style.css" />
</head>
<body background="../images/hg.jpg">
<hr size="10" color="#E2E2E2" />
<h3><VARIABLE name="message" /></h3>

<table><tr><th>Kennung</th><th>Name</th></tr>
<QUERY name="q1"><SQL>select * from Lerner order by Lerner</SQL>
<RESULTROW><tr><td><VARIABLE name="q1.Lerner" /></td><td><VARIABLE
name="q1.Vorname" /> <VARIABLE name="q1.Nachname" /></td></tr></RESULTROW>
</QUERY>
</table>

<form action="/gaon/servlet/Master" method="post">
<input type="hidden" name="task" value="A001" />
<input type="submit" value="zur&uuml;ck zum Hauptmen&uuml;" /></form>
<p />
</body>
</html>
```

Abbildung 33: Beispielmaske Sammelanzeige Login-Daten

Anhang D: Datenbankdesign

Als Nachschlagewerk während meiner Programmierung diente mir die folgende detaillierte Übersicht aller Tabellen der Datenbank.

Tabelle: Aufgaben

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Aufgabe	INT	Aufgabennummer
	Aufgabenbeschreibung	LONGBLOB	Beschreibung einer Situation (evtl. mit HTML-Code)
	Hilfe	CHAR(255)	URL für Hilfe-Seite
	Themengebiet	INT	Zuordnung zu einem Themengebiet des Projektes
	Schwierigkeitsgrad	INT	Schwierigkeitsgrad der Aufgabe
	Objektreferenz	CHAR(255)	Referenz zu HTML-Objekt
	ManuellKZ	TINYINT	Objekt darf nur manuell verändert werden Ja/Nein
	Punkte	INT	maximale Anzahl an Punkten für diese Aufgabe

Tabelle: Aufgabenpositionen

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Aufgabe	INT	Aufgabennummer
ja	Position	INT	Positionsnummer
	Fragetext	LONGBLOB	Fragetext dieser Position
	Hinweistext	LONGBLOB	Hinweistext ("Bitte kreuzen Sie an:") oder Applet-Code
	Antworttyp	CHAR(50)	Art der Teilfrage
	Begruendung	LONGBLOB	Begründung für Lösung

Tabelle: Droplisten

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Listenname	CHAR(50)	eindeutiger Name der Drop-Down-Liste
ja	Position	INT	Positionsnummer
	Schluessel	CHAR(50)	Vergleichsfeld für die Lösungsermittlung
	Wert	CHAR(50)	Anzeigetext innerhalb der Liste

Tabelle: Foren

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	ThreadId	CHAR(50)	Identifikation des aktuellen Threads
	Bezug	CHAR(50)	Bezug auf eine ThreadId
	Datum	DATETIME	Datum und Uhrzeit des Eintrags
	Lerner	CHAR(50)	Login-Kennung des Eintragenden
	Name	CHAR(50)	Name des Eintragenden
	Betreff	CHAR(50)	Betreff
	Beitrag	LONGBLOB	Beitrag
	Antworten	INT	Anzahl Antworten auf diesen Beitrag
	Host	CHAR(255)	Hostname des Clients
	IP	CHAR(50)	IP-Adresse des Clients

Tabelle: Frageboegen

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Fragebogen	INT	Fragebogennummer
ja	Position	INT	Positionsnummer
	Aufgabe	INT	zugeordnete Aufgabe

Tabelle: History

Key	Feldname	Felddatentyp	Beschreibung
ja	SessionId	CHAR(50)	Session Id des Clients
ja	UserId	CHAR(50)	Lerner-Kennung
	UserName	CHAR(50)	Benutzername
	IP	CHAR(50)	IP-Adresse
	Host	CHAR(50)	Host-Name
	LoginDate	DATETIME	Datum/Zeit der Anmeldung
	LogoutDate	DATETIME	Datum/Zeit der Abmeldung
	Counter0	INT	Anzahl Transaktionen gesamt
	Counter1	INT	Anzahl Tests gesamt
	Counter2	INT	Anzahl Fragen gesamt
	Counter3	INT	Anzahl beantworteter Fragen
	Counter4	INT	frei
	Counter5	INT	frei

Tabelle: Klassifikationen

Key	Feldname	Felddatentyp	Beschreibung
ja	Hierachie	INT	Sortierung für hierarchischen Zugriff
	Klassifikation	CHAR(50)	Name der Klassifikation
	Level	CHAR(3)	3-stelliges Kürzel
	Kommentar	CHAR(255)	Kommentar

Tabelle: Lerner

Key	Feldname	Felddatentyp	Beschreibung
ja	Lerner	CHAR(50)	Benutzerkennung des Lerners
	Vorname	CHAR(50)	Vorname des Lerners
	Nachname	CHAR(50)	Nachname des Lerners
	Anrede	CHAR(50)	Anrede (Herr/Frau/Prof./Dr. etc)
	Kennwort	CHAR(30)	Kennwort für Kennung
	Mail	CHAR(50)	E-Mail-Adresse
	Geburtsjahr	INT	Geburtsjahr 4-stellig (wird noch nicht benutzt)
	Von	DATETIME	Kennung ist gültig von
	Bis	DATETIME	Kennung ist gültig bis
	Klasse	CHAR(50)	Berechtigungsebene des Anwenders (Administrator, Autor, Lerner, Prüfling)
	Bild	CHAR(150)	lokale Adresse einer Bilddatei

Tabelle: Lernerstatistik

Key	Feldname	Felddatentyp	Beschreibung
ja	Lerner	CHAR(50)	Benutzerkennung des Lerners
ja	Projekt	CHAR(50)	Projektkennung
ja	Art	CHAR(50)	Testart (siehe Tabelle Testarten)
ja	Schluessel	INT	Fragennummer, Fragebogen, Themengebiet, Schwierigkeit
	Aufgaben	INT	Anzahl Aufgaben in diesem Test
	Versuche	INT	Anzahl Lösungsversuche
	Punkte	INT	erreichte Punktzahl (Summe)
	PunkteGesamt	INT	Summe erreichbarer Punkte
	Ergebnis	INT	prozentuales Ergebnis
	ErgebnisWidth1	INT	Breite des Ergebnisbalkens (richtig)
	ErgebnisWidth2	INT	Breite des Ergebnisbalkens (falsch)

Tabelle: Loesungspositionen

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Aufgabe	INT	Aufgabennummer
ja	Position	INT	Aufgabenposition
ja	Nummer	INT	Nummer der Lösungsposition
	TextVor	CHAR(255)	Text vor Eingabefeld
	Wert	LONGBLOB	Wert des Elements
	Loesung	LONGBLOB	Lösung dieser Position
	TextNach	CHAR(255)	Text nach Eingabefeld

Tabelle: Messageboard

