

Studiengang: Informatik
Prüferin: Dr. Waltraud Schweikhardt
Betreuerin: Dr. Waltraud Schweikhardt
Beginn am: 20.03.2002
Beendet am: 20.09.2002
CR-Nummer: H.5.2, H.5.3, H.5.4, I.7.2, K.3.1

Studienarbeit-Nr. 1852

**Übersicht und Untersuchung existierender
Autorensysteme und Entwurf für ein System
zum Erstellen multimedialer Lernprogramme**

Thomas Conradt, Swen Mühlleitner

Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20-22
70565 Stuttgart

Inhaltsverzeichnis

1 Einführung.....	6
2 Untersuchung bestehender Autorensysteme.....	8
2.1 Definitionen.....	8
2.2 Bewertungskriterien für Autorensysteme.....	9
2.2.1 Fachliche Kriterien.....	10
2.2.1.1 Didaktik.....	10
2.2.1.2 Methodik.....	10
2.2.1.3 Motivation.....	11
2.2.1.4 Medien.....	12
2.2.1.5 Zielgruppe.....	12
2.2.1.6 Lernkontrollen.....	13
2.2.2 Ergonomische Kriterien.....	13
2.2.2.1 Aufgabenangemessenheit.....	14
2.2.2.2 Selbstbeschreibungsfähigkeit.....	14
2.2.2.3 Steuerbarkeit.....	15
2.2.2.4 Erwartungskonformität.....	15
2.2.2.5 Fehlerrobustheit.....	16
2.2.2.6 Individualisierbarkeit.....	16
2.2.2.7 Lernförderlichkeit.....	17
2.2.3 Systemtechnische Kriterien.....	17
2.2.3.1 Anforderungen.....	17
2.2.3.2 Installation.....	18
2.2.3.3 Software Engineering.....	18
2.2.4 Wirtschaftliche Kriterien.....	18
2.3 Überblick über bestehende Autorensysteme.....	19
2.4 Bewertung bestehender Autorensysteme.....	20
2.4.1 Mediator.....	20
2.4.1.1 Bewertung.....	21
2.4.1.1.1 Fachliche Kriterien.....	21
2.4.1.1.2 Ergonomische Kriterien.....	23
2.4.1.1.3 Systemtechnische Kriterien.....	26
2.4.1.1.4 Wirtschaftliche Kriterien.....	27

2.4.1.2 Zusammenfassung der Ergebnisse.....	27
2.4.2 Question Mark Perception.....	27
2.4.2.1 Bewertung.....	28
2.4.2.1.1 Fachliche Kriterien.....	28
2.4.2.1.2 Ergonomische Kriterien.....	31
2.4.2.1.3 Systemtechnische Kriterien.....	35
2.4.2.1.4 Wirtschaftliche Kriterien.....	37
2.4.2.2 Zusammenfassung der Ergebnisse.....	37
2.5 Ergebnis der Untersuchung bestehender Autorensysteme.....	38
3 Entwurf eines Autorensystems.....	40
3.1 Terminologie.....	40
3.2 Grundlegender Aufbau des Autorensystems.....	40
3.3 Eingesetzte Technologien und Werkzeuge.....	44
3.3.1 Java.....	44
3.3.2 DOM.....	46
3.3.3 XML, XSL und XSLT.....	47
3.3.4 Servlets.....	48
4 Umsetzung des Entwurfs.....	50
4.1 Realisierung des Autorensystems.....	52
4.1.1 Die Klassen des Rahmenprogramms.....	52
4.1.1.1 Die Klasse Init.....	52
4.1.1.2 Die Klasse KomponentenUebersicht.....	53
4.1.1.2.1 Die Klasse DarstellungsPanel.....	55
4.1.1.2.2 Die Aktionsklassen der Schaltflächen im Funktionsbereich.....	56
4.1.1.2.3 Die Aktionsklassen des Popup-Menüs.....	61
4.1.1.3 Die Klasse KomponentenUebersichtIO.....	62
4.1.1.4 Die Klasse InformationsKomponente.....	64
4.1.1.5 Die Klasse KomponentenDaten.....	65
4.1.2 Unterstützende Klassen.....	65
4.1.2.1 Die Klasse Kommando.....	65
4.1.2.2 Die Klasse ExtensionFileFilter.....	66
4.1.2.3 Die Klasse Hilfsfunktionen.....	66
4.1.2.4 Die Klasse FrageAssistent.....	67

4.1.2.4.1 Die Klasse AbbrechenAction.....	68
4.1.2.4.2 Die Klasse HilfeAction.....	69
4.1.2.4.3 Die Klasse ZurueckAction.....	69
4.1.2.4.4 Die Klasse WeiterAction.....	69
4.1.2.4.5 Die Klasse FertigAction.....	69
4.1.3 Starten des Systems über die Klasse TestProgramm.....	70
4.1.4 Realisierung der Komponenten.....	71
4.1.4.1 Implementierung neuer Komponenten.....	71
4.1.4.2 Die Klasse TextKomponente.....	73
4.1.4.3 Die Klasse NumerischeFrage.....	74
4.1.4.4 Die Klasse BewertungsKomponente.....	77
4.2 Verzeichnisstruktur des Autorensystems.....	81
4.3 Darstellung der Lerneinheit.....	82
4.4 Antwortanalyse durch Servlets.....	83
4.4.1 Die Klasse Antwortanalyse.....	83
4.4.2 Die Klasse Auswertung.....	84
4.4.3 Die Klasse NumerischeFrageAuswertung.....	85
4.4.4 Die Klasse Gesamtbewertung.....	85
4.5 Die Konfigurationsdatei "ini.xml"	86
5 Schlußbetrachtung.....	89
5.1 Probleme der eingesetzten Technologien und Werkzeuge.....	89
5.2 Vorschläge für eine Weiterentwicklung.....	90

Abbildungsverzeichnis

Abbildung 1 - Projektübersicht	41
Abbildung 2 - Komponentenübersicht	42
Abbildung 3 - Frageassistent	43
Abbildung 4 - Funktionsweise von Servlets	48
Abbildung 5 - Übersicht über die Klassenstruktur	51
Abbildung 6 - Aufbau des Darstellungspanels	56
Abbildung 7 - Die Klasse FrageAssistent	67
Abbildung 8 - Die Klasse TextKomponente	73
Abbildung 9 - Die Klasse NumerischeFrage	74
Abbildung 10 - Die Klasse BewertungsKomponente	77
Abbildung 11 - Verzeichnisstruktur	81

1 Einführung

Das Lehren und Lernen in Bildungseinrichtungen - wie beispielsweise Schulen - wird heutzutage zunehmend durch den Einsatz von Computern erweitert. Zu bestehender Lernsoftware, die nur bestimmte Bereiche des Lehrstoffs abdecken kann, bilden Autorensysteme eine sinnvolle Alternative. Diese sollen es den Lehrern und Dozenten ermöglichen, auf einfache Art und Weise Lernsoftware für die unterschiedlichsten Gebiete selbst zu erstellen.

Diese neue Art des Lehren und Lernens wird zum Beispiel im Berufsbildungswerk der Paulinenpflege Winnenden eingesetzt, die in dieser Hinsicht mit der Abteilung Visualisierung und interaktive Systeme der Universität Stuttgart zusammenarbeitet. Im Berufsbildungswerk machen gehörlose, schwerhörige und sprachbehinderte Jugendliche ihre Ausbildung.

In einem Gespräch mit dem zuständigen Mitarbeiter wurde allerdings deutlich, daß die Bedienung bestehender Autorensysteme oftmals unhandlich ist bzw. entscheidende Aufgaben (wie z.B. Prüfungen) nicht realisiert werden können. Durch mangelnde Schulungen und fehlerhafte Dokumentationen bleiben den Autoren viele nützliche Funktionen verborgen, was wiederum nicht ermuntert, mit dem System zu arbeiten. Daher werden Autorensysteme in Bildungseinrichtungen nur wenig eingesetzt.

Im Rahmen dieser Studienarbeit soll neben einem Überblick über bestehende Autorensysteme untersucht werden, warum diese nur selten in Bildungseinrichtungen verwendet werden.

Dazu werden in Kapitel 2 zunächst Kriterien zur Bewertung solcher Systeme erarbeitet. Anschließend werden zwei ausgewählte Autorensysteme anhand dieser Kriterien untersucht und bewertet. Hierbei werden vor allem Gründe aufgezeigt, die den praktischen Einsatz durch Lehrer und Dozenten erschweren.

Durch die gewonnenen Erkenntnisse soll ein neues Autorensystem entworfen und das Grundgerüst implementiert werden, worauf in Kapitel 3 eingegangen wird. Dabei sollen die aufgestellten Bewertungskriterien beachtet werden um so ein

anwendungsfreundlicheres System zu entwerfen. Zunächst wird der grundlegende Entwurf eines solchen Systems skizziert sowie Technologien zur Umsetzung beschrieben.

In Kapitel 4 wird die Umsetzung des Autorensystems erläutert. Nach einer Übersicht über die Klassenstruktur werden die einzelnen Klassen detaillierter beschrieben. Außerdem werden weitere Programmstrukturen aufgezeigt sowie die Darstellung der Lerneinheiten und die Antwortanalyse vorgestellt.

Abschließend wird in Kapitel 5 auf Probleme während der Implementierung eingegangen. Daneben werden Vorschläge für eine Weiterentwicklung aufgezeigt.

2 Untersuchung bestehender Autorensysteme

Am Anfang dieses Kapitels werden die theoretischen Grundlagen zur Untersuchung von Autorensystemen gelegt. Ausgehend von einer Definition des Begriffs Autorensystem werden Kriterien zur Bewertung solcher Systeme herausgearbeitet. Nach einem kurzen Überblick über bestehende Autorensysteme werden anhand zweier Beispiele die Bewertungskriterien angewandt und dabei Vor- und Nachteile aufgezeigt.

2.1 Definitionen

Ein Autorensystem ist eine Entwicklungsumgebung, die speziell auf Autoren abgestimmt ist, um damit in einfacher Weise multimediale Software zur Wissensvermittlung und Befragung zu erstellen. Der Einsatzbereich von Autorensystemen reicht vom Lehren und Lernen in Bildungseinrichtungen bis hin zu Web-basierten Kundenbefragungen bzw. Personalentwicklung bei Unternehmen. Im Rahmen dieser Studienarbeit wird die Zielgruppe auf Bildungseinrichtungen eingeschränkt, d.h. Autorensysteme sollen der Erstellung von Lehr- bzw. Lernsoftware dienen. Die Autoren sind in diesem Falle Lehrer und Dozenten.

Als Grundlage von Autorensysteme kann man unterschiedliche Paradigmen heranziehen, die sich nach [Boles] grob in drei Kategorien einteilen lassen: Bildschirm-basierte, Timeline-basierte und Flowchart-basierte Autorensysteme.

"Bildschirm-basierte Autorenwerkzeuge lassen sich dadurch charakterisieren, daß die zu präsentierenden Medienobjekte auf Flächen gelegt werden, die als Karten, Seiten oder auch Dias bezeichnet werden und die im Prinzip einen Bildschirm widerspiegeln, wie ihn ein Benutzer während der Präsentation für ein bestimmten Zeitraum zu sehen bekommt. Eine komplette multimediale Präsentationsanwendung setzt sich aus einer Menge solcher Flächen zusammen, die einem Benutzer in einer bestimmten von ihm durch Interaktionen beeinflussbaren Reihenfolge gezeigt

werden."[Boles]

"Timeline-basierte Autorenwerkzeuge sind dadurch charakterisiert, daß die Medienobjekte auf einer Zeitachse angeordnet werden, die den zeitlichen Verlauf der Präsentation widerspiegelt."[Boles]

"Flowchart-basierte Autorenwerkzeuge sind dadurch gekennzeichnet, daß die Medienobjekte - durch Ikonen bzw. Miniaturen repräsentiert - in Diagrammen durch Kanten miteinander verbunden werden, die den möglichen Verlauf der Präsentation widerspiegeln. [...] Strukturierte flowchart-basierte Autorenwerkzeuge lassen sich dadurch charakterisieren, daß eine (hierarchische) Strukturierung der Diagramme möglich ist, d.h. bestimmte logisch bzw. funktional zusammenhängende Teile können zusammengefaßt und in ein externes Diagramm ausgelagert werden."[Boles]

2.2 Bewertungskriterien für Autorensysteme

Autorensysteme sollen der Erstellung von Lehr- und Lernsoftware dienen. Die Autoren - beispielsweise Lehrer - müssen dabei mit allgemeinen Kenntnissen im Umgang mit Computern und in akzeptabler Einarbeitungszeit die Möglichkeit haben, einfache aber dennoch ansprechende und pädagogisch sinnvolle Ergebnisse zu erzielen. Desweiteren sollte das Autorensystem problemlos auf den von der Zielgruppe eingesetzten Systemumgebungen einsetzbar sein und Erweiterungsmöglichkeiten - zum Beispiel in Form von Plugins - bieten. Letztlich muß der Einsatz eines Autorensystems ökonomisch vertretbar sein.

In den folgenden Abschnitten werden daher fachliche, ergonomische, systemtechnische und wirtschaftliche Kriterien zur Bewertung von Autorensystemen zusammengestellt.

2.2.1 Fachliche Kriterien

2.2.1.1 Didaktik

Unter der Didaktik versteht man die Wissenschaft des Lehrens und Lernens. Im Hinblick auf Autorensysteme soll untersucht werden, ob es möglich ist, didaktisch sinnvolle Lernsoftware zu erzeugen. Das Autorensystem sollte dem Autor Mechanismen zur Verfügung stellen, mit deren Hilfe sich das Lerntempo und die Lerninhalte individuell - beispielsweise für unterschiedliche Benutzergruppen sowie deren Fähigkeiten - gestalten und bewerten lassen. Wichtig ist in diesem Zusammenhang eine hohe Verarbeitungstiefe, d.h. es soll dem Lernenden die Möglichkeit der Wiederholung angeboten sowie verschiedene sensorische Kanäle angesprochen werden.

Der Autor sollte die Möglichkeit haben, Lernziele zu Beginn einer Lerneinheit und eine Zusammenfassung wesentlicher Punkte am Schluß darstellen zu können.

Zur Messung und Bewertung des Lernerfolgs muß das Autorensystem über entsprechende Konzepte, wie z.B. eine Punkteverwaltung, verfügen. Der Lernerfolg sollte dabei für den Lernenden in sinnvoll gewählten, regelmäßigen Zeitabständen präsentiert werden, abgerundet durch informative Rückmeldungen.

2.2.1.2 Methodik

Die Methodik ist die Wissenschaft vom planmäßigen Vorgehen beim Unterrichten. Es werden dabei die unterschiedlichen Unterrichtsmethoden untersucht.

Das Autorensystem sollte die vom Autor gewählte Methode bei der Konzeption der Lernsoftware unterstützen. Daher gilt es festzustellen, welche Vorgehensweisen vom jeweiligen System unterstützt werden. Hierbei wird besonders auf die Methoden "Drill and Practice", tutorielles Lehren, Lernen mit Hypermedia-Systemen und Lernen mit Hilfe von "Guided Tours" eingegangen.

"Drill and Practice" dient der Festigung des Wissens durch Wiederholen und

Anwenden. Es handelt sich dabei hauptsächlich um Übungsprogramme, die eher abfragen als erklären.

Bei tutoriellem Lehren werden einerseits Lehrinhalte aufbereitet und präsentiert und andererseits der Lernerfolg durch Kontrollfragen festgestellt. Das Ergebnis der Kontrollfragen beeinflusst den weiteren Verlauf im Lehrplan. Der Lernende wird also durch die vom Autor vorgegebene Lernreihenfolge, der sogenannten Autorensteuerung, geführt.

Im Gegensatz dazu gibt es beim Lernen mit Hypermedia-Systemen keine Führung. Dem Lernenden wird die Möglichkeit gegeben, ein Sachgebiet selbständig zu erarbeiten, was beispielsweise durch miteinander verlinkte Seiten realisiert werden kann.

Die "Guided Tour", also eine Art geführter Rundgang, bietet keine oder kaum Eingriffsmöglichkeiten für den Lernenden. Er bekommt den Lehrstoff in aufbereiteter Form präsentiert, wird aber nicht durch Rückfragen bzw. Tests geprüft, ob er die Inhalte begriffen hat.

2.2.1.3 Motivation

Den Gesichtspunkt der Motivation muß man differenziert betrachten. Zum einen stellt sich die Frage, ob der Autor dazu animiert wird, mit dem Autorensystem zu arbeiten, zum anderen ist zu klären, ob mit dem Autorensystem attraktive Lehr- und Lernsoftware erstellt werden kann, die zum Lernen motiviert.

Letzteres wird allerdings entscheidend durch den Autor beeinflusst, in dessen Verantwortung es liegt, die Fähigkeiten des Autorensystems sinnvoll einzusetzen um dadurch den Lehrstoff für den Lernenden ansprechend aufzubereiten. Deshalb wird hier nur untersucht, ob das Autorensystem entsprechende Fähigkeiten (z.B. unterschiedliche Medien) bereitstellt.

2.2.1.4 Medien

Mit einem Medium ist allgemein ein Träger von Informationen gemeint. Es ist zu untersuchen über welche Medien das Autorensystem verfügt bzw. welche Medien es unterstützt und ob diese Medien miteinander kombinierbar sind. Gleichzeitig stellt sich die Frage, ob der Einsatz der entsprechenden Medien an der jeweiligen Stelle Sinn ergibt, was jedoch größtenteils durch den Autor beeinflusst wird.

Im Grundrepertoire eines Autorensystems sollten die Medien Text, Grafik, Audio und Video in den gängigsten Formaten abgedeckt sein. Weiterhin sollten zukünftige oder seltener verwendete Formate leicht integrierbar sein.

2.2.1.5 Zielgruppe

Ein entscheidender Punkt zur Beurteilung ist die Feststellung der Zielgruppe des Autorensystems und die Untersuchung, ob das Programm dieser gerecht wird. Dabei spielen insbesondere die vom Autor erwarteten Voraussetzungen und Kenntnisse im allgemeinen Umgang mit Computern bis hin zu Programmierkenntnissen eine große Rolle. Die Erfüllung dieser Voraussetzungen trägt maßgeblich zum Arbeitsergebnis des Autors sowie zu dessen Motivation im Umgang mit dem Programm bei. Autorensysteme, die nicht zielgruppengerecht sind, können den Autor frustrieren und dazu führen, daß die Möglichkeiten zur Erstellung ansprechender Lehr- und Lernsoftware nicht oder nur teilweise ausgenutzt werden.

Die Zielgruppe der entstehenden Lehr- und Lernsoftware wird wiederum maßgeblich durch den Autor selbst bestimmt, wobei das Autorensystem die jeweils benötigte Funktionalität bereitstellen muß. Beispielsweise sollte eine kindgerechte Aufbereitung des Lernstoffs nicht nur auf textueller Basis erfolgen sondern auch Bilder enthalten. Dazu ist es erforderlich, daß das Autorensystem das Einbinden von Bildern ermöglicht, die evtl. auch im Rahmen der Antwortanalyse als Eingabefläche dienen können (sogenannte Hotspot-Fragen).

2.2.1.6 Lernkontrollen

Autorensysteme, die es dem Autor ermöglichen, den präsentierten Lehrstoff überprüfen zu lassen, stellen meist verschiedene Arten der Lernkontrolle zur Verfügung. Wichtige Arten der Lernkontrolle, wie beispielweise Lektionen mit angeschlossener Übung, Übungen und Prüfungen sollten vom Autorensystem angeboten werden.

Als Dialog für die Lernkontrollen sollten verschiedene Fragearten bereit gestellt werden, die zum jeweiligen Kontext des zu prüfenden Lerninhalts passen. Grundfragearten, wie Textfragen oder Multiple-Choice-Fragen sollten vorhanden sein.

Für die Überprüfung der Eingaben des Lernenden ist die Antwortanalyse zuständig. Sie bildet das Kernstück der Lernkontrolle und sollte daher mächtig sein. Wünschenswert ist unter anderem das Erkennen und Korrigieren von Tippfehlern sowie das Erkennen unterschiedlicher Eingaben mit semantisch gleicher Bedeutung (z. B. die Eingabe "zwei" statt der erwarteten Angabe "2" als Ziffer).

2.2.2 Ergonomische Kriterien

Die ergonomische Gestaltung des Autorensystems trägt in wesentlichem Maße zur Bedienungsfreundlichkeit und Motivationssteigerung des Autors bei. Es erleichtert dem Autor den Umgang mit dem System, verkürzt die Einarbeitungszeit und bietet somit die Grundlage für ansprechende und benutzergerechte Lehr- und Lernsoftware. Zur ergonomischen Bewertung von Autorensysteme können die selben Kriterien wie für jede andere Software herangezogen werden. Die Bewertung wird daher auf der Grundlage der Gestaltungsgrundsätze für Dialoge, die im ISO-Standard 9241 Teil 10 verankert sind, durchgeführt.[vgl. Herczeg]

In den folgenden Abschnitten werden die einzelnen Gestaltungsgrundsätze kurz dargestellt und erläutert, inwiefern sie für Autorensysteme wichtig sind.

2.2.2.1 Aufgabenangemessenheit

Unter Aufgabenangemessenheit versteht man, daß der Benutzer bei der Erledigung seiner Arbeitsaufgabe optimal unterstützt werden sollte. Der Dialog soll dabei den Arbeitsaufgaben angemessen sein, also den Benutzer nicht mit Eigenschaften des Dialogsystems unnötig belasten.

In Bezug auf Autorensysteme ist es daher erforderlich, daß dieses so gestaltet ist, daß der Autor seine Lehr- bzw. Lernsoftware einfach und ohne tiefergehende Computerkenntnisse konzipieren kann. Die Dialoge sollen den Autor bei seiner Arbeit in geeigneter Form unterstützen.

2.2.2.2 Selbstbeschreibungsfähigkeit

Selbstbeschreibungsfähigkeit bedeutet, daß Dialoge eines Systems für den Benutzer unmittelbar verständlich sein sollten bzw. auf dessen Verlangen eine angemessene Hilfestellung geben sollten.

Die Dialoge eines Autorensystems müssen so gestaltet werden, daß sich der Autor intuitiv zurechtfinden kann. Daneben sollte es eine gedruckte Dokumentation sowie eine Online-Hilfe geben. Beides sollte in der jeweiligen Landessprache verfaßt und inhaltlich korrekt und verständlich sein.

Die einzelnen Dialogelemente sollten klare Bezeichnungen und Direkthilfen (sogenannte Tooltips) haben. Dabei muß auf die jeweilige Zielgruppe (hier: Autoren von Lehr- und Lernsoftware) geachtet und die Hilfe an deren Kenntnisse angepaßt werden. Die Online-Hilfe sollte praktischerweise eine komfortable Suchfunktion enthalten.

2.2.2.3 Steuerbarkeit

Die Steuerbarkeit eines Dialogs ergibt sich daraus, daß der Benutzer die Geschwindigkeit des Ablaufs sowie die Auswahl und Reihenfolge von Arbeitsmitteln beeinflussen kann. Der Benutzer sollte gleichzeitig die Möglichkeit haben, einen oder mehrere Dialogschritte zurücknehmen zu können (Undo), wobei das Dialogsystem in den Zustand vor der Eingabe gehen muß.

