Entwurf und Implementierung einer vollständigen Infrastruktur für modulare E-Learning-Inhalte

Zur Erlangung des akademischen Grades
DOKTORINGENIEUR (Dr.-Ing.)
der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn
vorgelegte Dissertation

von
Dipl.-Inform. Michael Bungenstock
aus Hamburg

Referent:	Prof. Dr.-Ing. Bärbel Mertsching
Korreferent:	Prof. Dr.-Ing. Reinhard Keil-Slawik
Tag der mündlichen Prüfung:	6.4.2006

Paderborn, den 18.4.2006

Diss. 14/219

Kurzfassung

Mit dem Einzug des E-Learnings in Lehre und Ausbildung haben sich neue Anforderungen an die IT-Infrastrukturen ergeben, die mit den verfügbaren Techniken nicht adäquat gelöst werden können. Viele der neuen Teilaspekte, wie z.B. Lernobjekte, Metadaten und Kodierungen, werden zwar in wissenschaftlichen Arbeiten und Spezifikationen behandelt, aber leider fehlen die Zusammenhänge für die Implementierung eines vollständigen Systems. Hierdurch werden Seiteneffekte und Abhängigkeiten ignoriert, die in der Praxis zu essentiellen Problemen führen und proprietäre Lösungen hervorbringen. Der Akzeptanz von E-Learning ist diese Entwicklung abträglich, denn inkompatible E-Learning-Inhalte verursachen Kosten durch Konvertierung oder Neuentwicklung und mindern die Bereitschaft zur Verwendung. Die Verfügbarkeit von Inhalten beim E-Learning als wesentlicher Vorteil gegenüber konventionellen Formen wird auf diese Weise leider aufgehoben.

In dieser Arbeit wird ein ganzheitliches Konzept für modulare E-Learning-Inhalte hergeleitet und als praktische Anwendung realisiert. Als Grundlage dienen andere wissenschaftliche Arbeiten, die bereits wichtige und allgemein akzeptierte Ergebnisse hervorgebracht haben. Sie werden miteinander verbunden oder durch neue Konzepte ergänzt. Ein wesentliches Merkmal dieser Arbeit ist die Verwendung der Metaphern „multimedialer Baukasten“ und „Baustein“, die als Leitbild für alle Entscheidungen dienen. Sie vereinfachen den Entwurf der einzelnen Komponenten und prägen die spätere Benutzung des Systems. Hierdurch tragen die Metaphern zur Konsistenz und Vollständigkeit der Infrastruktur bei.

Entwurf und Umsetzung erfolgen in dieser Arbeit objektorientiert und bedienen sich der gängigen Mittel der Softwaretechnik. Aus Sicht der Benutzer/-innen wird ein fachliches Modell beschrieben, das durch Komponentenbildung in ein technisches überführt wird. Eine geringe Abhängigkeit gekoppelt mit einer hohen Kohäsion der Funktionen soll eine gute Skalierbarkeit für die unterschiedlichsten Einsatzgebiete garantieren. Durch die Flexibilität dieses Rahmenwerks lassen sich Einzelplatzlösungen genauso wie verteilte Anwendungen realisieren. Zur Demonstration werden in dieser Arbeit das Autorenwerkzeug Lyssa und ein Repository für die zentrale Datenhaltung entwickelt und in der Programmiersprache Java implementiert.

Das Rahmenwerk ist so konzipiert, dass es heutigen Standards entspricht und auch zukünftigen Entwicklungen gerecht wird. Ein Anliegen dieser Arbeit ist die Kompatibilität zu anderen Systemen, um eine breite Akzeptanz zu erreichen. Hierfür werden neben den Kodierungen in Standardformaten auch Konstruktionen zur Konvertierung auf Ebene der Standards, z.B. bei den Metadaten zwischen IEEE LOM und Dublin Core, sowie auf konzeptioneller Ebene, z.B. von verschachtelten zu einfachen Lernobjekten, vorgestellt. Denn die Wiederverwendbarkeit und die Vielseitigkeit von Inhalten gehören neben den multimedialen Möglichkeiten zu den herausragenden Stärken des E-Learnings. Mit dem Rahmenwerk dieser Arbeit sind nun die technischen Voraussetzungen geschaffen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AICC	**A**viation **I**ndustry **C**BT **C**ommittee
API	Application Program Interface
ARIADNE	**A**lliance Of **R**emote **I**nstructional **A**uthoring And **D**istribution **N**etworks for **E**urope
AWT	**A**bstract **W**indow **T**oolkit
BMBF	**B**undes**m**inisterium für **B**ildung und **F**orschung
BSCW	**B**asic **S**upport for **C**ooperative **W**ork
CAI	**C**omputer-**A**ssisted **I**nstruction
CBR	**C**ase-**B**ased **R**easoning
CBT	**C**omputer **B**ased **T**raining
CKS	**C**onstruction **K**it **S**erver
CMS	**C**ontent **M**anagement **S**ystem
CP	IMS **C**ontent **P**ackaging Information Model
CSS	**C**ascading **S**tyle **S**heets
DC	**D**ublin **C**ore
DCAP	**DC A**pplication **P**rofile
DII	**D**ynamic **I**nvocation **I**nterface
DRM	**D**igital **R**ights **M**anagement
DSSSL	**D**ocument **S**tyle **S**emantics and **S**pecification **L**anguage
DTD	**D**ocument **T**ype **D**efintions
GET	**G**rundlagen der **E**lektro**t**echnik
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**t**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotoco
I18N	**I**nternationalizatio**n**
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IGD	Fraunhofer **I**nstitut für **G**raphische **D**atenverarbeitung
ILS	**I**ntegrated **L**earning Management **S**ystem
IMC	**I**nternational **M**ail **C**onsortium
J2EE	**J**ava **2** Platform, **E**nterprise **E**dition
J2SE	**J**ava **2** Platform, **S**tandard **E**dition
JAX-RPC	**J**ava **A**PI for **X**ML-Based **RPC**
JAXM	**J**ava **A**PI for **X**ML **M**essaging
JPEG	**J**oint **P**hotographic **E**xperts **G**roup
JSP	**J**ava **S**erver **P**ages
KI	**K**ünstliche **I**ntelligenz
LCMS	**L**earning **C**ontent **M**anagement **S**ystem
LMS	**L**earning **M**anagement **S**ystemen
LO	**L**earning **O**bject
LODAS	**L**earning **O**bject **D**esign **a**nd **S**equencing Theory
LOM	**L**earning **O**bjects **M**etadata
LORI	**L**earning **O**bject **R**eview **I**nstrument
MathML	**Math**ematical **M**arkup **L**anguage

MDI ..... **M**ultiple **D**ocument **I**nterface
MIME ..... **M**ultipurpose **I**nternet **M**ail **E**xtensions
MVC ..... **M**odel, **V**iew, **C**ontroller
NFS ..... **N**etwork **F**ile **S**ystem
PC ..... **P**ersonal **C**omputer
PDA ..... **P**ersonal **D**igital **A**ssistants
PDF ..... **P**ortable **D**ocument **F**ormat
PIF ..... **P**ackage **I**nterchange **F**ile
PNG ..... **P**ortable **N**etwork **G**raphics
QName ..... **Q**ualified **Name**
RDF ..... **R**esource **D**escription **F**ramework
RIO ..... **R**eusable **I**nformation **O**bjects
RLO ..... **R**eusable **L**earning **O**bjects
RPC ..... **R**emote **P**rocedure **C**all
SCO ..... **S**harable **C**ontent **O**bjects
SCORM ..... **S**harable **C**ontent **O**bject **R**eference **M**odel
SCORM RTI .... **SCORM R**un**t**ime **I**nterface
SDK ..... **S**oftware **D**evelopment **K**it
SGML ..... **S**tandard **G**eneralized **M**arkup **L**anguage
SMB ..... **S**erver **M**essage **B**lock
SOAP ..... **S**imple **O**bject **A**ccess **P**rotocol
SWT ..... **S**tandard **W**idget **T**oolkit
TCP/IP ..... **T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol
TGN ..... **T**hesaurus of **G**eographic **N**ames
TP ..... **T**ransformation **P**ackage
UCS ..... **U**niversal Multiple-Octet **C**oded Character **S**et
UI ..... **U**ser **I**nterface
UML ..... **U**nified **M**odeling **L**anguage
URI ..... **U**niform **R**esource **I**dentifier
URIref ..... **URI Ref**erence
URL ..... **U**niform **R**esource **L**ocator
VFS ..... **V**irtual **F**ile **S**ystem
VLE ..... **V**irtual **L**earning **E**nvironment
VM ..... **V**irtual **M**achine
W3C ..... **W**WW **C**onsortium
WAM ..... **W**erkzeug, **A**utomat, **M**aterial
WBT ..... **W**eb **B**ased **T**raining
WebDAV ..... **Web**-based **D**istributed **A**uthoring and **V**ersioning
WSDL ..... **W**eb **S**ervices **D**escription **L**anguage
WWW ..... **W**orld **W**ide **W**eb
WYSIWYG ..... **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et
XML ..... E**x**tensible **M**arkup **L**anguage
XPath ..... **X**ML **Path** Language
XSD ..... **X**ML **S**chemas **D**efinition **L**anguage
XSL ..... E**x**tensible **S**tylesheet **L**anguage
XSL-FO ..... **XSL F**ormatting **O**bjects
XSLT ..... **XSL T**ransformations

Kapitel 1

Einleitung

„Non scholae, sed vitae discimus.“ „Nicht für die Schule, sondern für das Leben lernen wir“, steht es schon ungefähr seit Beginn unserer Zeitrechnung geschrieben.[1] Moderner wird heute im Kontext des stetigen gesellschaftlichen und wirtschaftlichen Strukturwandels vom *Lebenslangen Lernen* sowie der *Wissensgesellschaft* gesprochen. Jedem Individuum, unabhängig vom sozialen Stand, soll durch Wissen die Möglichkeit auf persönliche Verwirklichung und Anerkennung gegeben sein. Nur so lässt sich der Wohlstand einer Gesellschaft wahren und für eine ressourcenarme Nation wie Deutschland gilt dies um so mehr. Durch die Globalisierung steigt der weltweite Konkurrenzdruck, sodass Kompetenz und Qualifikation einen Standortvorteil bedeuten. Wissen muss folglich effektiv und effizient vermittelt werden.

Das *Bundesministerium für Bildung und Forschung* (BMBF) hat diesen Umstand erkannt und fördert unter anderem das Programm „Neue Medien in der Bildung“. Hierbei geht es um die breite Nutzung didaktisch hochwertiger Lehr- und Lern-Software in allen Bildungsbereichen, also einen Bereich des **E-Learnings**. Mit Hilfe von Maschinen soll die Wissensvermittlung verbessert werden.

Ein Projekt dieses Förderprogramms ist **mαth-kit** [Unger04; Unger02; Schiller02], das den Kontext dieser Arbeit darstellt. Ziel des Projekts ist die Erstellung eines **multimedialen Baukastens** [Thiere03b] zur Unterstützung der Mathematikausbildung im Grundstudium Mathematik, Technische Informatik, Maschinenbau und anderer Ingenieurwissenschaften [Thiere03a; Padberg02b; Padberg02a; Rehberg03]. Die Elemente des Baukastens eignen sich für Lehrende in Präsenz- und Fernlehre sowie für Studierende beim Selbststudium [Bauch03]. Als technische Realisierung ist eine flexible E-Learning-Plattform mit integriertem Autorensystem gedacht, die eine reibungslose Zusammenarbeit mit existierenden Systemen erlaubt. Da vier Universitäten[2] an diesem Projekt beteiligt sind, gilt ein besonderes Augenmerk der Erstellung bzw. Verwaltung von Inhalten in Gruppen. Insgesamt umfasst das Projekt mαth-kit somit die Erstellung sowie Veröffentlichung mathematischer Inhalte mit Hilfe eines eigens entwickelten Systems.

Der Fokus dieser Arbeit liegt auf der technischen Handhabung von Lehr- und Lerninhalten. Hierzu gehören neben Kodierungen, Protokollen und Datenhaltungsformen auch die geeignete Begriffs- bzw. Modellbildung. Beispiele für den konkreten Einsatz von E-Learning finden sich z.B. in [Dittler03].

[1] Laut Büchmann [Büchmann94] handelt es sich um die Umkehrung des Ausspruchs „Non vitae, sed scholae discimus“ aus den *Epistulae morales* von Seneca.

[2] Universitäten Paderborn, Hamburg, Bayreuth und Fernuniversität Hagen

1.1 Problemstellung

Bestrebungen nach technischen Vereinfachungen bei der Wissensvermittlung reichen bis ins 16. Jahrhundert zurück, wie ein Kupferstich aus dem Jahre 1588 von Agostino Ramelli belegt.[3] Abbildung 1.1 zeigt ein Bücherrad, das wahrscheinlich nie gebaut wurde.

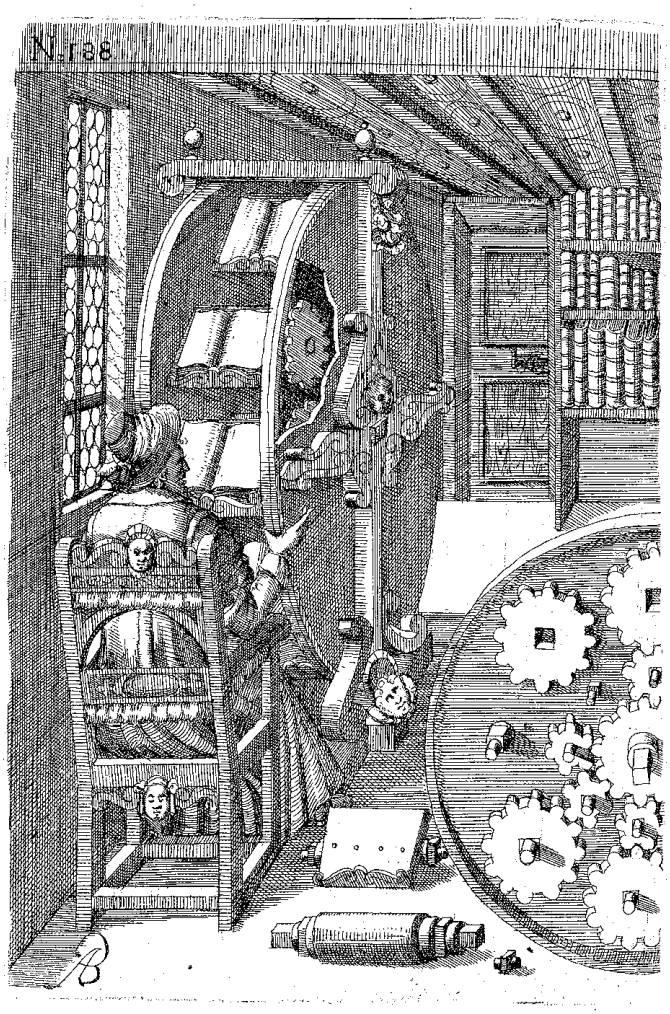


Abbildung 1.1: Das Bücherrad

Heute werden Computer als Hilfsmittel beim Lernen eingesetzt, weil sie den Zugriff auf Lerninhalte vereinfachen und die Darstellung komplexer Sachverhalte ermöglichen. Mit Hilfe des Internets kann von einfachen Abbildungen über Animationen bis hin zu interaktiven Simulationen alles auf heimischen PCs dargestellt werden. Tragbare *Laptops* und *Personal Digital Assistants* (PDA) erlauben in Kombination mit drahtlosen Übertragungswegen eine arbiträre Verfügbarkeit von Daten an fast jedem Ort der Erde. Die Möglichkeiten scheinen schier endlos zu sein. Doch reichen Internet und eine breite Palette von Anzeigeprogrammen für das E-Learning aus?

In Hinblick auf die Erwartungen und Hoffnungen, die sich an das E-Learning richten, kann die Antwort freilich nur „Nein“ lauten. Zu hoch sind die Ansprüche der Lehrenden und Lernenden, als dass sie in einfache HTML-Seiten gefasst werden könnten. Begriffe wie *Wiederverwendbarkeit*, *Sequenzing* oder *Personalized Learning* lassen erahnen, dass monolithisch erstellte Inhalte unzureichend sind. Es bedarf daher eines modularen Aufbaus, der auf die Anforderungen des E-Learnings eingeht.

[3] Agostino Ramelli (1531–1600) war ein italienischer Ingenieur des Königs von Frankreich. Der Originaltitel des Buches lautete „Le diverse et artificiose machine“.

Die Realisierung eines solchen Konzepts wirft jedoch in Theorie wie Praxis eine Fülle neuer Fragen auf, deren Beantwortung Bestandteil dieser Arbeit ist. Zwar gibt es eine Reihe von Modellen, Spezifikationen und Implementationen, die sich ausgiebig mit Teilfragen beschäftigen, aber der globale Zusammenhang fehlt. Besonders die Lücke zwischen den theoretischen Modellen und den existierenden Anwendungen scheint besonders groß zu sein. Konkret lassen sich folgende Fragen ableiten:

- *Was ist ein Modul im E-Learning-Kontext?* Die existierenden Auffassungen variieren in Umfang, Form und Funktion. Für die Beantwortung aller weiteren Fragen ist es wichtig, dass die gewählte Definition in Hinblick auf Didaktik und technischer Realisierung genügend Spielraum gewährt.
- *Wie können mehrere Module organisiert werden?* Von Interesse sind die verschiedenen Strukturen und Anordnungen, da sie die möglichen Aggregationen bestimmen.
- *Wie sollen Inhalte klassifiziert werden?* Die Module müssen mit Metadaten versehen werden, um auffindbar zu sein.
- *Wie sieht die technische Umsetzung eines Moduls aus?* Die Kodierung der Module spielt eine wesentliche Rolle für die Akzeptanz eines Systems.
- *Wie werden Module erzeugt und bearbeitet?* Entwicklungs- und Wartungsprozess müssen vom System unterstützt werden.
- *Wo werden Module gespeichert?* Es stehen z.B. Dateisysteme, Datenbanken und spezielle *Repositories* zur Verfügung.
- *Wie werden Module und deren Aggregate einheitlich dargestellt?* Inhalte aus verschiedenen Quellen sehen zwangsläufig unterschiedlich aus.
- *Wie lassen sich Objekte, Funktionen und Merkmale umgangssprachlich beschreiben?* Metaphern sind z.B. ein probates Mittel in der Informatik, um komplexe Sachverhalte zu veranschaulichen.

Die Vorteile des Computers beim Lernen können nur genutzt werden, wenn Produktion, Präsentation und Archivierung von E-Learning-Inhalten auf einem durchdachten Fundament beruhen. Um Allgemeingültigkeit zu erlangen, ist darauf zu achten, dass Einflüsse sozialer Art, wie z.B. Didaktik und Kultur, auf die technischen Konzepte vermieden werden. Bei der Umsetzung gilt es letztendlich, auf proprietäre Lösungen zu verzichten.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist, Entwurf, Implementation und Einsatz eines ganzheitlichen Konzepts für modulare E-Learning-Inhalte zu realisieren. Ihre Produktion, Präsentation und Archivierung soll gesamt abgedeckt werden, um einen konsistenten Umgang zu gestatten. Das reduziert die Kosten für Entwicklung, Wartung und Nutzung bei gleichzeitiger Qualitätssteigerung. Wenn vorhanden, sollen bewährte Modelle, Spezifikationen und Implementationen als Grundlage für die Realisierung eines technischen Systems dienen, das die genannten Merkmale aufweist. Für eine angemessene Verifikation und Validierung soll es als Prototyp implementiert werden.

Als Ausgangspunkt dient die Baukasten-Metapher des Projekts mαth-kit, die als Leitbild für das System dient. Alle Objekte, die während des Entwurfs benannt werden, sollen sich an den Begriff Baukasten anlehnen. Ein zentraler Gegenstand dieser Arbeit sind die modularen Inhalte und ihr Funktionsumfang. Sie sollen zu größeren Einheiten kombiniert und umgekehrt auch jederzeit in ihre Bestandteile zerlegt werden können. Durch eine Trennung von Inhalt

und Darstellung innerhalb der Module lassen sich so beliebige Lehr- und Lernmaterialien erstellen, die wie aus einem Guss wirken. Dies fördert die Wiederverwendung, unterstützt die Wartung und ermöglicht die Teamarbeit bei der Produktion. Auch Redundanzen bei der Archivierung lassen sich durch diesen Ansatz verhindern. Bei zentraler Datenhaltung muss jedes Modul nur einmal vorliegen, weil es lediglich referenziert wird. Die Metadaten gestatten bei der Präsentation eine gezielte Auswahl, da sie auch bei Aggregationen von Modulen den Zugriff auf einzelne Inhalte ermöglichen.

Für den Umgang mit modularen Inhalten werden passende Werkzeuge benötigt. Um den Aufwand der Implementation so gering wie möglich zu halten, sollen viele bewährte Applikationen eingesetzt werden. Hierzu gehören z.B. Web-Server, Datenbanken und Web-Browser. Nur Funktionen, die von anderen Programmen oder *Libraries* noch nicht angeboten werden, sollen in eigenen Werkzeugen umgesetzt werden.

1.3 Methodik

Diese Arbeit ist im angewandten Bereich angesiedelt, sodass die Ergebnisse durch Praxistauglichkeit überzeugen müssen. Ausgehend von der allgemeinen Problemstellung modularer E-Learning-Inhalte soll ein theoretisches Konzept entwickelt werden, das in einer speziellen Implementierung mündet. Die abstrakte Problemstellung konkretisiert sich somit durchgehend in immer genauere Teilprobleme. Am Ende steht ein Prototyp, mit dessen Hilfe die Funktionstüchtigkeit des Konzepts induktiv bewiesen werden soll.

Es gibt verschiedene Wege, einen Prototypen zu erstellen. In dieser Arbeit soll das *inkrementelle Modell* [Balzert00] eingesetzt werden, bei dem bereits am Anfang alle Anforderungen des Systems vollständig erfasst werden. Die Umsetzung gliedert sich in Ausbaustufen, die peu à peu die aufgestellten Anforderungen abdecken. Ein Vorteil dieses Verfahrens ist die schnelle Verfügbarkeit eines lauffähigen Systems, mit dem erste Tests gefahren werden können. Des Weiteren besteht keine Gefahr, dass die nächsten inkrementellen Erweiterungen nicht zu dem bestehenden System passen, da bereits im Vorfeld alle Anforderungen bekannt sind. Weil sich gewisse Abläufe bei den einzelnen Ausbaustufen wiederholen, wird bei dieser Art Prototyp von einem iterativen Modell gesprochen.

Bei der Entwicklung soll auf die Konzepte der objektorientierten Software-Entwicklung zurückgegriffen werden. Zur Notation wird die *Unified Modeling Language* (UML) [Obj03] verwendet, die sich als Standard durchgesetzt hat. Sie ist eine Sammlung verschiedener Diagrammarten, mit der sich dynamische wie statische Zusammenhänge darstellen lassen. Die eigentliche Implementierung wird mit der objektorientierten Programmiersprache *Java* [Gosling96] durchgeführt, da sie einfach im Funktionsumfang und plattformunabhängig ist.

1.4 Systematik

Der Aufbau dieser Arbeit leitet sich aus der Methodik ab und besteht aus vier Teilen. Der erste *Stand der Wissenschaft* gibt einen Überblick der aktuellen Arbeiten, die sich mit Aspekten der Problemstellung beschäftigen. Es wird jeweils erläutert, wie sich das vorliegende Thema in den Kontext der Arbeit fügt und welche relevanten Beiträge es gibt. Am Schluss dieses Teils wird eine Bewertung durchgeführt, indem die Zielsetzung den beschriebenen Arbeiten gegenüber gestellt wird, um bestehende Forschungslücken aufzudecken.

Der Teil *Entwurf* beschreibt detaillierter den benötigten Leistungsumfang und vermittelt ein theoretisches Modell vom System. Für die Beschreibung einzelner Komponenten, Beziehungen und Merkmale werden die Methoden der Software Technik herangezogen. Alle wichtigen Definitionen werden in UML-Notation angegeben und sind Basis für die Realisierung des Systems.

Die wird im Teil *Prototyp* ausführlich erläutert. Zuerst werden die kohärenten Funktionen in einzelne Libraries eingeteilt, um einen flexible Verwendung zu ermöglichen. Es wird nä-

her auf interessante Implementationsdetails eingegangen und beschrieben, wie ein praktischer Einsatz aussehen könnte. Danach werden die Applikationen von der Architektur bis zur Oberflächengestaltung vorgestellt und zu einem System zusammengeführt.

Abschließend wird im Teil *Analyse* die geleistete Arbeit untersucht und bewertet. Anhand ausgewählter Beispiele werden typische Situationen beim Umgang mit dem System durchgespielt. Hierbei stehen Praxistauglichkeit und Ergonomie im Vordergrund. Es folgt eine Beschreibung der erreichten Ergebnisse und eine Abschlussbewertung. Abgerundet wird die Analyse mit einem Ausblick auf weiterführende Themen, die sich während der Arbeit abzeichneten, aber nicht in dem vorgegebenen Zeitrahmen behandelt werden konnten.

Natürliche Sprachen unterliegen dem Zeitgeist. Aus diesem Grund sollen kurz ein paar formale Gesichtspunkte genannt werden, die für diese Arbeit gelten. Der Text hält sich an die 1996 reformierte deutsche Rechtschreibung. Soweit möglich, werden geschlechtsneutrale Substantive genutzt, wie z.B. Lernende statt Lerner und Lernerin. Lässt sich kein geeigneter Begriff bilden, werden beide Formen durch Schräg- und Bindestrich abgekürzt angegeben, wie z.B. Autor/-in. Anglizismen werden sparsam genutzt, jedoch sind sie in einer Domäne wie der Informatik unvermeidbar. Viele Begriffe haben sich bereits etabliert und sind in den alltäglichen Sprachgebrauch übergegangen. So wird niemand ernsthaft E-Learning mit E-Lernen oder Server mit Diener übersetzen. Zitate werden in ihrer Originalform belassen und weder übersetzt noch in der Rechtschreibung angepasst.

Für eine bessere Lesbarkeit des Textes werden bestimmte Wörter mit Formatierungen versehen, die sich vom restlichen Text abheben. Es soll folgende Konvention gelten:

Italic:	Namen und nicht integrierte englische Wörter, z.B. *Java, Repository*
Bold:	Einführung wichtiger Begriffe, z.B. **E-Learning**
`Typewriter`:	Quellcode, z.B. `System.out.println("Hello");`

Der Name des Projekts mαth-kit wird stets klein und mit dem griechischen Buchstaben Alpha geschrieben. Das wurde am Anfang von allen Beteiligten beschlossen und soll auch hier gelten.

Teil I

Stand der Wissenschaft

Kapitel 2

Lerntheorie

Die Hauptmotivation bei der Einführung neuer Medien beim Lernen ist die Optimierung des Lernprozesses. Alle Theorien, Entwürfe und Umsetzungen, die bei der Realisierung eines technischen Systems für E-Learning entstehen, müssen daher auf dieses Ziel ausgerichtet sein. Die Verbesserung eines Prozesses lässt sich jedoch nur erreichen, wenn die theoretischen Hintergründe bekannt sind. Aus diesem Grund wird ein kurzer Überblick über die Theorie des Lernens gegeben und ein Modell ausgewählt, mit dessen Hilfe weitere Klassifikationen von Lerninhalten durchgeführt werden.