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	MessageId	INT	Eintragsnummer
	Datum	DATETIME	Datum und Uhrzeit des Eintrags
	Lerner	CHAR(50)	Login-Kennung des Eintragenden
	Name	CHAR(50)	Name des Eintragenden
	Mail	CHAR(50)	E-Mail-Adresse des Eintragenden
	Url	CHAR(255)	Url einer Homepage
	Nachricht	LONGBLOB	Nachricht auf dem Board
	Kommentar	LONGBLOB	Kommentar des Moderators zu dieser Nachricht
	Host	CHAR(255)	Hostname des Clients
	IP	CHAR(50)	IP-Adresse des Clients

Tabelle: Projekte

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
	Projekttitel	CHAR(50)	Titel
	Projektbeschreibung	LONGBLOB	Beschreibung
	Hilfe	CHAR(255)	Link für eine erweiterte Hilfe
	Fragenanzahl	INT	Anzahl Fragen eines Testdurchlaufs
	Wissensbereich	CHAR(255)	Wissensbereich
	Version	CHAR(50)	Versionsnummer
	DatumAenderung	DATETIME	Datum der letzten Änderung
	BenutzerAenderung	CHAR(50)	BenutzerId der letzten Änderung

Tabelle: Projektthemen

Key	Feldname	Felddatentyp	Beschreibung
ja	Projekt	CHAR(50)	Projektkennung
ja	Themengebiet	INT	Themenkennung
	Themenbeschreibung	CHAR(50)	Beschreibung des Themengebiets

Tabelle: Projektzuordnungen

Key	Feldname	Felddatentyp	Beschreibung
ja	Lerner	CHAR(50)	Benutzerkennung des Lerners
ja	Projekt	CHAR(50)	zugeordnetes Projekt
	Von	DATETIME	Zuordnung ist gültig von
	Bis	DATETIME	Zuordnung verfällt nach dem
	Tutor	CHAR(50)	Benutzerkennung des zugeordneten Tutors
	Fragebogen	INT	Fragebogennummer, wenn Benutzer nur einen bestimmten Fragebogen zugelassen ist

Tabelle: Schwierigkeitsgrade

Key	Feldname	Felddatentyp	Beschreibung
ja	Schwierigkeitsgrad	INT	Schlüssel für Schwierigkeitsgrad
	Beschreibung	CHAR(50)	Beschreibung

Tabelle: Systemparameter

Key	Feldname	Felddatentyp	Beschreibung
ja	Systemname	CHAR(50)	Schlüssel "Gaon"
	Kennwort	CHAR(30)	Kennwort für root-Kennung

Tabelle: Testarten

Key	Feldname	Felddatentyp	Beschreibung
ja	Art	CHAR(50)	Testart
	Beschreibung	CHAR(50)	Beschreibung der Testart

Tabelle: Testeingaben

Key	Feldname	Felddatentyp	Beschreibung
ja	Identifikation	CHAR(50)	Test-Identifikation
ja	Testposition	INT	Positionsnummer innerhalb des Tests/Fragebogens
ja	Aufgabe	INT	Aufgabennummer aus dem Aufgabenpool
ja	Variable	CHAR(50)	Name der Eingabevariable
ja	Nummer	INT	Nummer des Vorkommens
	Wert	LONGBLOB	Wert der Eingabevariablen

Tabelle: Testidentifikationen

Key	Feldname	Felddatentyp	Beschreibung
ja	Lerner	CHAR(50)	Kennung des Lerners
ja	Projekt	CHAR(50)	Projektkennung
ja	Start	DATETIME	Startdatum, Zeit
ja	Identifikation	CHAR(50)	Test-Identifikation
ja	Art	CHAR(50)	Art des Tests (siehe unter Tabelle Testarten)
ja	Schluessel	INT	z.B. Fragebogennummer, Themengebiet
	Pruefungsmodus	TINYINT	Test im Prüfungsmodus Ja/Nein
	Ende	DATETIME	Endedatum, Zeit
	Aufgaben	INT	Anzahl Aufgaben in diesem Test
	Versuche	INT	Anzahl von Lösungsversuchen
	Punkte	INT	erreichte Punktzahl (Summe)
	PunkteGesamt	INT	Summe erreichbarer Punkte
	Bestanden	TINYINT	Prüfung bestanden Ja/Nein
	Ergebnis	INT	prozentuales Ergebnis
	ErgebnisWidth1	INT	Breite des Ergebnisbalkens (richtig)
	ErgebnisWidth2	INT	Breite des Ergebnisbalkens (falsch)
	NachbewertungDatum	DATETIME	tutorielle Nachbewertung Datum, Zeit
	NachbewertungTutor	CHAR(50)	Lerner-Kennung des Tutors

Tabelle: Testpositionen

Key	Feldname	Felddatentyp	Beschreibung
ja	Identifikation	CHAR(50)	Test-Identifikation
ja	Testposition	INT	Positionsnummer innerhalb des Tests/Fragebogens
ja	Aufgabenposition	INT	Positionsnummer der Aufgabe (Teilpositionen)
ja	Nummer	INT	Nummer der Loesungsposition
	Aufgabe	INT	Aufgabennummer aus dem Aufgabenpool
	Antwortelement	LONGBLOB	aufbereitete Antwort
	Bewertung1	TINYINT	ermittelte Bewertung: 1(Ja) richtig, 0(Nein) falsch
	Prozent1	INT	ermittelte prozentuale Bewertung, wenn teilweise richtig
	Bewertung2	TINYINT	tutorielle Bewertung: 1(Ja) richtig, 0(Nein) falsch
	Prozent2	INT	tutorielle prozentuale Bewertung, wenn teilweise richtig

Tabelle: Teststatistik

Key	Feldname	Felddatentyp	Beschreibung
ja	Identifikation	CHAR(50)	Testidentifikation
ja	Art	CHAR(50)	Testart (siehe Tabelle Testarten)
ja	Schluessel	INT	Fragennummer, Fragebogen, Themengebiet, Schwierigkeit
	Projekt	CHAR(50)	Projektkennung
	Aufgaben	INT	Anzahl Aufgaben in diesem Test
	Versuche	INT	Anzahl Lösungsversuche
	Punkte	INT	erreichte Punktzahl (Summe)
	PunkteGesamt	INT	Summe erreichbarer Punkte
	Ergebnis	INT	prozentuales Ergebnis
	ErgebnisWidth1	INT	Breite des Ergebnisbalkens (richtig)
	ErgebnisWidth2	INT	Breite des Ergebnisbalkens (falsch)

Tabelle: Teststruktur

Key	Feldname	Felddatentyp	Beschreibung
ja	Identifikation	CHAR(50)	Test-Identifikation
ja	Testposition	INT	Positionsnummer innerhalb des Tests/Fragebogens
	Aufgabe	INT	Aufgabennummer aus dem Aufgabenpool
	Punkte	INT	maximale Anzahl an Punkten für diese Aufgabe
	Beantwortet	TINYINT	Aufgabe beantwortet: 1(Ja), 0(Nein)

Anhang E: Source-Code ausgewählter Funktionen

E.1 Appletschnittstelle