In diesem Zusammenhang kann man zwischen drei verschiedenen Dialogarten unterscheiden: der systemgesteuerte Dialog, bei dem die Initiative beim System liegt und der Benutzer dabei keinen Einfluß auf die nächste Eingabe hat, der benutzergesteuerte Dialog, dessen Kontrolle beim Benutzer liegt, und der hybride Dialog, der eine Mischform der beiden ersten Formen darstellt.

Diese allgemeinen Grundsätze gilt es in einem Autorensystem umzusetzen. Dabei muß entschieden werden, welche Dialogart die für die jeweilige Aufgabe zweckmäßigste ist.

2.2.2.4 Erwartungskonformität

Die Erwartungskonformität eines Systems zeichnet sich durch ein konsistentes Konzept aus. Es entspricht den Erwartungen des Benutzers aus seinen Erfahrungen mit bisherigen Arbeitsabläufen oder aus Schulungen. Der Benutzer sollte das System intuitiv bedienen können. Bei der Konsistenz kann man verschiedene Arten unterscheiden: innere, äußere und methaphorische Konsistenz.

Dies bedeutet im Hinblick auf Autorensysteme, daß die Dialoge über das gesamte Programm hinweg einheitlich gestaltet sein sollten (innere Konsistenz). Gleichzeitig sollte sich die Gestaltung der Dialoge nach dem umschließenden Betriebssystem richten (äußere Konsistenz). Uneinheitliches Dialogverhalten würde den Benutzer unnötig belasten.

Allgemein sollten die Dialoge des Autorensystems die reale Arbeitsumgebung des Autors abbilden (metaphorische Konsistenz).

2.2.2.5 Fehlerrobustheit

Fehlerrobustheit bedeutet, daß ein System trotz fehlerhafter Benutzereingabe das beabsichtigte Arbeitsergebnis mit minimalem oder ohne zusätzlichem Aufwand erreichen kann. Bedienfehler müssen durch sinnvolle Fehlermeldungen gekennzeichnet werden. Sie sollten sich in der Gestaltung deutlich vom restlichen System unterscheiden, beispielsweise durch Verwendung geeigneter Farben. Gleichzeitig sollte das System trotzdem stabil weiterlaufen bzw. die Daten in einen konsistenten Zustand bringen, bevor es sich beendet. Auf Wunsch des Benutzers sollte das System versuchen, fehlerhafte Eingaben automatisch zu korrigieren und dem Benutzer anzuzeigen.

Systemfehler sollten automatisch protokolliert werden, damit die Fehlerbehebung durch den Hersteller leicht möglich ist.

Autorensysteme müssen hier in besonderer Weise beachten, daß Fehlermeldungen durch die Zielgruppe, also die Autoren, verstanden werden.

2.2.2.6 Individualisierbarkeit

Individualisierbare Systeme lassen sich entweder durch den Benutzer an dessen Vorlieben anpassen oder passen sich automatisch dem Verhalten des Benutzers an. Hierbei muß zwischen den Vorteilen der Individualisierung und der Konsistenzwahrung abgewogen werden. Die automatische Anpassung sollte auf Wunsch durch den Benutzer unterbunden werden können.

Beispiele für eine Individualisierbarkeit sind die Anpassung an die Fähigkeiten des Benutzers, z.B. die wahlweise Verwendung von Assistenten, oder die Voreinstellungen für Dateipfade und ähnliches.

2.2.2.7 Lernförderlichkeit

Unter Lernförderlichkeit eines Systems versteht man die sukzessive Anhäufung von Kenntnissen durch die Benutzung des Systems. Das heißt, der Benutzer erlernt das Programm durch den Umgang mit diesem. Da sich der Benutzer also (anfangs) durch Ausprobieren im Programm bewegt, muß sichergestellt sein, daß er bestehenden Daten keinen Schaden zufügen kann. Durch entsprechende Hinweise kann er das Programm Schritt für Schritt kennenlernen.

2.2.3 Systemtechnische Kriterien

Unter systemtechnischen Kriterien versteht man einerseits die Anforderungen des Autorensystems an die gegebene Hard- bzw. Software sowie Installationsaspekte. Andererseits können Prinzipien des Software Engineering als Kriterien zur Bewertung herangezogen werden.

2.2.3.1 Anforderungen

Zu den hardware-technischen Voraussetzungen zählen unter anderem Angaben zum Prozessor, Arbeits- und Festplattenspeicher. Daneben müssen Informationen über die erforderliche Multimedia-Ausrüstung (z.B. Grafik-, Video- und Soundkarte) sowie über die benötigten Peripheriegeräte (Ein- und Ausgabegeräte) benannt werden.

Die Anforderungen an die Software besteht im wesentlichen in der Angabe des Betriebssystems sowie weiterer benötigter Hilfsprogramme.

Wünschenswert sind eine minimale Verwendung der Systemressourcen sowie ein breiter Einsatz auf den unterschiedlichsten Systemen.

2.2.3.2 Installation

Die Installation des Autorensystems und aller notwendigen Hilfsprogramme sollte - sofern erforderlich - einfach gehalten sein. Sie sollte neben einer automatischen Variante dem Autor auch die Möglichkeit überlassen, benutzerdefinierte Einstellungen vorzunehmen. Der Ablauf sowie alle Funktionen sollten so weit wie möglich selbsterklärend und gut dokumentiert sein. Auch die Deinstallation des Programms sollte ohne weiteres möglich sein und das System in den ursprünglichen Zustand zurück versetzen.

Entsprechendes gilt für die Installation der Lehr- und Lernsoftware durch den Lernenden.

2.2.3.3 Software Engineering

Ein gutes Autorensystem zeichnet sich durch die Anwendung anerkannter Prinzipien des Software Engineering aus.

Dies spiegelt sich vornehmlich in offensichtlichen Programmierfehlern wider, die auf eine weniger gründliche Implementierung schließen lassen.

Weitere wichtige Aspekte sind die Ausbaufähigkeit des Autorensystems und das Vorhandensein von Schnittstellen zu anderer Soft- bzw. Hardware. Das Programm sollte modular durch Softwarebausteine erweiterbar sein, die entweder vom Hersteller zugekauft werden müssen oder durch erfahrene Anwender selbst erstellt werden können.

Die Portierbarkeit des Autorensystems ist angesichts der sich im Umlauf befindenden unterschiedlichen Betriebssysteme wünschenswert.

2.2.4 Wirtschaftliche Kriterien

Um den kommerziellen Einsatz eines Autorensystems in Bildungseinrichtungen oder Unternehmen zu ermöglichen, muß sich die Anschaffung und der Betrieb

ökonomisch rechnen.

Hierzu ist ein sinnvolles Preis/Leistungs-Verhältnis notwendig. Neben der Anschaffung spielen auch Wartungskosten sowie die Unterstützung durch den Hersteller eine wichtige Rolle.

2.3 Überblick über bestehende Autorensysteme

Es gibt eine Vielzahl von Autorensysteme auf dem Markt. Im folgenden wird ein Überblick über bestehende Autorensysteme der verschiedenen Kategorien gegeben. Die genannten Autorensysteme werden unter anderem bei [Boles] näher beschrieben.

Bei den Bildschirm-basierten Autorensysteme sind *HyperCard* von *HyperActive Software* für den Apple Macintosh und *Asymetrix ToolBook* für Windows die verbreitetsten.

HyperCard erfreut sich deshalb solch großer Popularität, da mit Hilfe dieses Programms die Entwicklung einer Vielzahl von Anwendungen möglich ist. Daneben ist die Bedienung einfach und die Einarbeitungszeit kurz.

In *ToolBook* entwickelt der Programmierer seine Anwendungen in Form eines Buches. Ähnlich wie *ToolBook* arbeiten die Autorensysteme *Bookmaker* und *Eckermann* aus dem Hause *octOpus*. *Bookmaker* eignet sich vorzugsweise für Präsentationen in Form eines Buches, während mit *Eckermann* auch Lehr- und Lernsoftware erzeugt werden kann.

Konkurrenz bekommen diese Systeme seit kurzem vom *Mediator*, welcher bereits in einigen Bildungseinrichtungen genutzt wird.

Zu erwähnen ist noch *MetaCard*, welches ein Bildschirm-basiertes Autorensystem für UNIX/X11-Systeme ist. *MetaCard* lehnt sich in der Funktionsweise stark an das bereits erwähnte *HyperCard* an.

Im Bereich der Timeline-basierten Autorensysteme gibt es vor allem den *Macromedia Director*. Mit Hilfe des *Macromedia Director* können interaktive, multimediale Präsentationsanwendungen erstellt werden. Der *Macromedia Director*

wird in Schulbuchverlagen, zum Beispiel beim Klett Verlag, zur Erstellung von Lernsoftware eingesetzt.

Weitere Produkte sind *Action!*, *M³Integrator* und *Passport Producer Pro*. Diese Systeme ähneln sich im Aufbau. Sie benötigen keine Programmiersprache sondern benutzen einen Timeline-Editor.

Beispiele für Flowchart-basierte Autorensysteme sind *Apple Media Tool* sowie *Authorware Professional* der Firma *Macromedia*.

Autorware Professional dient der Entwicklung interaktiver Lernprogramme und beinhaltet eigene Werkzeuge zur Erstellung verschiedener Medienobjekte. Desweiteren können über Schnittstellen andere Editoren eingebunden werden. Dabei wird keine Programmiersprache verwendet, sondern es handelt sich hierbei um ein rein graphisch-interaktives Werkzeug.

Apple Media Tool stellt dem Autor ebenfalls eine rein visuelle Programmierung zur Verfügung und arbeitet damit ähnlich wie *Authorware Professional*.

Ein weiteres Flowchart-basiertes Autorensystem ist *Question Mark Perception*, welches eine Alternative zu den bereits genannten Produkten darstellt.

Exemplarisch werden in den folgenden Kapiteln die Autorensysteme *Mediator* und *Question Mark Perception* herausgegriffen und näher betrachtet.

2.4 Bewertung bestehender Autorensysteme

2.4.1 Mediator

Mediator ist ein Autorensystem von Matchware. Es wird derzeit in einer Lite-Version dem Projekt "Lehren für die Zukunft" von Intel, Microsoft sowie der zuständigen staatlichen Stellen für Lehrerfortbildung beigelegt. Daher wird es vermutlich in Zukunft an vielen Schulen eingesetzt. Zur Untersuchung lag nur eine Demo-Version vor, die als Einschränkung nur fünf Seiten pro Projekt zulässt. Daher konnte das

System an einigen Stellen nicht intensiv getestet werden.

Auf den ersten Blick stellt sich der Mediator als Präsentationsprogramm - vergleichbar aber leistungsstärker wie PowerPoint von Microsoft - dar. Mediator ist ein Bildschirm-basiertes Autorensystem, das die Medienobjekte auf eigene Seiten legt, die dann auf dem Bildschirm dargestellt werden. Die Steuerung erfolgt durch den Ereignisdialog. Hier werden die Ereignisse verwaltet, auf die die unterschiedlichen Medienobjekte reagieren. Den jeweiligen Ereignissen können beliebige Aktionen zugeordnet werden.

2.4.1.1 Bewertung

2.4.1.1.1 Fachliche Kriterien

Didaktik

Mit dem Mediator kann man didaktisch sinnvolle Lerneinheiten erstellen. Der Lerninhalt ist nicht beschränkt, er kann durch den Autor individuell gestaltet werden. Es liegt ein linearer Ablauf vor, wobei Verzweigungen durch den Benutzer über sogenannte Hyperlinks auslösbar sind.

Das Lerntempo kann durch den Autor beeinflusst werden. Hierbei ist aufgefallen, daß die Voreinstellung deutlich zu kurz ist. Beispielsweise wird die Rückmeldung auf Fragen nur für etwa eine Sekunde eingeblendet.

Durch die Unterstützung unterschiedlicher Medien können verschiedene sensorische Kanäle angesprochen werden. Der Autor kann durch die Bereitstellung entsprechender Hyperlinks Wiederholungen in die Lerneinheit einbauen. So kann eine hohe Verarbeitungstiefe erreicht werden.

Der Lernerfolg wird beim Mediator in Prozent gemessen. Er ist nach Eingabe der Antwort sichtbar.

Methodik

Mediator unterstützt im wesentlichen zwei Methoden: Lernen mit Hilfe von "Guided Tours" sowie Lernen mit Hypermedia-Systemen. Diese Methoden lassen sich mit Hilfe des Mediators gut umsetzen, weil sie keine Antwortanalyse benötigen. "Drill and Practice" ist ebenfalls möglich, wobei man einschränkend hinnehmen muß, daß es nur zwei Arten von Fragen gibt.

Motivation

Durch die recht übersichtliche Gestaltung der Dialoge, die nur teilweise etwas überfrachtet sind, wird der Autor motiviert, mit dem Mediator zu arbeiten. Dies wird dadurch unterstützt, daß viele Interaktionen über "Drag und Drop" realisiert sind und ein WYSIWYG-Konzept zugrunde liegt.

Dem Autor wird damit ermöglicht, den Lehrstoff attraktiv für den Lernenden aufzubereiten und ihn damit zu motivieren.

Medien

Mediator bietet die Möglichkeit, eine Vielzahl unterschiedlicher Medien einzubinden. Es werden dabei alle gängigen Medienformate unterstützt. Gleichzeitig können neue Medientypen eingebunden werden.

Zielgruppe

Hauptzielgruppe des Mediator sind Lehrer an Bildungseinrichtungen, insbesondere an Schulen. Dies wird auch dadurch deutlich, daß das Programm im Rahmen des oben genannten Projekts an Schulen verteilt wird. Daneben richtet sich der Mediator auch an Unternehmen zur Erstellung hochwertiger Multimedia-Projekte.

Zur Erstellung ansprechender Präsentationen und einfachster Abfragen durch den Autor kann der Mediator sehr gut eingesetzt werden. Die Bedienung ist recht einfach, so daß kein Expertenwissen notwendig ist. Bei Erstellung komplexerer

Lerneinheiten, die die volle Funktionalität des Mediators ausnutzen, stoßen unerfahrene Autoren allerdings schnell an Grenzen. Hierfür sind tiefergehende Kenntnisse im Umgang mit Computern und dem Programm notwendig.

Es liegt in der Hand des Autors, für welche Zielgruppe er Lerneinheiten erstellt. Auch Anfänger können mit den Lerneinheiten problemlos arbeiten.

Lernkontrolle

Mediator stellt in seiner Grundversion die Möglichkeit zur Verfügung, Lektionen mit Übungen sowie Übungen zu erstellen. Prüfungen werden nicht unterstützt.

Allerdings werden lediglich zwei Fragearten angeboten: Textfragen und Multiple-Choice-Fragen. Die dazugehörige Antwortanalyse ist beim Mediator sehr schlecht ausgefallen. Bei der Textfrage werden jeweils drei Eingabefelder für Antworten angeboten, die alle richtig ausgefüllt werden müssen, damit die Frage als richtig gewertet wird. Die Eingabe einzelner richtiger Antworten reicht nicht aus und Tippfehler werden nicht erkannt. Die Erweiterung auf mehr als drei Antworten sowie die Bewertung einzelner richtiger Antworten ist nur mit hohem Aufwand und Programmierkenntnissen möglich.

Die Multiple-Choice-Frage beinhaltet maximal vier vorgefertigte Antwortmöglichkeiten, die nicht gemischt werden können.

Als Rückmeldung wird lediglich "richtig" oder "falsch" gemeldet. Auch ist die Zahl der Fragen auf maximal 100 begrenzt.

Laut Angaben des Herstellers lassen sich weitere und verbesserte Fragearten hinzukaufen, die für die Untersuchung im Rahmen dieser Studienarbeit nicht zur Verfügung gestellt wurden.

2.4.1.1.2 Ergonomische Kriterien

Mediator berücksichtigt wichtige ergonomische Gesichtspunkte, zeigt aber an manchen Stellen auch Schwächen. Im folgenden werden die Ergebnisse der Untersuchung kurz skizziert.

Aufgabenangemessenheit

Das Erstellen von Präsentationen mit dem Mediator ist problemlos und aufgabenangemessen möglich. Der Autor wird durch das System bei seiner Arbeit unterstützt. Lediglich einige Dialoge sind mit zuvielen Funktionalitäten überfrachtet, wie beispielsweise die Auswahl der Muster. Die Erstellung von Übungen mittels des CBT-Assistenten gestaltet sich hingegen deutlich schwieriger. Hier wird die Aufgabenangemessenheit eindeutig verletzt.

Selbstbeschreibungsfähigkeit

Im Konzept des Mediators ist die Selbstbeschreibungsfähigkeit gut verankert. Die Funktionen und Vorgänge erklären sich zum größten Teil selbst und werden nötigenfalls mit geeigneter Hilfe erklärt. Die Hilfe ist aber teilweise noch in englischer Sprache und das Ausdrucken der Hilfe ist nur mit dem externen Programm WordPad möglich.

Die gedruckte Dokumentation, die der getesteten Demoversion in Form eines PDF-Dokuments beilag, ist klar gegliedert und verständlich.

Steuerbarkeit

Die Erstellung von Lerneinheiten ist im Präsentationsteil sowohl system- als auch benutzergesteuert möglich, wohingegen die Erstellung von Fragen nur mit einem Assistenten möglich ist und nachträglich manuell angepaßt werden kann.

In beiden Fällen ist eine Rücknahme bereits gemachter Dialogschritte (Undo) problemlos möglich. Die Navigation durch die Lerneinheit wird durch einen sogenannten Hierarchie-Editor erleichtert.

Erwartungskonformität

Der Mediator ist im Hinblick auf die metaphorischen als auch die äußeren Konsistenz gut gelungen. Es wird versucht, Arbeitsabläufe möglichst genau abzubilden, wobei sich das System auf den Vorgang zum Erstellen einer

Präsentation konzentriert und nicht auf die Erstellung von Übungen. Die Bedienung paßt sich gut in das umschließende Betriebssystem Windows ein, was dem Autor das Arbeiten deutlich erleichtert.

Die innere Konsistenz ist zumeist gewahrt. Auch hier stehen vor allem die beiden Fragearten in der Kritik, die sich nicht wie erwartet verhalten.

Beispielsweise wird von der Antwortanalyse der Textfrage erwartet, daß die Eingabe einer von drei richtigen Antworten zumindest eine Teilpunktzahl gibt. Dies ist jedoch nicht der Fall, da die Eingabefelder mit der dahinterliegenden logischen Operation "UND" verknüpft sind, was bedeutet, daß alle Eingaben richtig sein müssen. Besser wäre hier die Wahlmöglichkeit zwischen einer "UND" und einer "ODER"-Operation.

Fehlerrobustheit

Mediator hat sich während der Untersuchung als einigermaßen fehlerrobust erwiesen. Bedienfehler erzeugen zwar keine irreparablen Schäden, jedoch wird der Autor bei der Behebung auch nicht unterstützt. Es bleibt ihm selbst überlassen herauszufinden, wie er sein Arbeitsergebnis erreichen kann.

Die Fehlermeldungen des Mediators sind in englischer Sprache und geben zumeist erste Hinweise für die Fehlersuche.

Individualisierbarkeit

Mit Mediator besteht die Möglichkeit, die Lerneinheit manuell oder mit Hilfe eines Assistenten (genannt Wizard) zu erstellen. Wird der Wizard verwendet, so kann der Autor nachträglich manuelle Korrekturen vornehmen.

Desweiteren kann der Autor nur noch Voreinstellungen in Bezug auf die Dateiverwaltung vornehmen.

Lernförderlichkeit

Der Mediator ist lernförderlich. Durch Ausprobieren gelingt es dem Autor recht einfach, die grundlegenden Funktionen und Vorgehensweisen zu erlernen. Speziellere Funktionen bleiben dem Autor durch Ausprobieren aber verborgen.

2.4.1.1.3 Systemtechnische Kriterien

Anforderungen

Mediator benötigt einen Intel Pentium Prozessor 166, 32 MB Ram sowie ca. 150 MB Festplattenspeicher. Er kann auf jedem Windows-System ab Version 95 laufen. Es werden eine Soundkarte, Tastatur, Maus und ein CD-Rom Laufwerk benötigt. Über die erforderliche Grafikkarte konnten keine Informationen gefunden werden.

Installation

Die Installation des Mediators gestaltet sich sehr einfach. Der Autor hat die Wahl zwischen automatischer oder benutzerdefinierter Installation. Die Dialoge sind selbsterklärend bzw. bieten ausreichende Hilfestellungen. Auch die Deinstallation ist problemlos möglich.

Für die Installation der Demo-Version wurde ein Registrierungscode benötigt, der schwer zu beschaffen war, da die Registrierungsroutine des Herstellers im Internet nicht ordnungsgemäß funktionierte.

Software Engineering

Mediator zeigte bei der Untersuchung keine Programmierfehler. Es konnten jedoch wegen der Einschränkungen der Demo-Version nicht alle Funktionen ausprobiert werden, so daß diese Aussage nur für die Grundfunktionen gilt.

Durch Zukauf von Plugins kann das System ausgebaut werden. Erweiterungen durch

eigene Programmierung ist mangels Schnittstellen nicht vorgesehen. Mediator ist nur auf Windows lauffähig und unterstützt keine anderen Betriebssysteme.

2.4.1.1.4 Wirtschaftliche Kriterien

Die aktuellen Versionen des Mediators kosten zwischen 69 US\$ für die Standardversion und 899 US\$ für die Expertenversion. Updates sind im Rahmen von Service Packs kostenlos, Upgrades hingegen kostenpflichtig.

Für die Wartung des Mediators steht keine telefonische Unterstützung zur Verfügung, dafür gibt es ein Diskussionsforum und einige FAQs.

2.4.1.2 Zusammenfassung der Ergebnisse

Mediator zeigt seine Stärken vor allem im Präsentationsbereich und bei der überwiegend ergonomischen Handhabung. Die vorliegende Version unterstützt die Abfrage des Lernstoffs nur spärlich und ist sehr kompliziert zu bedienen.

Im wesentlichen kann man Mediator zur komfortablen Präsentation von Lerneinheiten verwenden, z.B. als Ersatz für PowerPoint. Sollte man allerdings nur kleine Vorträge damit konzipieren wollen, ist es fraglich, ob sich der Kaufpreis lohnt.

2.4.2 Question Mark Perception

Question Mark Perception (QMP) ist ein Autorensystem aus dem Hause Questionmark. Zur Untersuchung lag die Version 2.1.5 vom 9.12.1999 vor.

QMP besteht aus vier Teilprogrammen: der Fragen-Manager, der Session-Manager, der Perception-Server und der Enterprise-Reporter.

Im Fragen-Manager kann der Autor einzelne Fragen zusammenstellen und bearbeiten. Im Session-Manager werden die zuvor erstellten Fragen zu

Lerneinheiten in Form von Lektionen, Übungen oder Prüfungen zusammengestellt. Mit Hilfe des Perception-Servers können die erstellten Lerneinheiten mit Zugriffsrechten und Zeitvorgaben versehen und über einen Web-Browser ausgeführt werden. Der Enterprise-Reporter erstellt Berichte über die Ergebnisse der Benutzereingaben.

2.4.2.1 Bewertung

2.4.2.1.1 Fachliche Kriterien

Didaktik

Mit QMP lassen sich didaktisch sinnvolle Lerneinheiten erstellen. Das Lerntempo sowie der Lerninhalt unterliegt keinen Beschränkungen. Es liegt in der Hand des Autors, die Lerninhalte zu gestalten und individuell zu bewerten.