2.1 Kompetenzstufen

Die Brüder Dreyfus haben ein fünfstufiges hierarchisches Lernmodell entwickelt, mit dem sich der Entwicklungsprozess von Lernenden beschreiben lässt [Dreyfus86; Humbert05; Klein99; Metzinger99]. Die zentrale Idee des Modells ist, dass Lernende von einem statischen Faktenwissen über ein dynamisch theoretisches Wissen zu intuitiven Fertigkeiten gelangen. Ein Mensch, der einer bestimmten Stufe zugeordnet werden kann, ist hierbei immer besser, als die höchstbegabten Menschen der darunter liegenden Stufen. Die fünf Stufen werden *Novice* (Neuling), *Advanced Beginner* (Fortgeschrittene/-r Anfänger/-in), *Competent* (Kompetenz), *Proficient* (Gewandtheit) und *Expertise* (Expertentum) genannt.

Neuling:
Der ersten Stufe sind alle Lernende zugeordnet, die sich einem ihnen unbekannten Thema zum ersten Mal annähern. Sie erlernen das Erkennen von Fakten und Mustern, um mit vorgegebenen Regeln ihre Handlungen zu bestimmen. Bei den Regeln handelt es sich um „kontextfreie Regeln“, da sie situationsunabhängig anhand eindeutig erkennbarer Elemente vom Neuling eingesetzt werden. Die Bewertung des Lernerfolgs umfasst lediglich, wie die erlernten Regeln befolgt wurden.

Fortgeschrittene/-r Anfänger/-in:
Beim ausgiebigen Lernen sammeln Menschen vielfältige Erfahrungen, wie mit realen Situationen umzugehen ist. Dies befähigt sie, mehr und kompliziertere kontextfreie Regeln in ihre Überlegungen einzubeziehen. Lernende fangen an, relevante Elemente selbständig und schneller zu erkennen, da sie aus vorherigen Beispielen schon bekannt sind. Auch für diese „situationalen“ Elemente gibt es Verhaltensregeln. Der Lernerfolg der Fortgeschrittenen zeigt sich in der gewonnenen Erfahrung, die nicht objektiv beschreibbar ist.

Kompetenz:
Die Handlung von Menschen auf dieser Stufe ist durch die Wahl eines Organisationsplans geprägt. Sie kennen bereits viele relevante Fakten und Regeln, die sie auf ein breites Spektrum von Fällen anwenden können. Durch die Organisation einer Situation muss

nur noch eine kleine Menge an Faktoren eines Plans berücksichtigt werden, wodurch die Komplexität einer Aufgabe reduziert wird. Für die Auswahl des Planes benötigt die kompetente Person jedoch einige Überlegungen, da die Tragweite der Wahl das gesamte Vorgehen entscheidet. Der Lernerfolg zeigt sich im Wandel der eigenen Beziehung zu der Umwelt. Die Kompetenten fühlen sich verantwortlich für ihr eigenes Handeln und den sich ergebenden Konsequenzen. Obwohl sie während des Entscheidungsprozesses Abstand zu den Dingen bewahren, sind sie mit den Auswirkungen ihres Handelns zutiefst verbunden.

Gewandtheit:
Die Fertigkeit der Lernenden nicht nur schlichte Regeln anzuwenden, sondern bewusst Entscheidungen zu treffen, wird mit wachsender Erfahrung intuitiver ausgeprägt sein. Eine gewandte Person fühlt sich wie der Kompetente mit ihrem Problem verbunden, jedoch wählt sie ihren Organisationsplan nicht aufgrund distanzierter und reflektierter Bewertungen. Vielmehr geschieht die Handlung automatisch, ohne bestimmte Überlegungen, da auf Erfahrungen vergangener Situationen zurückgegriffen wird. Diese Intuition ist eine Fähigkeit bei alltäglichen Problemen, und kein Raten oder eine übernatürliche Inspiration. Der Lernerfolg zeigt sich durch die intuitive Benutzung von Mustern, die nicht in einzelne Komponenten zerlegt werden müssen.

Expertentum:
Das Handeln von Personen auf dieser Stufe ist nicht distanziert von den Problemen, nicht von Gedanken der Auswirkungen geprägt und verläuft nicht nach Organisationsplänen — es ist bereits Bestandteil der Person geworden. Dies bedeutet allerdings nicht, dass Experten/-innen unüberlegt wichtige Entscheidungen treffen oder sich ihrer Handlungen nicht bewusst sind. Auch ihnen unterlaufen Fehler und unvorhersehbare Ereignisse können Probleme bereiten. Der Lernerfolg zeigt sich deshalb durch das Eingebunden-Sein in Problemsituationen.

2.2 Lernparadigmen

Neben den verschiedenen Stufen, die Menschen während des Lernens erreichen können, sind auch die verschiedenen Paradigmen des Lernens — historisch gewachsene Theorien, die in sich abgeschlossen sind — von Bedeutung. Im letzten Jahrhundert sind, hier in chronologischer Reihenfolge angegeben, **Behaviorismus**, **Kognitivismus** und **Konstruktivismus** als maßgebliche Theorien zu nennen [Baumgartner97]. Sie basieren alle auf bestimmten Annahmen zur Arbeits- und Funktionsweise des Gehirns, wodurch sich unterschiedliche Lehrstrategien und Lernziele ergeben.

2.2.1 Behaviorismus

Im Behaviorismus wird Lernen als konditionierter Reflex betrachtet, der durch Adaption erworben wird. Als Begründer des Behaviorismus gilt der amerikanische Psychologe John Broadus Watson von der Johns Hopkins Universität [Wozniak94]. Seit seiner Arbeit *„Psychology as the behaviorist views it“* (1913) werden alle Forschungen unter diesem Begriff zusammengefasst, deren Basiseinheiten aus Reiz-Reaktions- bzw. Stimulus-Response-Verbindungen (S-R-Verbindungen) bestehen. In den fünfziger Jahren wurde der Behaviorismus zum beherrschenden Paradigma der amerikanischen Psychologie und nur wenige Jahre im späteren Nachkriegseuropa übernommen.

Im Prinzip betrachtet der Behaviorismus das menschliche sowie das tierische Gehirn als eine Black-Box, bei der ein *Input* (Reiz) einen deterministischen *Output* (Reaktion) erzeugt (Abbildung 2.1).

Ob der gewünschte *Output* zum *Input* passt, also das gezeigte Verhalten richtig war, wird über ein extern gesteuertes Feedback vermittelt, den so genannten Konsequenzen. Sie sollten

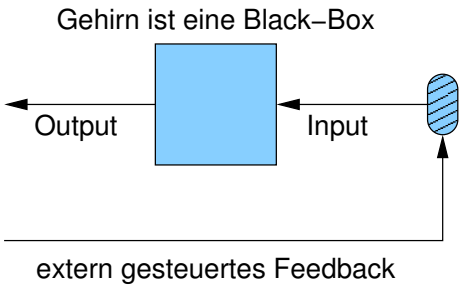


Abbildung 2.1: Schematisches Modell des Behaviorismus [Baumgartner99, S.102]

in einem kurzen Zeitraum, am besten unmittelbar, auf das Verhalten folgen, um eine Verstärkung bei positiven Leistungen bzw. eine Abschwächung oder Löschung bei negativen zu erreichen [Kerres98].

Die behavioristische Lehrstrategie setzt daher voraus, dass die Lehrenden genau wissen, was richtig und was falsch ist, da sie für die Konsequenzen verantwortlich sind. Dagegen stellt sich der Lernprozess für die Lernenden als eine Art Verhaltenssteuerung dar und steht damit im Gegensatz zum kognitiven Lernen. Obwohl diese Art des Lehrens heute als nicht mehr zeitgemäß erachtet wird, hat sie in gewissen Bereichen noch ihre Existenzberechtigung. Als Beispiel sei das *Drill & Practice* Muster in Sprachlabors genannt oder das erlernen körperlicher Fertigkeiten wie Maschinenschreiben, Jonglieren und Autofahren.

2.2.2 Kognitivismus

Die hauptsächliche Kritik am Behaviorismus, dass die inneren Prozesse des Gehirns ausgeblendet werden und folglich die komplexen Vorgänge des menschlichen Lernens keine Berücksichtigung finden, hat zum Kognitivismus geführt.

> „Die kognitionstheoretische Grundposition unterscheidet sich von der behavioristischen zunächst dadurch, daß der Lernende als ein Individuum begriffen wird, das äußere Reize aktiv und selbständig verarbeitet und nicht einfach durch äußere Reize steuerbar ist.“ [Tulodziecki96, S. 43]

Der Kognitivismus ist heute das noch dominierende Paradigma und es gibt eine Reihe von verschiedenen Ausprägungen. Allen gemein ist der Begriff der Informationsverarbeitung, was zu einer gewissen Äquivalenz von Computer und Gehirn führt. Abhängig von der Einschätzung dieser Annahme, kann von „starker“ oder „schwacher“ Künstlicher Intelligenz (KI) gesprochen werden [Searle86]. Die „starke“ KI geht von dem Standpunkt aus, dass die Beziehung zwischen menschlichem Gehirn und Computern eine Analogie ist und nicht bloß ein methodisches Verfahren. Minsky, als Vertreter dieser Ansicht, schreibt in dem Prolog seines Buchs Mentopolis:

> „Die meisten Leute glauben immer noch, daß keine Maschine je ein Gewissen, Ehrgeiz, Neid, Humor entwickeln oder andere geistige Lebenserfahrungen machen kann. Natürlich sind wir weit davon entfernt, Maschinen mit menschlichen Fähigkeiten bauen zu können. Aber das bedeutet, daß wir bessere Theorien über die Denkfähigkeit brauchen.“ [Minsky94, S. 19]

Anhänger der „schwachen“ KI gehen nicht so weit und sehen die Analogie als eine heuristische Annahme. Unabhängig von der Sichtweise sind sich die Kognitivisten darin einig, dass die Prozesse innerhalb des menschlichen Gehirns modelliert werden müssen. Hierfür müssen geeignete Wissensrepräsentationen und Algorithmen gefunden werden, mit denen die Fähigkeiten Lernen, Erinnern, Vergessen etc. modelliert werden können. Denkprozesse sind dann

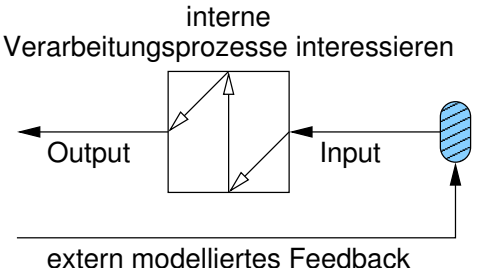


Abbildung 2.2: Schematisches Modell des Kognitivismus [Baumgartner99, S.105]

Wechselwirkungen von externen Angeboten und internen Strukturen. Abbildung 2.2 verdeutlicht diesen Zusammenhang.

Beim Lernen muss der Kognitivismus daher von einem objektiven externen Wissen ausgehen, welches in der Realität unabhängig vom Bewusstsein existiert. In dem schematischen Modell wird die Steuerung des externen Einflusses auf den Lernenden durch das *Feedback* angedeutet. Eine bedeutende Schwäche dieses Paradigmas ist die unmögliche bzw. umständliche Modellierung körperlicher Fertigkeiten.

2.2.3 Konstruktivismus

Im Konstruktivismus wird Lernen als aktiver Prozess betrachtet, bei dem der Mensch durch seine Sinne die Umwelt wahrnimmt und in Beziehung mit früheren Erfahrungen zu einem individuellen Wissen konstruiert. „Die Umwelt, so wie wir sie wahrnehmen, ist unsere Erfindung." [Foerster95, S.40]. Das Gehirn wird als selbstreferentielles System betrachtet, das Energien — nicht Informationen — über die Sinnesorgane verarbeitet und daraus neue individuelle Informationen erzeugt (Abbildung 2.3).

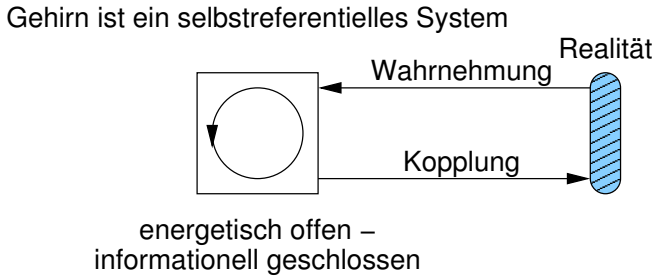


Abbildung 2.3: Schematisches Modell des Konstruktivismus [Baumgartner99, S.108]

Damit bildet dieses Paradigma den Gegensatz zum Kognitivismus, bei dem objektive Informationen aufgenommen und verarbeitet werden. Freilich verleugnet der Konstruktivismus nicht eine existierende Realität, jedoch geht er davon aus, dass sie nicht objektiv empfunden werden kann.

Für das Lehren und Lernen zieht diese Ansicht Konsequenzen nach sich. Streng genommen kann es z.B. keine optimale Wissensvermittlung geben, da sie individuell und unvorhersagbar ist. Die Lehrenden können unterstützend wirken, indem sie die Aktivierung von Vorkenntnissen, ihre Ordnung, Korrektur, Erweiterung, Ausdifferenzierung und Integration fördern.

> „Lernen bedeutet nach dem konstruktivistischen Paradigma: Wahrnehmen, Erfahren, Handeln, Erleben und Kommunizieren, die jeweils als aktive, zielgerichtete Vorgänge begriffen werden." [Klimsa93, S. 22]

Sie begleiten den Lernprozess und verhelfen den Lernenden zu eigenen Problemstellungen, die selbständig entwickelt und gelöst werden müssen. Die Problemfindung kann ein chaotischer, verwirrender Prozess sein, der aber dem Lernen zuträglich ist. Als einziges Paradigma der drei

genannten orientiert sich der Konstruktivismus mehr am Lernenden und macht die Qualität der Wissensvermittlung nicht nur an der Eingabe der Lehrenden fest.

Ende der achtziger, Anfang der neunziger Jahre haben sich drei Ansätze des „gemäßigten“ Konstruktivismus hervorgetan, denen gemein ist, dass sie Situiertheit und Anwendungsbezogenheit in den Vordergrund stellen. Die *Anchored-Instruction-Theorie* [CTGV90; CTGV93] verpackt authentische Probleme in Geschichten, die *Cognitive-Flexibility-Theorie* [Spiro88; Spiro91] setzt auf den Facettenreichtum realer Problemstellungen und die *Cognitive-Apprenticeship-Theorie* [Spiro88] nutzt authentische Aktivitäten und soziale Interaktionen in Expertenkulturen.

2.3 Lehrer/-in, Tutor/-in und Coach

Die beschriebenen Lernparadigmen wirken sich erwartungsgemäß auf die Lehre bzw. die Art der Wissensvermittlung aus. In [Baumgartner97] wird für jede der drei Theorien ein Begriff und eine Eigenschaftsbeschreibung für Lehrende definiert. Demnach werden Lehrende im Behaviorismus als *Lehrer/-innen* bezeichnet, die als Autoritätspersonen auftreten und genau wissen, was sie vermitteln möchten. Sie müssen nur die geeigneten Mittel und Wege für den „Wissenstransfer“ finden.

Im Kognitivismus werden gestellte Aufgaben von den Lernenden relativ selbständig bearbeitet, weshalb die Lehrenden den Lösungsprozess als *Tutoren* begleiten. Sie beobachten und geben bei Bedarf Hilfestellungen.

Im Konstruktivismus nehmen Lehrende die Rolle eines *Coaches* ein. Lernende generieren sich die Problemstellungen selbst, um komplexe Situation zu bewältigen. Hierdurch verlieren die Lehrenden ihre Unfehlbarkeit, da sie sich der Kritik der praktischen Situation aussetzen. Ihre lehrende Funktion nehmen sie durch ihre Erfahrung und die Fähigkeit der Betreuung war.

Abbildung 2.4 stellt die verschiedenen Lehrmodelle gegenüber und zählt die relevanten Eigenschaften auf.

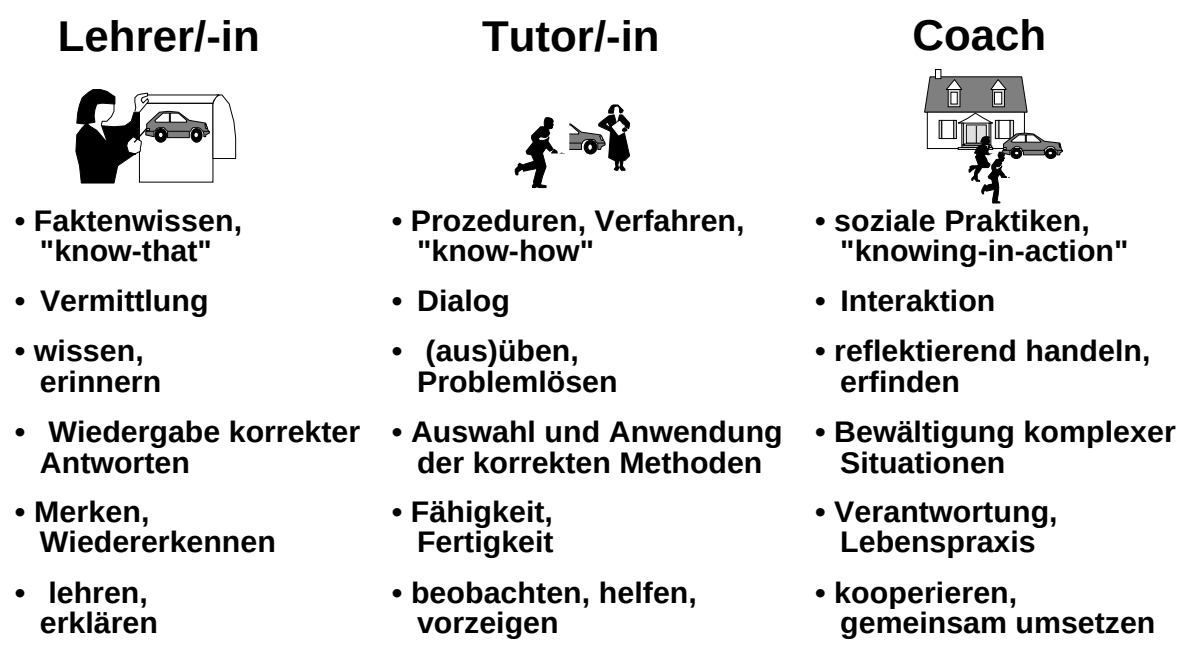


Abbildung 2.4: Drei Lehrmodelle [Baumgartner97]

2.4 Ein heuristisches Lernmodell

Die beschriebenen Kompetenzstufen mit ihren Lernelementen und das Lehrmodell können zu einem heuristischen Lernmodell zusammengefügt werden [Baumgartner99]. Hierdurch ergibt sich ein Modell, das die drei wichtigen Variablen Lernziele, Lerninhalte und Lehrstrategien beinhaltet. Die Beziehungen und Zusammenhänge zwischen den gleichrangigen Variablen können so dreidimensional dargestellt werden. Abbildung 2.5 zeigt das Modell als Würfel.

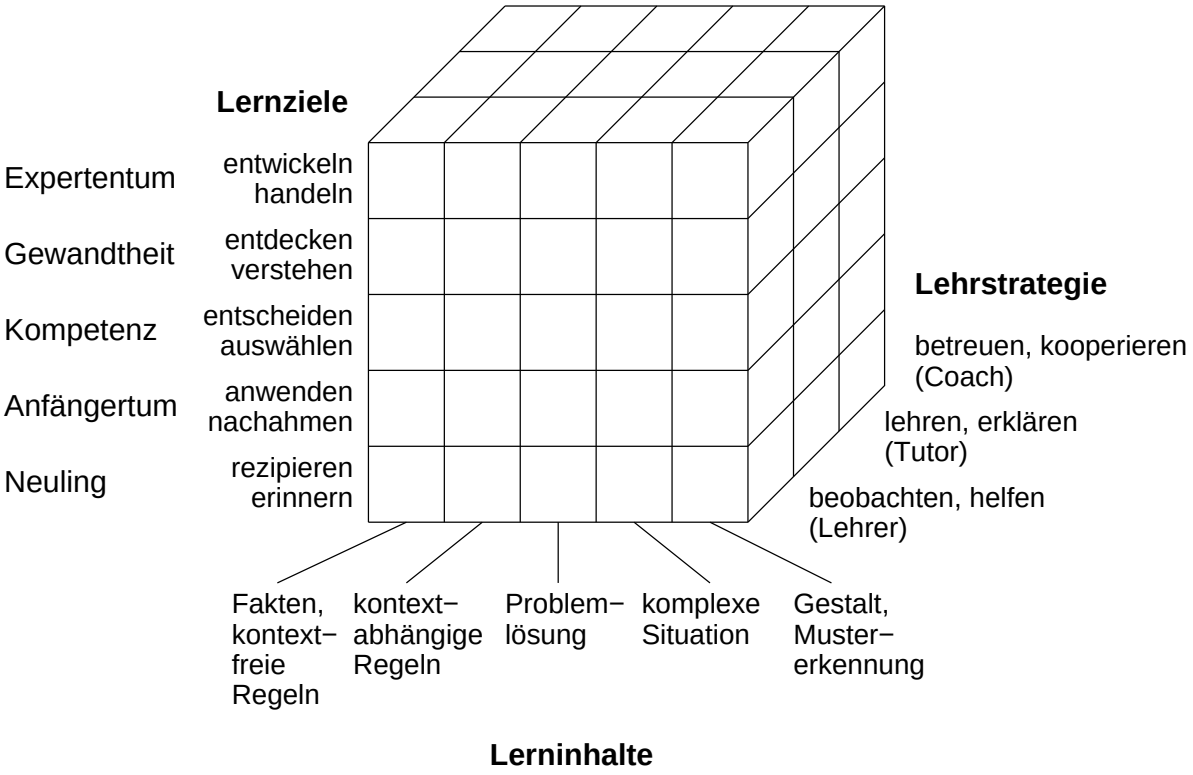


Abbildung 2.5: Ein heuristisches Lernmodell [Baumgartner99, S.96]

Ein wichtiger Punkt bei diesem Modell ist, dass es nur für Untersuchungszwecke eingesetzt werden kann und kein Entscheidungs- oder Vorgehensmodell ist. Schulmeister gibt in [Schulmeister03] die implizite Abgeschlossenheit zu bedenken. Ein Würfel kann nicht um eine vierte Dimension erweitert werden und die Skalierung der drei Dimensionen muss feststehen. Ob dieses Modell allen auftretenden Szenarien gerecht wird, kann in dieser Arbeit nicht betrachtet werden.

Dennoch soll Baumgartners Modell in dieser Arbeit als eine Lösung für die Taxonomie subjektiver Metadaten gesehen werden, wie sie in Kapitel 4 vorgestellt werden. Bei einer Beschreibung von Lernmaterialien kann mit Hilfe des Modells eine Beziehung zwischen dem Wissen in einer Disziplin, dem angestrebten Niveau des Lernens und den lerntheoretischen Ansätzen angegeben werden. Dies ist bereits viel mehr, als die verbreiteten Metadaten-Standards zu bieten haben.

2.5 E-Learning-Historie

Der Begriff **E-Learning** ist die Kurzfassung für *Electronic Learning* und umfasst die Gruppe der Lehr- und Lernverfahren, die Informations- und Kommunikationstechnologien einsetzen. Abhängig vom jeweiligen Stand der Technik, haben sich in der Entwicklung des E-Learnings unterschiedliche Systeme und Verfahren entwickelt. Bodendorf teilt diesen Prozess in drei Phasen ein, die in Abbildung 2.6 dargestellt sind.

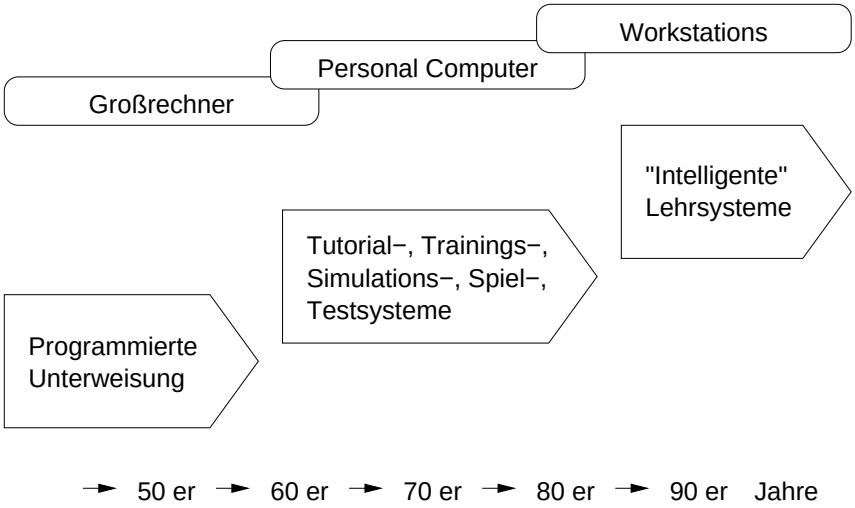


Abbildung 2.6: Entwicklung der computerunterstützten Ausbildung nach [Bodendorf90, S.15]

Burrhus Skinner entwickelte in den fünfziger Jahren eine erste „Lernmaschine“ [Skinner54]. Um das Lernen effizienter zu gestalten, zerlegte er den Lernprozess in so viele Teile, dass dieser nicht mehr von einer Lehrerin oder einem Lehrer vermittelt werden konnte und eine maschinelle Unterstützung bedurfte. Mit dem Einzug der Großrechner wurde zunächst versucht, den gängigen Unterricht nachzuimplementieren, allerdings mit mäßigem Erfolg. Die Programme waren in der Bedienung schlichtweg zu kompliziert und verlangten spezielle EDV-Kenntnisse.

In den siebziger Jahren kam mit dem *Personal Computer* (PC) eine preiswerte Alternative zu den Großrechnern auf den Markt. Sie wurden zunächst wie die **Telemedien** — hierzu zählen z.B. Diaprojektoren, Videorecorder und Bildplattenspeicher — als unterstützende Hilfsmittel eingesetzt. Mit der Zeit entwickelte sich dann das **Computer Based Training** (CBT), bei dem den Lernenden komplexe Sachverhalte multimedial vermittelt werden. Der Begriff **Multimedia** ist hierbei als technisches Attribut zu sehen:

> „The mixing of audio, video, and data is called multimedia; it sounds complicated, but is nothing more than commingled bits.“ [Negroponte96, S. 18]

Mit der Verbreitung des Internets entwickelte sich aus dem CBT das **Web Based Training** (WBT) und befreite die Lernenden aus ihrer Isolation. Neben der besseren Verfügbarkeit der Lernmaterialien über das Netzwerk werden verschiedene Kommunikationsmethoden wie z.B. E-Mail, Forum und Chat angeboten. Abbildung 2.7 verdeutlicht noch einmal die Gemeinsamkeiten und Unterschiede von CBT und WBT als Diagramm.