```
// transfer applet-variables to HTML-variables
function checkApplet() {
    for (I=0; I<document.applets.length; I++) {
       var applet=document.applets[I];
       var pos=applet.getNumberOfPositions();
       for (j=0; j<pos; j++) {
          var input=eval('document.forms[0].applet'+I+'Var'+j);
          input.value=applet.getValue(j+1);
       }
    }
    return true;
}

// transfer HTML-variables to applet-variables
function setAppletVar(noChange) {
    form=document.forms[0];
    elem=document.forms[0].elements;
    anzApplets=document.applets.length;
    if (anzApplets>0) {
        for (I=0; I<anzApplets; I++) {
            varName="applet"+I+"Var";
            var nr=0;
            if (noChange=='1') document.applets[I].setChange(false);
            else document.applets[I].setChange(true);
            for (j=0; j<elem.length; j++) {
                if (elem[j].name.substr(0, varName.length)==varName) {
                    nr=nr+1;
                    if (elem[j].value!="") document.applets[I].setValue(nr,
elem[j].value);
                }
            }
        }
    }
    return true;
}
```

Abbildung 34: JavaScript für die Kommunikation zwischen Applet und HTML

E.2 Servlet-Verarbeitung

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Master extends HttpServlet {
      HttpSession userSession = null;
            public void init(ServletConfig config) throws ServletException
{
            super.init(config); // call init from super class
            workDir=getInitParameter("workDir"); // get parameter
      } // end of init

      // redirect all gets to doPost
      protected void doGet (HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException {
            doPost(req, res);
      } // end of doGet

      // do all the work
      protected synchronized void doPost (HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException {
            String var1=req.getParameter("name"); // get the Var "name"
from client
            res.setContentType("text/html");
            PrintWriter out=res.getWriter(); // writer for response to
client
            out.println("<HTML><BODY>Your name is
"+var1+".</BODY></HTML>");
            out.close();
      } end of doPost
} // end of class
```

Abbildung 35: Servlet-Rumpf

```
public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException,IOException {
      try { startThread(req, res).join();
      } catch(InterruptedException e) {}
} // end of doPost

private synchronized Thread startThread(HttpServletRequest req,
HttpServletResponse res) throws IOException {
      for (int j=0; j<10; j++) {
            for (int I=0; I<POOL_SIZE; I++) {
                  if (thread[I]==null || !thread[I].isAlive()) {
                        if (worker[I]==null) {
                              worker[I]=new MasterWorker(I, workDir, dbName,
dbDriver, dbUser, dbPassword);
                        }
                        if (!worker[I].processorClassReady()) {
                              // error in processorClass start-up
```

```
                              return (new Thread());
                        } else {
                              thread[I]=new MasterThread(I, worker[I], req,
res);
                              thread[I].start();
                              return (thread[I]);
                        } // end if
                  } // end if
            } // end of loop next thread
            // all slots full, sleep and try again
            try { Thread.sleep(500);
            } catch(InterruptedException e) {}
      } // end of loop try slots
      // do some error handling: send overload error-message
      return (new Thread());
} // end of startThread
```