QMP sieht die Möglichkeit vor, Lernziele zu Beginn der Lerneinheit darstellen zu lassen, was aber wohl aufgrund eines Programmfehlers nicht funktioniert. Dafür kann nach dem Durcharbeiten einer Lerneinheit ein Schlußtext sowie die Angabe der erreichten Punktzahl mit einem individuellen Feedback problemlos dargestellt werden.

Wiederholungen können im Session-Manager unter Zuhilfenahme von Strukturierungsmechanismen einfach eingefügt werden. Die hohe Verarbeitungstiefe kann durch den Einsatz verschiedener Medien und die dadurch verbundene Ansprecherung verschiedener sensorischer Kanäle verstärkt werden.

Der Lernerfolg wird sowohl in Punkten als auch in Prozentwerten angegeben und ist nach dem Abschicken der Antwort sichtbar.

Methodik

Mit Hilfe von QMP lassen sich Lerneinheiten erstellen, die den meisten gängigen Unterrichtsmethoden entsprechen. Es ist ein leichtes, Programme zu erstellen, die

sich an das Konzept des "Drill and Practice" anlehnen. Gleichzeitig ist es auch möglich, die weitaus komplizierteren tutoriellen Lehrsysteme zu erstellen, wobei hier die Hauptarbeit beim Autor selbst liegt. Daneben können auch einfache "Guided Tour" Lernsysteme erzeugt werden.

Motivation

QMP motiviert den Autor nur begrenzt. Das Programm ist kompliziert zu bedienen und zeigt an vielen Stellen Inkonsistenzen. Auch nach längerem Umgang mit QMP bleiben dem Autor einige Funktionen verborgen. Der Ansatz, die Arbeit mit Hilfe von Assistenten zu erleichtern, wurde leider nicht konsequent vollendet. Autoren werden kaum motiviert, mit QMP zu arbeiten.

Medien

Durch den Einsatz unterschiedlicher Medien können ansprechende Lerneinheiten konzipiert werden. QMP unterstützt die wichtigsten Medien wie Text, Grafik, Audio und Video in gängigen Formaten. Bei Tests fiel allerdings auf, daß Bilder teilweise verzerrt dargestellt werden.

Zielgruppe

Die Zielgruppe von QMP sind laut Angaben im Prospekt des Herstellers nunmehr weniger Lehrer sondern eher professionelle Ausbilder und Personalverantwortliche. Dieser Personenkreis sollte auf jeden Fall über ausreichend fundierte Computerkenntnisse verfügen, da diese zur Installation und Bedienung von QMP erforderlich sind. Um ansprechende Ergebnisse zu erzielen sind HTML- und Java-Kenntnisse hilfreich.

Die entstehenden Lerneinheiten können auch von Anfängern verwendet werden, wobei das Starten des Perception-Servers unter Anleitung einer erfahrenen Person erfolgen sollte.

Lernkontrolle

QMP stellt alle wichtigen Arten der Lernkontrolle zur Verfügung. Mit Hilfe eines kombinierten Einsatzes des Session-Managers, des Perception-Servers und des Enterprise-Reporters lassen sich leicht Lektionen mit Übungen, Übungen und Prüfungen zusammenstellen. Im Session-Manager werden hierzu Strukturierungsmechanismen, wie die Gruppierung zu Blöcken, Sprungmarken und ähnliches bereit gehalten. Der Perception-Server stellt ein mächtiges Verwaltungswerkzeug der Lerneinheiten dar: Es können Lerngruppen gebildet und den einzelnen Lerneinheiten zugeordnet, Zugriffskontrollen über Paßwörter realisiert und Zeitangaben für Prüfungen festgelegt werden.

QMP implementiert eine Vielzahl unterschiedlicher Fragearten:

- Textfragen: die Antwort besteht aus einem Text
- Lückentext Fragen: der Benutzer muß die Lücken eines Textes ausfüllen
- Numerische Fragen: die Antwort der Frage besteht aus einer Zahl
- Multiple-Choice Fragen: eine Antwort aus mehreren Möglichkeiten ist die richtige
- Fragen mit Mehrfachantworten: mehrere Antworten können aus einer Liste gewählt werden
- Matrixfragen: es wird eine Matrix für die Antworten erstellt
- Zuordnungsfragen: die Antwort wird aus einer Liste gewählt
- Hotspot Fragen: der Benutzer muß einen bestimmten Punkt auf einem Bild anklicken

Die Fragearten sind mit Hilfe von Java erweiterbar, wobei die Vorgehensweise hier nicht dokumentiert ist.

Die Antwortanalyse von QMP muß differenziert bewertet werden. Bei der Erstellung der Antwortanalysen mit dem Assistenten bietet sich dem Autor eine zwar recht einfach zu bedienende aber dafür sehr eingeschränkte Funktionalität. Werden die Fragen - und damit auch die Antwortanalysen - jedoch manuell erstellt, bietet QMP

deutlich mehr Funktionalität: durch die Verknüpfung von Antworten mit logischen Operatoren lassen sich theoretisch alle möglichen Antwortkombinationen abdecken und Tippfehler bis zu einem gewissen Grad herausfiltern. Jedoch ist die Bedienung in der Praxis nur sehr umständlich möglich und erfordert Geduld und tiefergehende (Programmier-) Kenntnisse.

2.4.2.1.2 Ergonomische Kriterien

Die Stärken von QMP im fachlichen Bereich werden von deutlichen Schwächen im Bereich der Ergonomie begleitet. Im folgenden werden die einzelnen ergonomischen Kriterien beim Anwenden von QMP untersucht und anhand von Beispielen erläutert

Aufgabenangemessenheit

Beim Arbeiten mit dem Assistenten ist die Aufgabenangemessenheit im wesentlichen erfüllt. Für die unterschiedlichen Fragetypen stehen unterschiedliche Dialoge bereit, welche den Autor allerdings stark in seiner Arbeit einschränken, da nur Kernfunktionen bzw. -optionen zur Wahl angeboten werden. Beispielsweise ist es mit dem Assistenten nicht möglich, bei Fragen mit Mehrfachantworten für teilweise richtig beantwortete Fragen Punkte zu vergeben und den Benutzer durch entsprechende Rückmeldung darauf hinzuweisen. Auch die Tatsache, daß der Autor aus einer Fülle von Stilarten zur Gestaltung der Fragetexte jeweils nur einen gleichzeitig auswählen kann, stellt eine unnötige Einschränkung dar. Es ist nicht einsichtig, warum man einen Text beispielsweise entweder nur in farbiger Schrift darstellen oder umrahmen kann, aber nicht beides gleichzeitig.

Daneben gestaltet sich die Änderung bestehender Fragen schwierig, da dies nicht mit dem Assistenten durchgeführt werden kann sondern manuell erfolgen muß. Während z.B. die Hotspot-Frage mittels des Assistenten noch sehr einfach erstellt werden kann, in dem der Autor den Hotspot-Bereich mit der Maus markiert, kann eine Änderung nur durch Eingabe neuer Koordinaten erfolgen.

Die Fragenerstellung ohne Assistent bietet hier wesentlich mehr Möglichkeiten. Allerdings wird dabei für jede Frageart der selbe Dialog verwendet, so daß der Autor bei manchen Fragen durch unnötige Eingabefelder belastet wird. Bei jedem Dialog befinden sich beispielsweise Optionen, die lediglich bei der Matrix- oder Zuordnungs-Frage benötigt werden. Die Aufgabenangemessenheit ist hier nicht mehr gegeben.

Selbstbeschreibungsfähigkeit

Die in QMP verwendeten Dialoge sind vielfach nicht selbstbeschreibungsfähig. Autoren mit wenig Erfahrung im Umgang mit QMP sowie mit Computern im allgemeinen können das System kaum intuitiv bedienen.

Beispiele hierfür findet man an vielen Stellen im Ablauf: Bei der Erstellung einer Matrix-Frage ist nicht klar, an welcher Stelle man die Zeilen bzw. die Spalten eingibt. Der Assistent im Sessionmanager hat unklare, nicht dokumentierte Optionen. Auch bei der Eingabe eines Prozentwerts für die Gesamtbewertung kann der Autor nur raten, ob diese die Quote zum Bestehen oder zum Durchfallen angibt. Schwierig ist auch das Publizieren einer Session, da dies nicht intuitiv klar ist.

Die Dialogelemente haben teilweise verwirrende Bezeichnungen. So sind manche Bezeichnungen in deutsch, andere in englisch und die Bewertung einer Frage wird zum Beispiel als "Produkt" bezeichnet. Auch die einzelnen Fragearten haben teilweise unverständliche Bezeichnungen.

Auch Dokumentationen und Hilfe bietet QMP nicht in ausreichender Form. Die gedruckte Dokumentation ist zwar umfangreich, dafür aber schlecht strukturiert und teilweise inhaltsleer. Onlinehilfe gibt es nicht zu allen Vorgängen, sie ist überwiegend auf englisch. Positiv muß erwähnt werden, daß die Onlinehilfe eine Suchfunktion bietet.

QMP verfügt weiterhin über eine Rechtschreibprüfung, die aber weder bei deutsch- noch bei englischsprachigen Fragetexten funktioniert.

Steuerbarkeit

QMP bietet dem Benutzer einen systemgesteuerten Dialog (in Form des Assistenten) und einen benutzergesteuerten Dialog (durch die Erstellung "von Null auf"). Der Autor kann allerdings beim benutzergesteuerten Dialog die Reihenfolge bei der Auswahl der Arbeitsmittel nicht selbst bestimmen, sondern muß eine - nicht dokumentierte - Reihenfolge einhalten, da es sonst zu Programmabstürzen kommen kann. Auch das Publizieren einer Session muß trotz benutzergesteuertem Dialog in vorgegebenen Schritten durchgeführt werden, da hier sonst die Gefahr besteht, daß die komplette Datenbank zerstört wird.

Eine Rücknahme bereits gemachter Dialogschritte (Undo) ist bei Verwendung des Assistenten problemlos möglich, in dem der Autor die "Zurück"-Schaltfläche betätigt. Bei der manuellen Erstellung der Fragen ist eine Rücknahme nur durch Löschen der Frage oder Teilen davon möglich.

Erwartungskonformität

QMP verhält sich an vielen Stellen nicht so, wie der Autor es erwarten würde. Dabei wird vor allem die innere Konsistenz verletzt. So befinden sich die "OK"- und die "Abbrechen"-Schaltfläche nicht in allen Dialogen an der selben Stelle. Ein Autor, der sich nach einiger Zeit daran gewöhnt hat, daß sich die Schaltfläche zum Abbrechen einer Aktion zum Beispiel immer in der rechten unteren Ecke eines Dialogs befindet, löst möglicherweise unbeabsichtigte Folgen aus, wenn in einem anderen - selten verwendeten Dialog - an dieser Stelle die Schaltfläche zur Bestätigung platziert ist.

Daß das Konzept von QMP nicht konsistent ist, läßt sich beispielsweise auch daran erkennen, daß die Eingabefelder einiger Fragearten (z.B. das Beschreibungsfeld der Lückentext-Frage) vordefinierte Werte haben, während bei anderen diese Felder leer gelassen wurden. Auch ist bei manchen Fragen (z.B. dem Lückentext) kein individuelles Feedback für die richtige Antwort möglich und die zwangsweise zu bestimmende ID-Nummer jeder Frage wird nicht verwendet.

Man sieht also, daß die Bedienung von QMP oft nicht erwartungskonform ist. Auch die angebotenen Funktionen zur Antwortanalyse entsprechen nicht den Erwartungen des Autors. Ein Beispiel hierfür ist die Funktion "Tippfehler tolerant": Hier dürfen eine bestimmte Anzahl von Buchstaben fehlen, zuviel oder falsch sein. Der Autor erwartet nun, daß bei aktivierter Funktion Tippfehler aber auch richtig geschriebene Antworten erkannt werden. Allerdings muß der Lernende nun ausdrücklich einen Tippfehler eingeben, damit die Frage richtig beantwortet wird. Diesen Effekt kann man nur durch aufwendige manuelle Änderung der Frage beheben.

Das Hinzuzählen der "Erklärung", also eines Präsentationsblocks zu den Fragearten trägt ebenfalls nicht zur inneren Konsistenz bei.

Die äußere Konsistenz ist im wesentlichen gewahrt. Der Question-Manager sowie der Session-Manager passen sich gut in das umschließende Betriebssystem Windows ein. Die Bedienung und die dazu nötigen Elemente sind an das Look&Feel von Windows angepaßt, so daß sie dem Autor vertraut sind. Lediglich der Perception Server sowie der Enterprise Reporter fallen durch ihren ungewöhnlichen Aufruf durch eine dll-Datei im Browser aus dem Rahmen. Hierauf wird in der Dokumentation hingewiesen.

QMP bildet die Arbeitsabläufe zur Erstellung einer Lektion, Übung oder Prüfung relativ gut ab, so daß die metaphorische Konsistenz beachtet wird. Auch hier fällt lediglich der Schritt von der Zusammenstellung des jeweiligen Lehrstoffs hin zur Veröffentlichung im Perception Server aus dem Rahmen, der sich - wie bereits erwähnt - schwierig gestaltet.

Fehlerrobustheit

QMP ist in wichtigen Punkten nicht fehlerrobust. Fehlerhafte Benutzereingaben erfordern teilweise großen zusätzlichen Aufwand, um das gewünschte Ergebnis zu erreichen. Dies zeigt zum Beispiel die Tatsache, daß eine Frage komplett gelöscht werden muß, wenn der Autor die einzelnen Bestandteile in falscher Reihenfolge aufbaut. Treten während des Publizierens einer Session (durch den Autor verursachte) Fehler auf, muß die Datenbank bzw. das Autorensystem neu installiert werden.

Die Fehlermeldungen von QMP sind teilweise nicht vorhanden oder nicht verständlich formuliert. Auch sind sie nicht einheitlich in deutscher Sprache gehalten oder besonders gekennzeichnet.

Individualisierbarkeit

QMP bietet dem Autor die Möglichkeit, sowohl im Question Manager als auch im Session Manager zwischen der manuellen Vorgehensweise oder dem Einsatz des Assistenten zu wählen. Die getroffene Entscheidung ist für das zukünftige Arbeiten nicht bindend, d.h. es kann für jede einzelne Frage bzw. jede einzelne Session entschieden werden. Weitere Individualisierungen sind von QMP nicht vorgesehen.

Lernförderlichkeit

QMP ist nicht lernförderlich. Auch nach längerer Zeit bleibt die Benutzung des Systems vielfach unklar. Eine Vorgehensweise nach dem Prinzip "Trial and Error" ist mit QMP nicht möglich, da schon durch kleine Fehlbedienungen größere Schäden verursacht werden können. Hinweise, die dem Autor Aufschluß über seine Bedienfehler geben und ihm den richtigen Weg aufzeigen fehlen gänzlich.

2.4.2.1.3 Systemtechnische Kriterien

Anforderungen

Zum Betreiben von QMP ist mindestens ein Intel Pentium Prozessor notwendig. Die Größe des Arbeitsspeichers hängt vom jeweiligen Betriebssystem ab. Über den benötigten Festplattenplatz gibt es keine Angaben, man müsste jedoch mit ca. 25 MB auskommen. Die Multimedia-Ausstattung sollte eine Grafikkarte mit einer Auflösung von mindestens 800x600 Pixel haben. Unterstützt werden gängige Ein- und Ausgabegeräte wie Tastatur, Maus, Bildschirm und Drucker. Daneben ist ein CD-Rom Laufwerk notwendig.

Als Betriebssystem kann Windows ab der Version 95 eingesetzt werden. Zur Präsentation des Lehrstoffs wird entweder der Internet Explorer oder Netscape jeweils ab Version 4 benötigt.

Installation

Zum Betreiben von QMP ist eine Installation erforderlich. Die Systemteile Question Manager und Session Manager werden auf Wunsch automatisch installiert und ebenso deinstalliert. Der für den Perception Server und Enterprise Reporter erforderliche Webserver muß allerdings manuell installiert, konfiguriert und deinstalliert werden. Dies ist vor allem für unerfahrene Benutzer äußerst schwierig. Die Installation ist nicht ausreichend dokumentiert und nicht selbsterklärend.

Software Engineering

Bei der Untersuchung von QMP wurden mit Ausnahme des Programmfehlers beim Publizieren einer Session keine großen, offensichtlichen Fehler entdeckt. Jedoch gibt es sehr viele kleinere Fehler. Beispielsweise wird der Begrüßungstext nicht dargestellt und es gibt Formatierungsfehler beim Lückentext.

Die Antwortanalyse von QMP kann durch den Typ "Java" ausgebaut werden. Falls der Autor über entsprechende Programmierkenntnisse verfügt, kann er durch diese Schnittstelle eigene Fragearten erschaffen und die entsprechende Antwortanalyse individuell gestalten, wobei die Vorgehensweise nicht im Handbuch dokumentiert ist. Die Erweiterung durch Plugins ist nicht vorgesehen.

Das Autorensystem QMP selbst kann nur auf verschiedenen Windows-Varianten laufen. Eine Portierung auf andere Betriebssysteme ist bislang nicht möglich. Die Präsentation des Lehrstoffs und die dazugehörigen Fragen werden über den Perception Server angeboten. Daher ist es für den Lernenden möglich, auch mit anderen Betriebssystemen darauf zuzugreifen, vorausgesetzt es wird der entsprechende Browser dazu angeboten.

2.4.2.1.4 Wirtschaftliche Kriterien

Zum Preis der hier untersuchten Version gibt es keine Preisinformationen. Die aktuelle Version von QMP kostet zwischen 2000 Euro für Hochschulen und 4000 Euro für Unternehmen.

Die Wartung wird derzeit über einen sogenannten "Software Support Plan" angeboten. Hierbei erhält man für eine jährliche Gebühr von 25% des Lizenzpreises kostenlose Hilfe per E-Mail, Fax und Telefon sowie Softwareupdates.

2.4.2.2 Zusammenfassung der Ergebnisse

QMP unterstützt mit seiner Vielzahl von Fragearten sowie den dahinterstehenden Routinen zur Antwortanalyse den Autor bei der Erstellung von Lerneinheiten in verschiedensten Formen. Durch die Verwendung des eingebauten Assistenten hat er die Möglichkeit, den präsentierten Stoff durch einfache Fragen zu überprüfen oder durch manuelle Erstellung auch komplexere Fragen zusammenzustellen.

Diese guten Ansätze werden allerdings durch eine äußerst schlechte ergonomische Gestaltung des Autorensystems relativiert. Autoren, die nicht über die notwendigen Kenntnisse im Umgang mit Computern und Betriebssystem sowie in der Programmierung verfügen, werden nicht motiviert, mit QMP zu arbeiten.

Dies wird auch durch ein Gespräch mit Mitarbeitern des Berufsbildungswerks der Paulinenpflege in Winnenden unterstrichen, die das Autorensystem ursprünglich zu Ausbildungszwecken einsetzen wollten. Hier wurden auch die vielen Fragearten und deren Auswertung gelobt, die Bedienung allerdings als nicht sehr gelungen bezeichnet. Die Einlernzeit für Lehrer ist sehr groß, was gleichzeitig hohe Kosten verursacht. Auch die Dokumentation wurde bemängelt. Daher wird QMP inzwischen nicht mehr verwendet und ein Update auf eine aktuelle Version ist derzeit nicht geplant.

Durch den hohen Kaufpreis sowie die laufenden Kosten für den Support lohnt sich ein Einsatz an Bildungseinrichtungen kaum. Der Hersteller von QMP zielt mit seiner aktuellen Version auch zunehmend auf Firmenkunden ab, die das System zur

Schulung der Mitarbeiter sowie zu Umfragen bei den Kunden verwenden.

2.5 Ergebnis der Untersuchung bestehender Autorensysteme

Bei der Untersuchung der beiden Autorensysteme "Mediator" und "Question Mark Perception" stellte sich heraus, daß beide Systeme nicht den gewünschten Anforderungen entsprechen.

Der Mediator ist im Präsentationsbereich sehr leistungsfähig und zeichnet sich durch eine überwiegend ergonomische Handhabung aus. Dafür sind im Bereich der Antwortanalyse und der damit verbundenen Erstellung von Fragen deutliche Defizite erkennbar. Sinnvolle Übungen oder Prüfungen können mit dem Mediator nicht erstellt werden.

Question Mark Perception hingegen bietet eine Fülle von Fragearten sowie eine leistungsstarke Antwortanalyse. Gleichzeitig lassen sich einfache Präsentationen mit den gängigsten Medientypen gut verwirklichen. Die Schwäche des Systems liegt eindeutig in der Bedienung, da ergonomische Kriterien bei der Konzeption nicht berücksichtigt wurden. Dies macht den Umgang mit dem System unnötig schwer und demotiviert den Autor. Daneben sind noch etliche kleinere Programmierfehler vorhanden, so daß manche Funktionen nicht korrekt arbeiten.

Beide Systeme können nicht auf verschiedenen Betriebssystemen eingesetzt werden. QMP bietet im Gegensatz zum Mediator die Möglichkeit der Erweiterung durch den Benutzer. Hierzu sind allerdings Programmierkenntnisse erforderlich, da die Erweiterbarkeit durch eine Java-Schnittstelle realisiert wurde. Beim Mediator können zusätzliche Komponenten nur hinzugekauft werden.

Die Anschaffung der Systeme dürfte sich angesichts des Kaufpreises gerade für Bildungseinrichtungen kaum lohnen, da diese ohnehin nur über knappe finanzielle Mittel verfügen.

Um die beschriebenen Schwächen zu beseitigen ist es erforderlich, ein neues Autorensystem zu entwerfen, das sich in wesentlichen Teilen von den bereits bestehenden Systemen abhebt.

Das Autorensystem sollte offen, erweiterbar und benutzerfreundlich sein, wobei es

das Ziel ist, multimediale Präsentationskomponenten mit leistungsfähigen Antwortanalyseroutinen zu kombinieren.

Das Autorensystem soll auf unterschiedlichen Plattformen lauffähig sein, sowie ein Lernen über ein Netzwerk (z.B. Internet) ermöglichen. Durch die Bereitstellung geeigneter Schnittstellen soll die Erweiterbarkeit sichergestellt werden.

3 Entwurf eines Autorensystems

3.1 Terminologie

Im nachstehenden Entwurf sowie bei der Realisierung wird in Hinblick auf das Autorensystem die folgende Terminologie benutzt.

Die mit Hilfe des Autorensystems entstehende Lernsoftware wird hier "Lernprojekt" genannt. Es besteht aus einzelnen Projektseiten, den "Lerneinheiten". Diese wiederum sind aus einzelnen "Komponenten" aufgebaut, die Präsentations-, Frage- und Bewertungselemente darstellen.

Für die unterschiedlichen Personengruppen, die mit dem Autorensystem interagieren, wird folgende Terminologie genutzt: Als "Autoren" werden diejenigen Personen bezeichnet, die im Rahmen ihrer Lehrtätigkeit das Autorensystem zur Konzeption von Lernprojekten verwenden. Die "Lernenden" wenden diese Lernprojekte an. "Entwickler" sind die Personen, die das Autorensystem - z.B. durch Programmieren neuer Komponenten - erweitern.

3.2 Grundlegender Aufbau des Autorensystems

Der grundlegende Aufbau des Autorensystems gliedert sich in drei Ebenen: die Projektübersicht, die Komponentenübersicht und der Komponenteneditor. Das Autorensystem versetzt den Autor in die Lage ein Lernprojekt zu erstellen, welches sich in unterschiedliche Lerneinheiten gliedert. Die Ebenen spiegeln dabei den Grad der Detaillierung wieder, mit dem der Autor auf seine Lernprojekt blickt.