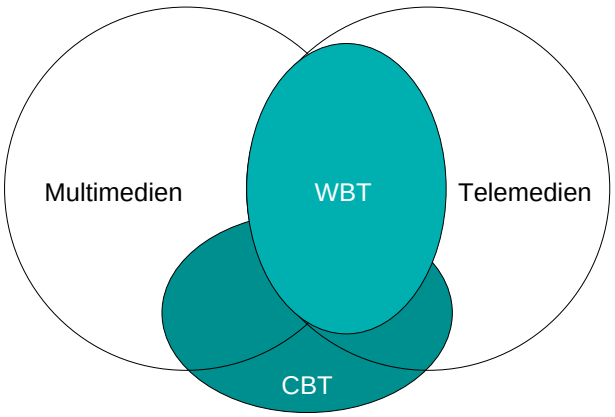


Abbildung 2.7: Begriffsbildung von WBT und CBT nach [Kerres98, S.14]

Kapitel 3

Lernobjekte

Inhalte für E-Learning benötigen eine gewisse Form, um sie elektronisch verarbeiten zu können. In der Literatur wird diese überwiegend als **Lernobjekt** (*Learning Object*) bezeichnet, ohne dass es einen gemeinsamen Konsens darüber gibt, was sich eigentlich hinter diesem Begriff verbirgt. Viele, die sich mit diesem Thema intensiv beschäftigen, stellen dieses Faktum fest und tragen ihre persönliche Definition bei, sodass ein Wust an Beschreibungen entstanden ist, der mühsam zu durchschauen ist.

Für diese Arbeit ist eine präzise Definition des Begriffs Lernobjekt von tragender Bedeutung, da er das Fundament für die Umsetzung des Baukastens ist. Aus diesem Grund werden verschiedene Aspekte der Lernobjekte betrachtet, um den gesamten Kontext dieses Begriffs auszuleuchten. Zuerst soll geklärt werden, was ein Lernobjekt ausmacht und welche Eigenschaften es im idealen Fall besitzt. Anschließend wird die Problematik der geeigneten Granularität angegangen (Abschnitt 3.3), die sich in der Praxis oft als schwierigste Aufgabe erweist.

3.1 Warum werden Lernobjekte benötigt?

Bevor die Lernobjekte näher untersucht werden können, muss als erstes ihr Bedarf festgestellt werden. Ein Beispiel von Stephen Downes [Downes00a] in leicht abgewandelter Form soll als erster Anhaltspunkt dienen.

Alle Universitäten in Europa zusammengenommen bieten sicherlich hunderte von Veranstaltungen an, in denen trigonometrische Funktionen behandelt werden. Die Mehrheit der Dozenten werden ihre selbst entwickelten Lehrmaterialien einsetzen, sodass die trigonometrischen Funktionen viele Male mit geringfügigen Unterschieden behandelt werden. Unter ökonomischen Gesichtspunkten ist diese Redundanz nicht akzeptabel, da die Erstellung guter Lehrmaterialien mit erheblichen Kosten verbunden ist. Es ist somit besser, ein Thema einmalig erschöpfend aufzubereiten und als Lernobjekt in allen Veranstaltungen — individuellen Animositäten beiseite gelassen — einzusetzen. Eine Kostenreduzierung durch Wiederverwendung ist somit ein Argument für Lernobjekte.

Aber nicht nur aus finanziellen Gründen, sondern auch für die Verwirklichung der Mehrwerte des E-Learnings, die bereits 1969, noch unter dem Namen *Computer-Assisted Instruction* (CAI), ausführlich in [Atkinson69] beschrieben wurden, sind Lernobjekte von Bedeutung. Demnach soll Lehrstoff beim E-Learning *adaptiv*, *generisch* und *skalierbar* sein [Gibbons02]. Die ersten zwei Adjektive sind Grundlage für ein individuelles, verbessertes Lernen und finden sich heute im *personalisierten Lernen* wieder [Martinez00]. Adaptiver Lehrstoff passt sich dem Wissens- und Leistungsstand der Lernenden an, indem z.B. geeignete Lernpfade oder ausgewählte Übungen angeboten werden. Daher kann die Zusammenstellung nicht im Voraus sondern nur generisch erfolgen. Es ist somit möglich, dass zwei Studierende mit ungleichen Voraussetzungen unterschiedlichen Lehrstoff bearbeiten, obwohl beide das gleiche Thema in einem Kurs behandeln. Personalisiertes Lernen benötigt zur Aufteilung der Inhalte folglich ei-

ne technische Einheit wie das Lernobjekt, die von einem System selbständig verarbeitet wird. Eine Verbesserung der Lehre ist somit ein weiteres Argument für Lernobjekte. Die Forderung des CAI nach skalierbaren Inhalten hingegen, bedeutet die Vervielfältigung von Lehrmaterialien nach industriellen Maßstäben und deckt sich mit den Anforderungen der Kostenreduktion, die bereits im Beispiel von Downes beschrieben sind.

3.2 Was ist ein Lernobjekt?

Das Lernobjekt ist für die technische Realisierung von E-Learning unerlässlich, doch wie ist es nun genau definiert? Um dieser Frage nachzugehen, werden verschiedene Ansätze erläutert und bewertet. Abschließend wird eine genaue Definition gegeben, die als Grundlage für weitere Betrachtungen gilt.

Bevor es weiter in die Details geht, soll hier ein etymologischer Fehler des Begriffs Lernobjekt ausgeräumt werden, der sehr häufig in der Literatur zu finden ist: Das Lernobjekt rühre vom objektorientierten Paradigma der Software-Technik her. Diese Herleitung ist definitiv falsch, da es weder Klassen noch erzeugte Objekte gibt. Wenn die Nähe zur Software-Technik gewünscht ist, wäre der Begriff *Modul* mit seinen Eigenschaften angebrachter. Tatsächlich gibt es in Wissenschaft und Wirtschaft eine Reihe anderer Begriffe, aber im Prinzip beschreiben sie und ihre Permutationen immer das gleiche Konzept, sodass sie als Synonyme für das Lernobjekt angesehen werden. Dies sind z.B.:

- Learning Module
- Instruction Object
- Educational Object
- Content Object
- (Reusable) Information Object
- Training Component
- Nugget
- Chunk

Weitere Bestrebungen für Lernobjekt-Ontologien finden sich z.B. in [Mosley05; Qin04].

3.2.1 Lernobjekte nach Cisco Systems

Das Unternehmen *Cisco Systems* beschreibt in seinem Strategie-Papier [Cis99] einen Ansatz für Lernobjekte auf Basis von `Reusable Information Objects` (RIO). Ein RIO ist eine granulare und wiederverwendbare Informationseinheit, die einmal entwickelt, auf verschiedenen Medien eingesetzt werden kann. Sie ist unabhängig von anderen RIOs, kann aber bei Bedarf mit ihnen zu höheren Strukturen, den `Reusable Learning Objects` (RLO), kombiniert werden. Ein RLO sollte, wie in Abbildung 3.1 dargestellt, aus Überblick, Zusammenfassung, Bewertung und 5–9 RIOs bestehen.

Mit diesem Ansatz soll ein Paradigmen-Wechsel herbeigeführt werden, weg von den traditionellen Kursen, die als unflexible, monolithische Blöcke daherkommen, hin zu den wiederverwendbaren Lernobjekten. Die Hauptkritik am Kurs liegt am vorgegebenen Lernpfad, der von den Lehrenden einmalig entwickelt wird und somit nur kognitivistisches Lernen (siehe z.B. [Searle86; Tulodziecki96]) erlaubt. Auf die individuellen Voraussetzungen der Lernenden, wie z.B. Vorwissen und Begabung, kann bei dieser uniformen Lehre nicht eingegangen werden.

Cisco unterscheidet zwischen den Vorteilen eines RIOs für Autoren und Lernende. Die Autoren profitieren von den RIO-spezifischen *Templates* als Grundlage für ein konsistentes

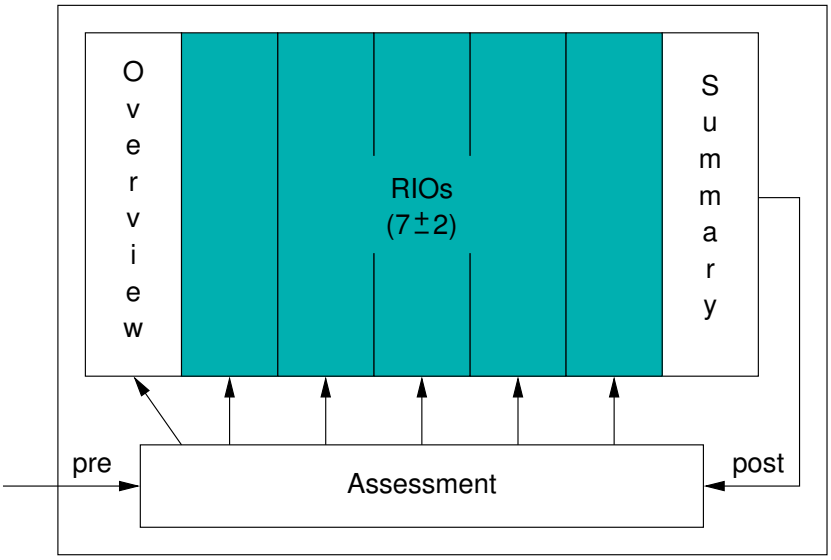


Abbildung 3.1: RLO-RIO-Struktur

Design, der Wiederverwendbarkeit von RIOs in zukünftigen Projekten und der Erstellung von höheren Strukturen, den so genannten RLOs. Für die Lernenden sind die RIOs konsistent in Design und Struktur, stehen jederzeit zur Verfügung und erlauben individuelle Lernpfade, die sich ihrem Wissen und ihren Fähigkeiten anpassen.

3.2.2 Lernobjekte nach Hodgins

Wayne Hodgins motiviert in seinem Whitepaper [Hodgins00] den Einsatz von Lernobjekten als Container für Information. In unserer Wissensgesellschaft ist das Wissen in den Köpfen von Experten die wertvollste Ressource, die, natürlichen Ressourcen gleich, ihren wahren Wert erst durch Extraktion, Aufbereitung und Veröffentlichung erhält. Ziel muss also sein, das vorhandene Wissen zu konvertieren, um es mit anderen in Konversation, Abbildung, geschriebenem Wort, Modell, Simulation und anderen Formen auszutauschen. Es ist jedoch wichtig, dass die Lernenden nur mit den Informationen konfrontiert werden, die sie wirklich benötigen.

Die Größe der Informationseinheiten spielt für Hodgins eine wesentliche Rolle:

> „Size matters: Smaller is better.“ [Hodgins00, S. 27]

Er kritisiert die bisherigen Kursgrößen, weil aus Gründen der Effizienz Kurse allumfassend und uniform gestaltet werden, um sie vielen Lernenden zur Verfügung zu stellen. Zudem werden meist proprietäre Datenformate genutzt, sodass es neben den bekannten Einschränkungen mit dem vorgegebenen Lernpfad auch noch zu Kompatibilitätsproblemen kommt. Informationen müssen also im Sinne der Wiederverwendbarkeit in kleinen, kompatiblen Einheiten bereit gestellt werden, die sich beliebig rekombinieren lassen. Hodgins schlägt für die technische Realisierung die *Reusable Information Objects* (RIO) von *Cisco* vor (siehe Abschnitt 3.2.1), die sich nach der Hierarchie in Abbildung 3.2 richten.

Für die Generierung dieser Strukturen werden zusätzlich Metadaten benötigt, die den Inhalt des jeweiligen Lernobjekts charakterisieren. Sie können entweder objektiv sein, d.h., eine automatische Vergabe durch das System ist möglich (z.B. das Erstellungsdatum), oder sind subjektiv und unterliegen der persönlichen Einschätzung einzelner Personen oder Gruppen.

Um die Eigenschaften der Lernobjekte verständlicher zu beschreiben, bedient sich Hodgins des LEGO-Bausteins als Metapher[1]. Mit den gleichen Bausteinen lassen sich Brücken, Häuser oder Raumschiffe bauen. Es ist zudem jederzeit möglich, die Gebilde wieder in ihre Bestandteile

[1]Eine Reihe weiterer Autoren (z.B. [Mason00]) gebraucht ebenfalls LEGO-Bausteine als Metapher. Es lässt sich jedoch nicht mehr genau feststellen, wer der Urheber ist.

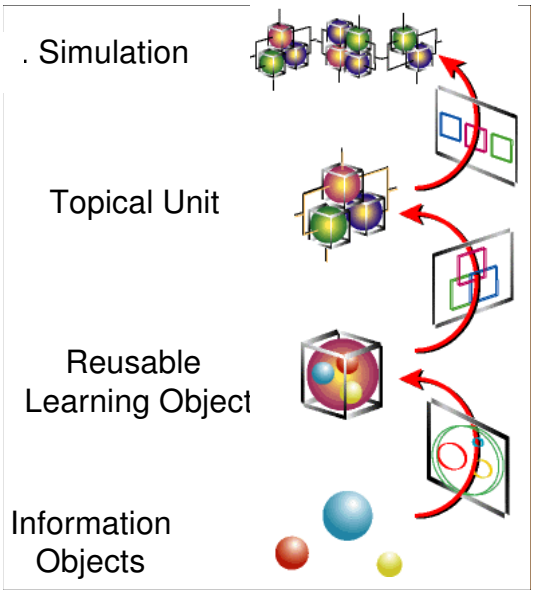


Abbildung 3.2: Lernobjekt-Hierarchie nach [Hodgins00, S. 28]

zu zerlegen und neu zu kombinieren. Genauso flexibel verhalten sich Lernobjekte, die beliebig kombinierbar bzw. zerlegbar sind und die Grundlage für ein personalisiertes Lernen bilden.

3.2.3 Lernobjekte nach Wiley

Für David A. Wiley hat das Internet die Kommunikation zwischen den Menschen verändert und wird auch die zukünftige Art des Lernens beeinflussen [Wiley02]. Daher ist es unvermeidlich, dass sich auch die Form des Lernmaterials anpassen wird. Er nennt als führende Technologie die Lernobjekte, da sie wiederverwendbar, generisch, anpassbar und skalierbar sind. Eine allgemeine Akzeptanz in Universitäten und Wirtschaft kann jedoch nur durch die Einigung auf verbindliche Standards erreicht werden.

> „Without such standards, universities, corporations, and other organizations around the world would have no way of assuring the interoperability of their instructional technologies, specifically their learning objects.“ [Wiley02, S. 4]

Wichtige Organisationen auf dem Gebiet der Lernobjekt-Standards sind z.B. LTSC[2], IMS[3], ADL[4] und ARIADNE[5]. Als Ausgangspunkt seiner eigenen Definition für Lernobjekte dient Wiley daher die des IEEE-LOM-Standards (siehe Abschnitt 4.3):

> „For this Standard, a learning object is defined as any entity — digital or non-digital — that may be used for learning, education or training.“ [IEE02a, S. 5]

Er stört sich an dem „non-digital“, da es nicht zu seiner Internet-Philosophie passt, und dem „may be used“, wodurch nicht wiederverwendbare Ressourcen einbezogen werden. Seine Umformulierung lautet nun:

> „... will define a learning object as ‘any digital resource that can be used to support learning.“ [Wiley02, S. 7]

Beim Umgang mit Lernobjekten müssen didaktische Theorien eine Rolle spielen, wenn sie die Lehre verbessern sollen. Für eine dynamisch automatische Komposition von Lernobjekten, die Voraussetzung für personalisiertes Lernen, müssen Metadaten über die angewandte

[2] `http://ieeeltsc.org` (29.10.05)
[3] `http://www.imsglobal.org` (29.10.05)
[4] `http://www.adlnet.org` (29.10.05)
[5] `http://www.ariadne-eu.org` (29.10.05)

Didaktik zur Verfügung gestellt werden. Nur auf diesem Weg kann eine sinnvolle Struktur mit Lernobjekten aufgebaut werden. Es reicht jedoch nicht aus, einfach Titel, Autoren und Versionen zu speichern, weil so lediglich eine einfache Suche nach Daten möglich ist. Leider finden die didaktischen Belange zu wenig Berücksichtigung bei den Standardisierungsprozessen.

Die von Hodgins angeführte LEGO-Metapher für Lernobjekte lehnt Wiley kategorisch ab, da sie falsche Assoziationen wecke und schlägt stattdessen das Atom als Alternative vor [Wiley99]. Wesentliche Eigenschaften des LEGO-Bausteins sind beliebige Kombinationsfähigkeit — jeder Baustein kann mit jedem zusammengesteckt werden—, keine Einschränkungen in der Struktur des Gebildes und eine „kinderleichte“ Bedienung. Übertragen auf das Lernobjekt, bedeutet dies eine inadäquate Vereinfachung, die schwerwiegende Konsequenzen hat. Lernobjekte müssen demnach lerntheoretisch neutral sein, weil sonst keine beliebige Kombinierung bzw. Strukturierung möglich ist. Hierdurch werden sie aber zu bloßen Informationsbehältern degradiert, worin Wiley ein ernstes Problem für die weiter Entwicklung der Lernobjekte sieht:

> „The learning object field must quickly make up its mind: are we in the information or the instruction business?“ [Wiley99, S. 3]

Die gleichen Probleme verursacht die Eigenschaft der „kinderleichten“ Bedienung, durch die jegliche Didaktik außen vor gelassen wird. Nur Experten mit einem lerntheoretischen Hintergrundwissen können sinnvolle Lernobjekte erstellen und deren geeignete Kombination sicherstellen.Die LEGO-Metapher hat sich somit selbständig gemacht und trägt maßgeblich zur Ungenauigkeit im Umgang mit Lernobjekten bei.

Mit Hilfe der Atom-Metapher versucht Wiley diesen Trend aufzuhalten. Ein Atom ist eine kleine Einheit, die mit anderen Atomen zu größeren Einheiten kombiniert werden kann. Soweit besteht eine Ähnlichkeit zu den LEGO-Steinen. Der wesentlich Unterschied liegt jedoch in den Bedingungen, unter denen der Aufbau geschehen kann, denn es gibt Einschränkungen. Atome können nicht willkürlich kombiniert werden, es gibt Vorgaben für die Struktur und der Umgang mit ihnen erfordert einiges an Wissen und Übung. Bei genauer Betrachtung offenbart die Metapher, dass Personen ohne didaktisches Wissen so wenig Lernobjekte sinnvoll kombinieren können, wie Personen ohne chemisches Wissen Kristalle aus Atomen wachsen lassen. Anstatt LEGO-Steine als Leitbild zu nehmen, sollten Lernobjekte lieber zu „Lernkristallen“ kombiniert werden.

3.2.4 Lernobjekte nach Downes

Stephen Downes möchte Lernobjekte nicht über ihre Eigenschaften, sondern über ihre Funktionen definieren, wie sie bestehende Probleme digitaler Lerneinheiten lösen können. Einer seiner Gründe für diese Betrachtungsweise ist die Uneinigkeit über eine genaue Definition im Lager der Lernobjekt-Forschung. Prinzipiell hat Downes nichts gegen die vorgestellten Modelle von Hodgins oder Wiley einzuwenden, da sie für sich betrachtet in ihrem Kontext sicher sinnvoll sind [Downes00b]. Doch die Unterschiede zwischen den Modellen sind zu groß und keines kann Allgemeingültigkeit für sich beanspruchen. Konsens herrscht nur darüber, dass die vom LOM-Standard vorgesehene Definition zu ungenau ist.

Als Rahmen für die Funktionen von Lernobjekten sieht Downes die **Lernobjekt-Wirtschaft** (*Learning Object Economy*). Hierbei handelt es sich um eine Verbindung von Netzwerken und Systemen, mit der Lehr- und Lernprozesse unterstützt werden. Innerhalb dieses Komplexes werden Lernobjekte erstellt und verteilt, wobei beliebige Materialien gemeint sind. Die genaue Interpretation, was ein Lernobjekt eigentlich ist, wird den Menschen überlassen. Egal ob nun Baustein oder Atom als Metapher herangezogen wird: Hauptsache eine gewisse Funktionalität ist vorhanden.

Ausgangspunkt für Downes Überlegungen ist das **Online-Lernen**. Durch das Internet eröffnen sich Möglichkeiten, die es vorher nicht gab. Als wesentliche Neuerungen nennt er den verbesserten Zugriff auf Materialien, bei dem unabhängig von Raum und Zeit mit eigener

Geschwindigkeit gelernt werden kann. Ein bedeutender Nachteil sind hingegen die mit der Produktion von E-Learning-Angeboten verbundenen Kosten. Auf herkömmliche Weise erstellt, sind sie sogar teurer als traditionelle Materialien wie z.B. Skripte oder Folien. Ursache dieser Diskrepanz ist die höhere Komplexität interaktiver Medien. Eine Lösung des Problems ist, wie in Abschnitt 3.1 bereits vorgeschlagen, die Wiederverwendung.

Es stellt sich die Frage, welche Form und Größe sich am besten für dieses Unterfangen eignet (mehr dazu auch in Abschnitt 3.3). Nach Downes sind es Kurse, die das aktuelle Angebot prägen. Kurse lassen sich aber nur schwer in verschiedene Veranstaltungen integrieren, weshalb Wiederverwendung bis jetzt wenig praktiziert wird. Durch eine Aufteilung von Kursen in Komponenten lässt sich jedoch auch diese Problematik entschärfen. Die einzelnen Teile können dann als Lernobjekte betrachtet werden. In diesem Zusammenhang tauchen auch Tausch und Zusammenstellung von Lernobjekten als weitere Fragestellungen auf.

Dritte können auf Komponenten nur zugreifen, wenn ihnen adäquate Suchmöglichkeiten angeboten werden. Zentrale Systeme wie z.B. Portale können diese Aufgabe gut übernehmen. Zum Nachteil gereicht diesem Ansatz die erschwerte Distribution über mehrere Portale. Was auf dem einen System angeboten wird, kann auf dem nächsten fehlen. Auch die Konsistenz der verschiedenen Materialien muss kritisch betrachtet werden. Unterschiedliche Formate, Präsentationen und Methoden verhindern die Kombination inhaltlich kohärenter Komponenten.

Aus den gestellten Anforderungen lassen sich leicht die Funktionen bestimmen, die aus einer Online-Ressource ein Lernobjekt machen. Eine Definition lässt sich aus den folgenden Eigenschaften ableiten:

- **Teilbar**[6]: Lernobjekte sind über das Internet erreichbar. Sie werden an einer zentralen Stelle erstellt und lassen sich beliebig in andere Kurse integrieren. Teilweise wird diese Eigenschaft unter „wiederverwendbar“ gefasst, was aber nicht das Gleiche bedeutet. „Teilbar“ schließt zusätzlich zur Wiederverwendung den Zugriff über Institutsgrenzen hinaus mit ein.

- **Digital**: Diese Eigenschaft ist Voraussetzung für das Online-Lernen. Physikalische Einheiten, wie z.B. Bücher, Mappen, Skripte, werden so von vornherein als Lernobjekte ausgeschlossen.

- **Modular**: Die Größe bestimmt die Wiederverwendbarkeit. Ein Lernobjekt ist nicht ein kompletter Kurs, aber ein Bestandteil von ihm. Daher müssen mehrere Lernobjekte wie Module zu größeren Einheiten zusammenführbar sein. Zudem soll ein Lernobjekt unabhängig von anderen sein.

- **Interoperabel**: Es muss möglich sein, Lernobjekte auch aus verschiedenen Quellen zu kombinieren. Für Personen, die einen Kurs zusammenstellen, darf es keine Rolle spielen, woher die einzelnen Komponenten stammen. Auch die Werkzeuge und die Infrastruktur sollten keinen Einschränkungen unterliegen.

- **Entdeckbar**: Für durchschnittliche Anwender/-innen muss es in vertretbarer Zeit möglich sein, die gewünschten Lernobjekte zu finden. Auf keinen Fall darf Spezialwissen vorausgesetzt werden.

Downes resümiert seine eigene Definition folgendermaßen:

> „In conclusion, learning objects are digital materials used to create online courses where these materials are sharable, modular, interoperable and discoverable.“ [Downes02]

[6] Downes benutzt das Wort „sharable“, das mehr auf die gemeinsame Nutzung abzielt als „teilbar“.

3.2.5 Lernobjekte nach Baumgartner

Peter Baumgartner führt ein recht einfaches Modell ein, bei dem der Begriff Lernobjekt, ähnlich wie bei *Cisco* (vgl. Abschnitt 3.2.1), als **Reusable Learning Object** (RLO) definiert ist. Zu Konfusion hinsichtlich der Verwendung des Begriffs Lernobjekt in dieser Arbeit kann es kommen, wenn Baumgartners Definition herangezogen wird:

> „Ein LO (Learning Object) ist die kleinste sinnvolle Lerneinheit, in die ein Online-Kurs zerlegt werden kann. Demnach kann ein LO entweder aus einem einzelnen Bild, einer Grafik, einem Text, einer Flash-Animation oder auch aus einer kurzen Anweisung mit einem definierten Lernziel und einem Test zur Lernerfolgskontrolle bestehen.“ [Baumgartner02b, S. 24]

Erst wenn ein Lernobjekt mit Metadaten angereichert wird, kann es wieder verwendet und zu höheren Kurseinheiten kombiniert werden. Diese eigenwillige Definition ist in Abbildung 3.3 zusammengefasst.

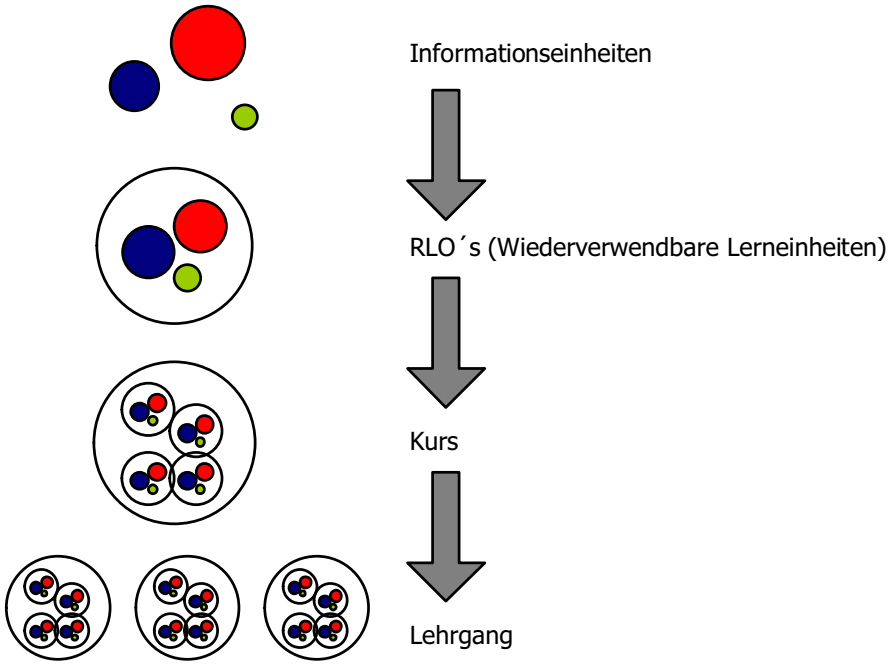


Abbildung 3.3: *Reusable Learning Objects* nach [Baumgartner02b, S. 24]

3.3 Granularität

Durch die Größe der Lernobjekte sind freilich Aspekte wie Entwicklung und Wiederverwendung tangiert. Sehr kleine Einheiten, oft auch Atome genannt, sind schnell realisiert und äußerst flexibel, jedoch ist ihre Verwendung mit Mehrarbeit verbunden. Eine simple Abbildung reicht beispielsweise meist nicht aus. Sie muss schon mit einem Text versehen werden, der sich in einen Gesamtkontext einbettet. Hingegen ist der Entwicklungsaufwand umfangreicher Einheiten enorm. Inhaltlich sind sie festgelegt und schwer an eigene Bedürfnisse anzupassen, sodass zwangsläufig Kompromisse bezüglich der Themen, Darstellung, Umfang, etc. eingegangen werden. Ein kompletter Kurs für ein Semester z.B. lässt kaum Spielräume für eigene Wünsche.