Abbildung 36: Auswahl eines Verarbeitungsthreads aus dem Pool

```
public void doPost(HttpServletRequest req, HttpServletResponse res) {
      try {
            // get task number from client
            putValue("task", req.getParameter("task"));
            // get the current userSession or create if necessary
            userSession=req.getSession(true);
            if (userSession.isNew()) login(req);
            // store variables, process event and branch to next location
            String newLocation=processorClass.processTask(req,
getValue("task"));
            // create page out of file
            res.setContentType("text/html");
            res.setHeader("pragma", "no-cache"); // disable cache
            //mix result with page deklaration
            HtmlPage page = convertXML(newLocation);
            page.setPrintWriter(res.getWriter());
            page.write(); // send page to client
      } catch (Exception e) { // error handling }
} // end of doPost
```

Abbildung 37: Hauptroutine eines MasterThreads

```
      protected boolean putValue(String varName, String value) {
            if (value==null) return putValues(varName, null);
            String[] valueArray= new String[1];
            valueArray[0]=value;
            return putValues(varName, valueArray);
      } // end of putValue
      protected boolean putValues(String varName, String[] value) {
            return putObject(varName, value);
      } // end of putValues
      protected boolean putObject(String varName, Object object) {
            if (varName==null) return false;
```

```
            try {
                  if (object==null) userSession.removeValue(varName);
                  else userSession.putValue(varName, object);
            } catch (java.lang.IllegalStateException e)  {}
            return true;
      } // end of putObject
```

Abbildung 38: Das Ablegen von Variableninhalten

```
      protected String getValue(String varName) {
            String[] values=getValues(varName);
            if (values!=null&&values.length>0) return values[0];
            else return null;
      } // end of getValue
      protected String[] getValues(String varName) {
            Object object=getObject(varName);
            if (object!=null) return ((String[])object);
            else return null;
      } // end of getValue
      protected Object getObject(String varName) {
            if (varName==null) return null;
            Object object=null;
            try {
                  object=userSession.getValue(varName);
            } catch (java.lang.IllegalStateException e)  {}
            return object;
      } // end of getObject
```

Abbildung 39: Das Lesen von Variableninhalten

E.3 Verarbeitung der XML-Masken

```
private HtmlPage convertXML(String newLocation) {
      HtmlPage page=new HtmlPage();
      DOMParser parser = new DOMParser();
      try {
            parser.parse(newLocation);
            Document d = parser.getDocument();
            page.addLine(convert(d));
      }
      catch (SAXParseException e) { e.printStackTrace(); }
      return(page);
} // end of convertXML

private String convert(Node node){
      String out="";
      if (node==null) return("");
      int type=node.getNodeType();
      switch (type) {
            // print document
```

```
            case Node.DOCUMENT_NODE: {
                  NodeList children = node.getChildNodes();
                  for (int iChild=0; iChild<children.getLength(); iChild++)
{
                        out+=convert(children.item(iChild));
                  }
                  break;
            }
            // print element with attributes
            case Node.ELEMENT_NODE: {
                  String nodeName=node.getNodeName();
                  NamedNodeMap attrs=node.getAttributes();
                  if (nodeName.equals("QUERY")){
                    String sqlString="";
                    Node resultrow=null;
                    NodeList children=node.getChildNodes();
                    if (children!=null) {
                      int len=children.getLength();
                      for (int I=0; I<len; I++) {
                        Node child=children.item(i);
                        if (child!=null&&
child.getNodeName().equals("SQL")) sqlString=convert(child);
                        if (child!=null&&
child.getNodeName().equals("RESULTROW")) resultrow=child;
                      } // end loop
                    } // end if
                    try {
                      Vector rsv=dbClient.executeRead (sqlString);
                      if (rsv!=null) for (int I=0; I<rsv.size(); I++) {
                        DBResultSet rs = (DBResultSet)rsv.elementAt(i);
                        DBResultSetMetaData rsmd =
dbClient.getResultSetMetaData();
                        for (int j=0;j<rsmd.getColumnCount(); j++)
                    master.put(rsmd.getColumnName(j+1), rs.getString(j+1));
                        out+=convert(resultrow)+"\n";
                      } // end loop next result set
                    } catch(SQLException e) {
                        // do some error handling
                    }
                  } else if (nodeName.equals(". . .")) {
                        // other gaon nodes
                  } else {
                        out+='<'+nodeName;
                        for (int I=0; I<attrs.getLength(); I++) {
                              Node attr = attrs.item(i);
                              out+=' '+attr.getNodeName();
                              out+="=\""+ attr.getNodeValue()+"\"";
                        } // end loop attributes
                        out+='>';
                        NodeList children=node.getChildNodes();
                        if (children!=null) {
                              int len=children.getLength();
                              for (int I=0; I<len; I++) {
                                    out+=convert(children.item(i));
                              } // end loop next child
                        } // end if
```

```
                  } // end if
                  break;
            } // end case
            // handle entity reference nodes
            case Node.ENTITY_REFERENCE_NODE: {
                  out+='&'+node.getNodeName()+';';
                  break;
            } // end case
            // print cdata sections
            case Node.CDATA_SECTION_NODE: {
                  out+="<![CDATA["+node.getNodeValue()+"]]>";
                  break;
            } // end case
            // print text
            case Node.TEXT_NODE: {
                  out+=node.getNodeValue();
                  break;
            } // end case
            // print processing instruction
            case Node.PROCESSING_INSTRUCTION_NODE: {
                  out+="<?"+node.getNodeName();
                  String data=node.getNodeValue();
                  if (data!=null) out+=' '+data;
                  out+="?>\n";
                  break;
            } // end case
      } // end switch
      if (type==Node.ELEMENT_NODE && node.hasChildNodes()) {
            out+="</"+node.getNodeName()+'>';
      } // end if
      return (out);
} // end of convert
```