Der Autor hat die Möglichkeit, sowohl reine Präsentationen als auch Präsentationen gemischt mit Übungsfragen oder Prüfungen zu erstellen.

Die Projektübersicht (siehe Abbildung 1) stellt eine globale Sicht auf das Lernprojekt dar. Hier hat der Autor eine Übersicht über die einzelnen Lerneinheiten und kann die Autorensteuerung, also eine vorgegebene Reihenfolge der Lerneinheiten, festlegen.

Desweiteren gibt es die Möglichkeit, neue Lerneinheiten anzulegen, bestehende zu laden sowie zu ändern. Zusätzlich können in der Projektübersicht die Voreinstellungen durch den Benutzer geändert sowie neue Komponenten durch den Entwickler registriert werden.

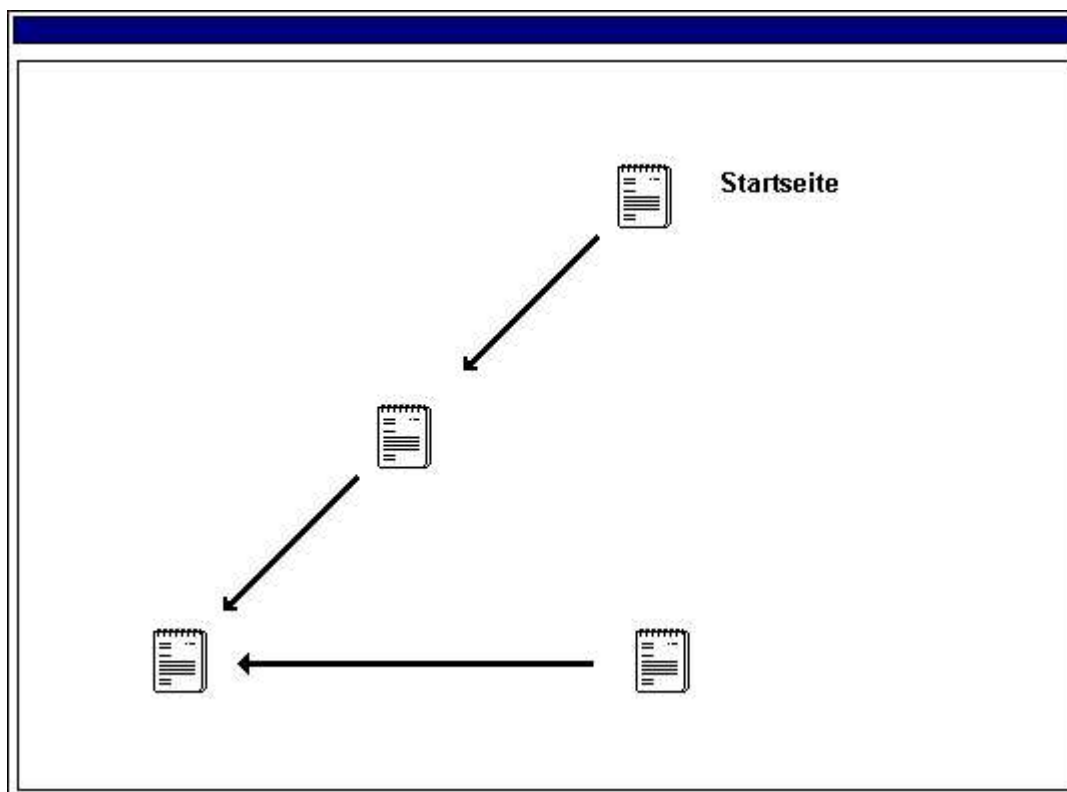


Abbildung 1 - Projektübersicht

Wird in der Projektübersicht eine neue Lerneinheit angelegt oder eine bereits bestehende editiert, so öffnet sich die Komponentenübersicht (siehe Abbildung 2). Hier hat der Autor einen Überblick über die Komponenten der Lerneinheit, kann diese bearbeiten, verschieben, löschen oder neue hinzufügen. Dazu stehen dem Autor drei Schaltflächen zur Verfügung, die ihm ermöglichen, Präsentationskomponenten (z.B. Texte, Bilder), Fragekomponenten (z.B. Textfrage, Multiple Choice Frage) sowie eine Gesamtbewertung hinzuzufügen. Die Gesamtbewertung gilt für die ganze Lerneinheit und darf daher nur einmal am Ende einer Lerneinheit stehen. Wenn keine Fragen vorhanden sind, darf auch keine Gesamtbewertung eingebaut werden.

Der Autor kann die Komponentenübersicht schließen, in dem er die eventuell

gemachten Änderungen speichert oder sie verwirft.

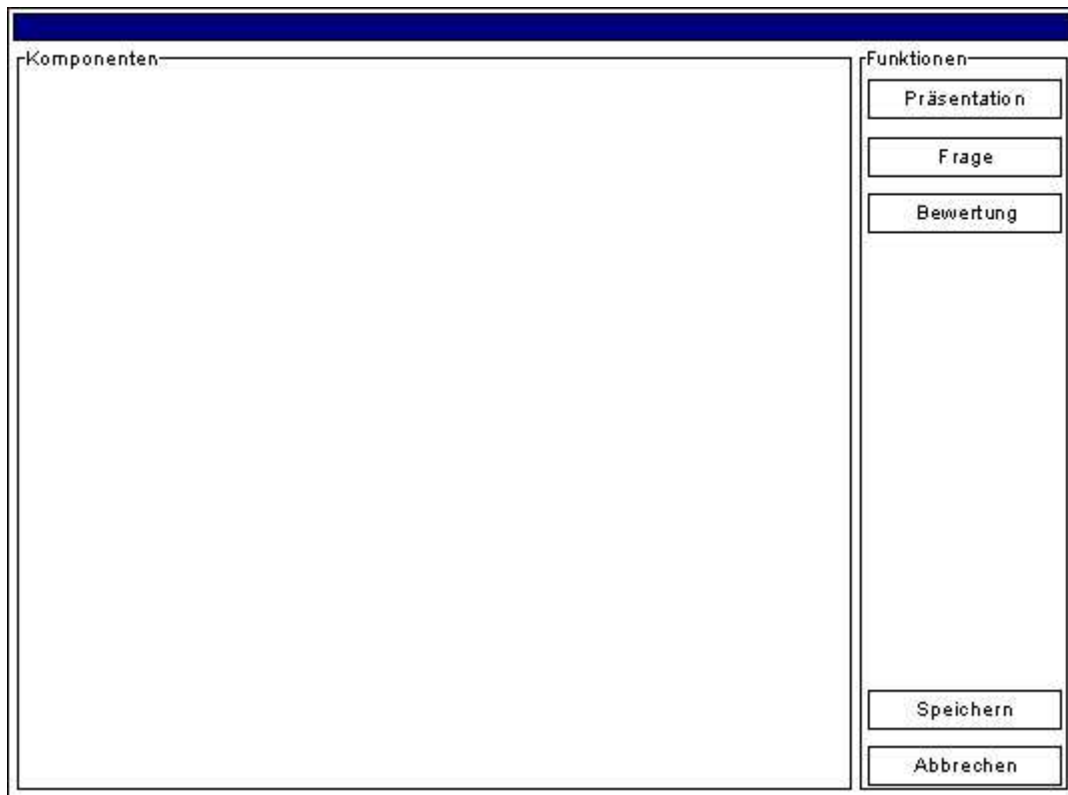


Abbildung 2 - Komponentenübersicht

Um auf die dritte Ebene, den Komponenteneditor, zu gelangen, muß man eine neue Komponente erstellen oder eine bereits bestehende bearbeiten. Hierbei muß zwischen zwei Arten von Editoren unterschieden werden: den internen und den externen Editoren. Interne Editoren sind die eigentlichen Komponenteneditoren. Sie verfügen über die notwendige Funktionalität, um mit dem Autorensystem zusammenzuarbeiten. Externe Editoren stellen ein Hilfsmittel für diejenigen Komponenten dar, die (noch) keinen eigenen Editor haben, und haben teilweise nur eingeschränkte Funktionen. Im Rahmen dieser Studienarbeit wird ein interner Editor exemplarisch für alle anderen realisiert werden. Er wird in Form eines Frageassistenten (siehe Abbildung 3) umgesetzt.

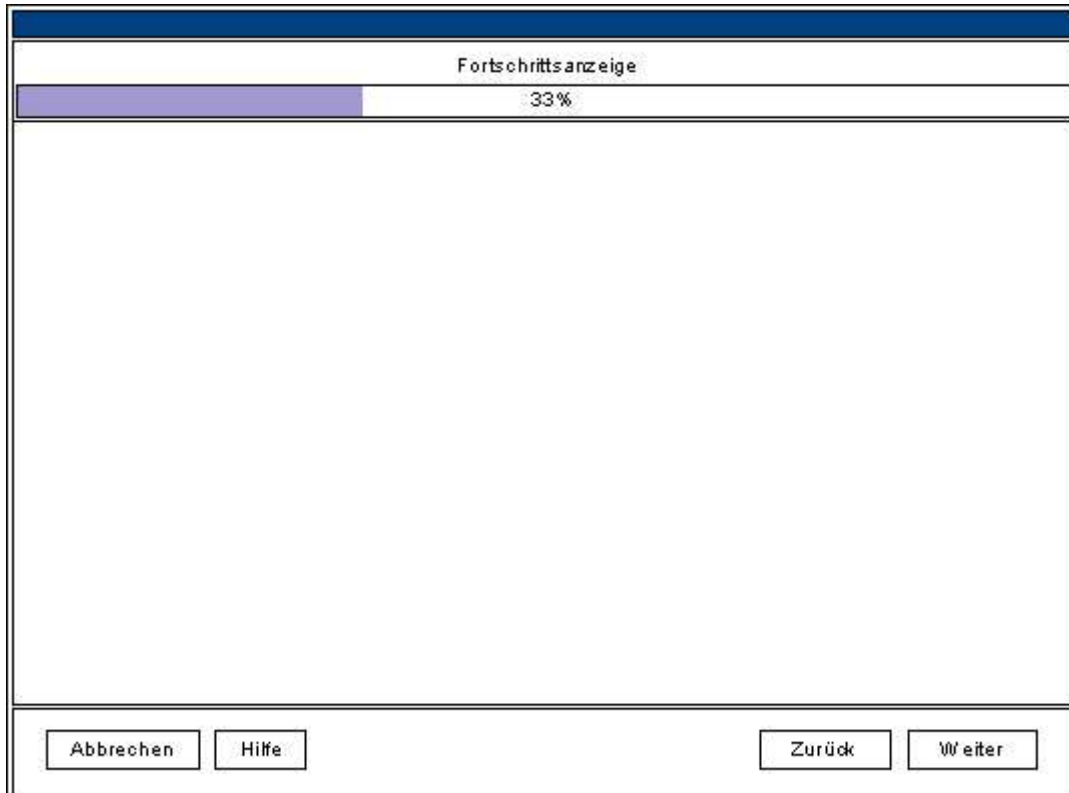


Abbildung 3 - Frageassistent

Der Lernende bedient das Lernprojekt über einen Web-Browser. Hierbei werden die Lerneinheiten in der vom Autor bestimmten Autorensteuerung präsentiert. Die Lerneinheiten stellen jeweils eine HTML-Seite dar und beinhalten die zuvor eingefügten Komponenten. Sind in einer Lerneinheit Übungsfragen vorhanden oder handelt es sich um eine Prüfung kann der Lernende über entsprechende Dialogelemente Eingaben tätigen. Nach Abschluß der Eingabe wird die Seite über eine Schaltfläche zur Auswertung abgeschickt. Das Ergebnis wird dem Lernenden auf der folgenden Seite gezeigt.

3.3 Eingesetzte Technologien und Werkzeuge

Um den Entwurf in der vorgestellten Weise zu realisieren, wurden verschiedene Technologien und Werkzeuge ausgewählt, die in den folgenden Abschnitten dargestellt werden. Dabei wird insbesondere darauf eingegangen, warum sie verwendet werden.

3.3.1 Java

Java wurde als Programmiersprache gewählt, weil sie verschiedene Vorteile bietet (vgl. [Horst/Cor]). Einer der wichtigsten Vorteile von Java ist seine Laufzeitbibliothek, die es ermöglicht plattformunabhängigen Code zu erzeugen, der unter allen Windows-Versionen, Solaris, Unix, Macintosh usw. lauffähig ist. Dies ist gerade im Hinblick auf die internetfähige Programmierung ein entscheidender Punkt.

Ein weiterer Vorteil ist, daß Java eine objektorientierte Sprache ist, die sich an die Syntax von C++ anlehnt. Im Gegensatz zur strukturierten Programmierung können mit Hilfe der Objektorientierung komplexere Programme leichter entwickelt werden. Bei der Entwicklung von Java wurde versucht, elf Entwicklungsziele zu realisieren: einfach, objektorientiert, verteilt, robust, sicher, architekturneutral, portabel, interpretiert, leistungsfähig, multithreaded und dynamisch.

Einfach: Im Gegensatz zu C++ stellt Java eine bereinigte C++-Syntax dar. So braucht man unter anderem nichts mehr über Zeigerarithmetik zu wissen, die bei falscher Anwendung bis zum Programmabsturz führen konnten. Jedoch wurde dies nicht konsequent durchgeführt und ein paar umständlichere Konstrukte (wie z.B. die switch-Anweisung) blieben noch in Java erhalten.

Objektorientiert: Die Objektorientierung hat sich gegen die strukturierte Programmierung weitestgehend durchgesetzt und in Java wird dem Entwickler die Objektorientierung angeboten.

Verteilt: Java bietet dem Netzwerkprogrammierer eine Fülle von Hilfsmittel. So ist unter anderem die serverseitige Programmierung mittels Servlets sehr einfach. Servlets werden auch in diesem Projekt verwendet. Auf sie wird später nochmals genauer eingegangen.

Robust: Java versucht schon zur Compilierzeit die meisten Fehler zu entdecken, damit es während der Laufzeit zu weniger unerwarteten Fehlern kommt.

Sicher: Auch Java ist nicht absolut sicher gegen Angriffe von außen. Es wird jedoch versucht, alle Sicherheitslücken so schnell wie möglich zu schließen.

Architurneutral: Da Java auf plattformunabhängigen Code abzielt, werden Bytecode-Anweisungen erzeugt, die nichts mit der darunterliegenden Plattform zu tun haben. Diese Bytecode-Anweisungen können von jedem System interpretiert werden, das einen Java-Interpreter (Java Virtual Machine) bereitstellt.

Portabel: Die Portabilität des in Java erzeugten Code ist dadurch gegeben, daß es keinerlei Konstrukte gibt, die vom darunterliegenden System abhängig sind. So haben zum Beispiel ganze Zahlen in Java immer dieselbe Größe von 32 Bit.

Interpretiert: Im Gegensatz zu Compilern, besitzt der Java-Interpreter hier seine Schwächen. Da der Code zeilenweise übersetzt wird und nicht auf einmal vor Ausführen des Programms, kommt es zu Leistungseinbußen, die man vor allem auf älteren Systemen zu spüren bekommt.

Leistungsfähig: Dieser Punkt hängt eng mit dem vorherigen Punkt zusammen. Durch den Java-Interpreter kommt es zu deutlich langsameren Ausführungszeiten als bei Compilern. Jedoch muß in Java ein Interpreter benutzt werden, da man sonst die Architurneutralität nicht erreichen kann.

Multithreaded: Im Gegensatz zu anderen Programmiersprachen, ist die Multithreading-Programmierung in Java relativ einfach zu realisieren.

Dynamisch: Java ist eine dynamische Sprache, was bedeutet, daß es während des Programmablaufs möglich ist, Typinformationen zu bekommen. Dies geschieht unter anderem durch den Reflection-Mechanismus, der es in dem vorliegenden Projekt ermöglicht, neue Komponenten hinzuzufügen, ohne dabei den bestehenden Quellcode ändern zu müssen.

Java eignet sich für das Projekt in mehreren Punkten. Aufgrund der Art, wie Java entwickelt wurde, ist es einfach, ein Programm zu schreiben, das auf verschiedenen Plattformen lauffähig ist. Auch die Integration neuer Komponenten (Plugins) ist leicht möglich. Der entscheidende Punkt aber ist, daß die zur Realisierung des Programms verwendeten Technologien bzw. Mechanismen allesamt mit Java zusammenarbeiten bzw. Java-Erweiterung sind. So sind die Servlets beispielsweise eine Java-Erweiterung für die serverseitige Programmierung. XML wurde an Java angelehnt und mit Hilfe von DOM ist es möglich, Dateien im XML-Format einfach zu bearbeiten.

3.3.2 DOM

Mit Hilfe von DOM lassen sich XML-Dateien auf einfache Weise in das Autorensystem laden bzw. können daraus gespeichert werden.

Hierbei wird das ganze Dokument eingelesen und in einer Baumstruktur gespeichert. Die Informationen befinden sich nun in den Knoten dieses Baumes. Mit Hilfe von Baumoperationen können diese Informationen nun anders angeordnet werden, so kann man z.B. ganze Teilbäume verschieben. In dem vorliegenden Projekt wird jedoch nur das Auslesen der Informationen benötigt.

Im umgekehrten Fall, beim Speichern einer XML-Datei, wird ein Baum angelegt und die Informationen werden in die Knoten des Baumes gelegt. Ist der Baum fertig gestellt, wird dieser als ganzes in eine XML-Datei geschrieben.

3.3.3 XML, XSL und XSLT

XML (Extensible Markup Language) ist eine Meta-Sprache und stammt von SGML ab. Mit Meta-Sprachen können andere Sprachen definiert werden. Der Vorteil von XML gegenüber dem älteren SGML liegt darin, daß XML nicht so komplex ist wie SGML (vgl. [McLau]).

Im Gegensatz zum HTML, in dem die Tags fest vorgeschrieben sind, ist dies in XML nicht so. Dadurch wird die Einschränkung von im voraus feststehender Tags aufgehoben. XML ermöglicht es den Inhalt von Daten beliebig zu beschreiben. Es ist lediglich erforderlich, daß man sich an die allgemeine Struktur hält, die von XML vorgegeben wird.

Zusammen mit XML und der Technologien, die auf XML aufsetzen, bekommt der Entwickler eine große Bandbreite bei der Verwaltung und Übertragung von Daten geboten. Zusätzlich wurde XML so entwickelt, daß vor allem Java-Entwicklern, damit leicht arbeiten können, was sich für das Autorensystem wiederum hervorragend eignet.

XML stellt also eine Möglichkeit dar, Daten mit Hilfe sogenannter Tags zu umschreiben. Eine Anwendung (z.B. ein Web-Browser) kann eine XML-Datei alleine jedoch nicht in der gewünschten Form darstellen. Die Datei muß zuerst umgewandelt werden, wofür XSL und XSLT verwendet werden.

In XSL (Extensible Stylesheet Language) können Vorlagen (sogenannte Stylesheets) geschrieben werden. In diesen Vorlagen wird beschrieben, wie die XML-Daten umgewandelt werden sollen. Dies ermöglicht es, XML-Daten von einem Format in ein anderes zu konvertieren. Ein gängiger Fall ist hierbei das Umwandeln einer XML-Datei in eine HTML-Datei. XSL sorgt dabei für eine komplette Trennung der Daten (dem Inhalt) von deren Darstellung.

Die eigentliche Übersetzung in ein anderes Format erfolgt durch die XSLT (Extensible Stylesheet Language Transformation). Während einer derartigen Transformation wird ein XML-Dokument und die zugehörigen XSL-Vorlagen miteinander kombiniert. Als Ergebnis erhält man die umgewandelten XML-Daten.

3.3.4 Servlets

Servlets sind Programme, die auf einem Server laufen und ankommende Anfragen von Clients bearbeiten. Sie sind daher sehr gut für die serverseitige Programmierung geeignet und stellen eine Java-Erweiterung dar (vgl. [Hall]).

Hierfür werden zuerst alle vom Benutzer gesendeten Daten gelesen, die zuvor von ihm in Formularfelder einer HTML-Seite eingegeben wurden. Aufgrund dieser Daten erfolgt eine Auswertung die in eine weitere HTML-Seite eingebettet werden. Dieses Dokument wird schließlich an den Client zurückgesendet.

Da man die Eingaben des Benutzers im voraus nicht wissen kann, muß die zurückgesendete Seite dynamisch erzeugt werden. Und hierin liegt der große Vorteil der Servlets, mit Hilfe derer man dies erreichen kann.

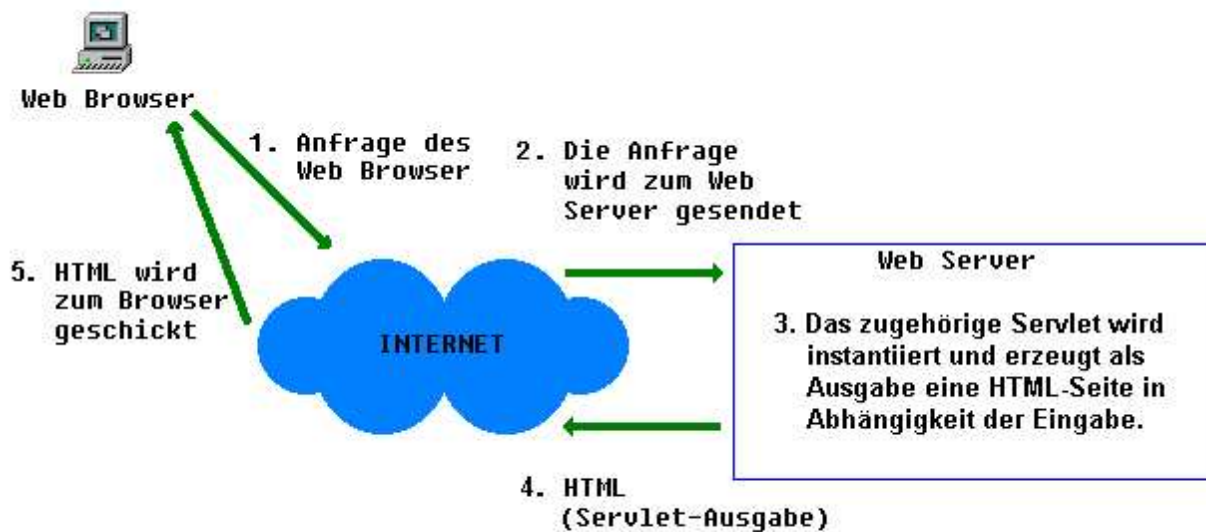


Abbildung 4 - Funktionsweise von Servlets

Außerdem bieten Servlets eine Reihe von Vorteilen gegenüber der CGI-Programmierung. Diese liegen dabei in der Effizienz, Bedienung, Mächtigkeit, Portierbarkeit, Sicherheit und beim Preis.

Effizienz: Im Gegensatz zu CGI wird nicht für jede HTTP-Anfrage ein neuer Prozess gestartet, sondern nur ein leichtgewichtiger Java-Thread, der das System weitaus

weniger belastet.

Bedienung: Servlets werden vollständig in Java programmiert, weshalb sie sich bestens für das Autorensystem eignen, da alle verwendeten Mechanismen mit Java zusammenarbeiten. Außerdem bieten Servlets dem Entwickler eine Vielzahl von Möglichkeiten an, um HTML-Formulardaten automatisch zu parsen und zu dekodieren, bzw. den HTTP-Anfrage-Header einfach auszulesen. Der Entwickler muß sich also nicht mehr darum kümmern, in welcher Form die Anfrage übertragen wird.