Doch wie lassen sich subjektive Größenangaben überhaupt definieren? Welche Größe ist gemeint, wenn der Umfang eines Lernobjekts angegeben ist? Bezieht sie sich auf den Inhalt oder das Medium? Mit Hilfe verschiedener Definitionen des Begriffs **Granularität** sollen diese

Fragen aufgeklärt werden. Der bereits für seine einfache Definition eines Lernobjekts gescholtene Standard LOM sieht vier Granularitätsstufen vor (der Attributname lautet *aggregation level*), die jeweils eine Aggregation der darunter liegenden darstellen. Angegeben durch arabische Zahlen (1–4), lässt diese Notation wahrlich Raum für Auslegungen. Als Beispiel für eine Interpretation soll die Festlegung der Firma *Autodesk*[7] dienen, die in Abbildung 3.4 illustriert ist. Es handelt sich um eine Definition mit fünf Ebenen, die nach [Duval03] auf die vier Granularitäten von LOM abgebildet werden:

1. **Data** oder **Raw Media Elements** sind die kleinsten Lernobjekte in diesem Modell. Es handelt sich um reine Daten, die nicht weiter zerlegt werden können, wie z.B. Absätze, Abbildungen und Animationen. Daten auf dieser Ebene können proprietär sein.

2. Die Ebene **Information Objects** ist unabhängig von bestimmten Medien. Lernobjekte dieser Größe enthalten soviel Informationen, dass sie oft wieder verwendet werden können.

3. Kohärente *Information Objects* zu einem Thema werden auf der Ebene **Application Objects** zusammengefasst. In der Regel ist dies die bevorzugte Größe zur Wiederverwendung von Lernobjekten.

4. Verschiedene Themenbereiche werden in den Ebenen **Aggregate Assemblies** oder **Collections** zusammengefasst. Um auf LOM abbildbar zu bleiben, haben sie beide die Größe 4.

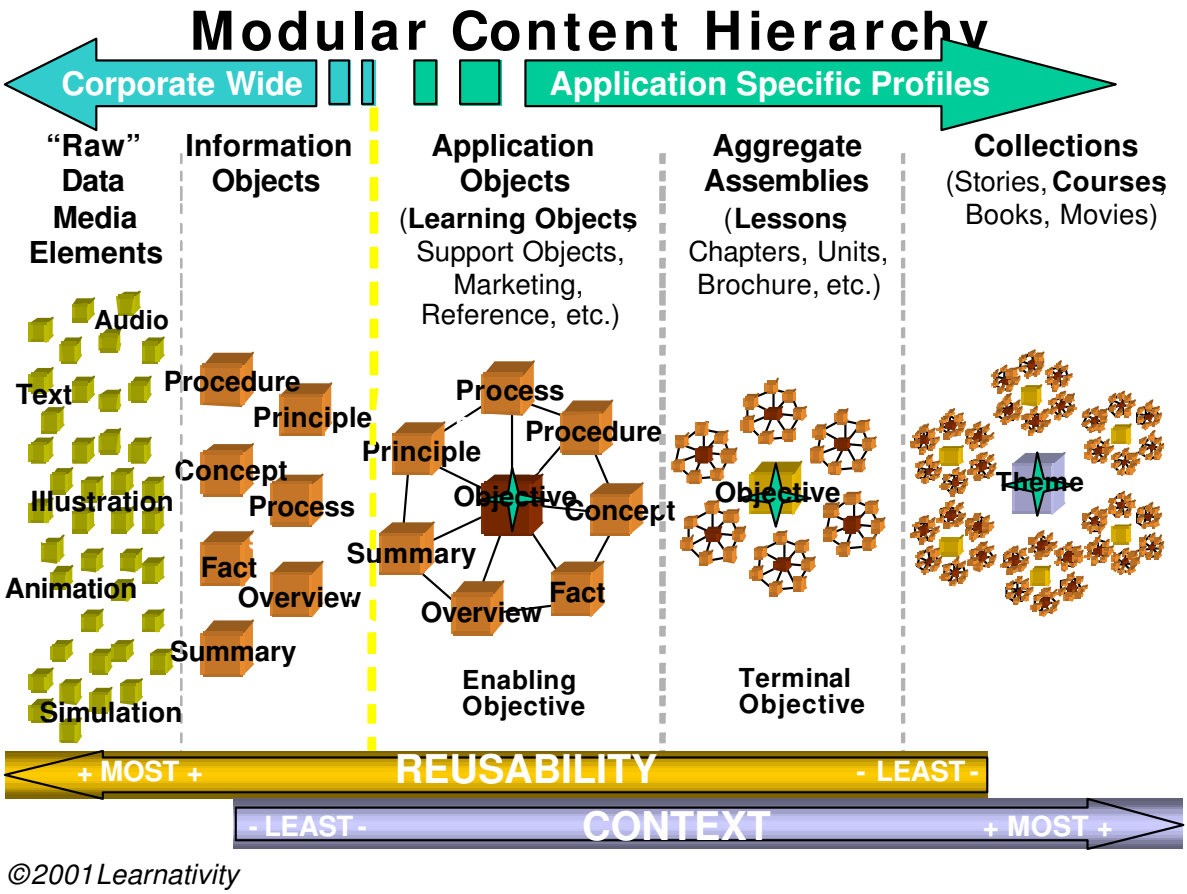


Abbildung 3.4: Lernobjekt-Hierarchie aus [Hodgins02, S. 78]

Eine alternative Definition der Granularität stellt Wiley in [Wiley00b] vor. Anstatt die Aggregationstiefe als Maß zu nehmen, kann auch die Komplexität von Inhalten herangezogen

[7] Autodesk ist der Arbeitgeber von Wayne Hodgins

werden. Auch wenn es eine Korrelation zwischen der Größe eines Lernobjekts und dessen Komplexität gibt, steht bei dieser Herangehensweise der Inhalt im Vordergrund. Folgende Erfahrung unterstützt diese Form von Granularität:

> „The optimal level of granularity must be determined for each project based on its individual goals. From the perspective of instructional developers, our experience is that it is most useful to move from the course level of granularity down to the concept level when designing, but not so far down as the individual media asset level. For our instructional needs, objects have the greatest potential for reuse when they center on a single, core concept.“ [South02, S. 6]

Eine praktische Anwendung, die sich intensiver mit der Aufteilung existierender Dokumente beschäftigt, ist *Slicing Books* [Dahn01; Dahn02]. Existierende Dokumente werden in semantische Einheiten aufgeteilt, die später individuell zusammengesetzt werden können.

3.4 Sequenzierung

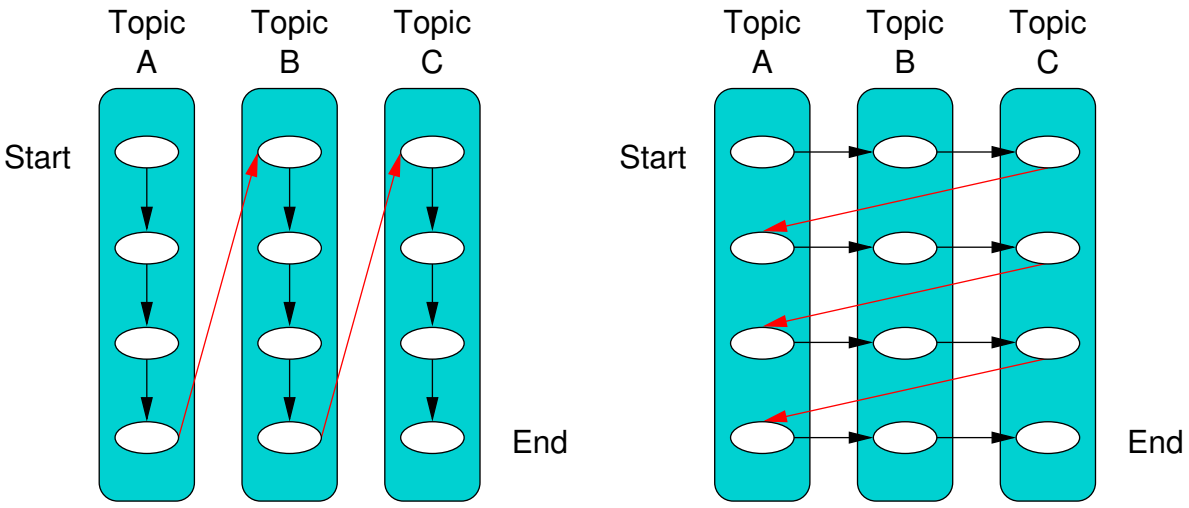
Eng verbunden mit der Granularität von Lernobjekten ist die Problemstellung der Reihenfolge, in der sie durchgegangen werden sollen. Diese Abfolge wird **Sequenz** genannt und ist ein didaktisches Grundproblem bei der Strukturierung von Lehr- und Lernmaterialien. Es lassen sich verschiedene Relationen zwischen den Lernobjekten bestimmen, aus denen sich die Sequenz ableiten lässt. Reigeluths „Evaluationstheorie“ [Reigeluth80; Reigeluth83; Reigeluth99] beinhaltet Vorschläge zur Sequenzierung von Lernmaterialien verschiedener Granularitäten. Ausschlaggebend für die Art der Sequenzierung sind Umfang und Zusammenhang der einzelnen Themen. Je enger die thematischen Verknüpfungen sind, desto stärker wirkt sich die Menge des Lernstoffes aus. Lernenden fällt es bei größeren Umfängen zunehmend schwerer, Schwächen und Ungereimtheiten in der Anordnung von Lernobjekten selbständig zu kompensieren. Hingegen erlaubt eine geringere Menge an Lernstoff, solche Mängel durch Erinnerung und Schlussfolgerungen auszugleichen.

Durch die Sequenzierung wird eine Relation zwischen den Lernobjekten festgelegt. Üblicherweise wird zwischen der chronologischen Abfolge (historische Sequenz), der Praxis üblichen Abfolge von Tätigkeiten (Prozeduren) und den Voraussetzungen bzw. dem Ausmaß der Komplexität unterschieden [Niegemann04]. Bei mehreren Themen wird zwischen der **linear-sukzessiven** und der **Spiral-Sequenzierung** Unterschieden. Abbildung 3.5 zeigt beide Sequenzmuster.

Bei der linear-sukzessiven Sequenzierung wird ein Thema intensiv durchgenommen, bevor es zum nächsten geht. Vorteil dieser Herangehensweise ist die Kontinuität, mit der ein Thema behandelt wird. Die Anordnung etwaiger Materialien fällt leichter und Lernende können sich auf ein Thema konzentrieren. Jedoch kann es bei einem Themenwechsel leicht passieren, dass zu spezielles Wissen verloren geht. Auch sind die Zusammenhänge zwischen den Themen nicht immer offensichtlich, da es durch die Separation nur wenig Anknüpfungspunkte gibt. Mit Überblicken, Rückblicken und Querverweisen kann diesem Problem allerdings in einem gewissen Maß entgegen gewirkt werden.

Bei der Spiral-Sequenzierung wird jedes Thema mehrmals durchlaufen. Erst werden die Grundlagen der einzelnen Themen behandelt, um sie jeweils soweit zu vertiefen, bis die erwünschte Kompetenzstufe erreicht ist. Im Gegensatz zur Sequenzierung gibt es viele Berührungspunkte zwischen den Themen, sodass sich die Zusammenhänge leichter erschließen. Nachteil dieser Herangehensweise sind die häufigen Unterbrechungen, die eine kontinuierliche Vertiefung erschweren.

Es gibt noch eine Reihe von Faktoren für die Sequenzierung, die stärker didaktisch ausgelegt sind. So werden z.B. die zu vermittelnden Kompetenzen unterschieden, ob sie mehr aufgaben- oder domänenorientiert sind [Reigeluth99]. Eine enge Verknüpfung von Reigeluths



(a) Linear-sukzessive Sequenzierung (b) Spiral-Sequenzierung

Abbildung 3.5: Linear-sukzessive Sequenzierung und Spiral-Sequenzierung nach [Reigeluth99, S. 432]

Arbeiten mit Lernobjekten findet sich in Wileys *Learning Object Design and Sequencing Theory* (LODAS) [Wiley00a] wieder.

Andere Kriterien für die Segmentierung sind unter anderen die Kapazität des menschlichen Arbeitsgedächtnisses [Case78; Case85]. Hierbei wird darauf geachtet, dass die „Informationseinheiten“ nur so groß sind, wie die Lernenden sie verarbeiten können.

3.5 IMS Content Packaging Specification

Die Spezifikation *IMS Content Packaging Information Model* (CP) [IMS04a] beschreibt Datenstrukturen, um die Zusammenarbeit von Internet-basierten Inhalten mit Autorensystemen, *Learning Management Systemen* (LMS) und Laufzeitumgebungen zu gewährleisten. Der Datenaustausch von E-Learning-Inhalten erfolgt über *Packages*, die im Wesentlichen aus einem *Manifest* und kohärenten Ressourcen, den *Physical Files*, bestehen. Weil das Manifest in XML (*Extensible Markup Language*) kodiert ist — es hat immer den Dateinamen `imsmanifest.xml` —, sind zusätzliche Kontrolldateien (DTD und XSD) für die Validierung enthalten. Im Inneren teilt sich das Manifest in die Bereiche *Metadata* (Metadaten), *Organizations* (Organisationen), *Resources* (Ressourcen) und *(sub)Manifest* auf. Abbildung 3.6 veranschaulicht den gesamten Aufbau eines *Packages*.

Physikalisch wird ein *Package* durch ein logisches Verzeichnis in einem Dateisystem repräsentiert, was für einen Datenaustausch, z.B. über das Internet, recht unhandlich ist. Anstatt das *Package* irgendwo in der Verzeichnisstruktur zu speichern, kann es inklusive aller Unterverzeichnisse auch in einer einzelnen Datei zusammengefasst werden, der *Package Interchange File* (PIF). Gültige Dateiendungen sind `.zip`, `.jar`, `.cab` und dergleichen. Wichtig ist lediglich, dass die interne Kodierung der Datei kompatibel zu RFC 1951 [Deutsch96] ist.

Als Austauschformat kommen jedoch nicht nur die *Package Interchange Files* in Frage, sondern auch Wechselmedien, wie z.B. die CD-ROM. In diesem Fall kann auf eine Komprimierung verzichtet werden. Nur Manifest sowie Kontrolldateien müssen im Wurzelverzeichnis liegen, um aus einem Datenträger ein gültiges *Package* zu machen.

Inhaltlich ist ein *Package* eine Einheit für wieder verwendbare E-Learning-Inhalte, z.B. als Teil eines Kurses, als eigenständiger Kurs oder als eine Sammlung verschiedener Kurse. Es muss möglich sein, ein *Package* mit anderen *Packages* zu kombinieren oder solche Verbünde in ihre Einzelteile zu zerlegen. Dies setzt voraus, dass ein *Package* für sich stehen kann und

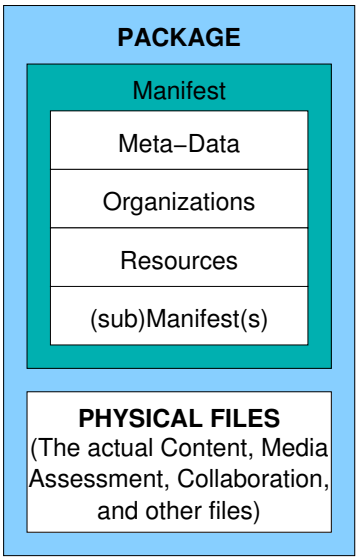


Abbildung 3.6: Die verschiedenen Bereiche innerhalb eines *Packages* [IMS04a]

keine Abhängigkeiten zu anderen Dateien besitzt. Wird es entpackt, müssen alle relevanten Daten für einen reibungslosen Ablauf vorhanden sein.

Der Aufbau des *Packages* und die Beschreibung der enthaltenen Ressourcen wird über das Manifest angegeben. Die interne Struktur ist stets statisch und kann in einer beliebigen Anzahl von Varianten angegeben sein. Durch die verschiedenen Funktionen und Größen eines *Packages* ist der semantische Gültigkeitsbereich der Strukturen nicht festgelegt, da es von einer kleinen Lerneinheit bis zur Kurssammlung alles beschreiben kann. Auf jeden Fall muss ein *Package* auf oberster Ebene, also direkt im logischen Verzeichnis, genau ein in XML kodiertes Manifest beinhalten, das als Beschreibung dient und *Top-level Manifest* genannt wird. Es kann auch (sub)Manifeste enthalten, die eine eigene semantische Ebene ausmachen. Enthält ein *Package* z.B. einen Kurs, dann können einzelne Kapitel oder Lerneinheiten durch (sub)Manifeste beschrieben werden. Dies ist sinnvoll, wenn der Inhalt so stark gekoppelt ist, dass er nicht für sich alleine stehen kann.

Eine sinnvolle aber freigestellte Herangehensweise ist, die Inhalte als Lernobjekte zu betrachten, sodass sie beliebig kombiniert, auseinander genommen und wieder verwendet werden können. Jedes Lernobjekt ist dann ein *Package* und besitzt sein eigenes Manifest. Bei einer Aggregation mehrerer Lernobjekte zu einem Kurs werden alle Lernobjekt-Manifeste in einem neuen Kurs-Manifest auf höherer Ebene zusammengeführt. Der Kurs kann wiederum mit anderen Kursen kombiniert werden, wobei das gleiche Procedere für die Manifeste zum Tragen kommt. Durch die Allgemeingültigkeit der *Packages* und die rekursive Definition des Manifests ist somit eine arbiträre Aufteilung von Inhalten möglich. Wie sie genau aussieht, liegt letztendlich bei den Autoren/-innen.

Die Ressourcen eines *Packages* sind beliebige Daten in Dateiform, wie z.B. HTML-Seiten, Texte, Sounds, Grafiken und Animationen. Sie sind entweder Bestandteil des *Packages* — sie liegen im Wurzel- bzw. einem Unterverzeichnis — oder werden über eine gültige URL referenziert, die eine Einbindung zur Laufzeit ermöglicht. Über eine definierte Auszeichnung können Ressourcen im Manifest deklariert werden, beispielsweise mit dem `<resource>`-Tag in XML-Notation. Da Ressourcen auch aus mehreren Dateien bestehen können, sollten integrierte Dateien explizit über das `<file>`-Tag bekannt gegeben werden. Im Falle einer URL-Adressierung ist diese Deklaration freilich nicht möglich, weil benötigte Dateien wiederum nur über URLs und nicht innerhalb des *Packages* erreichbar sind. Abhängigkeiten jeglicher Art zwischen Ressourcen können als *Dependency* (`<dependency>`-Tag) angegeben werden.

Handelt es sich bei den Ressourcen z.B. um Abschnitte und Unterabschnitte eines inhaltlich zusammenhängendes Textes, befinden sie sich strukturell auf verschiedenen Ebenen, die nicht

durch die einfache Deklaration im Manifest beschrieben sind. Mit Hilfe einer eigenen Datenstruktur, der Organisation (`<organization>`-Tag), werden solche hierarchischen Relationen angezeigt. Sie ist als Baum realisiert, dessen Knoten als *Items* (`<item>`-Tag) bezeichnet werden und Referenzen auf Ressourcen oder Submanifeste beinhalten. Die Submanifestverweise sind unerlässlich für die beschriebene Zusammenführung verschiedener *Packages*. Weil es möglich ist, Manifeste mit mehreren Organisationen zu erstellen, können die gleichen Ressourcen und Submanifeste innerhalb eines *Packages* verschieden angeordnet werden.

Nicht nur das Manifest kann mit Metadaten versehen werden, sondern auch Ressourcen, Dateien, Organisationen und Knoten. Die Beschreibung ist im IMS-eigenen Metadatenformat [IMS01] anzugeben und steht im `<meta-data>`-Tag. Abbildung 3.7 verdeutlicht nochmal die gesamte Datenstruktur eines Manifests.

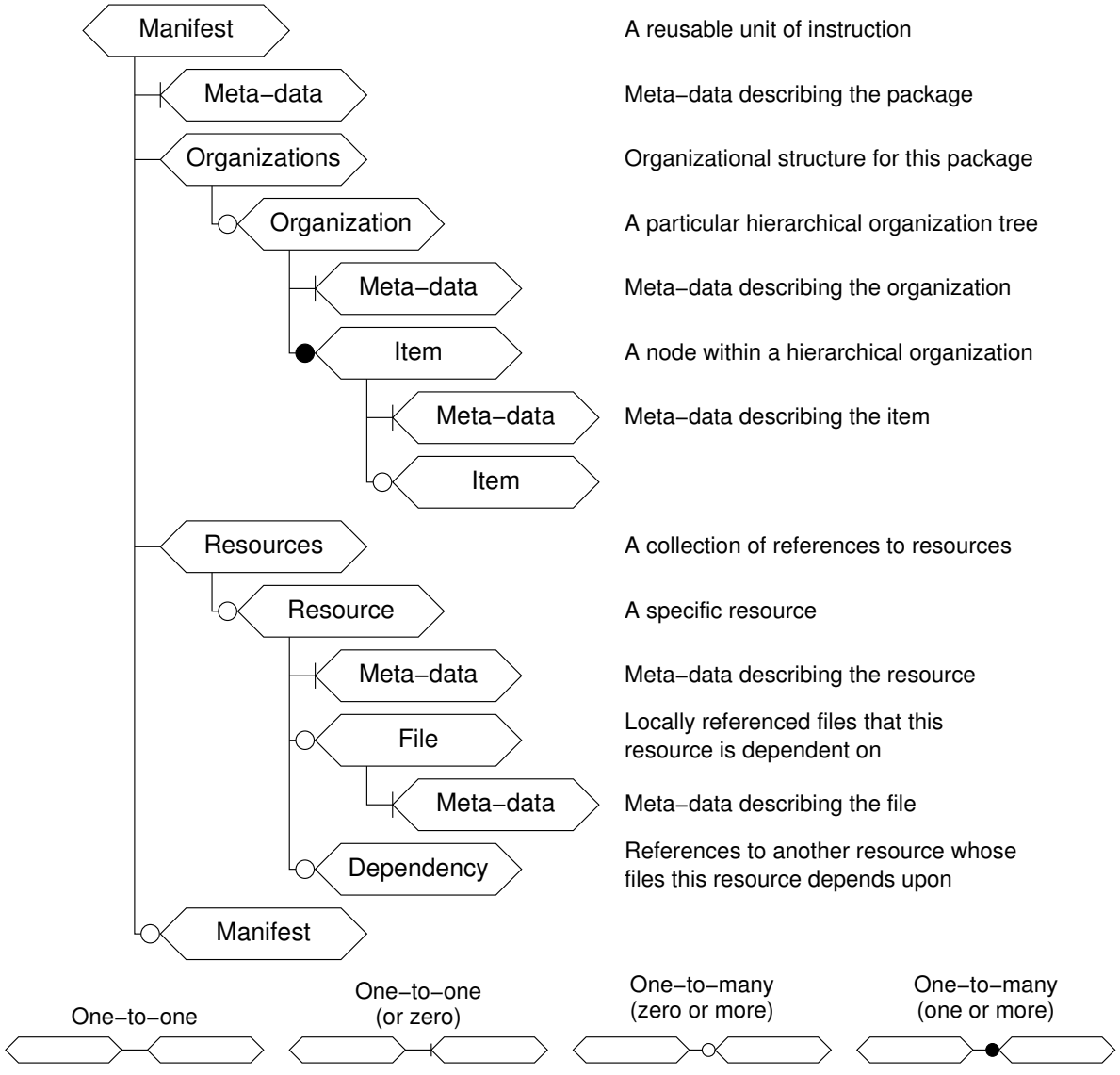


Abbildung 3.7: Datenstruktur eines Manifests [IMS04a]

Die Spezifikation des IMS CP ist für eigene Erweiterungen ausgelegt. Es ist erwünscht, dass Implementationen die Basisstrukturen der Organisationen erweitern und neue Typen für Ressourcen einführen. Bewährte Ergänzungen könnten letztendlich in zukünftige Versionen der Spezifikation aufgenommen werden.

Es werden aber nicht nur Datenstrukturen von der Spezifikation vorgegeben, sondern auch Algorithmen. Für diese Arbeit ist die Zusammenführung von Organisationen aus Manifesten und Submanifesten von Bedeutung, weshalb diesbezüglichen Verfahren besonderes Augenmerk

gilt. Immer wenn ein Knoten ein Submanifest anstatt einer Ressource referenziert, müssen die Strukturen beider Organisationen kombiniert werden. Je nach Anordnung und Aufbau der beiden Manifeste treten verschiedene Fälle auf, die differenziert behandelt werden. Für eine erfolgreiche Zusammenführung ist mindestens eine Organisation im Submanifest Voraussetzung. Ansonsten gilt die Referenz auf das Submanifest als nicht gesetzt. Stehen stattdessen mehrere Organisationen zur Auswahl, dann wird entweder die explizit deklarierte oder, wenn diese Auszeichnung fehlt, die erste verwendet. Mit der Zusammenführung selbst verhält es sich folgendermaßen. Die Organisation wird direkt mit dem referenzierenden Knoten vereint, wobei Konflikte durch gleiche Attribute — es sei z.B. ein Titel angeführt, welcher von Knoten und Organisation bestimmt werden kann — stets zugunsten des Knoten gelöst werden. Abbildung 3.8 veranschaulicht diesen Vorgang für einen Knoten ohne Subknoten.

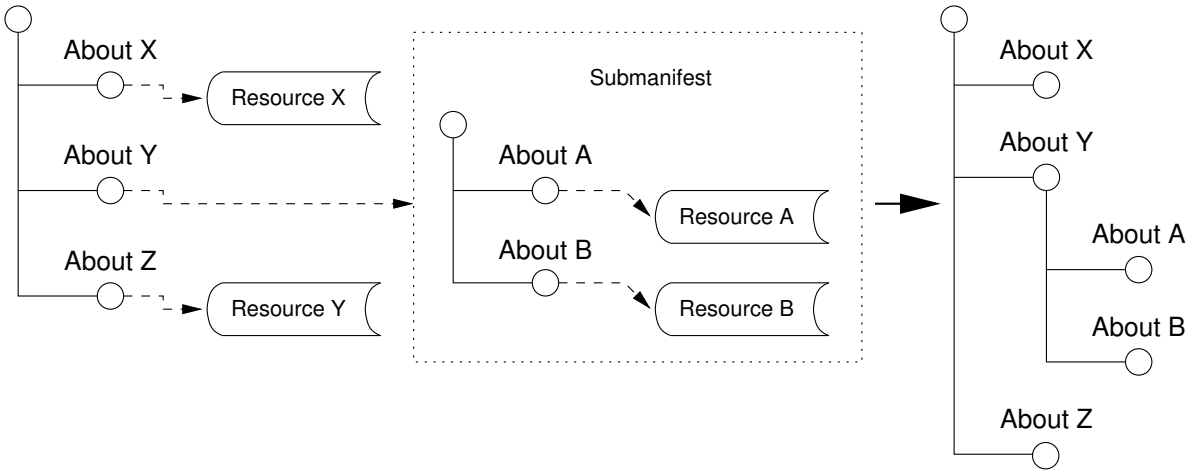


Abbildung 3.8: Einfache Auflösung von Referenzen

Das gleiche Beispiel für einen Knoten mit Subknoten ist in Abbildung 3.9 zu sehen.