Abbildung 40: Traversierung einer XML-Maskendefinition

E.4 Datenbank-Schnittstelle

```
// connect to database (local or non-local)
public boolean connectDB(String dbName, String dbDriver, String dbUser,
String dbPassword) throws SQLException, IOException, ClassNotFoundException
{
      String ODBCDriver="sun.jdbc.odbc.JdbcOdbcDriver";
      String MYSQLDriver="org.gjt.mm.mysql.Driver";
      // disconnect if still connected
      if (connected) { disconnectDB(); }
      // parse dbName: "jdbc:subprotocol://host:port/database"
      . . . // check and parse syntax, is db local ?
      this.dbName=dbName;
      if (dbDriver==null||dbDriver.equals("")) {
            if (dbName.startsWith("jdbc:odbc:")) dbDriver=_ODBC_Driver;
            if (dbName.startsWith("jdbc:mysql:")) dbDriver=_MYSQL_Driver;
      }
      if (dbLocal) {
            Class.forName(dbDriver); // load Driver
```

```
            if (dbUser==null) dbUser="";
            if (dbPassword==null) dbPassword="";
            con = DriverManager.getConnection(this.dbName, dbUser,
dbPassword); // connect to database
            return(true);
      } else {
            // send connect request to remote DBServer
      }
      return(false);
}
```

Abbildung 41: Verbindungsaufbau zur Datenbank

```
public Vector executeRead(String sql) throws SQLException {
      Vector rsv=new Vector(); // create new vector
      stmt=con.createStatement(); // create statement
      ResultSet rs1=stmt.executeQuery(sql); // execute query
      // Transform JDBC-ResultSet to DBResultSet
      DBResultSet rs= new DBResultSet(rs1, this, sql);
      // get all ResultSets of this query
      while (rs.next()) rsv.addElement((DBResultSet)rs.clone());
      stmt.close(); // close statement
      return rsv; // return result
}
```

Abbildung 42: Einlesen eines Ergebnisvektors

E.5 Verarbeitung der Benutzereingaben

```
// A010: query Login select
if (task.equals("A010")) newLocation="QueryLogin.xml";
// A011: query Login display multiple records
else if (task.equals("A011")) {
      if (master.getValue("queryLogin").equals("") &&
master.getValue("queryLastname").equals("") &&
master.getValue("queryMail").equals("")) master.putValue("queryLogin","*");
      newLocation="ResultLoginMultiple.xml";
}
// A012: query Login display Single record
else if (task.equals("A012")) {
      newLocation="ResultLoginSingle.xml";
}
// A013: update database: Login
else if (task.equals("A013")) {
      if (updateLogin()) newLocation="ResultLoginMultiple.xml";
      else newLocation="ResultLoginSingle.xml";
}
// A014: insert into database: Login
else if (task.equals("A014")) {
      if (insertLogin()) newLocation="ResultLoginMultiple.xml";
      else newLocation="NewLogin.xml";
}
// A015: delete from database: Login
```

```
else if (task.equals("A015")) {
      if (deleteLogin()) newLocation="ResultLoginMultiple.xml";
      else newLocation="ResultLoginSingle.xml";
}
// A016: new Login (send empty mask)
else if (task.equals("A016")) {
      master.putValue("newUser.Klasse", "Lerner");
      master.putValue("newUser.Bild", "../images/logo_vhs_hb.jpg");
      newLocation="NewLogin.xml";
}
```

Abbildung 43: Verknüpfung zwischen Menüsystem-Task und Funktionalität

```
// insert record into database (login table)
private boolean insertLogin() {
      String message="";
      // get input fields from client and store it
      storeField("user.Lerner");
      storeField("user.Vorname");
      storeField("user.Nachname");
      storeField("user.Klasse");
      storeField("user.Bild");
      // check input fields
      if (master.getValue("user.Lerner").equals("")) {
            message+="<li>"+rb.getString("ERR0003")+"</li>"; // user id
empty
      } else {
            try {
                  DBResultSet rs=dbClient.executeQuery("select Lerner from
Lerner where Lerner="+master.getValue("user.Lerner"));
                  if (rs!=null && rs.next())
message+="<li>"+rb.getString("ERR0011")+"</li>"; // user id duplicate
            } catch(Exception e) { dbg(e); }
      } // end if
      if (master.getValue("user.Nachname").equals("")) {
            message+="<li>"+rb.getString("ERR0004")+"</li>"; // user name
empty
      } // end if
      // build and execute sql instruction
      if (message.equals("")) {
            String sql="insert into Lerner(Lerner, Vorname, Nachname,
Klasse, Bild) values(";
            sql+="'"+master.getValue("user.Lerner")+"'";
            sql+=", '"+master.getValue("user.Vorname")+"'";
            sql+=", '"+master.getValue("user.Nachname")+"'";
            sql+=", '"+master.getValue("user.Klasse")+"'";
            sql+=", '"+master.getValue("user.Bild")+"'";
            sql+=")";
            try { dbClient.executeUpdate(sql);
            } catch(Exception e) {
message+="<li>"+rb.getString("SQL0002")+" "+e+"</li>"; dbg("["+sql+"] ",
e); }
      } // end if
      // send user message
      if (message.equals("")) {
```

```
            master.putValue("message", rb.getString("MSG0003"));
            return true;
      } else {
            master.putValue("message",
rb.getString("MSG0002")+"<ul>"+message+"</ul>");
            return false;
      } // end if
} // end of insertLogin
```