Mächtigkeit: Servlets bieten dem Entwickler mehr Möglichkeiten als die CGI-Programmierung. So können Servlets unter anderem direkt mit dem Web-Server kommunizieren, was bei CGI nicht so ohne weiteres realisierbar war.

Portierbarkeit: Servlets werden, wie bereits erwähnt, in Java geschrieben. Es ist dadurch möglich die Servlets plattformunabhängig zu programmieren. Deshalb können Servlets auf verschiedenen Servern laufen, ohne dabei das Servlet ändern zu müssen.

Sicherheit: CGI-Programmierer hatten das Problem, Sonderzeichen bei der Eingabe speziell zu behandeln, was bei den Servlets dadurch gelöst wird, daß die Umwandeln der Sonderzeichen automatisch geschieht.

Preis: Im Gegensatz zu CGI gibt es viele kostenlose oder sehr billige Webserver. Für das Autorensystem wurde der Apache Tomcat-Server benutzt, da dieser frei zugänglich ist. Probleme bei der Installation und Konfiguration des Apache Tomcat-Server wurden mittlerweile in den neuesten Versionen gelöst und die Konfiguration gestaltet sich nicht mehr so schwierig wie bei den älteren Versionen.

4 Umsetzung des Entwurfs

Das Projekt zur Erstellung eines Autorensystem ist in verschiedene Bereiche eingeteilt. Den Kern bietet das Rahmenprogramm, daß die Projektübersicht und die Komponentenübersicht umfaßt. Im Rahmen dieser Studienarbeit wird die Komponentenübersicht als Grundlage der Weiterentwicklung implementiert. Sie umfaßt die Klassen *Init*, *KomponentenUebersicht*, *KomponentenUebersichtIO*, *InformationsKomponente* und *KomponentenDaten* jeweils mit allen inneren Klassen.

Einen weiterer Bereich stellen die unterstützenden Klassen dar. Sie liefern Hilfsfunktionen sowie wiederverwendbare Systemkomponenten. Hierzu zählen die Klassen *Kommando*, *ExtensionFileFilter*, *Hilfsfunktionen* sowie *FrageAssistent* und deren innere Klassen.

Der erweiterbare Bereich des Autorensystems bilden die Komponenten. Sie leiten sich alle von der Klasse *InformationsKomponente* ab, die als Basisklasse im Rahmenprogramm zur Verfügung gestellt wird. Exemplarisch werden die Klassen *TextKomponente*, *NumerischeFrage* und *BewertungsKomponente* mit den jeweiligen inneren Klassen implementiert.

Die Darstellung der Lerneinheiten im Browser bildet einen weiteren Bereich. Hierbei wird eine Umwandlung der Lerneinheit im XML-Format in eine HTML-Seite mittels XSLT vollzogen. Dazu sind Vorlagen, sogenannte Stylesheets oder XSL-Dateien notwendig.

Den letzten Bereich bildet die Antwortanalyse der Eingaben von Lernenden. Hierbei werden Servlets eingesetzt, konkret die Klassen *Antwortanalyse*, *Auswertung*, *NumerischeFrageAuswertung* und *Gesamtbewertung*.

Abbildung 5 zeigt eine Übersicht über die Klassenstruktur. Die einzelnen Klassen sowie verwendete Strukturen im Autorensystem werden in den folgenden Kapiteln vorgestellt und erläutert. Für Detailinformationen sei auf den Quelltext des Autorensystems verwiesen, der sich auf der beiliegenden CD-Rom befindet.

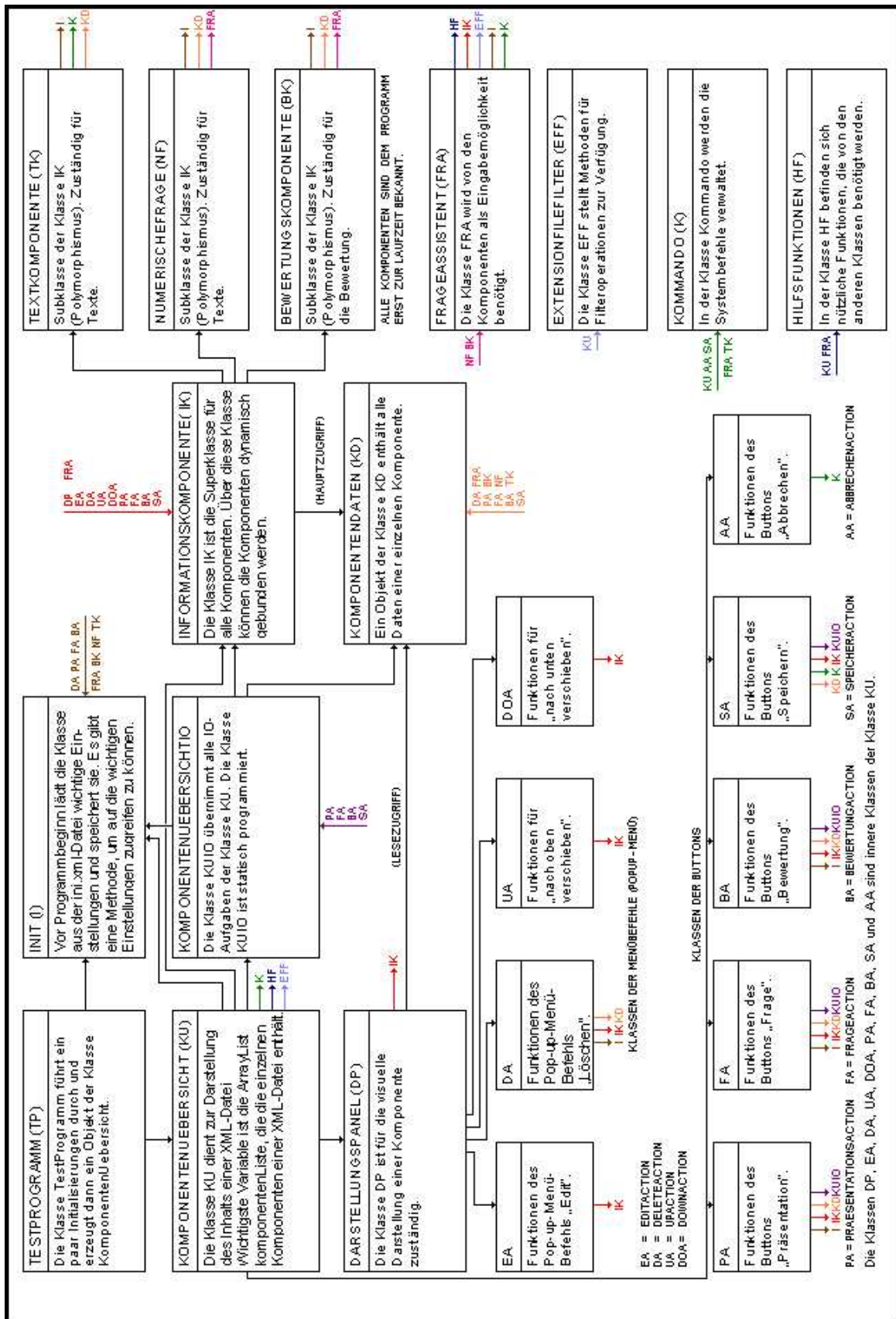


Abbildung 5 - Übersicht über die Klassenstruktur

4.1 Realisierung des Autorensystems

4.1.1 Die Klassen des Rahmenprogramms

4.1.1.1 Die Klasse *Init*

In der Klasse *Init* werden alle Programmeinstellungen gespeichert. Sie wurde als statische Klasse programmiert, d.h. die Methoden, Klassenfelder und Konstanten werden nur einmal für die Klasse angelegt. Daher müssen für die Klasse *Init* keine Objekte angelegt werden, da statische Methoden, Klassenfelder und Konstanten nicht zu einem Objekt gehören, sondern zu der Klasse selbst.

Die Entscheidung für eine statische Klasse erfolgte deshalb, weil für das Programm nur eine Kopie der in der Klasse *Init* stehenden Informationen benötigt wird und nicht für jedes angelegte Objekt eine eigene. Der Aufruf der Methoden und das Ansprechen der Klassenfelder und Konstanten erfolgt deshalb mit dem vorangestellten Bezeichner "*Init*." (beispielsweise erfolgt der Aufruf der Methode *map* aus dem Programm heraus mit der Anweisung *Init.map*).

Die Programmeinstellungen werden am Anfang des Programms mit der Methode *iniLaden* aus der Konfigurationsdatei geladen und in ein dreidimensionales Feld (bildlich ein Würfel) gespeichert. Da es sich bei der Konfigurationsdatei um eine XML-Datei handelt (*ini.xml*) wird zum Auslesen der Datei DOM verwendet. Ohne diese Programmeinstellungen wäre das Programm weitestgehend funktionsunfähig.

Die zweite wichtige Methode der Klasse *Init* ist die Methode *map*. Sie stellt eine Komfortmethode dar, die dem restlichen Programm die angefragten Programmeinstellungen zurückgibt. Diese werden dabei aus dem dreidimensionalen Array ausgelesen.

Die Klasse bietet weiterhin wichtige Klassenfelder und Konstanten an, die an vielen

Stellen des Programms genutzt werden. Durch die zentrale Speicherung der Klassenfelder und Konstanten wird eine einfache Anpassung des Programms an neue Gegebenheiten gewährleistet. Dies ist zum Beispiel nötig, wenn der Benutzer seine Dateien in anderen Verzeichnissen als den vorgegebenen abspeichern will, oder wenn neue Komponenten durch den Entwickler hinzukommen.

Die Möglichkeiten der Anpassung des Programms für den Benutzer auf der einen Seite und dem Entwickler auf der anderen Seite werden in einer späteren Version in der Klasse *ProjektUebersicht* angeboten.

Unter den zahlreichen Konstanten, die vor allem für den komfortableren Aufruf von *map* gedacht sind, ist noch die Konstante *separator* zu erwähnen. Da das Separator-Zeichen, daß Verzeichnisse und Dateien trennt, bei jedem Betriebssystem unterschiedlich realisiert wird (z.B. "\" bei Windows), wird dieses von der Java-Konstanten *File.separator* der Klasse *File* bezogen. Die Klasse *File* ist in der Lage, das jeweils für das Betriebssystem benötigte Separator-Zeichen zu ermitteln.

Da die Informationen der Klasse *Init* von zentraler Bedeutung sind, wird die Klasse an nahezu allen Stellen des Programms benutzt.

4.1.1.2 Die Klasse KomponentenUebersicht

Die Klasse *KomponentenUebersicht* dient der Verwaltung und Darstellung der Komponenten einer einzelnen Lerneinheit.

Sie erweitert die Klasse *JFrame* der Swing-Bibliothek von Java. *JFrame* ist ein Container und dient als Fenster für die Komponentenübersicht. Er enthält die für die Anwendung relevanten GUI-Komponenten, die im Konstruktor der Klasse erzeugt werden.

Das Fenster wird dabei zweigeteilt: Auf der rechten Seite (Bereich "Funktionen") werden die Schaltflächen "Präsentation", "Frage", "Bewertung" sowie "Speichern" und "Abbrechen" angelegt. Um diese funktionsfähig zu machen, benötigt die Klasse *KomponentenUebersicht* die inneren Klassen *PraesentationsAction*, *FrageAction*,

BewertungAction, *SpeicherAction* und *AbbrechenAction*. Dabei fügen die ersten drei Schaltflächen je eine Präsentations-, Frage- oder Bewertungskomponente ein. Die beiden letzten Button sind dafür zuständig, ob die vom Benutzer bearbeitete Datei gespeichert oder ob das Ergebnis verworfen werden soll.

Auf der linken Seite (Bereich "Komponenten") werden die einzelnen Komponenten der geladenen Datei mit Hilfe der Methode *panelAnzeigen* zur Anzeige gebracht.

Neben den grafischen Elementen werden im Konstruktor die wichtigsten Klassenfelder zur Verwaltung der Komponenten bzw. zur Verwaltung der Benutzereingaben erzeugt. Hierunter ist das Klassenfeld *komponentenListe* des Typs *ArrayList* die wichtigste. In ihr werden die Komponenten der geladenen Datei gespeichert und jede Änderungsoperation auf den Komponenten - sei es ein Hinzufügen einer neuen Komponente oder das Verschieben einer Komponente - greift auf dieses Klassenfeld zu. Um diesen Zugriff zu ermöglichen, wurden die entsprechenden Klassen, die die *komponentenListe* benötigen, als innere Klassen der Klasse *KomponentenUebersicht* konzipiert. Das eigentliche Laden der Lerneinheit (Datei im XML-Format) und damit der Komponenten übernimmt die Klasse *KomponentenUebersichtIO*.

Die Aufgabe der Methode *panelAnzeigen* ist es, für die visuelle Darstellung der Komponenten zu sorgen, die in der *komponentenListe* gespeichert sind. Hierzu bedient sich die Methode der inneren Klasse *DarstellungsPanel*, die aus den Angaben der Komponenten die visuelle Darstellung erzeugt und ein *JPanel* zurückgibt, welches die Methode *panelAnzeigen* in einen *Container* auf der linken Seite des Rahmens einbaut.

Die letzte Methode der Klasse ist die Methode *dateiAuswahl*. Diese Methode stellt einen Öffnen-Dialog für das Laden neuer Komponenten bereit (es kann sich dabei um Präsentations-, Frage- oder Bewertungskomponenten handeln). Um den richtigen Dateifilter zu erzeugen bedient sich die Methode der Klasse *ExtensionFileFilter*, die den gewünschten Filter zusammenstellt. Der Rückgabewert der Methode ist der vom Benutzer ausgewählte Pfad. Wird der Dialog abgebrochen wird *null* zurückgegeben. Wie bei jeder Pfadoperation, werden auch hier die Konstanten der Klasse *Init* benutzt.

4.1.1.2.1 Die Klasse *DarstellungsPanel*

Die Klasse *DarstellungsPanel* erzeugt die visuelle Darstellung einer Komponente und wird von der Methode *panelAnzeigen* der umschließenden Klasse *KomponentenUebersicht* benötigt.

Der Konstruktor bekommt hierfür folgende Parameter übergeben: Die *Informationskomponente*, von der die visuelle Darstellung erstellt werden soll, und zwei boolesche Klassenfelder, die für das Verschiebe-Menü wichtig sind. Damit die Klassen des Popup-Menüs auf der Komponente arbeiten können, wird die übergebene *Informationskomponente* im Klassenfeld *infKomp* gespeichert.

Aus der Klasse *KomponentenDaten* werden die Informationen über jede Komponente ausgelesen und anschließend ein *grundPanel* erzeugt (siehe Abbildung 6). Wichtig ist hierbei, daß in das *beschreibungPanel* das von der übergebenen Komponente bereitgestellte *JPanel* eingefügt wird.

Gleichzeitig wird ein Popup-Menü erzeugt, welches erscheint, wenn der Benutzer mit der rechten Maus auf die Komponente klickt. Derzeit sind die Menübefehle Editieren, Löschen und Verschieben realisiert. Die Funktionen der Menübefehle werden durch die Klassen *EditAction*, *DeleteAction*, *UpAction* und *DownAction* zur Verfügung gestellt. Es handelt sich wie bei der Klasse *DarstellungsPanel* selbst um innere Klassen der Klasse *KomponentenUebersicht*.

Das Verschiebemenü wird entsprechend der übergebenen booleschen Klassenfelder *nachoben* und *nachunten* angepaßt. So darf unter anderem die Bewertungskomponente einer Lerneinheit kein Verschiebemenü besitzen, da diese Komponente immer am Ende dieser stehen muß.

Zur Aktivierung des Popup-Menüs wird die Methode *setKontextMenu* mit dem *grundPanel* als Übergabeparameter aufgerufen. Die Methode *setKontextMenu* stellt eine Lösung für folgendes Problem dar: Damit das vom Konstruktor erzeugte Popup-Menü angezeigt wird, wenn der Benutzer ein grafisches Element mit der rechten Maustaste anklickt, muß dieses mit einem *MouseListener* verbunden werden. Das Problem liegt nun darin, daß mit der Verbindung von *grundPanel* und *MouseListener* nicht automatisch alle darin enthaltenen Elemente auch mit verbunden werden.

Damit aber beim Mausklick auf das *grundPanel* an jeder Stelle das Popup-Menü angezeigt wird, muß man den *MouseListener* mit jedem grafischen Element einzeln verbinden. Das Programm weiß allerdings erst zur Laufzeit, welche grafischen Elementen sich im *beschreibungPanel* befinden. Daher mußte die Methode *setKontextMenu* rekursiv programmiert werden. Zuerst ermittelt die Methode die Komponenten des übergebenen *Container grundPanel* und verbindet diese mit dem *MouseListener*. Handelt es sich bei einer der ermittelten Komponenten wiederum um einen *Container*, wird die Methode *setKontextMenu* mit diesem erneut aufgerufen. Somit wird letztlich jedes graphische Element mit dem *MouseListener* verbunden und das Popup-Menü funktioniert wie gewünscht.

Die Methode *getDarstellungsPanel* wird lediglich benötigt, damit die Methode *panelAnzeigen* auf das *grundPanel* zugreifen kann.

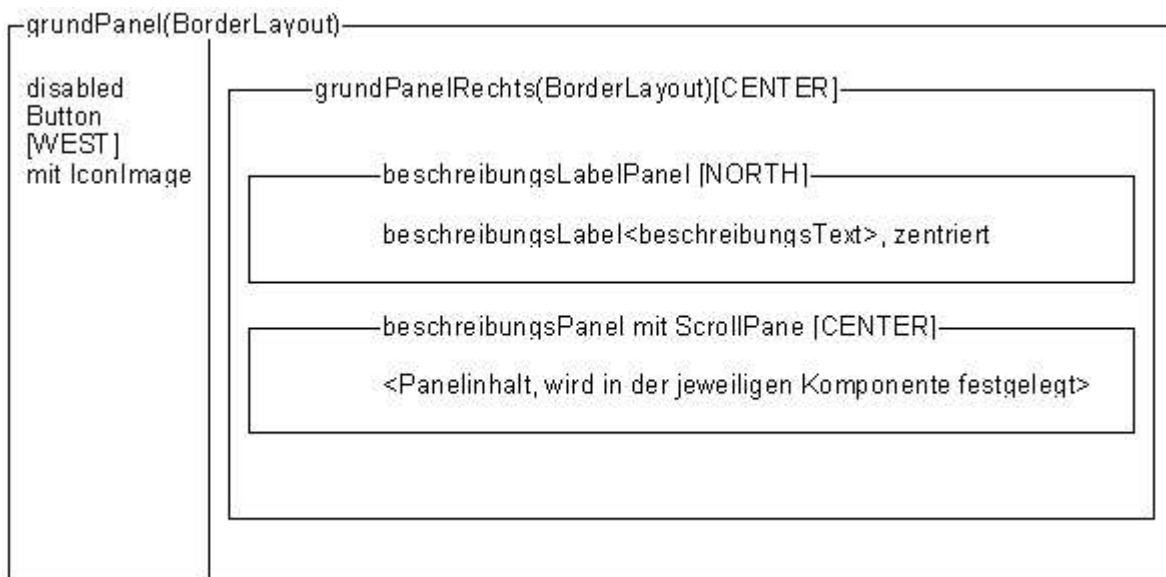


Abbildung 6 - Aufbau des Darstellungspanels

4.1.1.2.2 Die Aktionsklassen der Schaltflächen im Funktionsbereich

Die Swing-Bibliothek von Java stellt einen sehr nützlicher Mechanismus bereit, um Befehle zu kapseln und sie mit mehreren Ereignisquellen zu verbinden: die Action-Schnittstelle. Um nicht jede Methode der Action-Schnittstelle programmieren zu

müssen, wird dem Entwickler die Komfortklasse *AbstractAction* angeboten. Diese Komfortklasse wird durch die entsprechenden Aktionsklassen (*PraesentationsAction*, *FrageAction*, *BewertungAction*, *SpeicherAction* und *AbbrechenAction*) erweitert, wobei die Methode *actionPerformed* als einzige Methode überschrieben wird. Sie wird aufgerufen, wenn die jeweilige Schaltfläche aktiviert wird.

Der Vorteil, die Funktionalität einer Schaltfläche über diese Weise zu programmieren, liegt darin, daß in einer späteren Version des Programms andere Elemente der Benutzeroberfläche, wie z.B. Menüpunkte oder andere Schaltflächen mit dieser Klasse verbunden werden können, wenn sie die gleiche Funktionalität aufweisen sollten. Der Entwickler muß also keinen redundanten Code schreiben, sondern kann die betreffende Aktionsklasse wiederverwenden.

Da die Aktionsklassen auf Klassenfelder der umschließenden Klasse *KomponentenUebersicht* zugreifen müssen (insbesondere auf die *komponentenListe*), müssen sie innere Klasse sein.

Im Konstruktor der Aktionsklasse - z.B. in der Klasse *PraesentationsAction* - wird der Konstruktor der Superklasse (*AbstractAction*) aufgerufen und die Beschriftung der Schaltfläche übergeben. Der Konstruktor der Superklasse bewirkt, daß die Beschriftung auf der Schaltfläche erscheint.

In den folgenden Abschnitte werden die Aktionsklassen näher erläutert.

Die Klasse *PraesentationsAction*

Die Klasse *PraesentationsAction* ist für das Laden und später auch für das Erzeugen einer Präsentationskomponente verantwortlich.

Die überschriebene Methode *actionPerformed* bewirkt folgendes: Zuerst wird überprüft, ob eine der angebotenen Präsentationskomponenten einen internen Editor besitzt. Diese Information bekommt die Methode aus der Klasse *Init*. Wenn ein interner Editor vorhanden ist, wird dem Benutzer die Möglichkeit gegeben, über einen Dialog auszuwählen, ob er eine neue Komponente erstellen möchte oder eine bereits vorhandene Komponente laden möchte. Bei Auswahl von "Neu" wird die

Methode *neu* der vorher (mit Hilfe der Klasse *KomponentenUebersichtIO*) erstellten Komponente aufgerufen.

Hat keine der Präsentationskomponenten einen internen Editor (was derzeit der Fall ist) besteht nur die Möglichkeit des Ladens einer bereits bestehenden Komponente. Hierzu wird die Methode *dateiAuswahl* der umschließenden Klasse aufgerufen. Diese bietet einen Öffnen-Dialog an, über die die Komponente geladen werden kann. Um den Filter des Dialogs richtig zu setzen, wird der Methode *dateiAuswahl* die Art der Komponente (hier: "praesentation") übergeben. Nachdem die Datei ausgewählt wurde, wird die Komponente für das Programm erzeugt. Hierzu werden einige Informationen aus der Klasse *Init* und der Methode *erzeugeKomponente* der Klasse *KomponentenUebersichtIO* verwendet und die Komponente an die richtige Stelle in der *komponentenListe* eingefügt.

Als letztes wird die Methode *panelAnzeigen* der umschließenden Klasse aufgerufen, was bewirkt, daß die eventuell veränderte *komponentenListe* neu angezeigt wird.

Die Klasse FrageAction

Die Klasse *FrageAction* ist für das Laden oder Erzeugen von Fragekomponenten zuständig. Da für mindestens eine der Fragen ein interner Editor existiert (siehe auch Klasse *NumerischeFrage* und Klasse *FrageAssistent*), ist es möglich direkt im Programm selbst eine neue Komponente zu erzeugen.