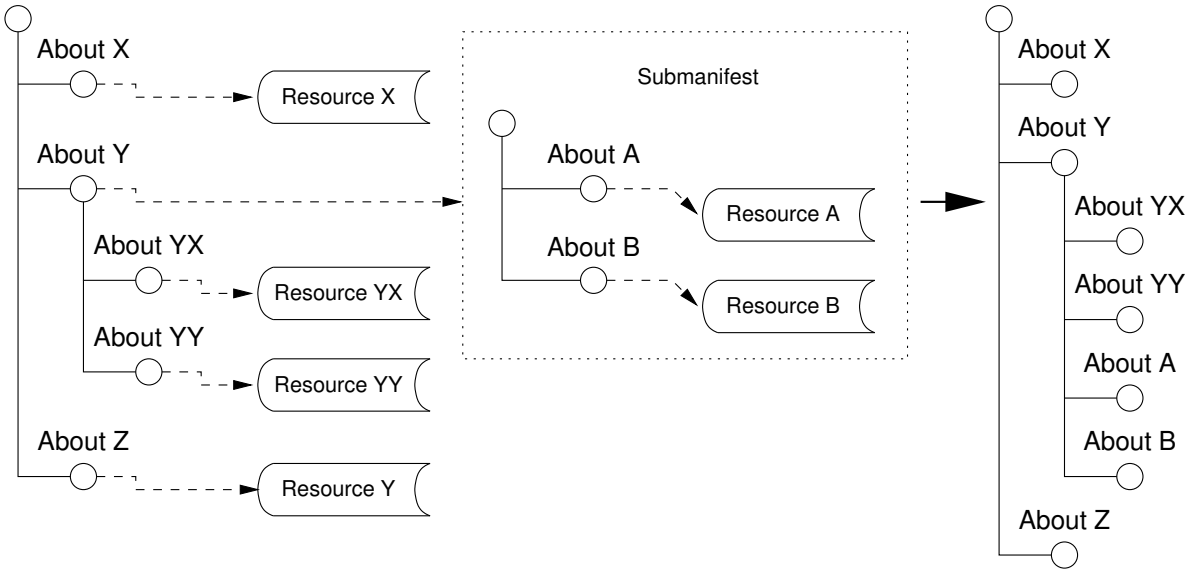


Abbildung 3.9: Auflösung von Referenzen mit Subknoten

Beispiele für den Einsatz von IMS CP finden sich in [Low02].

3.6 Sharable Content Object Reference Model

Beim *Sharable Content Object Reference Model* (SCORM) von *Advanced Distributed Learning* (ADL) [Dodd04c] handelt es sich um eine Spezifikation, die auf den Standards von AICC, IMS und IEEE aufbaut. SCORM definiert ein *Content Aggregation Model* und eine *Runtime Environment* für Lerneinheiten, die über das Internet veröffentlicht werden sollen. Das *Content Aggregation Model* beschreibt, wie kleinere Dateneinheiten zu Lerneinheiten, Kapiteln oder ganzen Kursen zusammengestellt werden. Im wesentlichen basiert es auf dem *Content Packaging* von IMS. Die *Runtime Environment* gibt einen Rahmen vor, wie die erzeugten Lerninhalte durch eine Lernplattform (siehe Kapitel 6) verwaltet und gesteuert werden. Abbildung 3.10 zeigt eine schematische Darstellung aus der Spezifikation.

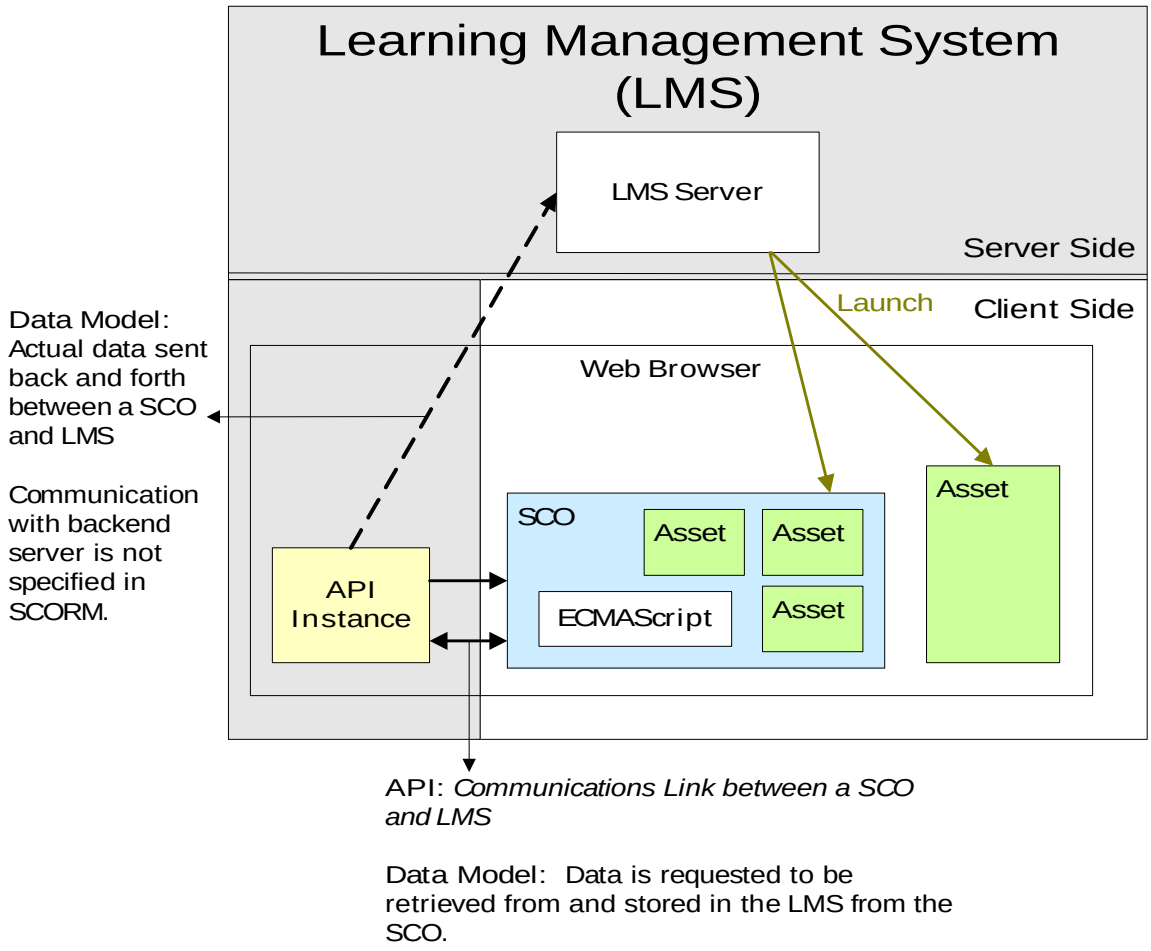


Abbildung 3.10: *Runtime Environment* aus [Dodd04b, S. 1-8]

Über einen Startmechanismus (*Launch*) werden die Web-basierten Inhalte aufgerufen und die Initialisierung durchgeführt. Die eigentliche Kommunikation erfolgt über eine API, die in der Abbildung beispielhaft über JavaScript angesprochen wird. Über definierte Datenstrukturen (*DataModel*) informiert der Client den *Server* (LMS) über den aktuellen Status.

Die gesamte Steuerung der Kommunikation auf Seiten des Clients läuft im Inhalt selbst ab und nicht im *Content Package*. Aus diesem Grund soll an dieser Stelle nicht weiter auf dieses Thema eingegangen werden. Mit dem Einsatz von SCORM setzen sich [Letts02; Shackelford02] auseinander und Beispiele für den Einsatz finden sich in [Newman03].

3.7 Formate

Die Inhalte von Lernobjekten müssen in einer Form vorliegen, dass die gesamte Infrastruktur, von den Autorensystemen bis hin zu den Lernplattformen, sie verarbeiten kann. Zu den gängigen Formaten für E-Learning Inhalte dürfen freilich HTML, PDF, *Microsoft Word/PowerPoint* oder *Macromedia Flash* zählen. Systeme mit einem solchen Repertoire sind hinsichtlich der Kompatibilität auf der sicheren Seite. Dennoch reichen diese überwiegend proprietären Formate für die Ansprüche des E-Learnings nicht aus. Die Gründe hierfür sind vielfältig. Zur Erstellung sowie Anzeige werden oft spezielle Programme benötigt, die teilweise horrende Kosten verursachen und oft nicht für alle Betriebssysteme zur Verfügung stehen. Folglich kommt es zu Einschränkungen, die zu Lasten der Lernenden gehen. Nicht selten müssen private Ressourcen eingesetzt werden, sodass hier ein geringer finanzieller Spielraum gewährt ist.

Ein anderes Problem ist die mangelnde Flexibilität der angesprochenen Formate. Unzulängliche Strukturinformationen erschweren die technische Umsetzung von Lernobjekten. Eigenschaften aus Abschnitt 3.2, wie z.B. Modularität, verschiedene Granularitäten, individuelle Lernpfade etc., sind nicht immer umsetzbar. Hinzu kommt die Vermengung von Darstellung und Inhalt. Beispielsweise kennt das Dateiformat von *MS Word* keine strukturelle Aufteilung eines Dokuments durch Kapitel, Abschnitt oder Absatz. Stattdessen hat eine Kapitelüberschrift eine bestimmte Formatierung, die sich in Schriftart und Größe ausdrückt. Ein anderes Textfragment mit zufällig gleichen Eigenschaften ist demnach nicht unterscheidbar. Für eine automatische Verarbeitung kann diese Verquickung unüberwindbare Probleme bereiten.

Abhilfe schafft wieder die **Extensible Markup Language** (XML), die bereits Gegenstand vieler Untersuchungen und Projekte war [Roisin98; Freitag02b; Teege02; Belqamsi02; Wollowski02; Balbieris02]. XML ist inzwischen so weit akzeptiert und bekannt, dass in dieser Arbeit nicht weiter auf technische Details eingegangen werden soll. Interessante Einführungen zu diesem Thema finden sich in [Ammelburger03; Mintert02; Ray01].

Ein Grund für die breite Akzeptanz von XML ist die einfache Definition eigener bzw. Anpassung existierender XML-Applikationen. Durch automatische Validierungsverfahren, die entweder durch *Document Type Definitions* (DTD) oder *XML Schemas* (XSD) [Binstock02; Vlist02] gesteuert werden, kann die Gültigkeit von XML-Dokumenten überprüft werden. Die DTDs stammen noch von der *Standard Generalized Markup Language* (SGML) [Goldfarb91] ab, aus der XML einst hervorging. Aufgrund verschiedener Mängel wurde vom W3C *XML-Schema* als Nachfolger der DTD eingeführt, dessen genaue Vorteile in [Hansch02] nachgelesen werden können.

Die Darstellung von XML-Dokumenten wird von außen gesteuert, wodurch verschiedene Layouts und Dateiformate unterstützt werden. Abhängig von der jeweiligen Anwendung, können XML-Dokumente z.B. in einem Webbrowser direkt dargestellt oder in einem zusätzlichen Verarbeitungsschritt umgewandelt werden. Die Steuerung erfolgt durch verschiedene Mechanismen. Mit Hilfe der *Extensible Stylesheet Language* (XSL) können Struktur und Darstellung gleichermaßen beeinflusst werden. Sie besteht aus drei Teilen: *XSL Transformations* (XSLT) zur Umwandlung der Struktur [Tidwell01], *XML Path Language* (XPath) zur Adressierung von XML-Fragmenten [Simpson02] und den *XSL Formatting Objects* (XSL-FO) zur Festlegung von Darstellungsregeln [Pawson02]. Auch die von HTML bekannten *Cascading Style Sheets* (CSS) [Meyer02] oder die *Document Style Semantics and Specification Language* (DSSSL) für SGML [Farreres03] lassen sich zur Formatierung von XML-Dokumenten nutzen.

Für die Kodierung von E-Learning-Inhalten gibt es bereits eine Reihe von XML-Applikationen, wie z.B. DocBook [Walsh02a; Walsh02b], OmDoc [Kohlhase00; Kohlhase02] und LMML [Freitag02a].

Kapitel 4

Metadaten

Die Lernobjekte aus Kapitel 3 können, wenn sie flexibel genug realisiert wurden, in vielen verschiedenen Kontexten eingesetzt werden. Für einen sinnvollen Einsatz durch Dritte ist jedoch eine präzise Identifizierung der Lern- und Lehrmaterialien unerlässlich. Völlig inakzeptabel ist, bei jeder Recherche nach geeigneten Materialien über die Inhalte selbst zu suchen. Bei Angeboten mit mehreren hundert oder sogar tausenden Lernobjekten steht der Nutzen in keiner Relation zum Aufwand. Auch mit etwaigen maschinellen Hilfen, wie z.B. Volltextsuche, ist diesem Problem nicht beizukommen. Aus diesem Grund sollten Lernobjekte mit zusätzlichen Beschreibungen versehen werden. Da sie nicht direkt zum Inhalte gehören, werden sie **Metadaten** genannt. Im Umgang mit Metadaten ist der Mensch vertraut und sie sind aus dem alltäglichen Leben nicht wegzudenken. Beispielsweise steht auf einer Milchverpackung, dass es sich um Milch handelt, wie hoch der Fettgehalt ist und das Mindesthaltbarkeitsdatum. Niemand würde auf die Idee kommen, die Verpackung zu öffnen, um den Inhalt festzustellen. Die Metadaten reichen als Information für eine Auswahl aus.

Ähnlich verhält es sich bei E-Learning-Angeboten, wenn auch ganz andere Metadaten benötigt werden. Doch welche sind es? Zusätzliche Daten über den/die Autoren/-in und das Institut können helfen, die Qualität der Inhalte abzuschätzen. Ein Lebenslauf oder unterrichtete Lehrveranstaltungen helfen, die Reputation zu beurteilen [Kortzfleisch99, 54]. Aber auch technische Voraussetzungen, rechtliche Rahmenbedingungen, benötigte Vorkenntnisse, didaktische Methoden, etc. spielen eine Rolle. Nach [Gill98] gibt es drei wesentliche Eigenschaften, die bei allen Informationsobjekten — einschließlich Lernobjekten — durch Metadaten beschrieben werden können: Inhalt, Kontext und Struktur. Bedauerlicherweise herrscht Uneinigkeit darüber, welche einzelnen Attribute der Metadaten relevant sind, besonders zwischen den Lagern Technik und Didaktik. Dennoch hat es wenig Sinn, auf individuelle Lösungen zu setzen, weil dies die Vorteile der Metadaten kompensiert.

Metadaten brauchen einen gemeinsamen logischen Raum, der Strukturen und Datentypen vorgibt [Simon01]. Dieser lässt sich nur über Standards bilden, die formal und informell Vorgaben machen. Welchen Umfang solch eine Spezifikation haben sollte, wird z.B in [Griffin97; Ahronheim98] beschrieben, wobei es hauptsächlich um Strukturelemente, Datenelemente, Methoden zur Manipulation, Verfügbarkeit von Werkzeugen und Verantwortlichkeiten geht.

> „Mit Hilfe von E-Learning-Standards lässt sich also die Recherchierbarkeit, Austauschbarkeit und Wieder- bzw. Weiterverwendung von Lernressourcen gewährleisten, indem sie mit Metadaten nach einem einheitlichen Muster in maschinenlesbarer Form beschrieben werden. Diese Standards sind eine zwingende Voraussetzung für die Interoperabilität von Lernressourcen und Lernsystemen, da sie die Schnittstellen und Referenzmodelle für den E-Learning-Bereich definieren.“ [Niegemann04, S. 270]

Im Zusammenhang mit Metadaten tauchen immer wieder Begriffe auf, die teilweise verschieden interpretiert werden. Um Konfusionen zu vermeiden, werden kurz die wichtigsten

benannt und für diese Arbeit definiert. Ein **Metadaten-Element** ist als abstrakte Einheit zu verstehen, die zur Strukturierung — umfasst die erlaubten Unterelemente und ihre Kardinalität — und Notation von Daten — gibt die Syntax gültiger Literale (Zeichenketten) vor, auch **Wertebereich** genannt — dient. Das Metadaten-Element „Ersteller“ könnte z.B. ein Unterelement „Namen“ haben, dessen Wertebereich die Namen aller Mitarbeiter/-innen eines Unternehmens ist. Mehrere kohärente Metadaten-Elemente werden zu einem **Metadaten-Schema** zusammengefasst und mit einem Namen versehen. Wenn nun für Metadaten bekannt ist, nach welchem Schema sie sich richten, können z.B. Validierungen und Interpretationen durchgeführt werden.

Bei der anschließenden Beschreibung der heute relevanten Metadaten-Standards wird sich zeigen, dass die Spezifikationen entweder einen technischen oder didaktischen Schwerpunkt haben. In der praktischen Anwendung führen solche Spezialisierungen jedoch zu Problemen. Kein Satz an domänenspezifischen Metadaten-Elementen wird ausreichen, um alle Aspekte des Lernobjekts zu beschreiben. Abhilfe versprechen **Application Profiles** für Metadaten, bei denen verschiedene Metadaten-Schemata zusammengeführt und angepasst werden.

> „An application profile is an assemblage of metadata elements selected from one or more metadata schemas and combined in a compound schema. Application profiles provide the means to express principles of modularity and extensibility. The purpose of an application profile is to adapt or combine existing schemas into a package that is tailored to the functional requirements of a particular application, while retaining interoperability with the original base schemas.“ [Duval02]

Ein *Application Profile* bedient sich mehrerer Mechanismen, bei denen die semantische Interoperabilität bestehen bleibt. Durch **Namespaces** lassen sich Datenelemente der einzelnen Schemata direkt adressieren, sodass auch gleichnamige Attribute unterscheidbar sind. Sie erlauben ebenfalls die Erweiterung mit eigenen Elementen. Eine Applikation, die nur standardisierte Schemata verarbeitet, wird zwar die selbst definierten Elemente nicht adäquat interpretieren können, aber ohne weiteres die bekannten. Hierdurch bleiben die Metadaten für die einzelnen Domänen immer gültig. Handelt es sich bei *Namespaces* noch um eine echte Erweiterung, sind die restlichen Mechanismen Einschränkungen der gültigen Möglichkeiten. In einem *Application Profile* können die Kardinalitäten strenger ausgelegt werden, als sie ursprünglich waren. Dann ist z.B. ein optionales Element als ein obligatorisches umdefiniert. Ähnlich restriktiv können die gültigen Wertebereiche verkleinert werden. Mit der Festlegung von Beziehungen zwischen Datenelementen und ihren Wertebereichen können bestimmte Strukturen zugelassen werden. So kann z.B. die Existenz eines Datenelements ein anderes ausschließen, oder ein bestimmter Wert den Wertebereich eines anderen Datenelements einschränken. Mehr zu dem Thema *Application Profile* mit Beispielen findet sich in [Heery00; Dekkers01; Baker01].

Ein weiterer Ansatz zur Handhabung verschiedener Metadaten-Standards sind die **Metadata Registries**. Hierbei handelt es sich um Systeme, die eine Reihe bestimmter Dienstleistungen anbieten. Der Funktionsumfang kann dabei stark variieren. Nach [Baker03, S. 12] können folgende Fokusse für *Metadata Registries* ausgemacht werden:

- Individueller Standard: Beinhaltet allgemeine Informationen über einen bestimmten Metadaten-Standard und Richtlinien zur Verwendung.
- Erweiterungen: Gibt an, wie ein Standard von Gruppen erweitert bzw. in eine andere Sprache übersetzt wurde.
- Data Warehouse: Speichert Definitionen von Datenelementen und Typen mit dem Ziel, verschiedene Datenbanken an einer zentralen Stelle zu halten.
- Domäne: Benennt interessante Metadaten-Schemata für Domänen, wie z.B. E-Learning, Kultur und Wirtschaft.

- Funktionen: Ordnen Metadaten-Schemata bestimmten Funktionen zu, wie z.B. Suchen, Rechteverwaltung oder Leistungsbewertung.
- Unternehmen: Zugriff auf Taxonomien von Unternehmen oder anderen Gruppen.
- Anwendung: Bietet Schemata in verschiedenen Formaten und Syntaxen für spezielle Anwendungen an.
- Konvertierung: Übersetzt Metadaten in das Format eines anderen Metadaten-Standards.

Ein interessantes Konzept zur Konvertierung auch inkompatibler Standards findet sich in [Blanchi01]. Entwürfe von Systemen werden in [Heery03; Heery02; Nagamori01] beschrieben.

Die technische Umsetzung von Metadaten lässt sich in drei Schichten einteilen (vgl. Abbildung 4.1).

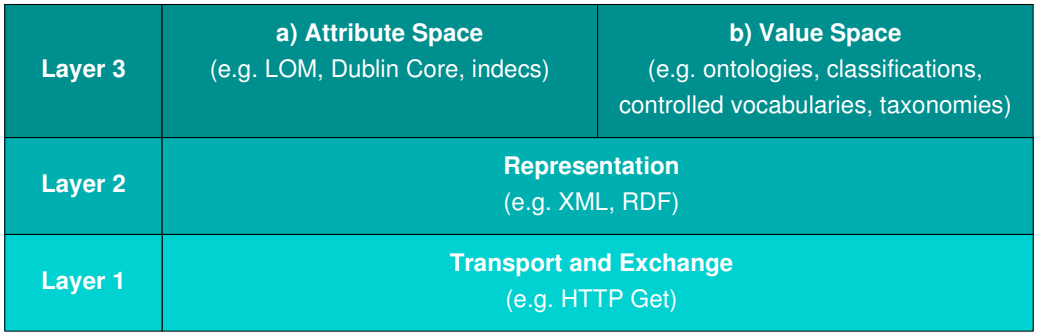


Abbildung 4.1: Schichten für Metadaten-Umsetzung nach [Baker03, S. 6]

Schicht 3 beinhaltet Strukturen und Wertebereiche, die von verschiedenen Organisationen empfohlen bzw. standardisiert wurden. In dieser Arbeit wird besonders auf die Standards von **Dublin Core** (DC, siehe Abschnitt 4.2) und **Learning Objects Metadata** (LOM, siehe Abschnitt 4.3) eingegangen. Wegen ihrer Verbreitung und der Relevanz für die Praxis sind sie in die engere Wahl gekommen. Schicht 2 umfasst die möglichen Kodierungen, die auch maschinell verarbeitbar sind, wie z.B. das allgemeine Rahmenwerk für Metadaten Namens **Resource Description Framework** (RDF). Für DC und LOM gibt es jeweils entsprechende Abbildungen. XML wurde bereits in Abschnitt 3.7 als Format für Inhalte vorgestellt, eignet sich aber auch hervorragend für die Kodierung komplexer Datenstrukturen wie Metadaten. Andere Kodierungen werden aufgrund ihrer geringen Bedeutung nicht betrachtet. Auch der Transport von Metadaten, in der Schicht 1 beschrieben, wird nicht weiter behandelt, weil etablierte Infrastrukturen, wie z.B. das Protokoll HTTP, genutzt werden.

4.1 Resource Description Framework

Das *Resource Description Framework* (RDF) dient zur Auszeichnung von Web-Ressourcen mit zusätzlichen Metadaten. Dies sind z.B. Angaben über den/die Autor/-in, das Datum der Änderung oder Lizenzbedingungen. Als Web-Ressource kann jedes Objekt bezeichnet werden, welches sich über das Web identifizieren lässt. Auf eine direkte Verfügbarkeit über das Internet kommt es dabei nicht an. Gegenstände eines Web-Shops können z.B. mit Preisen, Verfügbarkeit etc. ausgezeichnet sein, obwohl sie freilich nur über den Postweg zu den Kunden gelangen.

RDF dient in erster Linie der maschinellen Verarbeitung und ist weniger für eine direkte Verwendung durch Menschen gedacht. Mit seiner Hilfe sollen Metadaten direkt von einer Anwendung zur nächsten übertragen werden, ohne dass es zu einem Informationsverlust kommt. Weil RDF eine offene, allgemein gehaltene Spezifikation ist, können die Autoren/-innen von Web-Ressourcen auf eine Reihe von RDF-Libraries und Werkzeugen zurückgreifen. Die folgende Beschreibung beruht auf dem *RDF Primer* von [Manola03].

Die Identifikation von Ressourcen erfolgt über die *Uniform Resource Identifier* (URI) [Berners-Lee98]. Im Gegensatz zu den *Uniform Resource Locators* (URL) [Berners-Lee94], die im *World Wide Web* (WWW) den Zugriff auf physikalisch existierende Ressourcen regeln, sind die URIs allgemeiner definiert[1]. Mit ihrer Hilfe können alle Dinge bezeichnet werden, die Bestandteil einer Modellierung sind und bilden somit die Grundlage für die Auszeichnung von Web-Ressource mit Metadaten. Um den Inhalt einer Ressource genauer differenzieren zu können, nutzt RDF die *Fragment Identifier*, die stets durch ein `#` an eine URI angehängt werden. Dieses Konstrukt nennt sich dann *URI Reference* (URIref), dargestellt im folgenden Beispiel:

`http://www.upb.de/index.html` $\Longrightarrow$ eine HTML-Seite
`http://www.upb.de/index.html#section2` $\Longrightarrow$ ein Abschnitt

Alle zusätzlichen Metadaten einer Ressource werden durch *Properties* (Eigenschaften) angegeben, die jeweils aus einem Namen und zugehörigen Werten bestehen. Beispielsweise kann die HTML-Seite mit dem Ersteller „Michael Bungenstock“ versehen werden, was in natürlicher Sprache wie folgt beschrieben werden kann:

`http://www.upb.de/index.html` hat ein **Ersteller** mit dem Wert **Michael Bungenstock**

Die wichtigen Bestandteile dieser Aussage sind durch Formatierung hervorgehoben. Für eine genauere Strukturbeschreibung besitzt RDF eine eigene Terminologie, die in Tabelle 4.1 erläutert ist.

Wort	**Begriff**	**Beschreibung**
`http://www.upb.de/index.html`	Subjekt	Ressource
Ersteller	Prädikat	Eigenschaft
Michael Bungenstock	Objekt	Wert

Tabelle 4.1: RDF-Terminologie

Nun dient RDF in erster Linie der maschinellen Verarbeitung, sodass eine Sprache benötigt wird, mit deren Hilfe Subjekt, Prädikat und Objekt eindeutig angegeben werden können. Da für den Zugriff auf Ressourcen bzw. das Subjekt bereits die URIs eingeführt wurden, liegt es nahe, dies auch für die beiden anderen Begriffe zu tun.