Abbildung 44: Funktionalität für das Einfügen eines neuen Benutzers

E.6 Generierung der HTML-Fragedokumente

```
// insert template
inputStream = new BufferedReader(new FileReader(questionTemplate));
outputStream= new HtmlPage(new PrintWriter(new BufferedWriter(new
FileWriter(questionFile))));
String line=null;
while ((line=inputStream.readLine())!=null) {
      outputStream.addLine(line);
} // end loop next line
inputStream.close();

// create question instance
Question question=null;
try {
      question=new Question(projekt, aufgabe, dbClient, rb);
} catch(Exception e) { return false; }

if (question!=null) try {
      outputStream.addLine(HTML.p(HTML.font(HTML.att(HTML.COLOR,
"#3333FF"), HTML.b(question.getDescription()))));
      QuestionPositionList qpl=question.getPositionList();
      if (qpl!=null) for(int I1=0; I1<qpl.getLength(); I1++) {
            QuestionPosition qp=qpl.item(I1);
            int position=qp.getPosition();
            if (qp.getQuestion()!=null)
outputStream.addLine(HTML.p(HTML.font(HTML.att(HTML.COLOR, "#3333FF"),
HTML.b(qp.getQuestion()))));
            if (qp.getDeclaration()!=null)
outputStream.addLine(HTML.p(HTML.b(qp.getDeclaration())));
            String antworttyp=qp.getType();

            QuestionResultPositionList qrpl=qp.getResultList();
            if (qrpl!=null) {
                  outputStream.addLine(HTML.tag(HTML.P)+
HTML.tag(HTML.TABLE, HTML.att(HTML.BORDER,"1")+
HTML.att(HTML.WIDTH,"100%")));
                  for(int I2=0; I2<qrpl.getLength(); I2++) {
                        QuestionResultPosition qrp=qrpl.item(I2);
                        int nummer=qrp.getNumber();
                        outputStream.addLine(HTML.tr( HTML.td("<GAONINPUT
type=\""+antworttyp+"\" name=\"inputVar"+position+"\"
onChange=\"disableNavigate()\"><VARIABLE
```

```
name=\"inputVar"+position+"["+(nummer-1)+"]\"
/></GAONINPUT>")+HTML.td(qrp.getValue())));
                  } // end loop next Loesungsposition
                  outputStream.addLine(HTML.tagEnd(HTML.TABLE));
            } // end if
      } // end loop next Aufgabenposition
} catch(Exception e) { return false; }
outputStream.addLine(HTML.tagEnd(HTML.FORM));
outputStream.addLine(HTML.tagEnd(HTML.BODY));
outputStream.addLine(HTML.tagEnd(HTML.HTML));
outputStream.write();
return true;
```

Abbildung 45: Generieren der HTML-Fragedokumente

E.7 Überprüfung von Eingaben gegen die Lösungsmenge

```
if (antworttyp.equals("checkbox")||antworttyp.equals("radio")){
      String[] inputVar2=(String[])inputVar.clone();
      // initialize array
      inputVar=new String[countResultPositions];
      for (int I=0; I<inputVar.length; I++) inputVar[I]="0";
      // transfer values to array
      for (int I=0; I<inputVar2.length; I++) {
            int cval=Integer.parseInt(inputVar2[I]);
            inputVar[cval-1]="1";
      } // end loop next input
} // end if
```

Abbildung 46: Transformieren von Benutzereingaben

```
boolean neg=false;
Vector schlagwort=new Vector();
Vector negativwort=new Vector();
StringTokenizer st=new StringTokenizer(loesung.toLowerCase(), ":!", true);
while (st.hasMoreElements()) {
      String token=(String)st.nextElement();
      if (token.equals("!")) neg=true;
      else if (token.equals(":")) neg=false;
      else {
            if (neg) negativwort.addElement(token);
            else {
                  Vector orList=new Vector();
                  StringTokenizer st2=new StringTokenizer(token, "/");
                  while (st2.hasMoreElements())
orList.addElement(st2.nextElement());
                  schlagwort.addElement(orList);
            } // end if
      } // end if
} // end loop next token
```

Abbildung 47: Vorbereitung der Lösungsmenge

```
Vector antwort=new Vector();
StringTokenizer st1=new StringTokenizer(inputValue.toLowerCase(), "
,;.:!?+*-/|§$%&<>()[]{}\"'\\");
while (st1.hasMoreElements()) antwort.addElement(st1.nextElement());
boolean negativ=false;
// check all words
for (int I=0; I<schlagwort.size(); I++) {
      boolean found=false;
      Vector orList=(Vector)schlagwort.elementAt(i);
      for (int j=0; j<orList.size(); j++) {
            for (int k=0; k<antwort.size(); k++) {
                  if
(((String)orList.elementAt(j)).equals((String)antwort.elementAt(k))) {
                        // remove object from vector as keyword may be
required more than one time
                        antwort.removeElementAt(k);
                        found=true;
                        break; // break if word was found in answerList
                  } // end if
            } // end loop next element of answer
            if (found) break; // break if one of the orList was found
      } // end loop next element of orList
      if (found) woerter++;
} // end loop next word
// check negativ words
for (int I=0; I<antwort.size(); I++) {
      for (int j=0; j<negativwort.size(); j++) {
            if
(((String)antwort.elementAt(i)).equals((String)negativwort.elementAt(j))) {
                  negativ=true;
                  break; // break if negative hit
            } // end if
      } // end loop next negative word
      if (negativ) break; // break if negative hit
} // end loop next element of answer
if (woerter==schlagwort.size()&&!negativ) {
      result.result=true;
      result.percent=1;
} else {
      result.result=false;
      if (negativ||woerter==0) {
            result.percent=0;
      } else {
            result.percent=(float)woerter/(float)schlagwort.size();
      } // end if
} // end if
```