Der weitere Ablauf ähnelt dem Ablauf der Methode *actionPerformed* von der bereits beschriebenen Klasse *PraesentationsAction*. Die Methode hier unterscheidet sich lediglich in einem Punkt: Da es für die Speicherung der Lerneinheit wichtig ist, die Anzahl der Fragekomponenten zu kennen, wird eine Zähl-Variable hochgezählt, die diese Information beinhaltet. Außerdem sorgt die Methode *actionPerformed* dafür, daß die Schaltfläche "Bewertung" aktiviert wird, falls dies nicht schon der Fall war, da eine Lerneinheit mit Fragen immer eine Bewertungskomponente beinhalten muß.

Die Klasse **BewertungsAction**

Diese Klasse ermöglicht das Einfügen einer Bewertungskomponente in die Lerneinheit. Dabei kann der Benutzer wählen, ob er eine Komponente neu erstellen oder laden will.

Das Erstellen einer neuen oder das Laden einer bereits vorhandenen Komponente, läuft analog ab wie bei den beiden vorherigen Klassen. Da es pro Lerneinheit nur eine Bewertungskomponente geben darf, wird die Schaltfläche "Bewertung" deaktiviert, sobald eine Bewertungskomponente eingefügt wurde.

Gleichzeitig gilt, daß sie immer an letzter Stelle in der *komponentenListe* stehen muß. Dies wird durch das Setzen eines booleschen Klassenfelds gewährleistet. Ist es wahr, werden neue Komponenten (entweder Präsentations- oder Fragekomponenten) nicht an letzter, sondern an vorletzter Stelle eingefügt. Anschließend wird die Methode *panelAnzeigen* der Klasse *KomponentenUebersicht* aufgerufen, wenn sich die *komponentenListe* geändert hat.

Die Klasse **SpeichernAction**

Die Klasse *SpeichernAction* realisiert die Funktionen der Schaltfläche "Speichern". Die Methode *actionPerformed* speichert die *komponentenListe* als Datei ab. Hierzu wird zuerst geprüft, ob sich die Lerneinheit in einem konsistenten Zustand befindet. Es gibt genau zwei konsistente Zustände: Die Lerneinheit enthält keine Fragekomponenten und keine Bewertungskomponente. In diesem Fall handelt es sich also um eine reine Präsentation. Ansonsten enthält sie eine oder mehrere Fragekomponenten und genau eine Bewertungskomponente.

Sollte sich die Lerneinheit in keinem der beiden Zustände befinden, wird eine entsprechende Fehlermeldung ausgegeben, da entweder eine Fragekomponente fehlt, falls eine Bewertungskomponente vorhanden ist, oder das Umgekehrte ist der Fall. Die Konsistenz wird über die Klassenfelder *fragezaehler* bzw. *bewertungda*, die in den Klassen *PraesentationsAction*, *FrageAction* und *DeleteAction* gesetzt werden, festgestellt.

Wenn eine Bewertungskomponente vorhanden ist, wird als nächstes die Gesamtanzahl der Punkte der Fragekomponenten errechnet und an die

Bewertungskomponente übergeben. Im anderen konsistenten Zustand entfällt dieser Schritt.

Nun erfolgt die eigentliche Speicherung mit Hilfe der Klasse *KomponentenUebersichtIO* über deren Methode *speichern*. Sollte die Größe des Klassenfelds *dateiLoeschenListe* nicht *null* sein, werden nun die in der Liste vorhandenen Komponenten physikalisch gelöscht.

In der Liste standen die Komponenten, die vom Autor während der Bearbeitung der Lerneinheit gelöscht wurden, aber deren zugehörigen Dateien noch physikalisch vorhanden waren. Diese Vorgehensweise ist notwendig, da der Autor die Lerneinheit auch nicht speichern sondern über "Abbrechen" verlassen kann.

Da der Autor die Lerneinheit hier aber im neuen Zustand abspeichert, müssen jetzt die Dateien endgültig gelöscht werden. Anschließend wird das Programm beendet. In einer späteren Version wird an dieser Stelle zur Projektübersicht (Klasse *ProjektUebersicht*) zurückgekehrt.

Die Klasse AbbrechenAction

Die Klasse *AbbrechenAction* ermöglicht es dem Autor, die Projektübersicht verlassen zu können, ohne dabei die Lerneinheit abspeichern zu müssen. Es muß natürlich gewährleistet sein, daß die Lerneinheit beim Abbrechen unverändert bleibt, was dadurch erreicht wird das die zugehörige XML-Datei nicht verändert wird. Hat der Autor zuvor während der Bearbeitung neue Komponenten eingefügt, so müssen nun die physikalischen Kopien der neuen Komponenten gelöscht werden. Dies geschieht mit Hilfe des Klassenfelds *dateiNeuListe*, das in den Klassen *PraesentationsAction*, *FrageAction* und *BewertungAction* gesetzt wurde, und der Klasse *Kommando*, die die physikalische Löschung der Dateien vornimmt.

Nach Leeren der Listen *dateiNeuListe* und *dateiLoeschenListe*, wird das Programm beendet. In einer späteren Version wird an dieser Stelle, ähnlich zu der Klasse *SpeichernAction*, das Programm nicht beendet, sondern es wird in die Projektübersicht zurückgekehrt.

4.1.1.2.3 Die Aktionsklassen des Popup-Menüs

Die folgenden vier Klassen realisieren die Funktionen des Popup-Menüs, welches in der Klasse *DarstellungsPanel* erzeugt wurde. Es werden hierbei dieselben Mechanismus benutzt wie bei den Aktionsklassen für die Schaltflächen. Im Konstruktor der Klasse wird zusätzlich die übergebene Informationskomponente in ein internes Klassenfeld gespeichert, da diese Komponente von der Methode *actionPerformed* benötigt wird.

Die Klasse *EditAction*

Durch Auswahl des Menübefehls "Editieren" wird die Methode *actionPerformed* der Klasse *EditAction* aufgerufen. In dieser Methode wird die jeweils zur Komponente gehörenden Methode *edit* aufgerufen. Liefert diese *true* zurück, wird die Methode *visuellerUpdate* der Komponente aufgerufen, die die interne visuelle Darstellung der Komponente aktualisiert. Da die Komponente nach dem Editieren geändert wurde, muß danach noch die Methode *panelAnzeigen* aufgerufen werden, die den Komponentenbereich des Fensters aktualisiert.

Die Klasse *DeleteAction*

Die Klasse *DeleteAction* realisiert den Menübefehl „Löschen“ des Popup-Menüs. Die Methode *actionPerformed* löscht die aktuelle Informationskomponente aus der *komponentenListe*. Das physikalische Löschen der Datei erfolgt nicht hier, sondern in der Klasse *SpeichernAction* (siehe auch dort). Die nötigen Informationen, daß die Komponente physikalisch gelöscht werden muß, übergibt die Methode *actionPerformed* dem Klassenfeld *dateiLoeschenListe*.

Wenn es sich bei der zu löschenden Komponente um eine Fragekomponente handelt, wird der Fragezähler um eins verringert. Wenn es sich um die Bewertungskomponente handelt, wird das boolesche Klassenfeld *bewertungda* auf *false* gesetzt und je nachdem, ob eine weitere Fragekomponente in der Lerneinheit existiert, wird der Button "Bewertung" aktiviert. Die Änderungen im Komponentenbereich des Fensters werden mit der Methode *panelAnzeigen* aktualisiert.

Die Klasse UpAction

Die Methode *actionPerformed* der Klasse *UpAction* ist für das Verschieben einer Komponente nach oben zuständig. Hierzu vergleicht die Methode alle Komponenten der *komponentenListe* mit der aktuellen Informationskomponente. Nachdem die Methode festgestellt hat, an welcher Stelle sich die Informationskomponente befindet, verschiebt die Methode die Komponente nach oben. Danach wird die Methode *panelAnzeigen* aufgerufen.

Die Klasse DownAction

Die Methode *actionPerformed* der Klasse *DownAction* ist analog zur Klasse *UpAction* für das Verschieben nach unten verantwortlich. Hierzu vergleicht die Methode alle Komponenten der *komponentenListe* mit der aktuellen Informationskomponente. Nachdem die Methode festgestellt hat, an welcher Stelle sich die Informationskomponente befindet, verschiebt die Methode die Komponente nach unten. Um ein weiteres Verschieben nach unten zu verhindern, verläßt das Programm die Schleife sofort und ruft die Methode *panelAnzeigen* auf.

4.1.1.3 Die Klasse KomponentenUebersichtIO

Auch diese Klasse wurde aus den gleichen Gründen wie die Klasse *Init* statisch programmiert. Die Klasse enthält drei Methoden: *speichern*, *laden* und *erzeugeKomponente*.

Die Methode *speichern*, speichert die *komponentenListe* der Klasse *KomponentenUebersicht* in einer Datei im XML-Format ab. Dazu wird der Methode die *komponentenListe* übergeben, sowie der Projektname und der Seitenname (Dateiname). Die Methode *speichern* wird von der inneren Klasse *SpeichernAction* der Klasse *KomponentenUebersicht* aufgerufen.

Um die einzelnen Komponenten in einer XML-Datei abspeichern zu können, wird

DOM benötigt. Nachdem die benötigten grundlegende Elemente erzeugt wurden, wird durch die *komponentenListe* gegangen und Schritt für Schritt die Daten jeder einzelnen Komponente ausgelesen.

Da die Methode nicht wissen kann, wie die einzelnen Komponenten aufgebaut sind, da diese dem Programm erst zur Laufzeit bekannt sind, ist jede Komponente für das korrekte Speichern selbst zuständig. Die Methode ruft deshalb selbst die Methode *speichern* der jeweiligen Komponenten auf, die als Ergebnis einen *Node* liefern, der direkt in die XML-Datei eingefügt wird.

Nachdem die Datei komplett aufgebaut wurde, wird sie schließlich gespeichert.

Das Pendant zur Methode *speichern* ist die Methode *laden*. Die Methode lädt eine XML-Datei und gibt der aufrufenden Stelle die daraus erzeugte *komponentenListe* zurück. Bei der aufrufenden Stelle handelt es sich hierbei um den Konstruktor der Klasse *KomponentenUebersicht*. Zum korrekten Laden benötigt die Methode den Projektnamen und den Seitennamen. Mit Hilfe von DOM wird die gewünschte Datei ausgelesen und es werden die einzelnen Komponenten mittels der Methode *erzeugeKomponente* erzeugt.

Für das spezielle Laden der einzelnen Komponenten ist aus dem gleichen Grund, wie beim Speichern, die Komponente selbst verantwortlich.

Schließlich gibt es noch die Methode *erzeugeKomponente*. Diese Methode ist von zentraler Bedeutung für das Programm. Sie nutzt einen sehr leistungsfähigen Mechanismus von Java aus, den sogenannten Reflection-Mechanismus.

Mit Hilfe dieses Mechanismus kann man zur Laufzeit eine Klasse einbinden, die dem Programm bis dahin nicht bekannt war. Durch diese Fähigkeit ist es erst möglich, ein Programm zu schreiben, welches erlaubt, Komponenten nachträglich einzufügen, ohne dabei den Quelltext des Programms ändern zu müssen.

Dies wird nun ausgenutzt, um die einzelnen Komponenten (Präsentations-, Frage- oder Bewertungskomponenten) dynamisch während der Laufzeit in das Programm einzubinden. Hierzu wird der Methode der Klassenname der einzubindenden Komponente übergeben und dann eine neue Instanz der Klasse erzeugt. Die benötigten Parameter für den Konstruktor wurden zuvor der Methode *erzeugeKomponente* übergeben.

Das Resultat ist ein Objekt der Klasse *Object*, mit dem das Autorensystem aber nicht direkt arbeiten kann. Um das Objekt nun in ein Objekt der jeweils dazugehörenden Klasse umzuwandeln, wird die Eigenschaft der Polymorphie verwendet.

In der Programmiersprache Java sind Objekte polymorph. Dies bedeutet, daß ein Objekt einer Klasse, auf ein Objekt vom Typ der Klasse verweisen kann oder auf ein Objekt einer beliebigen Subklasse dieser Klasse. Da alle Komponenten die Klasse *InformationsKomponente* erweitern, wird das Objekt der Klasse *Object* in ein Objekt des Typs *InformationsKomponente* umgewandelt (sogenanntes Casting).

Das Objekt ist dann vom Typ *InformationsKomponente*, aber verweist trotzdem auf die richtige Subklasse. Erfolgt nun ein Methodenaufruf, so wird nicht die Methode der Klasse *InformationsKomponente* benutzt, sondern - sofern vorhanden - die überschriebene Methode der entsprechenden Komponente. Dieser Vorgang wird dynamisches Binden genannt.

4.1.1.4 Die Klasse *InformationsKomponente*

Die Klasse *InformationsKomponente* dient nur dem Zweck, den Polymorphismus von Java ausnutzen. (siehe auch Klasse *KomponentenUebersichtIO*). Es werden hier alle Methoden bereitgestellt, die von den einzelnen Komponenten benötigt werden. Die Methoden sind zumeist ohne Code bzw. setzen Standardwerte.

Die Methode *neu* wird aufgerufen, wenn eine neue Komponente erzeugt wird. Zum Editieren dient die Methode *edit*.

Die Methoden *laden* und *speichern* realisieren das Laden bzw. Speichern einer Komponente in der Lerneinheit (XML-Datei).

Über *editorSpeichern* wird hingegen die Komponente in einer separaten Datei (im XML-Format) gespeichert, damit sie in weiteren Lerneinheiten wiederverwendet werden kann.

Die interne Aktualisierung der visuellen Darstellung der Komponente wird über die Methode *visuellerUpdate* gemacht. Schließlich kann über *datenVerarbeitung* die Eingabe des Autors beim Erzeugen neuer Komponenten eingelesen und auf Fehler überprüft werden.

Als einziges Klassenfeld besitzt die Klasse *InformationsKomponente* ein Objekt der Klasse *KomponentenDaten*. In diesem Objekt werden die für das Programm benötigten Daten der Komponente gekapselt.

Um auf dieses Klassenfeld zugreifen zu können, gibt es in der Klasse zwei Methoden: *getKomponentenDaten* und *setKomponentenDaten*.

4.1.1.5 Die Klasse KomponentenDaten

Die Klasse *KomponentenDaten* kapselt alle Informationen einer Komponente, die an einer beliebigen Stelle des Programms benötigt werden. Die Klasse besitzt deshalb zwei Arten von Methoden: Setzen und Auslesen der privaten Klassenfelder. Dies sind zum einen die *set*-Methoden, die die Klassenfelder auf die übergebenen Werte setzen, und zum anderen die *get*-Methoden, um die Werte aus den Klassenfeldern auszulesen.

Jede Komponente besitzt ein Objekt der Klasse *KomponentenDaten*. Auf dieses Objekt wird an allen Stellen des Programms zugegriffen, an denen die Komponenten manipuliert werden.

4.1.2 Unterstützende Klassen

4.1.2.1 Die Klasse Kommando

Die Klasse *Kommando* führt Systembefehle aus. Hierzu gehört das Kopieren und Löschen einer Datei.

Das Kopieren von Dateien wird benötigt, damit das Programm auf der Kopie (im Verzeichnis "projektdaten" abgelegt; siehe auch Verzeichnisstruktur) arbeitet und nicht auf dem Original. Löschen wird benötigt, um gelöschte Komponenten auch physikalisch von der Festplatte zu löschen.

Um einen Systembefehl ausführen zu können, stellt der Konstruktor fest, ob dazu eine Shell benötigt wird. Sollte dies der Fall sein, wird die Methode *runShellCommand* aufgerufen, die vor Ausführung des Systembefehls eine Shell startet. Da der Aufruf der Shell von Betriebssystem zu Betriebssystem variiert, wird der Aufruf entsprechend des festgestellten Betriebssystems getätigt. Die Methode *runCommand* schließlich ist für das eigentliche Ausführen des Systembefehls zuständig.

4.1.2.2 Die Klasse *ExtensionFileFilter*

Die Klasse *ExtensionFileFilter* erweitert die Klasse *FileFilter*. Sie erzeugt mit ihren Methoden einen Dateifilter, den beispielsweise der Öffnen-Dialog der Methode *dateiAuswahl* der Klasse *KomponentenUebersicht* benötigt.

Mit der Methode *addExtension* kann eine weitere Dateierweiterung in den Filter aufgenommen werden. Bevor dies geschieht, wird der Erweiterung ein Punkt (".") vorangestellt, wenn dieser nicht vorhanden sein sollte. Durch die Methode *setDescription* kann eine Beschreibung für den Filter gegeben werden, die mittels der Methode *getDescription* ausgelesen wird. Die Methode *accept* sorgt dafür, daß nur die Dateien mit der gewählten Dateierweiterung angezeigt werden.

4.1.2.3 Die Klasse *Hilfsfunktionen*

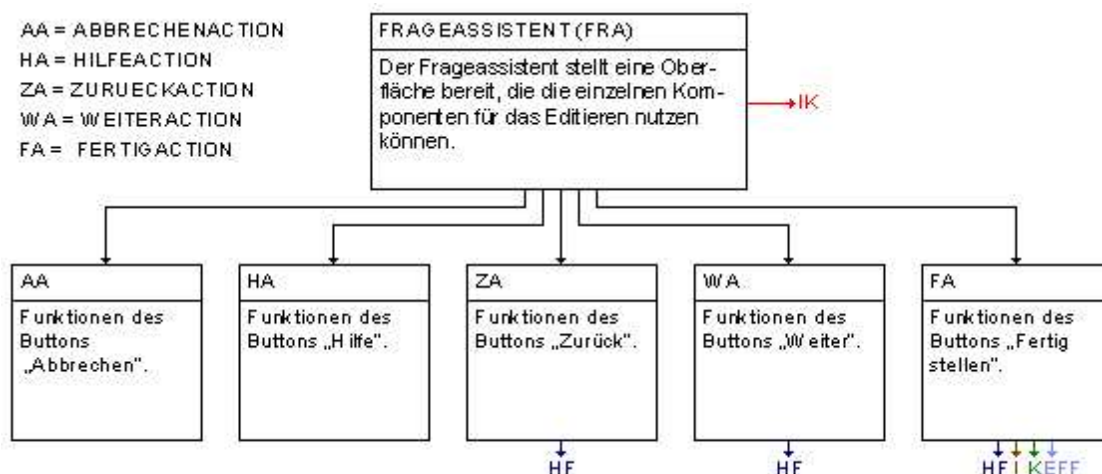
Die Klasse *Hilfsfunktionen* bietet nützliche Methoden an, die von Teilen des Programms genutzt werden und logisch zu keiner anderen Klasse gehören.

So bewirkt die Methode *zentrieren*, daß eine grafische Komponente über einer anderen grafischen Komponente zentriert wird und dabei nicht über den Bildschirmbereich herausragt. Das Setzen der grafischen Komponenten wird nicht von der Methode *zentrieren* selbst durchgeführt, sondern es wird lediglich die richtige Position berechnet und zurückgegeben.

Die Methode *zeitstempel* liefert den aktuellen Zeitstempel zurück, gemessen in Millisekunden seit dem 1. Januar 1970. Der Zeitstempel wird benötigt, um eine in das Verzeichnis "projektdaten" (siehe auch Verzeichnisstruktur) kopierte Datei eindeutig zu bezeichnen. Dabei wird der Zeitstempel dem eigentlichen Namen der Datei vorangestellt. Somit wird gewährleistet, daß es keine zwei Dateien mit dem gleichen Namen gibt, die sich eventuell überschreiben könnten.

Die letzte Methode der Klasse, ist die Methode *prozentAngabe*. Sie berechnet den Anteil des einen Parameters zum anderen Parameter und liefert das Ergebnis in einer Prozentangabe über einen *String* zurück.

4.1.2.4 Die Klasse FrageAssistent



Die Klassen AA, HA, ZA, WA und FA sind innere Klassen der Klassen FRA.

Abbildung 7 - Die Klasse FrageAssistent

Die Klasse *FrageAssistent* ist der interne Editor für die Fragekomponenten und die Bewertungskomponente. Der Aufruf erfolgt demnach entweder in der Methode *neu* bzw. *edit* der einzelnen Komponenten. Die Klasse *FrageAssistent* erweitert dabei die Java-Klasse *JDialog*, mit der man einen modalen Dialog implementieren kann. Ein modaler Dialog erlaubt für die Dauer seiner Anzeige keine Eingaben im Fenster der aufrufenden Klasse. Dies ist notwendig, um die Übergabe der Daten zwischen beiden Fenstern konsistent zu halten. Die Klasse *FrageAssistent* besteht aus dem

Konstruktor und der Methode *getResult*.

Dem Konstruktor werden verschiedene Parameter übergeben: Wichtig hierbei sind die Übergabe der Informationskomponente, für die der Frageassistent aufgerufen wurde, die Übergabe der Eingabeseiten der Komponente und die Angabe, welche der Eingabeseiten zuerst angezeigt werden soll. Mit Hilfe der Übergabeparameter werden verschiedene Klassenfelder gesetzt. Anschließend werden die grafischen Elemente initialisiert, wozu auch die fünf Schaltflächen ("Abbrechen", "Hilfe", "Zurück", "Weiter" und "Fertig stellen") gehören, deren Funktionen durch die inneren Klassen *AbbrechenAction*, *HilfeAction*, *ZurueckAction*, *WeiterAction* und *FertigAction* realisiert werden. Je nach Seitenanzahl erscheint entweder die Schaltfläche "Weiter" oder "Fertig stellen". Im oberen Bereich des Assistenten ist eine Fortschrittsanzeige implementiert.

Die Methode *getResult* liefert als Ergebnis den Wert des privaten Klassenfelds *result* zurück. Sie wird von den einzelnen Komponenten benötigt, um festzustellen, ob der Frageassistent abgebrochen wurde, oder die vorgenommenen Änderungen gespeichert wurden.

Die weitere Funktionalität der Klasse liegt in den fünf inneren Klassen der Schaltflächen, die in den nächsten Abschnitten kurz beschrieben werden. Der prinzipielle Aufbau der Klassen entspricht dem der anderen Aktions-Klassen (siehe Kapitel 4.1.1.2.2).

4.1.2.4.1 Die Klasse *AbbrechenAction*

Die Aufgabe der Klasse *AbbrechenAction* ist es, dafür zu sorgen, daß keine Änderungen in den Eingabefeldern in die jeweilige Komponente übernommen werden. Dafür wird das Klassenfeld *result* auf 0 gesetzt. Dies zeigt der aufrufenden Komponente an, daß der Frageassistent abgebrochen wurde. Gleichzeitig wird der Frageassistent geschlossen, in dem er unsichtbar gemacht wird.

4.1.2.4.2 Die Klasse HilfeAction

Die Klasse *HilfeAction* besitzt derzeit noch keine Funktionalität. In späteren Version wird eine Onlinehilfe zum Frageassistenten implementiert.

4.1.2.4.3 Die Klasse ZurueckAction

Die Klasse *ZurueckAction* realisiert die Navigation in rückwärtiger Richtung. Nach Betätigen der Schaltfläche "Zurück" muß im Frageassistenten die jeweils vorhergehende Eingabeseite angezeigt werden. Sollte sich der Frageassistent nach dem Anzeigen auf der ersten Eingabeseite befinden, wird die Schaltfläche deaktiviert. Daneben wird die Fortschrittsanzeige mit Hilfe der Methode *prozentAngabe* der Klasse *Hilfsfunktionen* aktualisiert.