`http://www.upb.de/index.html` $\Longrightarrow$ Subjekt
`http://purl.org/dc/elements/1.1/creator` $\Longrightarrow$ Prädikat
`http://www.getlab.de/staffid/83427` $\Longrightarrow$ Objekt

Die URI für das Subjekt ist klar, da sie gleichzeitig als URL für das Dokument interpretiert werden kann. Bei der Wahl des Prädikats ist nicht auf den ersten Blick ersichtlich, warum diese kryptische URI eines anderen Anbieters eingesetzt wird. Würde ein einfaches Literal wie z.B. `creator` nicht ausreichen? Das Problem ist die Eindeutigkeit, die bei einem einfachen Wort wie `creator` sicherlich nicht gewährleistet wäre. Abhängig vom Kontext kann es zu verschiedenen Interpretationen kommen, da kein eindeutiges Konzept identifiziert wird. Bei der Beispiel-URI ist das anders. Eine mit *Dublin Core* (siehe Abschnitt 4.2) vertraute Person kann sofort die Bedeutung des Prädikats erkennen. Aus dem Beispiel lässt sich ebenfalls die Schlussfolgerung ziehen, dass bekannte URIs eigenen vorzuziehen sind. Was nützen die schönsten URIs, wenn weder Mensch noch Maschine sie sinnvoll interpretieren können? Beim Objekt ist dies freilich anders, weil die Werte zu speziell sind, als dass sie allgemein definiert werden könnten. Je nach Komplexität des Wertes, können entweder Ressourcen oder Literale als Objekt dienen. Im Beispiel werden Daten über Mitarbeiter/-innen als Ressourcen modelliert, um mehr Informationen über die betreffende Person bereitstellen zu können, wie z.B. Abteilung, Telefonnummer und E-Mail-Adresse.

[1] Alle URLs sind eine echte Teilmenge der URIs

Diese Relationen lassen sich auch in Form von Graphen illustrieren: Subjekt und Objekt sind Knoten, die durch das Prädikat, dargestellt als Pfeil, verbunden sind. Abbildung 4.2 zeigt das gleiche Beispiel in grafischer Form.

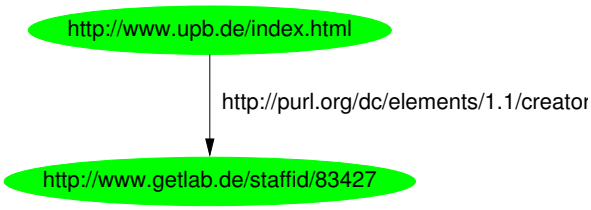


Abbildung 4.2: RDF-Graph für den Mitarbeiter Michael Bungenstock

Nun soll das Beispiel durch zwei weitere Aussagen über Erstellungsdatum und Sprache erweitert werden:

`http://www.upb.de/index.html` hat ein **Erstellungsdatum** mit dem Wert **4.11.2002**
`http://www.upb.de/index.html` hat eine **Sprache** mit dem Wert **deutsch**

Diese Werte werden als Zeichenketten angegeben, weil sie keine weitere relevante Struktur aufweisen und die Interpretation des Wertes von der Applikation abhängt. Es sei ausdrücklich darauf hingewiesen, dass Literale lediglich für Objekte genutzt werden und nie für Subjekte bzw. Prädikate. Durch den Zeichensatz Unicode [Aliprand03] können Werte in vielen Sprachen direkt dargestellt werden. Bei der grafischen Darstellung werden die Literale in Kästen gezeichnet, wie in Abbildung 4.3 zu sehen ist.

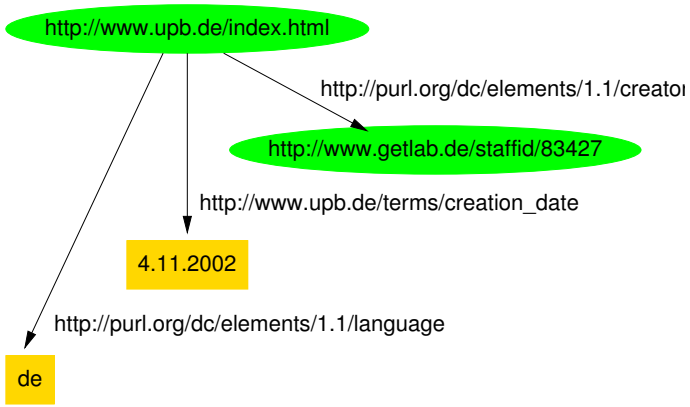


Abbildung 4.3: RDF-Graph mit Ressourcen und Literalen

Um die maschinelle Verarbeitung dennoch stabiler zu halten, können in RDF *Typed Literals* definiert werden. Das sind Zeichenketten mit einem bestimmten Typ — wie von Programmiersprachen oder Datenbanken her bekannt —, der Wertebereich, Ordnung, und Operationen festlegt. Anstatt das Erstellungsdatum “4.11.2002” als bloße Aneinanderreihung von Zeichen zu deuten, lassen sich Tag, Monat und Jahr ablesen. Ohne eine Interpretation des Literals würden syntaktische und semantische Fehler (z.B. ein falsches Datum ”4.13.2002” oder ”.11.2002”) nicht frühzeitig erkannt werden. Im Gegensatz zu den genannten Programmiersprachen und Datenbanken bietet RDF keine eingebauten Datentypen an, weshalb sie extern definiert und über Datentypen-URIs referenziert werden. Ein Vorteil dieses Ansatzes ist die Flexibilität beim Umgang mit Datentypen aus verschiedenen Quellen, da für eine direkte Darstellung der Werte keine Umwandlung auf vordefinierte Datentypen nötig ist. Bei der Definition orientiert sich RDF konzeptionell an einer Typdefinition der XML-Schemata-Spezifikation [Biron01], wonach ein Datentyp wie folgt beschrieben ist:

- eine definierte Menge von Werten (Wertebereich genannt),

- eine definierte Menge von Zeichenketten (lexikalischer Bereich genannt)
- und eine Abbildung vom lexikalischen Bereich in den Wertebereich.

Die Zeichenkette "4.11.2002" des Beispiels kann folglich bei einem Datumstypen auf das Datum 4. April 2002 abgebildet werden. Vorstellbar sind auch die Schreibweisen "2002-11-4", "11/4/2002", etc. als gültige Literale für dasselbe Datum, wenn es eine entsprechende Abbildung gibt.

Die Notation von typisierten Literalen in RDF setzt sich aus einer Zeichenkette und einer URIref für den Datentyp zusammen. Als Separator dient die Zeichenfolge `^^`, sodass auch in der grafischen Repräsentation lediglich ein Knoten benötigt wird. Abbildung 4.4 zeigt das Beispiel aus Abbildung 4.3 erweitert um Typinformationen.

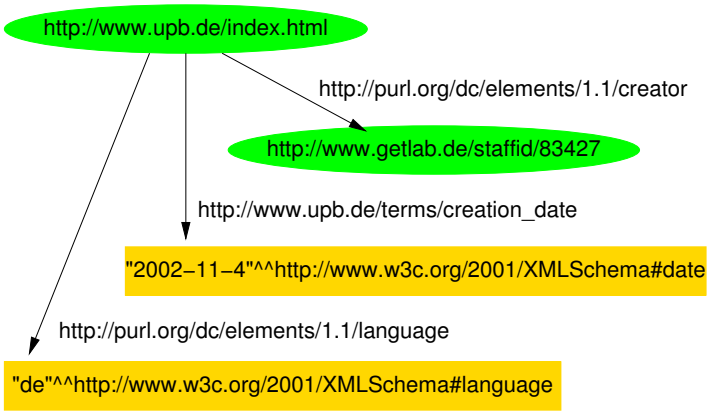


Abbildung 4.4: RDF-Graph mit typisierten Literalen

Gelegentlich ist die grafische Darstellung von RDF-Daten ungeeignet und eine schriftliche vorzuziehen. Anstatt der natürlichsprachlichen Schreibweise sieht RDF *Tripel* vor, die aus Subjekt, Prädikat und Objekt bestehen. URIrefs werden in spitzen Klammern geschrieben, Literale in Anführungszeichen und jedes Tripel entspricht einem Pfeil im Graphen mit Start- sowie Endpunkt.

```
<http://www.upb.de/index.html> <http://purl.org/dc/elements/1.1/creator> ↩
     <http://www.getlab.de/staffid/83427>
<http://www.upb.de/index.html> <http://www.upb.de/terms/creation_date> ↩
     "2002-11-4"^^<http://www.w3c.org/2001/XMLSchema#date>
<http://www.upb.de/index.html> <http://purl.org/dc/elements/1.1/language> ↩
     "de"^^<http://www.w3c.org/2001/XMLSchema#language>
```

Augenfällig ist der Platzbedarf dieser Darstellung, bei der die Tripel nicht in eine Zeile passen (↩ zeigt den Fortgang des Tripels ohne Zeilenumbruch). Dies liegt einerseits an der Redundanz, da ein Subjekt bzw. Objekt in jedem Tripel explizit angegeben werden muss, andererseits an den langen URIrefs. Letztere lassen sich durch **qualifizierte Namen** (QNames) in eine kürzere Form überführen. Ein QName besteht aus einem *Präfix*, das für eine *Namespace URI* steht, einem Doppelpunkt als Separator und einem *lokalen Bezeichner*. Für das Beispiel werden nun folgende QName-Präfixe definiert:

Präfix `dc`, Namespace `http://purl.org/dc/elements/1.1/`
Präfix `xsd`, Namespace `http://www.w3.org/2001/XMLSchema#`
Präfix `upb`, Namespace `http://www.upb.de/`
Präfix `upbt`, Namespace `http://www.upb.de/terms/`
Präfix `get`, Namespace `http://www.getlab.de/staffid/`

Daraus folgt für die Tripel des Beispiels:

```
upb:index.html  dc:creator          get:83427
upb:index.html  upbt:creation_date  "2002-11-4"^^xsd:date
upb:index.html  dc:language         "de"^^xsd:language
```

Bei RDF werden Mengen zusammenhängender URIrefs als **Vokabular** bezeichnet. Wenn sich alle URIrefs ein gemeinsames Präfix teilen, dann lassen sich mit QNames effizient die Elemente der Menge bestimmen. Teilweise werden die Präfixe der QNames selbst als Bezeichner für Vokabulare genutzt, z.B. `dc`-Vokabular für die Menge der URIrefs von Dublin Core.

Abschließend soll noch auf komplexere Datenstrukturen eingegangen werden, wie sie in praktischen Anwendungen vorkommen. Als Ausgangspunkt dient die Ressource `get:83427` des vorangegangen Beispiels, die den Mitarbeiter Michael Bungenstock identifiziert. Eine Reihe von Metadaten bieten sich an, die mit dieser Person in Verbindung stehen, wie z.B. die Adresse. In Tripel-Form, das Prädikat wird als ungetyptes Literal geschrieben, sieht diese Eigenschaft wie folgt aus:

```
get:83427  upbt:address  "Pohlweg 47-49, 33098 Paderborn"
```

Eine Analyse des Literals fällt der vielen Daten wegen — Straße, Hausnummer, Postleitzahl und Ort müssen unterschieden werden — recht aufwendig aus. Daher ist es sinnvoll, die Adresse in ihre Bestandteile zu zerlegen. Die Konsequenz dieser Umstrukturierung ist eine neue Ressource *address* für jeweils jede/-n Mitarbeiter/-in mit vier neuen Prädikaten. Zur Identifikation dieses Objekts wird der *Namespace* `http://www.getlab.de/addrid/` mit dem Präfix `getaddr` definiert. Die Tripel lauten:

```
get:83427       upbt:address     getaddr:83427
getaddr:83427   upbt:street      "Pohlweg"
getaddr:83427   upbt:street_no   "47-49"
getaddr:83427   upbt:city        "Paderborn"
getaddr:83427   upbt:zip         "33098"
```

Abbildung 4.5(a) zeigt den passenden Graphen.

(a) Eigene Ressource für Adressen

(b) Anonyme Ressource

Abbildung 4.5: Strukturierte Adresse

Auf diese Weise müssen bei komplexen Datenstrukturen eine Reihe von zusätzlichen URIrefs erzeugt werden, die aber niemals benötigt werden. Es gibt keinen vernünftigen Grund, die Ressource `getaddr:83427` jemals direkt aufzurufen. Sie ist nur ein Konstrukt, um eine Adresse adäquat zu modellieren. Abbildung 4.5(b) zeigt eine alternative Darstellung, bei der die zusätzliche URIref ausgelassen wird.

Knoten ohne URIref werden als **leere Knoten** bezeichnet und referenzieren **anonyme Ressourcen**. Sie können beliebig oft in Graphen eingesetzt werden, sind aber nicht unterscheidbar. Diese Eigenschaft führt bei der Tripel-Darstellung unweigerlich zu Problemen. Da Tripel nur indirekt über Bezeichner verbunden sind, muss ein Symbol für anonyme Ressourcen definiert sein. Wenn keine zusätzlichen Regeln bei der Verarbeitung gelten, wie z.B. die Reihenfolge oder eine Trennung durch Leerzeilen, dann ist die korrekte Interpretation unmöglich. Aus diesem Grund definiert RDF für die Tripel-Schreibweise leere Knoten mit Bezeichner. Die Notation ist ein Unterstrich, gefolgt von einem Doppelpunkt und einem Namen, wie z.B. `_:bungeaddr`. Daraus ergibt sich für die Tripel des Beispiels:

```
get:83427    upbt:address    _:bungeaddr
_:bungeaddr  upbt:street     "Pohlweg"
_:bungeaddr  upbt:street_no  "47-49"
_:bungeaddr  upbt:city       "Paderborn"
_:bungeaddr  upbt:zip        "33098"
```

Neben den Graphen und Tripel hat RDF noch eine Syntax für XML, die **RDF/XML** [Beckett03] genannt wird. Sie ist die normative Syntax für RDF, auf die sich alle Implementationen stützen. Für die direkte Betrachtung durch den Menschen sind die XML-Konstrukte jedoch wenig geeignet und bringen konzeptionell nichts Neues, weshalb eine ausführliche Behandlung an dieser Stelle nicht erforderlich ist. Weitere Informationen finden sich in z.B. [Powers03; Hjelm01]. Die Realisierung von *Application Profiles* mit Hilfe von RDF wird in [Hunter01] beschrieben und Anfragen auf Metadaten im RDF-Format in [Nejdl02].

4.2 Dublin Core Metadata

Bei den *Dublin Core* Metadaten (DC) [Dub99] handelt es sich um einen Konsens über Kernelemente, dessen Ursprung und Namensgebung in einem Workshop im März 1995 in Dublin (Ohio) liegt. Zur Beschreibung von Ressourcen sind fünfzehn verschiedene Elemente definiert, deren Semantik durch internationale, interdisziplinäre Gruppen von Bibliothekaren, Informatikern und Angehörigen verwandter Wissenschaften definiert wurde. Die Werte für jedes Element können frei gewählt werden, jedoch gibt es teilweise Empfehlungen für den Einsatz definierter Vokabulare.

1. **Title**: Ist der Name, unter dem eine Ressource bekannt ist.
2. **Creator**: Sind beispielsweise Personen, Organisationen oder Dienste, die sich verantwortlich für die Erstellung zeigen.
3. **Subject**: Umfasst das Thema der Ressource. Meist werden Schlüsselwörter oder Codes für Kategorien eingesetzt.
4. **Description**: Beschreibt den Inhalt. Hierzu gehören z.B. Zusammenfassungen oder Inhaltsverzeichnisse.
5. **Publisher**: Sind beispielsweise Personen, Organisationen oder Dienste, die sich verantwortlich für die Veröffentlichung zeigen.
6. **Contributor**: Sind beispielsweise Personen, Organisationen oder Dienste, die in irgendeiner Form beteiligt sind.
7. **Date**: Ist ein Datum eines Ereignisses im Lebenszyklus einer Ressource. Empfehlenswert ist das Erstellungsdatum.
8. **Type**: Beschreibt den Typ oder die Art einer Ressource.

9. **Format**: Gibt die physikalische Form einer Ressource an.
10. **Identifier**: Ist eine eindeutige Referenz in einem gegebenen Kontext.
11. **Source**: Referenziert eine andere Ressource, von der die beschriebene abgeleitet wurde.
12. **Language**: Identifiziert die Sprache des Inhalts.
13. **Relation**: Beschreibt Beziehungen jeglicher Art zu anderen Ressourcen.
14. **Coverage**: Gibt den räumlichen und zeitlichen Rahmen vor.
15. **Rights**: Informiert über Eigentums- und Nutzungsrechte.

Wie das Wort *Core* im Namen bereits andeutet, handelt es sich um einen Satz fundamentaler Metadaten, die sich ebenfalls in koexistierenden Spezifikationen wiederfinden. DC ist somit prädestiniert für *Application Profiles* (siehe Empfehlungen für den Einsatz in [CEN03]). Für eine Identifizierung der Elemente sieht z.B. [Bearman99] Qualifizierer vor. Eine andere Möglichkeit ist, *Dublin Core* als eine Art Sprache auszulegen, in der bestimmte Klassen von Begriffen für Ressource definiert sind [Baker00]. In diesem Fall sind es Nomen für die Elemente und Adjektive als Kennzeichen, die zusammen mit den Nomen in einer einfachen Syntax, wie z.B. RDF (siehe Abschnitt 4.1), notiert werden. Aus dieser einfachen Definition lässt sich schließen, dass *Dublin Core* einfach einzusetzen, jedoch weniger für komplexe Beziehungen oder Konzepte geeignet ist. In [CEN03] stehen Empfehlungen für den Einsatz von DC *Application Profiles* (DCAP) und ein Erfahrungsbericht findet sich z.B. in [Friesen02].

Die Bedeutung dieses Standards für diese Arbeit zeigt sich in den Elementabbildungen anderer relevanter Metadaten-Standards wie z.B. LOM (nächster Abschnitt). In der Spezifikation von LOM steht genau beschrieben, wie einzelne Einträge auf die fünfzehn Elemente von DC abgebildet werden.

4.3 Learning Object Metadata

Der Standard *Learning Object Metadata* (LOM) [IEE02a] spezifiziert die Semantik und Syntax von Metadaten für Lernobjekte (siehe Kapitel 3). Hauptsächlich geht es um Datenstrukturen, mit denen die Eigenschaften von Lernobjekten vollständig beschrieben werden. Neben dem Konzept, das hier vorgestellt wird, gibt es noch Spezifikationen für XML- [IEE02b] sowie *RDF-Bindings* [IEE; Nilsson03] (siehe Abschnitt 4.1), die jedoch nur für technische Umsetzungen interessant sind und an dieser Stelle nicht weiter behandelt werden.

Für LOM sind Lernobjekte alle digitalen oder nicht digitalen Einheiten, die zum Lernen bzw. Lehren genutzt werden. Mit Hilfe der Metadaten können sie charakterisiert werden, was ihre Suche, Evaluation, Beschaffung und Nutzung vereinfacht. Ein Problem ist die große Menge an potentiellen Attributen, die sich als Metadaten eignen. Um Struktur in die Datenmenge zu bringen, erfolgt eine Aufteilung in Kategorien, von denen es bei LOM insgesamt neun gibt: **General**, **Life Cycle**, **Meta-Metadata**, **Technical**,**Educational**, **Rights**, **Relation**, **Annotation** und **Classification**. Der Aufbau der Attribute kann recht unterschiedlich sein. Es gibt einfache Werte, zusammengesetzte Felder, Listen und hierarchische Datenstrukturen, die alle als obligatorisch oder optional markiert werden können. Da auch die besten Strukturen nur dann helfen, wenn die Anzahl der jeweiligen Attribute handhabbar ist, wurde bei der Festlegung von LOM darauf geachtet, ihre Menge so gering wie möglich zu halten.

In der Datenstruktur von LOM bilden die neun Kategorien die oberste Ebene mit folgender Bedeutung:

a) *General*: Umfasst generelle Daten, die sich auf das gesamte Lernobjekt beziehen.

b) *Life Cycle*: Gruppiert alle Eigenschaften, die mit dem Verlauf bzw. dem aktuellen Stand des Lernobjekts in Verbindung stehen.

c) *Meta-Metadata*: Beschreibt zusätzliche Daten über die Metadaten selbst.

d) *Technical*: Gibt die technischen Voraussetzungen an, die für einen reibungslosen Einsatz notwendig sind.

e) *Educational*: Charakterisiert die didaktischen Eigenschaften der Inhalte.

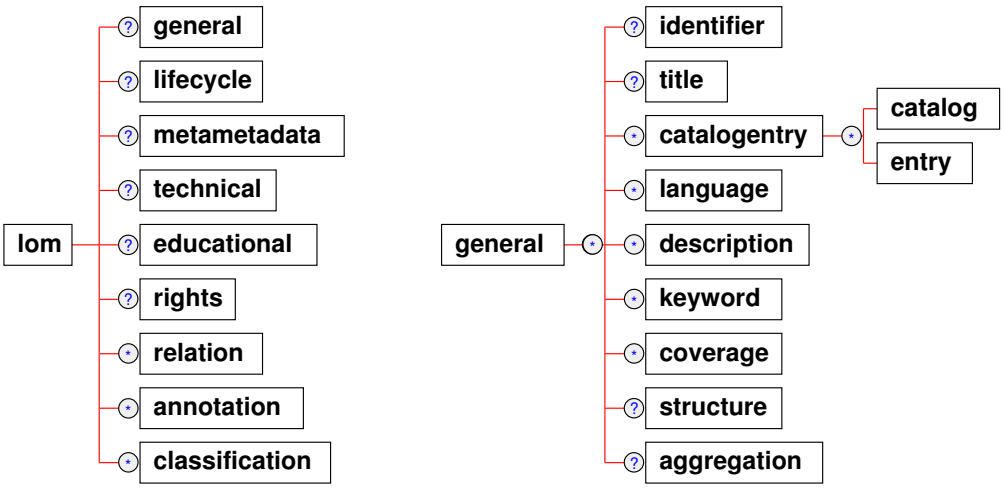
f) *Rights*: Ermöglicht Nutzungsklauseln und Copyright-Vermerke.

g) *Relation*: Beschreibt Beziehungen und Abhängigkeiten zu anderen Lernobjekten.

h) *Annotation*: Speichert alle Kommentare inklusive Verfasser/-in über den eigentlichen Einsatz.

i) *Classification* Erlaubt die Nutzung eigener Klassifikationssysteme.

Der Aufbau der Struktur lässt sich auch grafisch darstellen. Abbildung 4.6(a) zeigt die neun Kategorien mit den möglichen Kardinalitäten (im folgenden durch n abgekürzt). Das Symbol ? steht für ein optionales Element ($n \in \{0, 1\}$). Mit * wird eine beliebige Kardinalität angegeben ($n \in \{0, 1, 2, 3, ...\}$).



(a) Die oberste Ebene (b) Die Kategorie *General*

Abbildung 4.6: Beispiele für Strukturen in LOM (von [IEE02a] abgeleitet)

Die komplette Struktur von LOM ist zu umfangreich, als dass sie hier vollständig erörtert werden könnte. Abbildung 4.6(b) zeigt exemplarisch die Kategorie *General*. Unterhalb des Elements **general** kommen die eigentlichen Datenelemente, die verschieden komplex sind. Es gibt einfache Datenelemente für Werte und zusammengesetzte für komplexere Strukturen. In Anlehnung an einen Baum in der Graphentheorie spricht die LOM-Spezifikation bei einfachen Datenelementen von Blättern, bei zusammengesetzten von Knoten. Abbildung 4.7 zeigt die verschiedenen Ebenen innerhalb des Baums.

Jeder Knoten, wie z.B. das Datenelement **catalogentry**, dient nur zur Strukturierung und kann niemals einen eigenen Wert besitzen. Neben dem Namen sieht die Spezifikation für jedes Datenelement noch eine Kardinalität und — wenn es sich um Blätter mit einer Kardinalität $n > 0$ handelt — eine Ordnung vor. Trifft die letztgenannte Bedingung zu, wird auch von einer **Liste** gesprochen. Bei einer **geordneten Liste** muss die Reihenfolge der Blätter berücksichtigt werden, wohingegen sie bei einer **ungeordneten Liste** keine Bedeutung hat. Einfache Datenelemente sind zudem noch mit einem **Typ** versehen, der den Wertebereich der erlaubten Werte angibt. Vorgesehen sind die Typen **LangString**, **DateTime**, **Duration**, **Vocabulary**, **CharacterString** und **Undefined**.

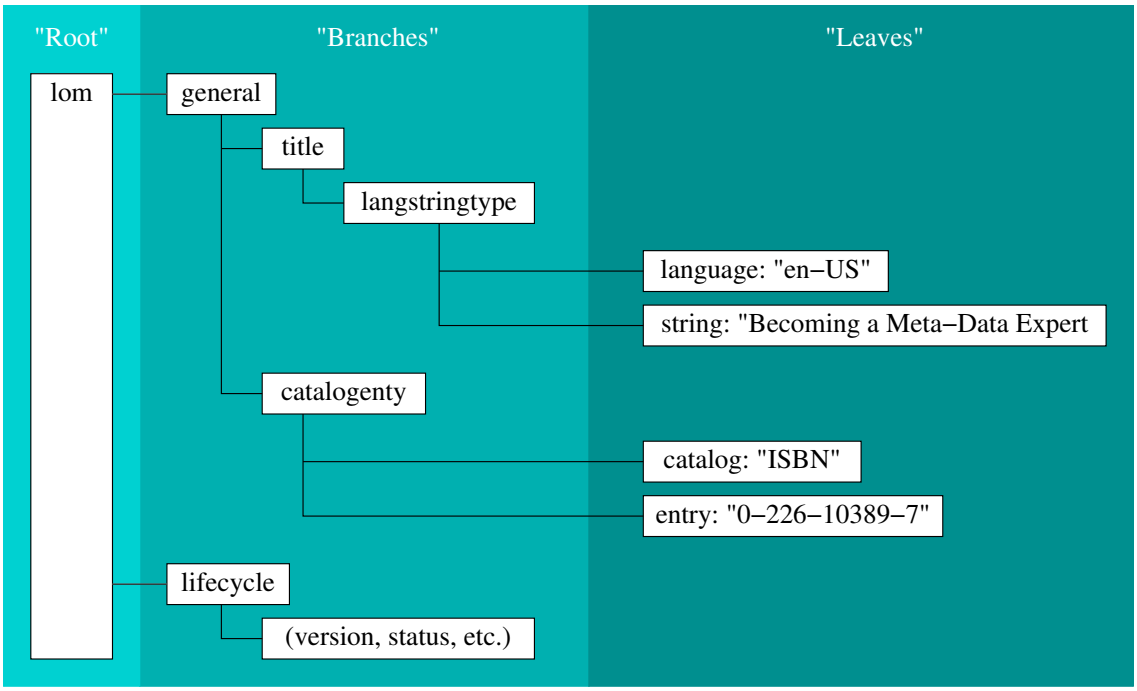


Abbildung 4.7: Aufbau von LOM als Baum nach [IMS03b]

Ein *LangString* wird für unterschiedliche lokale Werte eingesetzt, dient also der Internationalisierung (I18N[2]) von Metadaten. Er besteht aus einem Sprach-Code nach ISO-639 [Int88; Int98] in Kombination mit einem optionalen Länder-Code [Int97] und dem eigentlichen Literal in der Codierung *Universal Multiple-Octet Coded Character Set* (UCS) [Int02]. Für jede unterstütze Sprache wird ein *LangString* angelegt, was z.B. in XML folgendermaßen aussieht:

```
<desciption>
  <langstring xml:lang="en">Leaving Certificate</langstring>
  <langstring xml:lang="en–GBR">A–Level</langstring>
  <langstring xml:lang="fr–FRA">Baccalauréat</langstring>
  <langstring xml:lang="de–DEU">Abitur</langstring>
  <langstring xml:lang="de–AUT">Matura</langstring>
</description>
```

Mit *DateTime* werden Datums- und Zeitangaben ab dem Jahr 1 gemacht. Daten ab dem 15. Oktober 1582 gelten nach dem gregorianischen Kalender, davor liegende nach dem julianischen. Die Werte werden als Zeichenketten angegeben, deren Wertebereich in ISO-8601 [Int00] definiert ist. Gültige Literale für Datums- und Zeitangaben sind z.B. `1984`, `1857-06-06` und `1969-07-21T03:53`. Zeitspannen basieren auf der Norm ISO-8601 und werden ebenfalls als Zeichenketten mit dem Typ *Duration* notiert.