Abbildung 48: Freitextauswertung

Literaturverzeichnis

[Apache 2000] *The Apache Software Foundation:* Xcerces Java Parser 1.3.1. `http://xml.apache.org/dist/xerces-j/`

[Baumgartner 1994] *Peter Baumgartner, Sabine Payr:* Wie Lernen am Computer funktioniert. Auszug aus: Lernen mit Software. Didaktik von Bildungssoftware. Innsbruck: Österreichischer Studienverlag, 1994. In: c't 1994, Heft 8

[Behme 1998] *Henning Behme, Stefan Mintert:* XML in der Praxis. Bonn: Addison-Wesley-Longmann, 1998

[Booch 1994] *Grady Booch:* Objektorientierte Analyse und Design. Bonn: Addison-Wesley, 1994

[Booch 1999] *Grady Booch, Jim Rumbaugh, Ivor Jacobson:* Das UML-Benutzerhandbuch. Bonn: Addison-Wesley, 1999

[Bruns 2000] *Beate Bruns, Petra Gajewski:* Multimediales Lernen im Netz. Berlin: Springer, 2000

[CW 2001] *Computerwoche*. Ausgabe 17, 2001

[Enlight 2001] *Enlight:* Teststation – Grundlagen Online-Tests. `http://www.enlight.net/germany.asp`

[Eurostat 1999] *Europäische Kommission/Eurydice/Eurostat:* Schlüsselzahlen zum Bildungswesen in Europa 1999/2000. `http://www.eurydice.com`

[Flanagan 1996] *David Flanagan:* Java in a Nutshell, Deutsche Ausgabe für Java 1.0. Bonn: O'Reilly/International Thomson Verlag, 1996

[Flechsig 1996] *K-H. Flechsig:* Kleines Handbuch didaktischer Modelle. Neuland Verlag, 1996

[Franklin 2001] *Derek Franklin, Brooks Patton:* Flash 5! Animationen fürs Web. München: Markt+Technik Verlag, 2001

[Freire 1998] *Pedro Freire:* export Access to SQL version 2.0. CYNERGI, 1998 `http://www.mysql.com/`

[Gogolla 1999] *Martin Gogolla:* Skript zur Vorlesung Datenbanksysteme. Universität Bremen, Fachbereich 3 – Informatik, 1999

[Goldfarb 1999] *Charles F. Goldfarb, Paul Prescod:* XML-Handbuch. München, London u.a.: Prentice Hall, 1999

[Harold 2000a] *Elliotte Rusty Harold:* XML DTD's, 2000. http://metalab.unc.edu/xml

[Harold 2000b] *Elliotte Rusty Harold:* Processing XML with Java, 2000. http://www.ibiblio.org/xml

[Hoffmann 1999] *Berthold Hoffmann (Betreuung), Projektgruppe Bali:* Projektbericht des studentischen Projektes Bali. Universität Bremen, Fachbereich 3 – Informatik, 1999 http://www.tzi.de/~bali

[Idris 1999a] *Nazmul Idris:* Introduction to DOM 1.0 API, 1999. http://developerlife.com/domintro/default.htm

[Idris 1999b] *Nazmul Idris:* XML and Java Tutorial, Part I, 1999 http://developerlife.com/xmljavatutorial1/default.htm

[Idris 1999c] *Nazmul Idris:* Introduction to XML, Java, Databases and the Web, 1999 http://developerlife.com/dbsourceintro/default.htm

[Kerres 1998] *M. Kerres:* Multimediale und telemediale Lernumgebungen. München, Wien: Oldenbourg, 1998

[Kunze 2001] *Jürgen Kunze:* Computerlinguistik: Vorraussetzungen, Grundlagen, Werkzeuge. Berlin: Humboldt Universität, Lehrstuhl für Computerlinguistik, 2001

[Krüger 2000] *Guido Krüger*. Go To Java 2, Zweite Auflage. Bonn; München; Paris: Addison Wesley, 2000

[Mandel 1998] *Prof. Dr. Heinz Mandel, Dr. Gabi Reinmann-Rothmeier, Dr. Cornelia Gräsel:* Gutachten zur Vorbereitung des Programms „Systematische Einbeziehung von Medien, Informations- und Kommunikationstechnologien in Lehr- und Lernprozesse“. In: Materialien zur Bildungsplanung und Forschungsförderung, Bund-Länder-Kommission 1998, Heft 66

[Matthews 1999] *Mark Matthews:* JDBC Driver for MySQL. Open Source Software, 1999. http://www.mysql.com

[Microsoft 2001] *Microsoft Corp.:* Active Server Pages Developers Network, 2001. http://msdn.microsoft.com/asp/