4.1.2.4.4 Die Klasse WeiterAction

Die Klasse *WeiterAction* realisiert das Gegenteil der Klasse *ZurueckAction*. Sie sorgt dafür, daß die nächste Eingabeseite angezeigt wird. Außerdem aktiviert sie bei Bedarf die Schaltfläche "Zurück". Befindet sich der Autor auf der letzten Eingabeseite wird die Schaltfläche "Weiter" durch "Fertig stellen" ersetzt.

4.1.2.4.5 Die Klasse FertigAction

Die letzte der Aktions-Klassen besitzt eine weitaus komplexere *actionPerformed*-Methode als die vorigen. Prinzipiell läßt die Methode die Eingaben des Autors auf Fehler überprüfen und bringt den Frageassistenten durch Speicherung der neuen Werte zum Abschluß. Hierzu wird die gerade aktuelle Liste der Eingabeseiten, an die Methode *datenVerarbeitung* der Komponente geschickt, die für die Fehlerüberprüfung zuständig ist. Sofern ein Fehler aufgetreten ist, werden die Seiten mit den fehlerhaften Eingaben in einer Liste zurückgegeben, ansonsten *null*. Diese Liste wird

dem Klassenfeld *panelListe* des Assistenten übergeben. Dadurch wird es dem Autor ermöglicht, die Eingabe zu korrigieren. Die entsprechenden Klassenfelder für den Ablauf des Assistenten müssen dafür auf die korrekten Werte gesetzt und die Schaltflächen sowie die übrigen grafischen Elemente an die neuen Gegebenheiten angepaßt werden.

Im Korrektheitsfall wird die Variable *result* auf eins gesetzt um anzuzeigen, daß der Frageassistent durch Speicherung der Änderungen abgeschlossen wurde. Es wird ein Speicherndialog geöffnet, der es ermöglicht, die Komponente in einer Datei im XML-Format abzuspeichern. Hierfür wird unter anderem die Klasse *Init* aufgrund einiger Informationen und die Klasse *ExtensionFileFilter* aufgrund des Dateifilters benötigt. Wurde der Speicherndialog über die Schaltfläche "Speichern" verlassen, wird die Methode *editorSpeichern* der Komponente aufgerufen. Danach wird mit Hilfe der Klasse *Kommando* die Datei in das Verzeichnis "projektdaten" kopiert (siehe auch Abbildung 11 auf Seite 81). Wird stattdessen die Schaltfläche "Abbrechen" betätigt, kehrt der Autor auf die letzte Eingabeseite zurück.

4.1.3 Starten des Systems über die Klasse *TestProgramm*

Als einzige Methode beinhaltet die Klasse *TestProgramm* die Methode *main* und stellt damit den Start des Programms dar. In der *main*-Methode werden zuerst mit Hilfe der Klasse *Init* die Programmeinstellungen geladen. Danach wird ein Objekt der Klasse *KomponentenUebersicht*, mit der anzuzeigenden Lerneinheit (XML-Datei) als Parameter, erzeugt.

Für die Klasse *TestProgramm* wurde absichtlich dieser (während der Entwicklung entstandene Name) beibehalten, um zu verdeutlichen, daß die Implementierung des Autorensystems noch nicht abgeschlossen ist.

Später wird sie durch die Klasse *ProjektUebersicht* vollständig ersetzt, die dann die Autorensteuerung und die Auswahl der zu bearbeitenden Lerneinheit des Projekts realisiert. Außerdem wird die Klasse *ProjektUebersicht* eine Möglichkeit bereitstellen, das Autorensystem durch den Benutzer zu individualisieren, sowie die Einbindung neuer Komponenten durch den Entwickler wesentlich zu vereinfachen.

4.1.4 Realisierung der Komponenten

In diesem Kapitel werden zunächst allgemeine Implementierungshinweise zum Erstellen einer Komponente gegeben. Diese sollten von Entwicklern unbedingt beachtet werden. Im Anschluß daran werden exemplarisch die Implementierungsansätze der Textkomponente, der numerischen Frage sowie der Bewertungskomponente beschrieben.

4.1.4.1 Implementierung neuer Komponenten

Die Klasse der neue Komponente muß immer von der Klasse *InformationsKomponente* abgeleitet werden und über die Konfigurationsdatei *ini.xml* registriert werden.

In der Komponente sollten jeweils zwei Klassenfelder geführt werden. Eine für den konsistenten Zustand, die beim Speichern benutzt wird. Das andere Klassenfeld für vorübergehende inkonsistente Zustände, beispielsweise während der Eingabe der Daten der Komponenten. Nach dem Speichern werden die Werte des inkonsistenten Klassenfelds auf das konsistenten Klassenfeld geschrieben. Beim Abbrechen wird genau umgekehrt verfahren.

Im Konstruktor der neuen Komponente sollten die bekannten Daten der Komponente auf Standardwerte gesetzt werden.

Die Methode *edit* wird so implementiert, daß sie das Editieren der Komponente realisiert. Wird dabei der Frageassistent benutzt, so muß diesem eine *ArrayList* mit *JPanel* übergeben werden. Um zu den anderen Komponenten konform zu bleiben, sollte eine Methode *setAssistentenPanelListe* programmiert werden, der diese *ArrayList* erzeugt. Der Rückgabewert gibt an, ob die Methode *edit* erfolgreich ausgeführt wurde.

Die Methode *visuellerUpdate* sorgt für den visuellen Update der Komponente. Dies wird zum Beispiel nach dem Editieren notwendig.

Sollte der Frageassistent benutzt werden, muß die Methode *datenVerarbeitung* implementiert werden, da diese nach Betätigen der Schaltfläche "Fertigstellen" im Frageassistenten aufgerufen wird. Hier wird die eigentliche Fehlerkontrolle für die Eingaben des Autors beim Erstellen einer neuen Komponente durchgeführt. Der Rückgabewert der Methode *datenVerarbeitung* ist eine *ArrayList*, die die fehlerhaften Panel enthält.

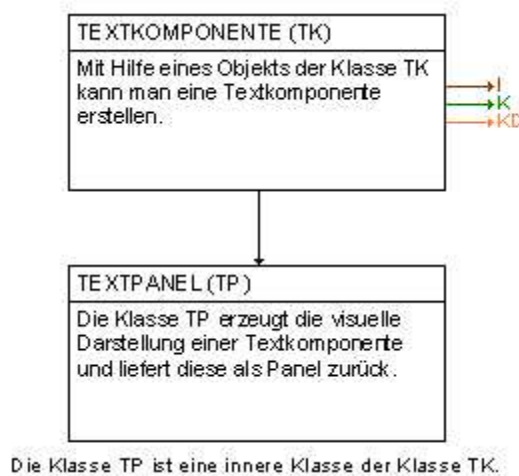
Die Methode *neu* muß ebenfalls implementiert werden, da diese aufgerufen wird, wenn eine Komponente neu erzeugt wird. Auch hier besteht die Möglichkeit den Frageassistenten zu nutzen. Der Rückgabewert gibt an, ob die Methode *neu* erfolgreich ausgeführt wurde.

Die beiden Methoden *laden* und *speichern* realisieren das Laden bzw. Speichern der Lerneinheit. Die Methode *speichern* liefert dabei einen *Node* zurück. Sie sind in jedem Falle zu implementieren.

Die Methode *editorSpeichern* hingegen realisiert die Speichern-Funktion des internen Editors, damit die einzelnen Komponenten wiederverwendet werden können und später wieder geladen werden können. Sofern kein interner Editor vorhanden ist, muß diese Methode nicht implementiert werden.

Abschließend benötigt jede Komponente eine Möglichkeit, ein *JPanel* für die Darstellung zu erzeugen, daß von der Klasse *DarstellungsPanel* benötigt wird. Es wird empfohlen, dies als innere Klasse der Komponente zu realisieren.

4.1.4.2 Die Klasse TextKomponente



Die Klasse TP ist eine innere Klasse der Klasse TK.

Abbildung 8 - Die Klasse TextKomponente

Die Klasse *TextKomponente* ist ein Beispiel einer Präsentationskomponente. Sie erweitert wie alle Komponente die Klasse *InformationsKomponente*, überschreibt jedoch nicht alle Methoden. Der Grund hierfür ist, daß die Textkomponente derzeit keinen internen Editor besitzt, was sich in späteren Versionen ändern wird. Da einige der Methode nur für interne Editoren verwendet werden (wie z.B. *editorSpeichern*), müssen sie nicht überschrieben werden.

Die Textkomponente setzt in ihrem Konstruktor Standardwerte und speichert sie ab. Die Methoden, die überschrieben werden, sind *edit*, *visuellerUpdate*, *speichern* und *laden*.

In *edit* wird der externe Editor aufgerufen und anschließend *true* zurückgeliefert. Nach jedem Editieren einer Textkomponente muß die Methode *visuellerUpdate* aufgerufen werden. Sie sorgt dafür, daß die visuelle Darstellung der Textkomponente - es handelt sich dabei um ein Objekt der Klasse *TextPanel* - aktualisiert wird. Hierzu lädt sie die editierte Datei neu und ruft dann die Methode *setPanel* der Klasse *TextPanel* auf, die das im Konstruktor erzeugte Panel mit neuen Werten belegt. Dieses Panel wird von der Klasse *DarstellungsPanel* benötigt, die dieses in ihr *grundPanel* einbaut (siehe auch Klasse *DarstellungsPanel* und Abbildung 6 auf Seite 56).

Die Methode *speichern* muß ebenfalls überschrieben werden, da sie von der Methode

speichern der Klasse *KomponentenUebersichtIO* aufgerufen wird. Sie liefert einen *Node* zurück, der diese Textkomponente repräsentiert. Hierzu wird einfach ein Textknoten mit dem Text der Komponente erzeugt.

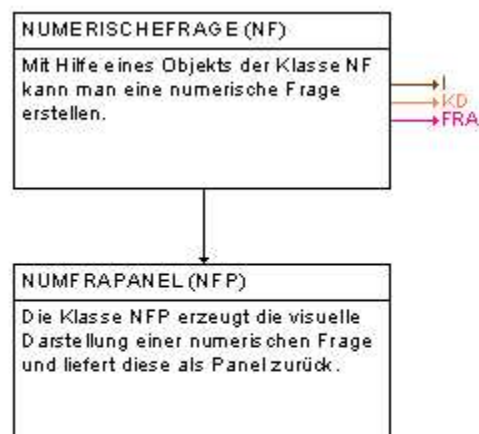
Zuletzt wurde die Methode *laden* implementiert, mit deren Hilfe eine Datei geladen wird und der Inhalt in einem Klassenfeld gespeichert wird.

Die Klasse *TextKomponente* wird in den Fällen benutzt, wenn ein Objekt der Klasse *InformationsKomponente* vom Typ *TextKomponente* ist.

Die Klasse *TextPanel*

Die Klasse *TextPanel* ist für die visuelle Darstellung einer Textkomponente verantwortlich. Hierzu erweitert sie die Java-Klasse *JPanel*. Die Klasse *TextPanel* realisiert ein Panel, welches ein Textfeld enthält, in dem der Inhalt der geladenen Datei steht.

4.1.4.3 Die Klasse *NumerischeFrage*



Die Klasse *NFP* ist eine innere Klasse der Klasse *NF*.

Abbildung 9 - Die Klasse *NumerischeFrage*

Bei der Klasse *NumerischeFrage* handelt es sich um ein Beispiel einer Fragekomponente, nach deren Konzept auch die anderen Fragekomponenten

aufgebaut werden sollten. Da die Implementierung der numerische Frage die Klasse *FrageAssistent* als internen Editor verwendet, müssen im Gegensatz zur Textkomponente mehr Methoden der Klasse *InformationsKomponente* überschrieben werden.

Die Methode *neu* wird beim Erstellen einer neuen Komponente aufgerufen. Es wird ein Objekt der Klasse *FrageAssistent* erzeugt und damit der Dialog "Frageassistent" angezeigt. Schließlich wird abgefragt, ob der Assistent abgebrochen wurde oder ob die Änderungen gespeichert wurden. Über die Methode *getResult* der Klasse *FrageAssistent* wird der Rückgabewert abgefragt. Damit entscheidet das System, ob die Komponente in die *komponentenListe* der Klasse *KomponentenUebersicht* aufgenommen werden muß.

Um eine bereits bestehende Komponente editieren zu können, muß die Methode *edit* überschrieben werden. In dieser Methode wird zuerst festgestellt, ob der interne oder externe Editor verwendet werden soll. In aktuellen Fall wird der interne Editor (Klasse *FrageAssistent*) benutzt. Der interne Editor erwartet eine Liste von Eingabeseiten (als *JPanel* realisiert), in die die Werte der numerischen Frage geändert bzw. eingegeben werden können. Diese Eingabeseiten werden durch die Methode *setAssistentenPanelListe* erzeugt.

Anschließend wird wiederum geprüft, ob die Änderungen gespeichert oder ob abgebrochen wurde. Während bei Änderungen eine visuelle Aktualisierung erfolgen muß, werden im Falle eines Abbrechens die Klassenfelder, die die Informationen der numerischen Frage beinhalten, und diejenigen, die zur Fehlerüberprüfung notwendig sind, auf die richtigen Wert gesetzt.

Die Methode *datenVerarbeitung* wird von der Methode *FertigAction* der Klasse *FrageAssistent* aufgerufen. Die Aufgabe dieser Methode ist es, die Eingaben des Autors bei der Erzeugung bzw. Änderung einer Frage auf Fehler zu überprüfen. Hierzu werden der Methode die Eingabeseiten vom Assistenten übergeben. Hierbei entscheidet der Entwickler der Komponente über die Art und den Umfang der Fehlerüberprüfung. In dieser Klasse werden die Klassenfelder *felderFehler* und *panelNoetig* dazu benutzt, um anzugeben, welche Eingabefelder auf welchen

Eingabeseiten Fehler enthalten. Dies wird dann in der Methode *setAssistentenPanelListe* berücksichtigt. Um eine Fehlerüberprüfung durchführen zu können, muß man den Aufbau der Eingabeseiten genau kennen. Sollte ein Fehler aufgetreten sein, wird eine Fehlerinformationsseite (*FehlerPanel*) erzeugt, welches den fehlerhaften Eingabeseiten vorangesetzt wird. Diese Liste von Eingabeseiten wird dem Frageassistenten erneut übergeben.

Sobald die Eingaben des Autors korrekt sind, werden das Klassenfeld des inkonsistenten Zustands auf das des konsistenten Zustand kopiert und die Fehlerüberprüfung zurückgesetzt.

Nach korrektem Editieren einer Komponente bzw. am Anfang des Programms wird die Methode *visuellerUpdate* aufgerufen, die für die Aktualisierung der Darstellung der numerischen Frage in der Komponentenübersicht sorgt. Zur Darstellung wird ein Objekt der Klasse *NumFragePanel* verwendet.

Die Methoden *speichern* und *laden* müssen auch hier - analog zur Textkomponente - implementiert werden, um das Speichern bzw. Laden der Komponente in der Lerneinheit zu realisieren. Daneben muß noch die Methode *editorSpeichern* überschrieben werden, die die numerische Frage in einer separaten Datei abspeichert, so daß sie wiederverwendet werden kann. In allen drei Methoden wird DOM verwendet.

Beim Laden werden zusätzlich wichtige Klassenfelder gesetzt: Das Klassenfeld des konsistenten sowie des inkonsistenten Zustands werden auf die geladenen Werten gesetzt, da während des Ladens seitens des Autors noch keine Änderung an der Komponente vorgenommen werden kann. Daneben werden die Klassenfelder der Fehlerüberprüfung auf Anfangswerte gesetzt. Zusätzlich wird im Klassenfeld *kompDaten* die Maximalpunktzahl (*bewertungOK*) gesetzt. Dies wird später vom Objekt der Klasse *Bewertungskomponente* benötigt. Abschließend wird die Methode *setAssistentenPanelListe* aufgerufen.

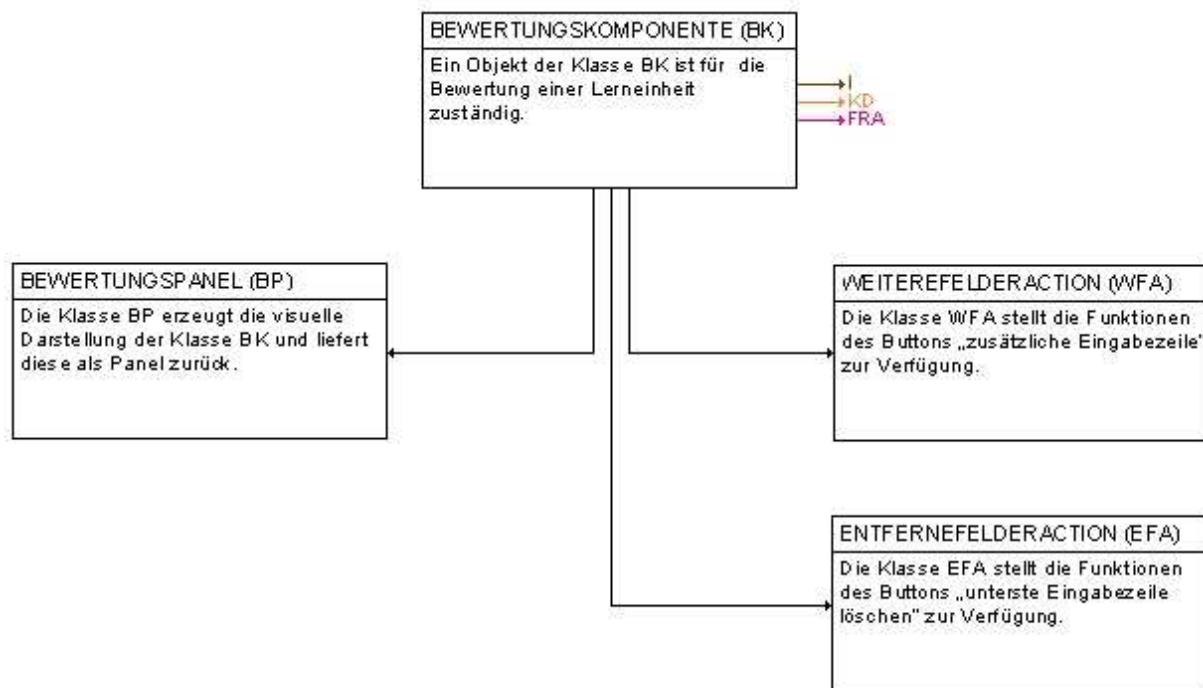
Die Methode *setAssistentenPanelListe* ist eine private interne Methode die keine Methode der Klasse *InformationsKomponente* überschreibt. Die Aufgabe der Methode ist von zentraler Bedeutung für den internen Editor der Klasse *FrageAssistent*. Sie ist

für die Zusammenstellung der Eingabeseiten in Abhängigkeit der Fehlervariablen zuständig. Fehlerhafte Eingabefelder werden dabei besonders gekennzeichnet.

Die Klasse NumFragePanel

Die Klasse *NumFragePanel* ist für die visuelle Darstellung einer numerischen Frage verantwortlich und ist analog zur Klasse *TextPanel* realisiert. Das erzeugte Panel enthält die wichtigsten Informationen der numerischen Frage im Überblick.

4.1.4.4 Die Klasse BewertungsKomponente



Die Klassen BP, WFA und EFA sind innere Klassen der Klasse BK.

Abbildung 10 - Die Klasse BewertungsKomponente

Die Klasse *BewertungsKomponente* realisiert die mögliche Gesamtbewertung am Ende einer Lerneinheit. Es handelt sich dabei um eine besondere Komponente, da diese entweder gar nicht in der Lerneinheit existiert oder nur einmal vorkommen darf. Wenn die Lerneinheit ein BewertungsKomponente enthält, steht diese immer am

Ende. Vom Aufbau ähnelt sie den anderen Komponenten. Sie überschreibt dieselben Methoden wie die Klasse *NumerischeFrage*.

Wie bei allen Komponenten werden auch im Konstruktor der Bewertungskomponente die Komponentendaten auf Standardwerte gesetzt. Hierzu wird die Klasse *Init* wegen einiger Informationen benötigt. Anschließend werden wiederum die Klassenfelder für den konsistenten und den inkonsistenten Zustand gesetzt sowie die Fehlervariablen - die in der Methode *datenVerarbeitung* gesetzt und in der Methode *setAssistentenPanelListe* benötigt werden - mit ihren Ausgangswerten belegt. Abschließend werden die Eingabeseiten für den internen Editor (Klasse *FrageAssistent*) sowie das Panel für die visuelle Darstellung der Komponente erzeugt.

Die Methode *neu* wird aufgerufen, wenn eine Bewertungskomponente neu erstellt wird. Das Ergebnis wird an die innere Klasse *BewertungAction* der Klasse *KomponentenUebersicht* zurückgeliefert, die entsprechend dem Ergebnis fortfährt.

Mit Hilfe der Methode *edit* kann man die Informationen der Bewertungskomponente auf andere Werte setzen. Dabei wird zuerst festgestellt, ob ein interner Editor vorhanden ist, was in dieser Version der Fall ist. Nun werden mit der Methode *setAssistentenPanelListe* die Eingabeseiten für den Frageassistenten erzeugt und der Klasse *FrageAssistent* übergeben. Nachdem der Autor den Frageassistenten beendet hat, wird festgestellt, ob gespeichert oder abgebrochen wurde, indem der Wert der Methode *getResult* geprüft wird. Wenn sich die Komponente verändert hat, erfolgt ein visueller Update. Im Falle des Abbrechens müssen die Werte des Klassenfeldes des inkonsistenten Zustands auf die Werte des Klassenfeldes des konsistenten Zustands gesetzt werden, da die inkonsistenten Klassenfelder aktuell falsche Werte besitzen. Gleichzeitig werden auch die Fehlervariablen auf ihre Ausgangswerte gesetzt.

Nach korrektem Editieren einer Komponente bzw. am Anfang des Programms wird die Methode *visuellerUpdate* aufgerufen, die für die Aktualisierung der Darstellung der numerischen Frage in der Komponentenübersicht sorgt. Zur Darstellung wird ein Objekt der Klasse *BewertungsPanel* verwendet.

Die Methode *datenVerarbeitung* bekommt von der inneren Klasse *FertigAction* der

Klasse *FrageAssistent* die Eingabeseiten mit den vom Autor eingegebenen Werten geliefert. Tritt bei der folgenden Überprüfung ein Fehler auf, werden die entsprechenden Fehlervariablen gesetzt. Das Vorgehen unterscheidet sich hierbei von dem bei der Klasse *NumerischeFrage*, wobei das Grundprinzip erhalten bleibt. Bei der Bewertungskomponente existiert zum Beispiel nur eine Eingabeseite. Auch hier wird vor die eigentlichen Eingabeseite eine Seite mit der Fehlermeldung gesetzt. Die Fehlerüberprüfung wurde in dieser Version einfach gehalten, ist aber jederzeit ausbaubar. Eine mögliche Erweiterung wäre ein Abprüfen der Eingaben für die Prozentbereiche auf mögliche Grenzverletzungen.

Im Fehlerfall wird die erzeugte Fehlerseite sowie die Eingabeseite an die aufrufende Stelle zurückgegeben. Im korrekten Fall wird null zurückgegeben und die Klassenfelder der Komponente werden auf die richtige Werte gesetzt.