Eine Eigenheit dieser Notation sind die vielen Kombinationsmöglichkeiten der Werte, durch die leider keine Eindeutigkeit mehr gegeben ist. So entsprechen z.B. `PT2H30M`, `PT150M`, `PT120M1800S`, etc. ein und der selben Zeitspanne. Noch problematischer ist die Interpretation von Monaten und Jahren. Ein Monat kann 28, 29, 30 oder 31 Tage haben und bei einem Jahr kann es sich eventuell um ein Schaltjahr handeln. Das Literal `P1M` steht folglich für verschiedene Zeitspannen, die unterschiedlich sind.

$$\texttt{P1M}=\texttt{P30T} \wedge \texttt{P1M}=\texttt{P31T} \text{ aber } \texttt{P30T}\neq\texttt{P31T}$$

Es liegt somit in den Händen der Entwickler/-innen, eine einheitliche Darstellung in ihren Anwendungen zu finden.

[2]I18N ist das Akronym für den englischen Begriff *Internationalization*.

Metadaten sind oft subjektiver Natur, was zu einer individuellen Begriffsbildung führen kann. Die natürliche Sprache lässt genug Raum für Interpretationsschwierigkeiten, sodass der Nutzen von Metadaten für Mensch und Maschine beeinträchtigt werden kann. Um diesem Problem entgegen zu treten, definiert LOM den Datentyp *Vocabulary*, mit dem eine Menge von Zeichenketten als gültiger Wertebereich definiert wird. Durch ihn wird die semantische Interoperabilität erhöht, da nur eine begrenzte Anzahl von Begriffen zur Auswahl steht. Bei der Festlegung dieser Wortmengen ist es sogar möglich, die Semantik der einzelnen Begriffe explizit zu erläutern. Auch die maschinelle Verarbeitung wird sicherer, da eine Überprüfung durch das System möglich ist.

Für beliebige Werte sieht LOM den Typ *CharacterString* vor, bei dem nicht einmal die Codierung vorgegeben wird. Lediglich die zu unterstützende Mindestlänge der Zeichenketten muss von den Systemen unterstützt werden. Bei manchen Datenelementen wird der Wertebereich auf einen anderen Standard eingeschränkt, wie z.B. für Personendaten (**VCard** [Howes98; Dawson98]) und Formatkennungen (**MIME Types** [Freed96]). Datenelemente vom Typ *Undefined* sollten nicht gesetzt werden, da sich ihre Bedeutung im weiteren Verlauf des Standardisierungsprozesses ändern kann.

Anhand der verfügbaren Anwendungen und Systeme kann mit Recht behauptet werden, dass sich LOM als der Standard im E-Learning-Bereich durchgesetzt hat. Hiervon zeugen auch die vielen Publikationen zu diesem Thema. In [Neven02] werden verschiedene Repositories evaluiert, die LOM zur Auszeichnung der Metadaten nutzen. Eine Erweiterung von LOM für multimediale Komponenten wird in [Saddik00; Saddik01] vorgestellt. Das europäische Projekt ARIADNE (*Alliance Of Remote Instructional Authoring And Distribution Networks for Europe*), hat ein eigenes *Application Profile* auf Basis von LOM entwickelt [Najjar03; Duval00] und ist der Versuch, mehr auf die europäischen Anforderungen für Metadaten einzugehen.

Die Beschreibung von LOM verdeutlicht, dass der Schwerpunkt der beschriebenen Metadaten eindeutig technisch ist. Die bestehende Kritik, besonders aus dem didaktischen Lager, soll nicht verschwiegen werden.

> „Zurzeit ist es weder möglich, die Eignung von Ressourcen für konkrete didaktische Methoden zu bestimmen, noch können pädagogische Planungsdetails (wie zum Beispiel die Kommunikationsstruktur, Evaluation) erschlossen werden. Die Akzeptanz der weniger technisch orientierten Lehrenden und Trainer hängt maßgeblich von derartigen Erweiterungen ab.“ [Pawlowski01, S. 107]

> „Die von der IEEE LTSC empfohlenen LOM-Spezifikationen werden zunehmend in aktuelle Lernplattformen implementiert und erlangen damit immer mehr praktische Relevanz. Zugleich stellen sie aber auch die pädagogische Eignung dieser Lernumgebungen in Frage. Der Verzicht auf die Beschreibung didaktisch-methodischer Aspekte, z.B der Verweis auf wichtige Kontextinformationen oder der Bezug zu konkreten Anwendungsszenarien, provoziert geradezu bereits jetzt zu beobachtende Extremformen des elektronisch vermittelten Lernens, die als »just enough learning«, »granulares Lernen« und bisweilen auch als »Fast-Food-Learning« bezeichnet werden.“ [Niegemann04, S. 272]

Kapitel 5

Autorenwerkzeuge

Für die Erstellung von Lernobjekten werden spezielle Werkzeuge gebraucht, so genannte **Autorenwerkzeuge**. Abhängig von der gewünschten Granularität (siehe Abschnitt 3.3) und dem Format (siehe Abschnitt 3.7) können durchaus verschiedene Programme zum Einsatz kommen. In der Regel werden z.B. Abbildungen mit Grafikprogrammen, *Java Applets* mit Programmierumgebungen und XML-Seiten mit XML-Editoren erstellt. Eine Anwendung für alle Aufgaben scheint daher wegen der vielfältigen Inhalte unrealistisch zu sein. Aufgrund der Komplexität des Erstellungsprozesses ist auch eine gemeinsame Betrachtung mit Lernplattformen (siehe Kapitel 6) nicht ratsam, da die jeweiligen Anforderungen zu weit auseinander liegen. Dennoch muss ein Kriterium bei der Bewertung von Autorensystemen und Lernplattformen die Möglichkeit zum Datenaustausch sein.

Andere Kriterien wie z.B. Funktionsumfang, Einarbeitungsaufwand, Preis, Systemvoraussetzungen und Kompatibilität werden allerdings ausschlaggebender sein. Ausgehend von der Fragestellung, was eigentlich produziert werden soll, muss das richtige Produkt gewählt werden. Manche der Kriterien wirken gegeneinander. So sind Autorenwerkzeuge mit einem größeren Funktionsumfang teurer und anspruchsvoller in der Bedienung. Was aber nützt das beste Autorenwerkzeug, wenn es nicht für das eingesetzte Betriebssystem erhältlich ist?

Neben den Werkzeugen sind auch Vorgehensmodelle für den Erstellungsprozess von Bedeutung. Häufig sollen multimediale Inhalte erschaffen werden, was den Entwicklungsaufwand massiv erhöht. Um eine gewisse Qualität zu erreichen, wird Wissen aus verschiedenen Disziplinen benötigt, wobei die Vorgehensmodelle hierbei meist auf Modellen der Softwaretechnik basieren, wie z.B. in [Depke99] beschrieben.

Zu der Gruppe der Autorenwerkzeuge zählen viele Programme, die sich in Aufgabe, Darstellung und Funktionsumfang teilweise sehr unterscheiden. Es ist daher sinnvoll, vorweg eine Klassifizierung einzuführen, um eine bessere Vergleichbarkeit zwischen den Systemen anzubieten, die ähnliche Aufgaben erfüllen. Im nächsten Abschnitt werden nur externe Programme vorgestellt, die nicht Bestandteil einer Lernplattform sind.

5.1 Klassifizierung

Alle vorgestellten Programme dienen zur Erstellung Web-basierter E-Learning-Inhalte, wobei die hierfür nötigen Sprachen bzw. Formate für die Anwender/-innen keine Rolle spielen sollen. Grafische Oberflächen, möglichst mit „What You See Is What You Get“ (WYSIWYG), erlauben eine schnelle Einarbeitung und vereinfachen die Arbeit. Anstatt z.B. HTML-Seiten direkt zu erstellen, können Texte, Grafiken und dergleichen intuitiv mit der Maus zusammengestellt werden. Es ist dann Aufgabe des Autorensystems, die Inhalte in eine entsprechende Kodierung zu überführen. Trotz dieser verallgemeinerten Anforderung nach grafischer Bedienbarkeit gibt es wesentliche Unterschiede, die eine Klassifizierung rechtfertigen.

Nach [Häfele03] lassen sich Autorensysteme grundsätzlich in 6 Gruppen einteilen:

1. Professionelle Werkzeuge mit integrierter Programmiersprache und hohem Einarbeitungsaufwand

2. Standard WYSIWYG-HTML-Editoren mit speziellen Plugins für E-Learning-Inhalte
3. *Rapid Content Development Tools* mit geringem Einarbeitungsaufwand
4. *Content Converter* für eine Umwandlung bestehender Dokumente in geeignete Formate
5. *Live Recording* zum Mitschneiden von Vorlesungen, Vorträgen und Präsentationen
6. *Screen Movie Recorder* zur Aufzeichnung von Programmsteuerungen

Eine Ausnahme sind noch die Editoren für mathematische Formeln, da sie meist nicht Bestandteil der anderen Autorensysteme sind. Obwohl es sich um eigenständige Programme handelt, werden sie in der Klassifizierung nicht explizit hervorgehoben. Die ersten beiden Klassen 1 und 2 gehören zu den klassische Autorensysteme, wie sie heute gängig sind. Der Trend geht jedoch hin zur schnellen Entwicklung von E-Learning-Inhalten, ermöglicht durch Programme der Klassen 3 bis 6. Abbildung 5.1 illustriert die Klassifizierung und nennt bereits passende Produkte.

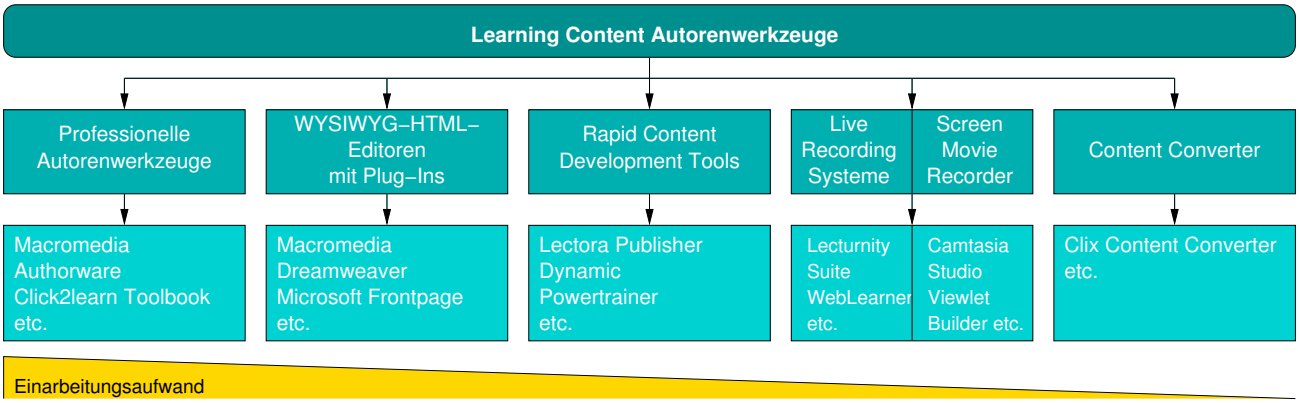


Abbildung 5.1: Systematik der Autorenwerkzeuge [Häfele03]

5.1.1 Professionelle Autorenwerkzeuge

Zu den hier vorgestellten professionellen Werkzeugen gehören **Macromedia Authorware**[1], **Click2Learn ToolBook**[2] und **Matchware Mediator**[3]. Sie zeichnen sich alle durch eine detaillierte Programmierbarkeit und Gestaltung der Inhalte aus, setzen jedoch ein gewisses Expertenwissen voraus.

Die aktuelle Version 7 von Authorware ist für die Erstellung von multimedialen E-Learning-Anwendungen auf CD-ROM/DVD und im Internet ausgerichtet. Zu den neuen Leistungen dieses Produkts zählen die Unterstützung von Standards für Lernplattformen und der Import von *Microsoft PowerPoint-Folien*. Durch eine automatische Überwachung können die Lernerfolge der Lernenden verfolgt werden. Personen, die bereits mit anderen *Macromedia* Produkten vertraut sind, werden sich schnell an das *Graphical User Interface* (GUI) gewöhnen und einen schnellen Einstieg finden. Viele Arbeitsschritte können mit *Drag'n'Drop* erledigt werden und durch das *One-button publishing*, bei dem lediglich ein Knopfdruck für die Erstellung des Endprodukts getätigt wird. Automatisch wird ein *Content Package* (siehe Kapitel 3) erzeugt und auf die Lernplattform hochgeladen. Auf diese Weise lassen sich sehr schnell Ergebnisse erzielen. Die eigentliche Stärke liegt aber in der Programmierbarkeit, die jedoch einiges an Erfahrung mit Skriptsprachen, z.B. *JavaScript*, abverlangt. Für Standardanwendungen, wie z.B. Logins, Kurs-Rahmenwerke, Übungen und Quiz, gibt es Templates und Wizards, die ohne Programmierung auskommen. Abbildung 5.2 zeigt einen exemplarischen Screenshot.

[1] `http://www.macromedia.com/software/authorware` (29.10.05)
[2] `http://www.toolbook.com` (29.10.05)
[3] `http://www.matchware.net/ge/products/mediator/default.htm` (29.10.05)

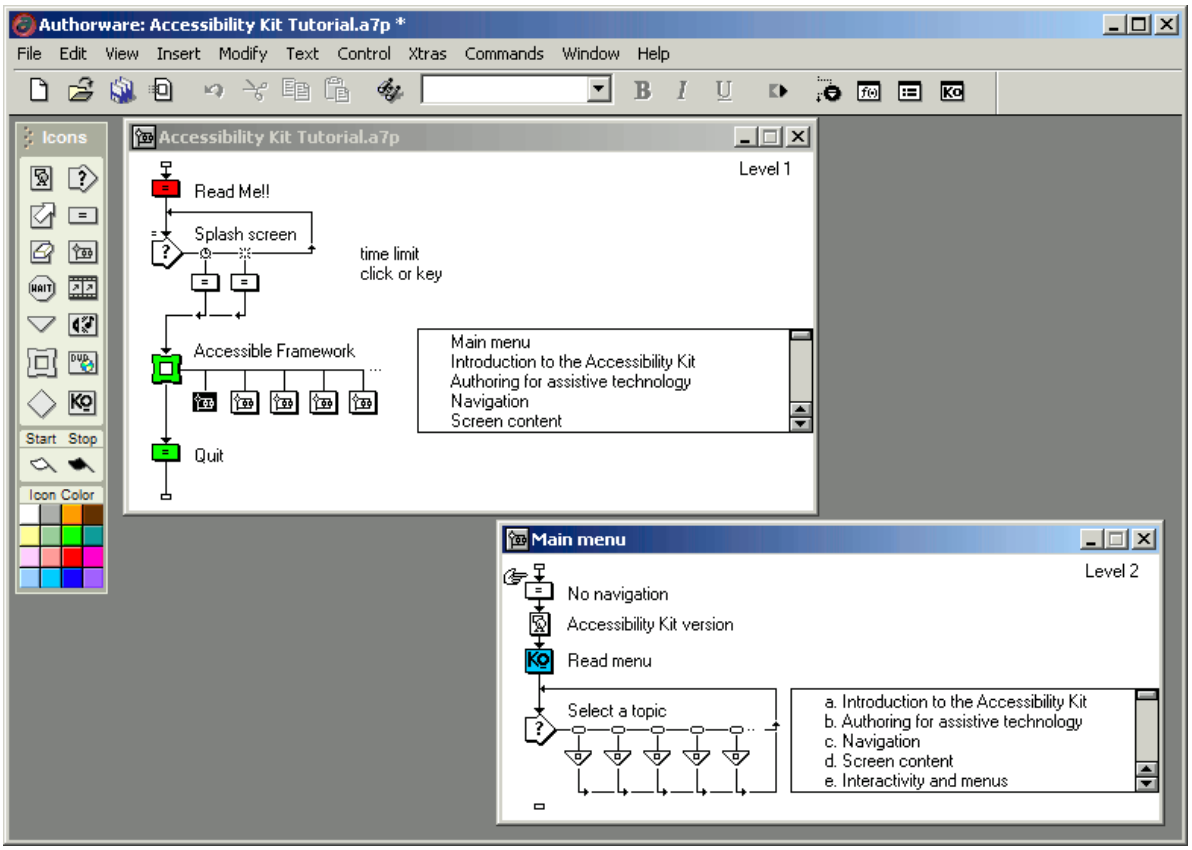


Abbildung 5.2: *Macromedia Authorware 7*

Bei ToolBook handelt es sich um eine Produktlinie, die aus **ToolBook Instructor** und **ToolBook Assistant** besteht. Beide Programme sind für die Erstellung von Web-basierten E-Learning-Inhalten in Unternehmen ausgelegt, unterscheiden sich aber in der Zielgruppe. Richtet sich der Assistant mehr an fachliche Experten, die wenig technisches Wissen haben, können mit dem *Instructor* Simulationen und interaktive Inhalte programmiert werden. Eine WYSIWYG-Oberfläche mit *Drag'n'Drop* erlaubt eine einfache Nutzung beider Produkte und *Wizards*, *Templates* sowie eine große Auswahl an fertigen Objekten erleichtern eine schnelle Produktion. Die erzeugten Inhalte können von beiden Programmen verarbeitet werden. So ist es z.B. möglich, komplexere Templates mit dem Instructor zu erstellen, die dann im Assistant zum Einsatz kommen. Ein weiteres Plus von *ToolBook* ist die nahtlose Anbindung zu verschiedenen Lernplattformen.

Den *Mediator* in der Version 7 gibt es als *Standard*, *Pro* und *EXP*. Die Ausgabe Standard bietet einen Einstieg in die Erstellung multimedialer CDs und Flash-Seiten. Ohne Programmierkenntnisse und mit *Drag'n'Drop* können auf diese Weise einfache Ergebnisse produziert werden. Für die Erstellung von E-Learning-Inhalten empfiehlt der Hersteller selbst die Version *Pro*, da weitergehende Funktionen angeboten werden. In einem Ereignisdialog können Ereignisse wie „Maus loslassen“ oder „Taste gedrückt“ mit Aktionen wie „Seitenwechsel“ oder „Cursor verschieben“ grafisch verknüpft werden. Hierdurch können echte Interaktionen realisiert werden, aber eine direkte Programmierung mit *JavaScript* ist nur in der Ausgabe EXP möglich.

5.1.2 WYSIWYG-HTML-Editoren

Mit WYSIWYG-HTML-Editoren können auf einfache Weise HTML-Seiten erstellt werden. Per Maus lassen sich Texte, Bilder, Animation etc. kombinieren und ausrichten, ähnlich einer Textverarbeitung. Mittlerweile sind die Programme so weit im Funktionsumfang fortgeschrit-

ten, dass so gut wie keine Programmierung von Hand mehr durchzuführen ist. Besonders bei der Erstellung von Lösungen wiederkehrender Aufgaben in *JavaScript*, wie z.B. Navigation oder der Wertüberprüfung in Formularen, stellen die WYSIWYG-HTML-Editoren eine große Hilfe dar. Zu den oft genutzten Programmen dieser Gattung zählen sicherlich **Macromedia Dreamweaver**[4], **Adobe GoLive**[5], **NetObjects Fusion**[6] und **Microsoft Frontpage**[7]. Für die direkte Erstellung von E-Learning-Inhalten sind sie jedoch nur bedingt geeignet, weil sie meist keine Standards wie SCORM oder AICC unterstützen.

Für *Dreamweaver* gibt es eine Reihe an Erweiterungen speziell für E-Learning-Inhalte, die als *Plugins* direkt in das Programm integriert werden und kostenfrei bei *Macromedia* im E-Learning-Bereich[8] heruntergeladen werden können. Die wichtigste Erweiterung ist der **CourseBuilder**, mit dem über 40 vorgefertigte Lerninteraktionen, Quiz- und Bewertungsvorlagen zur Verfügung stehen. Abbildung 5.3 zeigt das Dialogfenster dieser Erweiterung.

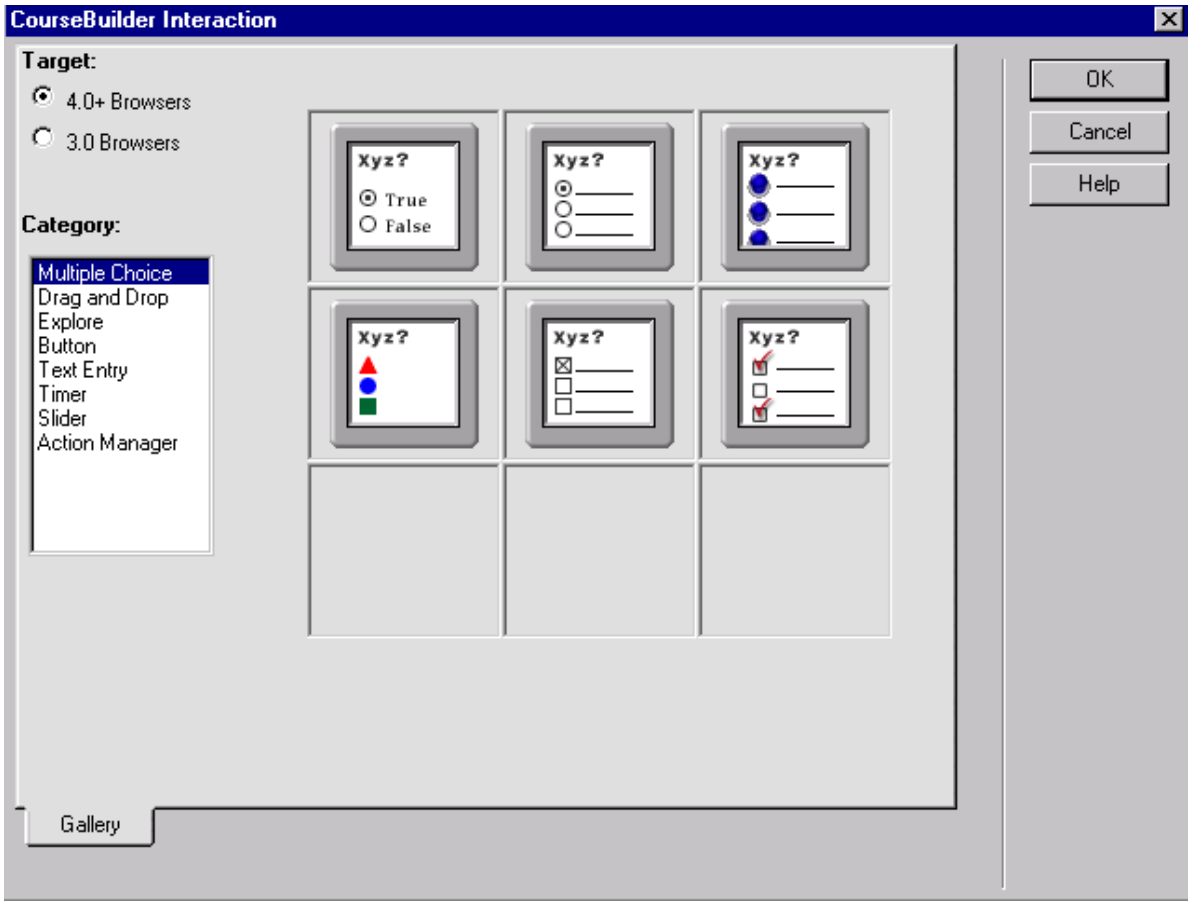


Abbildung 5.3: *Macromedia CourseBuilder*-Erweiterung für *Dreamweaver*

Mit Hilfe der **Learning Site**-Befehlserweiterung können Kursaktivitäten, wie z.B. ein Quiz, aus verschiedenen Quellen zusammengestellt werden. Über integrierte Navigationsfunktionen und *Tracking-Features* lassen sich die Aktivitäten der Lernenden in einer Datenbank protokollieren. Durch die **SCORM RTI**-Erweiterung (*Sharable Content Object Reference Model Runtime Interface*) können die Inhalte so gespeichert werden, dass sie den SCORM-Standards für die Runtime-Umgebung entsprechen. Die **Manifest Maker**-Erweiterung generiert aus der Struktur eine Manifest-Datei in XML, die allen Spezifikationen des IMS *Content Packaging* entspricht (siehe Abschnitt 3.5).

[4] `http://www.macromedia.com/software/dreamweaver` (29.10.05)
[5] `http://www.adobe.com/products/golive` (29.10.05)
[6] `http://www.netobjects.com` (29.10.05)
[7] `http://www.microsoft.com/frontpage` (29.10.05)
[8] `http://www.macromedia.com/de/resources/elearning/extensions/dw_ud` (29.10.05)

5.1.3 Content Converter

Aus gehaltenen Vorlesungen, Vorträgen und Präsentationen existieren oftmals Materialien in digitaler Form, die sich jedoch nicht direkt für das E-Learning eignen. Durch Anpassung bzw. Umwandlung können jedoch adäquate Ergebnisse erreicht werden, die eine einfache Integration in ein LMS ermöglichen. Die Klasse der *Content Converter* richtet sich an Personen mit erstellten Inhalten in gängigen Formaten, wie z.B. **OpenOffice**[9] oder **PowerPoint**[10], die mit wenigen Schritten solche Dateien in E-Learning-Inhalte umwandeln wollen. Das Zielformat ist meistens HTML und ein typisches Programm dieser Gattung ist der **Content Converter**[11] des LMS Clix Campus.

Der *Content Converter* liest *Word*-Dokumente ein, analysiert die Struktur, erzeugt ein XML-Dokument und extrahiert Abbildungen in separaten Dateien. Hierbei wandelt das System den Dokumentaufbau in eine Navigationslogik um, die für die spätere Steuerung benötigt wird. Hierbei ist darauf zu achten, dass die Überschriften richtig gesetzt sind, da sie die Anhaltspunkte der Analyse darstellen. Abschließend erfolgt die Umwandlung des XML-Dokuments in einzelne HTML-Seiten und eine Navigation. Über Format- und Design-Vorlagen kann das Layout den eigenen Bedürfnissen angepasst werden.

Mittlerweile können auch Textverarbeitungsprogramme ihre Dokumente in HTML oder XML speichern, weshalb die Relevanz der einfachen *Content Converter* abnimmt. Die Resultate ähneln jedoch mehr Texten als multimedialen E-Learning-Inhalten. Umfangreichere *Content Converter*, die z.B. Metadaten extrahieren, feinere Granularitäten erlauben und Standards wie SCORM oder AICC unterstützen, haben jedoch das Potential, eine echte Alternative gegenüber spezialisierten Programmen zu sein.