[Münz 1998] *Stefan Münz:* SelfHTML Version 7.0. http://www.teamone.de/selfhtml

[Neuhaus 1998] *Uwe Neuhaus:* JavaScript 1.1. Bonn: Addison Wesley Longman Verlag GmbH, 1998

[Netscape 1999] *Netscape Communications Corporation:* How SSL Works. http://developer.netscape.com/tech/security

[Oestereich 1998] *Bernd Oestereich:* Objektorientierte Softwareentwicklung. München,

Wien: Oldenbourg, 1998

[Peleska 1996] *Jan Peleska:* Formal Methods and the Development of Dependable Systems. Kiel: Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, 1996

[Pomberger 1996] *Gustav Pomberger, Günther Blaschek:* Software Engineering: Prototyping und objektorientierte Software-Entwicklung. München; Wien: Hanser, 1996

[Seeboerger 2000] *Michael Seeboerger-Weichselbaum:* XML – Das Einsteigerseminar. Kaarst: bhv Verlag, 2000

[Steinmetz 2000] *Ralf Steinmetz:* Multimedia-Technologie – 3., überarb. Auflage. Berlin: Springer, 2000

[Sun 2000] *Sun Microsystems:* Java Server Web Development Kit. `http://java.sun.com/products/servlet`

[Sun 2001] *Sun Microsystems:* Java Server Pages. `http://java.sun.com/products/jsp`

[Thome 1990] *R. Thome:* Wirtschaftliche Informationsverarbeitung. München: Vahlen, 1990

[W3C 1998] *W3C Recommodation:* Working Draft Namespaces in XML, 1998 `http://www.w3.org/TR/WD-xml-names`

[W3C 2000a] *W3C Recommodation:* Document Object Model (DOM) Level 2 Core Specification, 2000 `http://www.w3.org/TR/DOM-Level-2-Core`

[W3C 2000b] *W3C Recommodation:* XML Specification 1.0 (Second Edition), 2000 `http://www.w3.org/TR/REC-xml`

[W3C 2000c] *W3C Recommodation:* XHTML - 1.0 The Extensible HyperText Markup Language, 2000 `http://www.w3.org/TR/xhtml1`

[Wilhelms 1999] *Gerhard Wilhelms, Markus Kopp:* Java professionell. Bonn: MITP-Verlag, 1999

[Zeiger 1999] *Stefan Zeiger:* Servlet Essentials. 1999 `http://www.novocode.com/doc/servlet-essentials/`

Glossar

Dieses Kapitel soll zur Erklärung benutzter Fachbegriffe dienen. Dabei wird kein Anspruch auf Vollständigkeit erhoben.

Applet Java-Programm, das in eine HTML-Seite eingebunden ist. Es wird vor Ausführung mit vom Server geladen und dann auf dem Client ausgeführt.

Array Gruppe von gleichartigen Feldern, auf dessen Elemente ähnlich einer Tabelle über die Spaltenposition zugegriffen werden kann.

Browser Programm zur Darstellung von HTML-Seiten. Die bekanntesten Browser sind Netscape und der Internet Explorer.

Browser-Sandbox Um zu verhindern, dass Applets auf dem Client-Rechner Schaden anrichten, werden ihre Zugriffe auf Rechner-Ressourcen limitiert. Daher spricht man davon, dass sie in einer „Sandbox“ ablaufen.

Client Der „Kunde“. Innerhalb einer Client-Server-Umgebung derjenige Teilnehmer, der die Dienstleistungen konsumiert.

DOM Document Object Model. Ein Modell, um XML-Dokumente als Java-Objekt zu beschreiben

HTML Hyper-Text-Markup-Language. Beschreibungssprache für den Inhalt und das Aussehen einer Internetseite.

PGP Pretty Good Privacy. Verschlüsselungsalgorithmus zum Schutz vor unberechtigtem Zugriff auf Daten.

SAX Simple API for XML. Eine Programmschnittstelle, um aus Java heraus XML-Dokumente zu verarbeiten.

Server Der „Dienstleister“. Innerhalb einer Client-Server-Umgebung derjenige Teilnehmer, der die Dienstleistung erbringt. Gemeint ist im allgemeinen sowohl Computer als auch die entsprechende Software.

Servlet Serverseitiges Programm, dass die Anfragen eines Clients bearbeitet und selbsttätig HTML-Seiten generiert. Ein Servlet ist eingebunden in den eingesetzten Webserver (der nur vorgefertigte HTML-Seiten aus einer vorgegebenen Verzeichnisstruktur an den Client übertragen kann).

SQL Structured Query Language. Standardisierte Abfragesprache für Datenbanken.

SSL Secure Socket Layer. Von der Firma Netscape entwickelter Aufsatz auf das TCP/IP-Protokoll um dieses um die Aspekte der sicheren Übertragung zu ergänzen.

TCP/IP Kommunikationsprotokoll zur Übertragung von Datenpakten über das Internet.

Tag Ein Tag kennzeichnet eine bestimmte Auszeichnung für den darauffolgenden Text (z.B. Beginn Fettdruck hat in HTML das Tag `<b>`). Siehe auch Anhang A: kleiner

Exkurs in XML.

Thread Ein Programmteil, der mehrfach parallel laufen kann. Dabei wird jedoch kein neuer Prozess im Hauptspeicher erzeugt, sondern ein Teil des eigentlichen Programm-Prozesses gemeinsam von allen Threads genutzt. Diese Vorgehensweise ist somit deutlich schneller als das Multitasking über neu erzeugte Prozesse.

XHTML Die XML-Version von HTML.

XML Extensible Markup Language. Metasprache, um Grammatiken von Beschreibungssprachen zu definieren.