Die Methoden *laden*, *speichern* und *editorSpeichern* sind analog zu den entsprechenden Methoden der Klasse *NumerischeFrage* implementiert, so daß an dieser Stelle nicht weiter darauf eingegangen wird.

Abschließend werden noch die zwei privaten Methoden der Klasse *Bewertungskomponente* erläutert. Bei der Methode *erzeugeEingabePanel* handelt es sich um eine Hilfsmethode der Methode *setAssistentenPanelListe*. Da sich der Aufbau der Bewertungskomponente dadurch auszeichnet, daß jede Eingabezeile die gleichen Felder besitzt, bietet es sich an, diese durch eine Hilfsmethode zu erzeugen. Wie bei der Methode *setAssistentenPanelListe* werden auch bei der Methode *erzeugeEingabePanel* die Fehlervariablen berücksichtigt, um ein Eingabefeld mit fehlerhaften Werten hervorzuheben. Die Eingabezeile wird als Panel an die Methode *setAssistentenPanelListe* zurückgegeben.

Für die Methode *setAssistentenPanelListe* gelten dieselben Bedingungen wie bei der gleichnamigen Methode der Klasse *NumerischeFrage*. Die Methode erzeugt die Eingabeseite für den internen Editor und berücksichtigt dabei die Fehlervariablen, die eine Besonderheit besitzt: Dies sind die beiden Schaltflächen "zusätzliche Eingabezeile" und "unterste Eingabezeile löschen", deren Funktionalität durch die folgenden inneren Klassen realisiert wird.

Die Klasse *WeitereFelderAction*

Im Konstruktor der Klasse *WeitereFelderAction* wird der Konstruktor der Superklasse aufgerufen, der für die Beschriftung der Schaltfläche zuständig ist.

Die Methode *actionPerformed* realisiert die Funktion der Schaltfläche. Wird diese betätigt, erscheint eine weitere Eingabezeile auf der Eingabeseite. Dadurch hat der Autor die Möglichkeit, einen zusätzlichen Bewertungsbereich einzugeben.

Die Klasse *EntferneFelderAction*

Die Aufgabe der Methode *actionPerformed* ist es, die unterste Eingabezeile auf der Eingabeseite der Bewertungskomponente zu löschen. Die Methode muß hierfür auch die entsprechenden Klassenfelder löschen. Sollte es nur noch eine Eingabezeile geben, wird die Schaltfläche deaktiviert, da mindestens eine Eingabezeile vorhanden sein muß.

Die Klasse *BewertungsPanel*

Die Klasse *BewertungsPanel* ist für die visuelle Darstellung einer Bewertungskomponente verantwortlich und ist analog zur Klasse *TextPanel* bzw. *NumFragePanel* realisiert. Das erzeugte Panel enthält die wichtigsten Informationen der Bewertungskomponente im Überblick.

4.2 Verzeichnisstruktur des Autorensystems

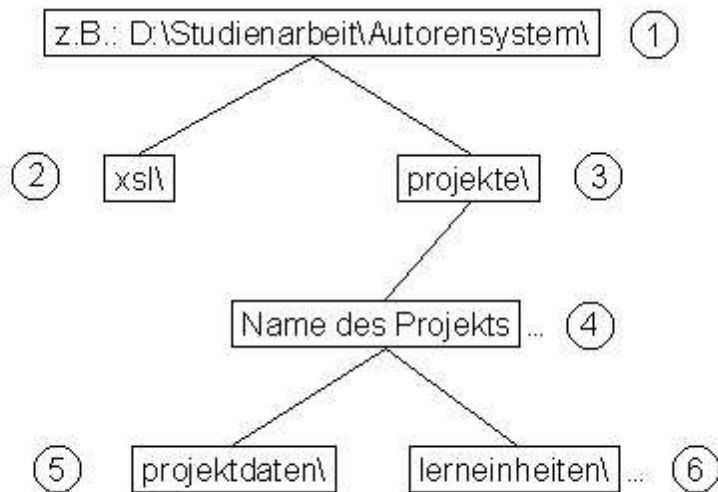


Abbildung 11 - Verzeichnisstruktur

Abbildung 11 zeigt die Verzeichnisstruktur des Autorensystems. Das Basisverzeichnis (1) beinhaltet die einzelnen Klassen und die Ausführungsdatei *TestProgramm*. Es wird in der Klasse *Init* im Klassenfeld *basisVerzeichnis* gespeichert. Das Basisverzeichnis hat das Unterverzeichnis "xsl" (2), in dem die XSL-Dateien zur Aufbereitung der Lerneinheit für den Browser liegen.

Das Projektverzeichnis "projekte" (3) liegt normalerweise unterhalb des Basisverzeichnis, kann sich aber an jeder beliebigen Stelle in der Verzeichnisstruktur des Computers befinden. Das Projektverzeichnis wird im Klassenfeld *projektVerzeichnis* in der Klasse *Init* gespeichert.

Der Name eines Projekts (4) wird vom Benutzer festgelegt. Das Verzeichnis liegt immer unterhalb des Verzeichnisses "projekte". Es enthält immer das Unterverzeichnis "projektdatei" (5), in dem sich die verwendeten Dateien des Projekts befinden, sowie das Verzeichnis "lerneinheiten" (6), in dem die einzelnen Lerneinheiten (XML-Dateien) abgelegt werden.

4.3 Darstellung der Lerneinheit

Die Lerneinheit wird im XML-Format gespeichert. Sie enthält alle Informationen über die Komponenten, die der Autor zuvor eingefügt hatte. Um die Datei in einem Browser anzeigen zu können, muß sie umgewandelt werden, d.h. aus der XML-Datei wird eine HTML-Datei erzeugt, die von jedem gängigen Browser darstellbar ist.

Für die Umwandlung ist XSLT zuständig, das eine XML-Datei mit Hilfe von Vorlagen (sogenannten Stylesheets) in das Zielformat HTML konvertieren kann. Die verwendeten Stylesheets werden im folgenden vorgestellt.

Für jede Komponente muß der Entwickler auch eine Stylesheet-Datei programmieren, die die Informationen der Komponente in entsprechenden HTML-Code umwandelt. Derzeit gibt es drei verschiedene Komponenten: die Textkomponente, die numerische Frage und die Bewertungskomponente. Daher gibt es drei Vorlagen mit demselben Namen.

Diese werden alle in die Hauptvorlage (Komponente.xsl) importiert, auf die in der Lerneinheit referenziert wird.

Die Hauptvorlage ist dafür zuständig, den grundlegenden Aufbau der HTML-Datei zu realisieren, wie beispielsweise `<HEAD>`- oder `<TITLE>`-Tag. Die notwendigen Informationen für die einzelnen Komponenten werden aus den importierten Vorlagen bezogen.

Die Vorlage der Textkomponente ist relativ einfach gehalten und übernimmt nur den Text zwischen den `<Inhalt>`-Tags der Textkomponente und fügt diesen in die HTML-Seite ein.

Die Vorlage der numerischen Frage ist hingegen wesentlich komplexer. Informationen, die für die Servlets notwendig sind, aber vom Lernenden nicht gesehen werden dürfen, werden im HTML-Code versteckt, in dem sie innerhalb eines `<INPUT>`-Tags des Typs `HIDDEN` übergeben werden. Solche Informationen sind beispielsweise die korrekte Antwort, die Punkteverteilung sowie die Rückmeldungen. Daneben müssen Verwaltungsinformationen übergeben werden (z.B. die Angabe, welches Servlet für die Auswertung zuständig ist). Gleichzeitig wird durch die Vorlage ein Eingabefeld erzeugt, in das der Lernende seine Antwort eingeben kann.

Die Vorlage der Bewertungskomponente, erzeugt die Schaltfläche "Abschicken" in der HTML-Seite. Damit kann der Lernende seine Eingaben zur Überprüfung abschicken. Daneben werden auch hier Verwaltungsinformationen sowie die Punkteverteilung der Gesamtbewertung versteckt übergeben.

4.4 Antwortanalyse durch Servlets

Zur Auswertung der Eingaben der Lernenden werden Servlets verwendet, da sie ebenfalls in der Programmiersprache Java implementiert werden und sich daher sehr gut in die bisherigen Mechanismen einfügen. Entsprechend der Vorlagen existiert für jede Komponente auch ein eigenes Servlet, d.h. eine eigene Klasse. Diese werden nun im folgenden vorgestellt.

4.4.1 Die Klasse Antwortanalyse

Die Klasse *Antwortanalyse* bearbeitet jede Anfrage, die von einer HTML-Seite einer Lerneinheit kommt und stellt mit Hilfe anderer Klassen eine HTML-Seite für die Rückmeldung zusammen. Damit sich diese Klasse als ein Servlet auszeichnet, muß sie die Java-Klasse *HttpServlet* erweitern.

Die Anfrage einer HTML-Seite kann über zwei Arten geschehen: Zum einen über eine *Get*-Anfrage, zum anderen über eine *Post*-Anfrage. Der Unterschied liegt darin, daß bei einer *Get*-Anfrage die Anfrageinformationen in der URL-Zeile des Browser steht. Dies ist in diesem Fall nicht sinnvoll, da der Lernende ansonsten Daten lesen kann, die nicht für ihn bestimmt sind. Bei der *Post*-Anfrage hingegen werden die Daten im Kopf der Anfrage (*Header*) versteckt. Diese Anfrageart wird von der durch die Vorlagen generierten HTML-Seite genutzt. Um dennoch beide Anfragearten abzufangen, wurden die Servlets so programmiert, daß beide zuständigen Methoden (*doPost* für Post-Anfragen, *doGet* für Get-Anfragen) überschrieben wurden. Damit das Servlet keinen redundanten Code aufweist, ruft die *doPost*-Methode lediglich die

doGet-Methode auf, in der die eigentliche Funktionalität realisiert ist.

Die Methode *doGet* stellt die HTML-Seite zusammen, die als Antwort an den Lernenden zurückgegeben wird. Hierzu werden zuerst alle Parameternamen der Anfrage abgerufen und gespeichert: Die Methode, die dies realisiert liefert einen Wert des Typs *Enumeration* (Aufzählung) zurück, der leider keine vordefinierte Reihenfolge bei seinen Elementen besitzt. Durch eine spezielle Implementierung wird diese Unzulänglichkeit aber umgangen. Dazu werden die Werte der Parameter *servlet* und *timestamp* bestimmt. Der Parameter *servlet* beinhaltet, welche Servlets der Reihenfolge nach aufgerufen werden. Dabei wird der bereits vorgestellte Reflection-Mechanismus verwendet, um ein einfaches Einbinden zukünftiger Servlets zu ermöglichen. Mit Hilfe des Parameters *timestamp* werden die Übergabeparameter für die einzelnen Klassen bestimmt, die für die Bearbeitung der Komponenten zuständig sind. Da keinerlei Garantie für die Reihenfolge der übergebenen Parameter existiert, werden die Parameternamen zusammen mit ihren Werten in einem zweidimensionalen Feld gespeichert, welches dann an die entsprechenden Servlets übergeben wird.

4.4.2 Die Klasse Auswertung

Die Klasse *Auswertung* besitzt an sich keine Funktionalität, wird aber wegen des Polymorphismus benötigt (vgl. Klasse *InformationsKomponente*). Alle Servlets erweitern die Klasse *Auswertung* und überschreiben die Konstruktoren und Methoden. Schließlich werden in der Klasse *Antwortanalyse* Objekte der Klasse *Auswertung* erzeugt und über diese die entsprechenden Klasse dynamisch gebunden.

4.4.3 Die Klasse *NumerischeFrageAuswertung*

Die Klasse *NumerischeFrageAuswertung* ist dafür zuständig, die Eingaben des Lernenden auf eine numerische Frage auszuwerten und eine Rückgabe zu generieren.

Dem Konstruktor wird das zweidimensionale Feld (mit Parameternamen und deren Werte) übergeben sowie der *PrintWriter*, der angibt, wohin die Ausgabe geschrieben werden soll. Die Aufgabe des Konstruktors ist es, das übergebene Feld auszulesen und in private Variablen zu speichern. Die Implementierung ist dabei so gehalten, daß keine bestimmte Reihenfolge erwartet wird. Anschließend ruft der Konstruktor die Methode *auswertung* auf.

Die Methode *auswertung* realisiert die eigentliche Auswertung der Eingaben des Lernenden und generiert mit Hilfe der übergebenen Parameter die entsprechende Rückmeldung, die auf dem *PrintWriter* ausgegeben wird. Die Antwortanalyse kann an dieser Stelle beliebig komplex implementiert werden. In der jetzigen Version wird jedoch nur geprüft, ob die eingebene Antwort der korrekten Antwort entspricht.

Die Methode *getPunkte* gibt den Wert der Variablen *punkte* zurück. Dieser Wert wird von der Klasse *Gesamtbewertung* zur Durchführung der Bewertung der Lerneinheit benötigt.

4.4.4 Die Klasse *Gesamtbewertung*

Die Klasse *Gesamtbewertung* genießt eine Sonderrolle und wird nicht dynamisch gebunden.

Der Konstruktor bekommt von der Klasse *Antwortanalyse* den *PrintWriter* für die Ausgabe übergeben, sowie das zweidimensionale Feld mit allen notwendigen Informationen und zwei Parametern mit Punktwerten. Dies sind zu einem die erreichte Punktzahl und zum anderen die maximal erreichbare Punktzahl. Der

Konstruktor erledigt ansonsten die selben Aufgaben wie der Konstruktor der Klasse *NumerischeFrageAuswertung*.

In Abhängigkeit der erreichter Punktzahl und der Informationen aus dem übergebenen Feld wird eine entsprechende Bewertung (Rückmeldung) ausgegeben. Diese Aufgabe übernimmt die Methode *auswertung*.

4.5 Die Konfigurationsdatei "ini.xml"

Neben den Dateien der Lerneinheiten wird auch die Konfigurationsdatei im XML-Format gespeichert. Ohne diese Datei würde das Programm nicht funktionieren, denn sie enthält alle wichtige Informationen über die verfügbaren Komponenten des Autorensystems. Zusätzlich sind noch Informationen über das Basis- und Projektverzeichnis enthalten.

Der Aufbau der Datei "ini.xml" ist im folgenden mit den Komponenten Textkomponente, numerische Frage und Bewertungskomponente exemplarisch dargestellt:

```
<?xml version="1.0"?>

<Ini>
  <Basisverzeichnis>d:\autorensystem\</Basisverzeichnis>
  <Projektverzeichnis>d:\autorensystem\projekte\</Projektverzeichnis>
  <Komponenteninformationen>
    <Typ gruppe="praesentation">TextKomponente</Typ>
    <Beschreibung>Text</Beschreibung>
    <Editor intern="nein" verwenden="nein">notepad.exe</Editor>
    <Endung>txt</Endung>
  </Komponenteninformationen>
  <Komponenteninformationen>
    <Typ gruppe="frage">NumerischeFrage</Typ>
    <Beschreibung>Numerische Frage</Beschreibung>
    <Editor intern="ja" verwenden="ja">Assistent</Editor>
    <Endung>numfra</Endung>
  </Komponenteninformationen>
  <Komponenteninformationen>
    <Typ gruppe="bewertung">Bewertungskomponente</Typ>
    <Beschreibung>Gesamtbewertung</Beschreibung>
    <Editor intern="ja" verwenden="ja">Assistent</Editor>
    <Endung>bewko</Endung>
  </Komponenteninformationen>
</Ini>
```

Beispielhaft sei hier die numerische Frage herausgezogen. Unter dem Attribut *gruppe* des *<Typ>*-Tags steht die Art der Komponente. Es handelt sich hierbei um eine Frage. Zwischen den *<Typ>*-Tags steht der Klassenname der Komponente (*NumerischeFrage*). Diese Information wird vor allem für den Reflection-Mechanismus benötigt. Der Wert zwischen den *<Beschreibung>*-Tags erscheint auf den Panels der Komponente in der Komponentenübersicht. Bei numerischen Fragen handelt es sich um den Wert "Numerische Frage". Die Informationen zwischen den *<Editor>*-Tags

gibt den Namen des zu verwendeten Editors für die Komponenten an. Die Attribute geben an, ob es einen internen Editor gibt und ob dieser verwendet werden soll. In dem beispielhaften Fall wird der interne Editor, der Frageassistent, verwendet. Die Information zwischen den *<Endung>*-Tags, gibt die Dateiendung einer numerischen Frage - "numfra" - an.

Die Konfigurationsdatei wird von der Klasse *Init* des Programms geladen und ausgelesen und stellt den anderen Teilen des Programms die wichtigen Informationen zur Verfügung.

5 Schlußbetrachtung

Die Untersuchung der bestehenden Autorensysteme ergab, daß die Bedenken von Lehrern und Dozenten in vielen Fällen berechtigt sind. Daher ist es sinnvoll, trotz der Vielzahl von Systemen, ein neues zu entwerfen, das nicht ausschließlich aber dennoch besonders für die Verwendung in Bildungseinrichtungen konzipiert wurde. Dabei wurde die Bedienung bewußt einfach gehalten, damit die Autoren nicht durch unnötige Funktionalität belastet werden.

Als Ergebnis liegt eine lauffähige Version des Autorensystems vor, in der die Komponentenübersicht und beispielhaft mehrere Komponenten implementiert wurden. Diese - sowie alle notwendigen Werkzeuge und Hilfsprogramme - befinden sich auf der beiliegenden CD-Rom.

Im folgenden werden auf Probleme während der Implementierung eingegangen und Erweiterungsmöglichkeiten genannt.

5.1 Probleme der eingesetzten Technologien und Werkzeuge

Bei der Implementierung des Autorensystems traten jedoch auch einige Nachteile der eingesetzten Technologien und Werkzeuge auf, die die Realisierung des Systems erschwert und verzögert haben.

Wird das Autorensystem beispielsweise auf älteren Systemen mit langsameren Prozessoren eingesetzt, kommt es zu deutlich langsameren Ausführungsgeschwindigkeiten. Dies liegt daran, daß Java den Quellcode nicht compiliert sondern interpretiert. Dadurch erscheint das System etwas schwerfällig. Java überzeugt durch eine klare Struktur, die leider nicht konsequent durchgehalten wurde. So gibt es immer wieder kleinere Inkonsistenzen, die die Implementierung verzögern. Es wäre daher wünschenswert, den guten Ansatz von diesen Inkonsistenzen zu bereinigen.

Auch DOM zeigt Schwächen, wenn große Dateien verarbeitet werden müssen. Dies liegt daran, daß DOM immer das gesamte Dokument in den Speicher einliest und parst. Da die Lerneinheiten im Normalfall aber wesentlich kleiner sind als ein Megabyte, fällt dieser Nachteil nicht so schwer ins Gewicht. Schwieriger hat sich die Implementation der Baumoperationen gestaltet, da sich DOM hier nicht wie erwartet und dokumentiert verhalten hat. Daher mußten umfangreiche Tests durchgeführt werden, um die Funktionen von DOM herauszufinden.

XML ist zwar standardisiert, wird aber von verschiedenen Browsern unterschiedlich unterstützt und dargestellt, was sich während der Implementierung herausgestellt hat. Daher ist die aktuelle Version des Autorensystems noch nicht optimal. Um die teilweise auseinanderlaufenden Entwicklungen in diesem Bereich zu überwinden, wird in zukünftigen Versionen das Problem auf eine andere Weise gelöst (siehe Kapitel 5.2)

5.2 Vorschläge für eine Weiterentwicklung

Das im Rahmen dieser Studienarbeit entstandene Autorensystem kann an verschiedenen Stellen weiter ausgebaut und verbessert werden. Neben der Beseitigung einiger verbleibender Programmierfehler werden im folgenden Erweiterungsmöglichkeiten aufgezeigt.

Im Bereich des Rahmenprogramms sollte eine Autorensteuerung implementiert werden, die verschiedene Lernmethodiken unterstützt. Dies sollte zusammen mit Konfigurationsmöglichkeiten für Autor und Entwickler in der Projektübersicht realisiert werden.

Wie bereits im vorigen Kapitel erwähnt, sollte die Lerneinheit nicht wie bisher vom Browser in HTML umgewandelt werden. Die Transformation der XML-Datei nach HTML kann auch direkt in Java - unter Verwendung von XSL und XSLT - implementiert werden. Dies würde garantieren, daß die Lerneinheit mit allen gängigen Browsern dargestellt werden kann, unabhängig von deren XML-

Unterstützung.

Die Portabilität des Autorensystems ist momentan durch die Klasse *Kommando* teilweise eingeschränkt. Ziel ist es, in späteren Versionen diese Klasse zu eliminieren und die benötigten Befehle auf andere Weise im Programm zu realisieren, um auch hier weitestgehend plattformunabhängig zu sein.

Im Bereich der Komponenten sind ebenfalls Verbesserungen möglich. Die Fehlerüberprüfung der einzelnen Komponenten kann ausgebaut und an die jeweiligen Gegebenheiten angepaßt werden. Jede Komponente sollte einen internen Editor besitzen, der implementiert werden muß, damit langfristig auf die Verwendung externer Editoren verzichtet werden kann. Die bereits vorgesehenen Attribute der Komponenten können sinnvoll eingesetzt werden. So ist zum Beispiel eine Option denkbar, mit der der Autor entscheiden kann, ob er vorformatierten Text oder Fließtext anzeigen möchte, was sich deutlich in der Anzeige der Lerneinheit niederschlägt.

Daneben sollten selbstverständlich neue Komponenten entwickelt und dem Programm hinzugefügt werden, um einen sinnvollen Einsatz des Autorensystems zu gewährleisten.

Ein wichtiger Punkt, der bei der Implementierung der aktuellen Version noch nicht zum Tragen kam, ist die ergonomische Gestaltung des ganzen Systems. Hierbei sind zahlreiche Verbesserungen möglich, die beispielsweise auch durch Befragungen von Experten sowie Anwendern erfolgen können.

Letztlich sollte das Autorensystem gegen mutwillige Beschädigungen sicher gemacht werden. So gilt es, das Rahmenprogramm gegen "gefährliche" Komponenten zu schützen, die den Betrieb gefährden könnten. Die versteckten Elemente der Lerneinheiten (z.B. richtige Antwort, Punktevergabe, usw.) sollten verschlüsselt werden, damit die Lernenden diese nicht unbefugt einsehen können.

Literaturverzeichnis

- [Boles] Boles, D.: Vorlesungsskript Multimedia-Systeme im Sommersemester 1997, Universität Oldenburg;
www-is.informatik.uni-oldenburg.de/~dibo/teaching/mm97/node27.html
- [Hall] Hall, M.: Core Servlets und JavaServer Pages, München: Markt+Technik Verlag 2001
- [Herczeg] Herczeg, M.: Software-Ergonomie - Grundlagen der Mensch-Computer-Kommunikation, Bonn Paris Reading, Mass. [u.a.]: Addison-Wessley 1994
- [Horst/Cor] Horstmann, C. S.; Cornell, G.: Core Java 2 Band 1 - Grundlagen, München: Markt + Technik Verlag 2001
- [Horst/Cor2] Horstmann, C. S.; Cornell, G.: Core Java 2 Band 2 - Expertenwissen, München: Markt + Technik Verlag 2002
- [McLau] McLaughlin, B.: Java und XML, Köln: O'Reilly Verlag 2001
- [Seeboe] Seeboerger-Weichselbaum, M.: Das Einsteigerseminar XML, Landsberg: verlag moderne industrie Buch 2001
- [Seeboe2] Seeboerger-Weichselbaum, M.: Java/XML, Landsberg: verlag moderne industrie Buch 2002