5.1.4 Live Recording Systeme

Systeme zur Aufzeichnung von Audio- und Videodaten ermöglichen eine schnelle Erstellung multimedialer E-Learning-Inhalte. Auf diese Weise lassen sich Vorlesungen, Vorträge und Präsentationen festhalten und bei Bedarf abrufen. Nach der Aufzeichnung lassen sich die Daten bearbeiten, z.B. durch Anmerkungen oder die Erstellung eines Indexes. Im Gegensatz zu einer herkömmlichen Aufzeichnung mit einer einfachen Videokamera erlauben die *Live Recording* Systeme eine Verknüpfung mit dem präsentierten Material, wie z.B. einer *PowerPoint*-Präsentation. Exemplarisch für diese Klasse werden **IMC Lecturnity Suite**[12] und **Tegrity WebLearner**[13] vorgestellt.

Ausgangspunkt der *Lecturnity Suite* ist eine *PowerPoint*-Präsentation, die im Vorwege erstellt wird. Ausgestattet mit einem Headset oder Mikrofon trägt der/die Dozent/-in wie gewohnt die Vorlesung vor und wird von einer Kamera aufgezeichnet. Eventuelle Annotationen lassen sich auf einem Smart- oder Whiteboard anbringen. Abbildung 5.4 zeigt ein Szenario für den Einsatz von *Lecturnity*. Alle Quellen werden synchron aufgezeichnet und in ein Lernmodul gepackt, das anschließend mit dem **Lecturnity Player** abgespielt werden kann oder mit dem **Lecturnity Converter** in ein Format für den **Real Media Player**[14] oder **Windows Media Player**[15] umgewandelt wird.

Das Prinzip der Aufzeichnung ist beim *Tegrity WebLearner* gleich. Ein besonderer Punkt ist das **WebLearner Studio**, ein Komplettpaket versehen mit der nötigen Hardware (PC, Bildschirm, Projektor, Kamera und Mikrofon), das für die Aufgabe optimal abgestimmt ist. Probleme mit inkompatibler Hardware treten somit nicht auf. Eine Nachbearbeitung der Aufzeichnung ist mit dem **WebLearner Editor** möglich und eine Wiedergabe erfolgt über den

[9] `http://www.openoffice.org` (29.10.05)
[10] `http://office.microsoft.com` (29.10.05)
[11] `http://www.im-c.de` (29.10.05)
[12] `http://www.im-c.de` (29.10.05)
[13] `http://www.tegrity.com` (29.10.05)
[14] `http://www.real.com` (29.10.05)
[15] `http://www.microsoft.com/windows/windowsmedia` (29.10.05)

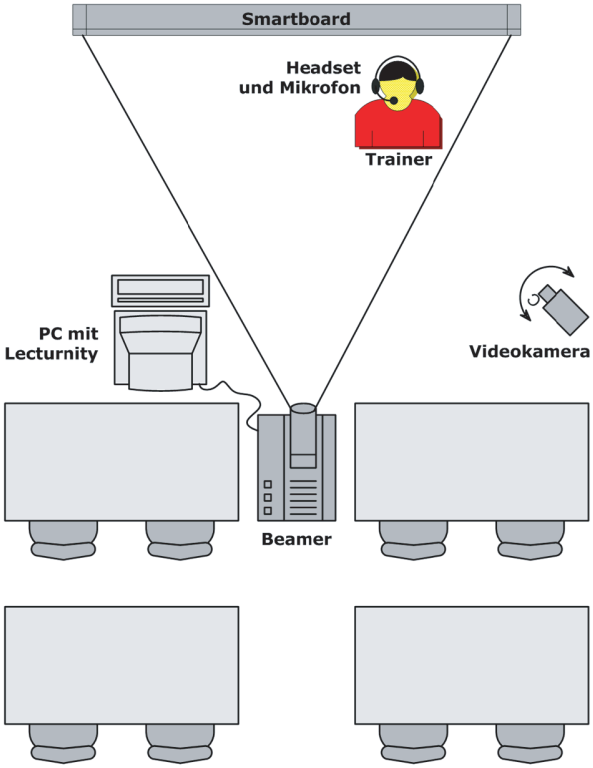


Abbildung 5.4: Einsatz von *Lecturnity* (Aus einer Werbebroschüre)

WebLearner Server. Zusätzliche Module bieten Funktionen an, die in der *Lecturnity Suite* nicht angeboten werden. Beispielsweise ermöglicht das Modul **Webcast** synchrone *Live*-Sessions mit anderen Teilnehmern/-innen, inklusive Interaktionen via Chat und Mikrofon.

5.1.5 Screen Movie Recorder

Screen Movie Recorder dienen zur direkten Aufzeichnung von Aktivitäten am Rechner und einer gleichzeitigen oder späteren Vertonung. Der Bildschirm oder ein ausgewähltes Fenster wird direkt „abgefilmt“ und an zusätzlicher Hardware werden lediglich ein Headset oder Mikrofon benötigt. Auf diese Weise lassen sich Tutorien für die Bedienung von Programmen einfach und schnell erstellen. Stellvertretend für diese Klasse werden **Camtasia Studio**[16], **Turbo Demo**[17] und **Qarbon ViewletBuilder**[18] vorgestellt.

Das *Camtasia Studio* ist ein sehr professionelles Programm, mit dem z.B. *Microsoft* seine *How-To*-Videos erstellt hat. Es besteht aus den drei Komponenten **Recorder**, **Producer** und **Effects**. Der *Recorder* erstellt einen Film, der mit dem *Producer* nachbearbeitet werden kann, wie z.B. Schneiden, Vertonen und Konvertieren in andere Videoformate. Spezielle Ergänzungen wie Annotationen, Bilder und dergleichen werden mit *Effects* eingefügt.

Bei *Turbo Demo* und *ViewletBuilder* werden im Gegensatz zum *Camtasia Studio* keine kompletten Filme aufgenommen. Eine Animation besteht bei diesen Programmen aus einzelnen Bildern, auch Schlüsselszenen genannt, zu denen lediglich die Mausbewegung und Ereignisse aufgezeichnet werden. Dieser Ansatz erzeugt wesentlich kleinere Dateien als eine vollständige Aufzeichnung. Auch die nachträgliche Bearbeitung erweist sich als einfacher, da Screenshots hinzugefügt oder misslungene ausgetauscht werden können. Neben der Ergänzung mit Textfeldern, Sprechblasen, Bildern und Vektorgrafiken sind sogar einfache Interaktionen mit der Maus möglich.

[16] `http://www.techsmith.com` (29.10.05)
[17] `http://www.turbodemo.de` (29.10.05)
[18] `http://www.qarbon.com` (29.10.05)

5.1.6 Rapid E-Learning Content Development

Die *Rapid E-Learning Content Development* Werkzeuge kommen ohne jegliche Programmierung aus und erlauben dennoch die Entwicklung interaktiver Elemente wie Quiz und Tests in standardkompatiblen Formaten. Hierfür stehen Schablonen einzelner Seiten zur Verfügung, die mit eigenen Texten, Abbildungen und Animationen ausgefüllt werden. Mehrere Seiten lassen sich in Kapiteln organisieren und zu vollständigen Kursen zusammensetzen. Alle Programme bieten WYSIWYG und *Drag'n'Drop* für eine einfache Benutzung an. Je nach Leistungsumfang können Fremdformate für Audio, Video und Multimedia eingebunden werden, sodass auch komplexere Elemente möglich sind. Bekannte Vertreter dieser Klasse sind **NIAM-TMS EasyGenerator**[19] , **ITACA EasyProf**[20] und **Trivantis Lectora Publisher**[21].

Der *EasyGenerator* ist eine *E-Learning-Suite* und teilt die Arbeit in vier Bereiche auf, die jeweils von einer Komponente unterstützt werden. Schulungen und Tests werden mit dem *EasyGenerator* erzeugt und gewartet. Die Resultate lassen sich in einem Format speichern, das mit Hilfe des *EasyPlayers* von einer CD-ROM oder des *EasyWebPlayers* über das Internet abgespielt wird. Zudem werden SCORM und AICC für den Einsatz in einem LMS unterstützt. Welche Fortschritte die Lernenden machen, wird durch *EasyProgress* überwacht. Sämtliche Verwaltungstätigkeiten, die bei der Erstellung, Bereitstellung und Einsatz anfallen, werden mit *EasyCourseManager* erledigt.

Mit *EasyProf* lassen sich einfach multimediale E-Learning-Inhalte und Präsentationen in den Ausgabeformaten HTML, HTML mit SCORM sowie CD-ROM erzeugen. Ein Schwerpunkt dieses Programms liegt bei Testfragen, die sich mit der Maus zusammenstellen lassen. Die Tracking-Daten werden in XML kodiert und können entweder per E-Mail oder FTP übertragen werden.

Neben den Schablonen bietet *Lectora* zusätzlich Assistenten an, mit denen Kursstrukturen automatisch erzeugt werden. Ähnlich den programmierbaren Konkurrenzprodukten, lassen sich Ereignisse — ausgelöst von Maus und Tastatur — mit Grafik- und Textobjekten verknüpfen. Zu den Stärken des Programms gehört die Vielfalt an unterstützen Fremdformaten, wie z.B. das IPIX-Format[22], das 360×360 Grad Panorama Bilder ermöglicht. Als Ausgabeformat stehen AICC sowie SCORM zur Wahl, HTML, CD-ROM/DVD und die Erstellung eines ausführbaren Programms (`.exe`) für Windows. Abbildung 5.5 zeigt einen typischen Screenshot von *Lectora*.

5.2 Bewertung

In den Tabellen 5.1 und 5.2 sind die vorgestellten Programme für einen direkten Vergleich aufgelistet. Auch wenn die Ansätze teilweise sehr verschieden sind, wie z.B. programmierbare und aufzeichnende Systeme, lassen sich Parallelen beim Erstellungsprozess erkennen, die problematisch sind. Hierzu gehört die Wiederverwendung, da die meisten Autorensysteme nur optimale Ergebnisse erzielen, wenn die Ergebnisse in proprietären Formaten gespeichert werden. Dateien verschiedener Anwendungen lassen sich dann nur mit Aufwand kombinieren und die Resultate sind oft suboptimal. Ein einheitliches Layout ist nur mit viel Disziplin bei der Erstellung zu erreichen und Inhalte fremder Anbieter stechen schon aufgrund des Erscheinungsbildes hervor. Da die vorgestellten Programme meist auf Systemen mit dem Betriebssystem Windows laufen, werden einige Zielformate auf Rechnern mit *Linux* oder *Mac OS X* nicht unterstützt. Folglich wird beim Einsatz eines proprietären Formats bereits durch das Autorensystem die Infrastruktur eingeschränkt. Mit den Definitionen für Lernobjekte aus Kapitel 3, besonders Downes essentielle Anforderung an die Interoperabilität (siehe S. 22) sei hier hervorgehoben, ist dies freilich schwer vereinbar.

[19] `http://www.easygenerator.de` (29.10.05)
[20] `http://www.easyprof.com` (29.10.05)
[21] `http://www.lectora.com` (29.10.05)
[22] `http://www.ipix.com` (29.10.05)

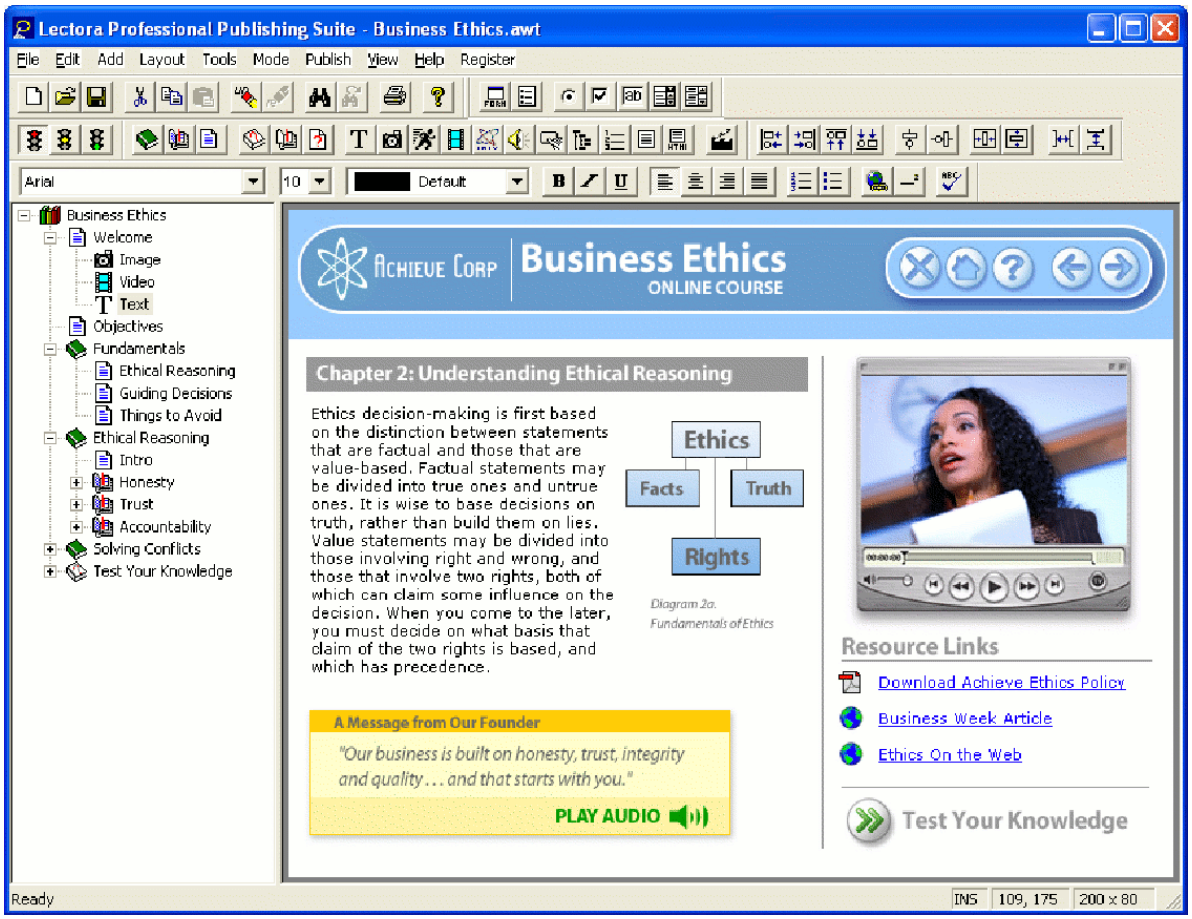


Abbildung 5.5: Screenshot von *Lectora*

Ein ähnliches Problem ergibt sich aus der unterstützten Granularität. Viele der Programme erlauben lediglich in sich abgeschlossene Einheiten, wie z.B. Kurse oder Vorträge. Dies führt zu Behinderungen bei einer nachträglichen Änderung der Sequenzierung (siehe Abschnitt 3.4), was wiederum zu Lasten der Wiederverwendung geht. Programmen mit kleineren Einheiten, wie z.B. *Dreamweaver* oder die *Screen Movie Recorder*, fehlen hingegen die adäquaten Mittel für komplexere Strukturen. Auch wenn *Dreamweaver* einer guten Autorenumgebung schon nahe kommt, ist der Ansatz falsch gewählt. Anstatt einen HTML-Editor mit E-Learning-Plugins zu erweitern, sollte besser eine E-Learning-Umgebung mit HTML-Plugins gewählt werden.

Ganz anders stellt sich das Problem Standardkompatibilität dar. Wahrscheinlich liegen die Gründe im Marketing begründet, dass auf vielen Produkten dieses Siegel zu finden ist und in der Tat ermöglichen es diese Programme auch, Dateien in Formaten wie beispielsweise *SCORM*, *IMS Content Packaging* und *LOM* zu speichern. Doch genauer betrachtet, zeigt sich schnell der Etikettenschwindel. Die Ursachen liegen wieder in den proprietären Formaten und den eingeschränkten Granularitäten. Eine mit *Authorware* gespeicherte SCORM-Datei ernüchtert schnell. Die Strukturierungsmöglichkeiten des Manifests (siehe Abschnitt 3.5) werden nicht genutzt, denn es handelt sich vielmehr um eine *Flash*-Datei, eingepackt als Ressource, was sicherlich nicht der Sinn eines *Sharable Content Objects* (SCO) ist. Als Fazit auf ein anderes Produkt umzuschwenken löst das Problem nicht. Die anderen Produkte schneiden nicht besser ab.

Bei der Zusammenarbeit mit Learning Management Systemen wird nur von wenigen Autorensystemen, wie z.B. *Authorware* und *ToolBook*, eine direkte Verbindung angeboten. Gerade bei der Erstellung neuer Inhalte ist aber wichtig, das Layout auf der Zielplattform zu überprüfen. Funktionen wie WYSIWYG unterstützen die Autoren/-innen bei der Arbeit, jedoch

prägt letztendlich das LMS das Erscheinungsbild. In Hinblick auf eine Separation von Inhalt und Darstellung, wie in Abschnitt 3.7 beschrieben, gilt dies besonders zu berücksichtigen. Aus diesem Grund muss sich die Datenübertragung nahtlos in den gesamten Prozess der Erstellung und Kontrolle einfügen.

Die vorgestellten Autorensysteme gehören zu den Programmen, die heutzutage im Einsatz sind. Ihre Analyse hat die in Abschnitt 1.1 angesprochene Diskrepanz zwischen Theorie und Praxis bestätigt, sodass folglich neue Ansätze für Autorensysteme gefunden werden müssen.

	System	Bedienung	Eingabe	Import	Ausgabe	Standard	Größe
Professionell	Authorware 7	Drag'n'Drop, WYSIWYG, Programmierung	Text, Grafik, Audio, QuickTime, Director, Flash, JavaScript, ActiveX	PowerPoint, XML, RTF	Flash, Windows Media Player, DVD, Mac OS X Playback	SCORM, IMS, AICC, LOM	Kurse, Templates
	ToolBook (Instructor)	Drag'n'Drop, WYSIWYG, Programmierung	Text, Grafik, Audio, PowerPoint, Word, Flash, JavaScript, ActiveX, RealMedia		HTML, .exe	SCORM, AICC	Kurse, Templates
	Mediator 7 (EXP)	Drag'n'Drop, WYSIWYG, Programmierung	Text, Grafik, Audio, Flash, JavaScript, Visual Basic, ActiveX, QuickTime	Flash	DHTML, Flash, CD-ROM/DVD, ScreenSaver, .exe, Mediator		Kurse, Templates
HTML	Dreamweaver (mit Plugins)	Drag'n'Drop, WYSIWYG, Programmierung	Text, Grafik, Audio, Video, Flash, ActiveX, JavaScript	HTML, XML, Tabellendaten	HTML, XHTML, JSP, PHP	SCORM, IMS, AICC, LOM	Seiten, Kurse
Conv.	Clix Campus Content Converter	Maus		MS Word	XML, HTML		Abschnitte, Kurse
Live Recording	Lecturnity Suite	Maus, Tastatur	Text, Grafik	PowerPoint, analoge Videodaten	CD/DVD, Internet-Streaming, RealMedia, Windows Media		Vortrag, Film
	Tegrity WebLearner	Maus, Tastatur	Text, Grafik	PowerPoint, Windows Media, QuickTime, Videodaten	CD-ROM, Windows Media, QuickTime	SCORM, IMS, AICC	Vortrag, Film

Tabelle 5.1: Übersicht der Autorensysteme (Teil 1)

	System	Bedienung	Eingabe	Import	Ausgabe	Standard	Größe
Screen Recorder	Camtasia Studio	Maus, Tastatur	Text, Grafik, Audio, HTML, PowerPoint, Word		AVI, Flash, Windows Media, QuickTime, RealMedia, .exe, Animated GIF		Film
	Turbo Demo	Maus, Tastatur	Text, Grafik, Audio		Java/HTML, Flash, AVI, .exe, Animated GIF, PDF		Film
	Viewlet-Builder	Maus, Tastatur,	Text, Grafik, Audio	Flash	Flash, .exe, PDF, Bilder	SCORM	Film
Rapid Development	Easy-Generator	Maus, Drag'n'Drop, WYSIWYG, Tastatur	Text, Grafik, Audio		Eigenes Format	SCORM, AICC	Kurs, Schablone
	EasyProf	Maus, Drag'n'Drop, WYSIWYG, Tastatur	Text, Grafik, Audio, RealMedia, Windows Media, QuickTime, Flash	PowerPoint, Word	HTML, CD-ROM, .exe, .jar	SCORM, AICC	Kurs, Schablone
	Lectora	Maus, Drag'n'Drop, WYSIWYG, Tastatur	Text, Grafik, Audio, RealMedia, Windows Media, Flash		HTML, .exe	SCORM, AICC	Kurs, Schablone

Tabelle 5.2: Übersicht der Autorensysteme (Teil 2)

Kapitel 6

Lernplattformen

Der Einsatz von E-Learning lässt sich nur mit einer geeigneten Infrastruktur sinnvoll realisieren, weil viele Inhalte inklusive Metadaten einer großen Zahl von Personen mit unterschiedlichen Aufgaben zugänglich gemacht werden müssen. Einen wesentlichen Anteil übernimmt die **Lernplattform**, die eine zentrale Anlaufstelle für ein breites Angebot an Diensten ist. Um den Leistungsumfang eines solchen Systems genauer bestimmen zu können, werden folgend die Rollen und Tätigkeiten beschrieben, die hauptsächlich mit einer Lernplattform zu tun haben. Ein/eine **Dozenten/-in** plant und organisiert Lehrveranstaltungen für **Studierende** und wird bei der Durchführung zumeist von **Tutoren/-innen** unterstützt. Jede dieser Rollen muss über eine **Benutzerverwaltung** dem System bekannt gegeben werden. In der Regel legt die Rolle **Administrator/-in** den/die Dozenten/-in an, eventuell auch die Tutoren/-innen. Diese können wiederum Studierende eintragen, die sich für einen Kurs angemeldet haben.

An dieser Stelle wird bereits erkennbar, dass eine **Rollen- und Rechtevergabe** benötigt wird. Lediglich den Administratoren/-innen soll gestattet sein, neue Zugänge für Dozenten/-innen anzulegen. Dozenten/-innen und Tutoren/-innen werden mit weniger Rechten bedacht und bei Studierenden sollte im Hinblick auf möglichen Missbrauch ganz von jeglichen administrativen Möglichkeiten abgesehen werden.

Eine Lernplattform muss also „wissen“, wer gerade mit ihr arbeitet, um die Rechte der Rolle überprüfen zu können. Hierbei sei erwähnt, dass es durchaus üblich ist, wenn ein und die selbe Person in mehreren Rollen auftritt. So kann z.B. der/die Dozent/-in in die Rolle **Designer/-in** wechseln, in der andere Rechte zugebilligt werden. Dieses Vorgehen erlaubt eine bessere Kontrolle der einzelnen Aktivitäten und erhöht die Sicherheit des Systems. Eine Person identifiziert sich über die **Authentifizierung** der Lernplattform und bekommt eine Rolle zugewiesen. Vorwiegend erfolgt der Zugang über einen Namen und ein Passwort, aber auch Schlüssel in Dateien oder *Cookies* der Web-Browser sind denkbar. Nach einer erfolgreichen Authentifizierung überprüft die Lernplattform jede Aktion über die **Autorisierung**. Anhand der eingenommenen Rolle und den vergebenen Rechten wird kontrolliert, ob die gewünschte Operation ausgeführt werden darf oder die nötige Berechtigung fehlt. Komfortable Systeme zeigen allein die Operationen an, die auch ausgeführt werden dürfen. So würde z.B. die Rolle Designer/-in eine Operation „Bearbeiten“ in ihrer Kursansicht vorfinden und die Rolle Studierende nicht.

Neben der Benutzerverwaltung muss eine Lernplattform auch eine **Kursverwaltung** anbieten. Hierbei sind die unterstützten Granularitäten (siehe Abschnitt 3.3) und Formate (siehe Abschnitt 3.7) wesentliche Merkmale. Je nach Abstraktionsniveau können u.a. HTML-Seiten, Bilder, Lernobjekte oder vollständige Kurse in einem Repository gespeichert werden. Die Lernplattform sollte den/die Dozenten/-in bei der Suche nach geeigneten Materialien unterstützen und beliebige Kombination fremder und eigener Inhalte ermöglichen. Durch Wiederverwendung können somit Kosten reduziert werden. Verschiedene Ausgabeformate, wie z.B. HTML oder PDF runden den Funktionsumfang ab, wobei die direkte Erstellung von Inhalten nicht zu den Aufgaben einer Lernplattform gehört. Im folgenden Abschnitt 6.1 wird allerdings auf Sys-

teme eingegangen, die diesen Prozess unterstützen, z.B. durch einen einfachen Datenaustausch mit Autorensystemen.

Begleitend zu einem Kurs sollte eine Lernplattform Daten über die Leistungen der Studierenden sammeln: „Wie lange ist die letzte Anmeldung her?“, „Wie viel Zeit wurde in welchem Bereich verbracht?“ und „Welche Aufgaben wurden absolviert?“ sind nur einige Informationen, die in Statistiken die Lernaktivitäten veranschaulichen. Auf diese Weise können individuelle Probleme erkannt und gezielt behoben werden.

Auch die Kommunikation zwischen den Personen soll eine Lernplattform unterstützen. Bekannte **Kommunikationsmethoden** sind u.a. **Foren**, in denen Beiträge nach Themen geordnet sind und *Threads* (Diskussionsfäden) entstehen. Dozent/-in und Tutoren/-innen können bei Bedarf Hilfestellungen geben, indem sie eigene Beiträge hinzufügen. Schneller geht es beim **Chat** zu. Texteingaben werden direkt auf den Bildschirmen aller oder ausgewählter Personen angezeigt und Reaktionen können prompt folgen, sodass im Vergleich zu den Foren agilere Diskussionen möglich sind. Beide Kommunikationsmethoden, Foren und Chat, sollten bei Bedarf moderierbar sein, d.h. eine Person oder Gruppe ist mit Sonderrechten ausgestattet. Sie überwacht das „Niveau“ der Beiträge und kann gegebenenfalls intervenieren. Wichtige Nachrichten für alle Studierenden, wie z.B. Terminankündigungen oder Terminänderungen, sollten per **E-Mail** bekannt gegeben werden.

Der Lernprozess der Studierenden soll von der Lernplattform unterstützt werden. Besonders die Organisation und Vorgehensweise sollen durch **Werkzeuge für Studierende** vereinfacht werden, wie z.B. mit einem Notizbuch für persönliche Annotationen zu den Inhalten oder einem Kalender. In eigenen Bereichen soll die Möglichkeit zum Nachlernen und zur Selbstevaluation gegeben sein.

6.1 Definitionen

Es soll nun der Versuch unternommen werden, eine Definition bzw. einen Anforderungskatalog für Lernplattformen zu erstellen. Da es sich um komplexe Systeme handelt, bei denen es auf Praxistauglichkeit ankommt, gibt es so gut wie keine theoretischen Arbeiten. Vielmehr handelt es sich um Praxiserfahrungen, die in eigenen Implementierungen oder Evaluationen (siehe Abschnitt 6.2) gesammelt wurden.

Von der damaligen EDUCOM-Kommission (heute: EDUCAUSE) stammt folgender Versuch, die wesentlichen Eigenschaften einer Lernplattform festzulegen. Die Definition stammt aus [Schulmeister01, S. 132ff]:

Kurse: Die Einrichtung und Durchführung von Kursen ist möglich.

Akteure: Lernsysteme sollten mindestens die Rollen für folgende Akteure vorsehen: Studierende, Dozenten, Tutoren, Administratoren.

Dienste: Dienste müssen über eine eigene Funktionalität verfügen:
Administrative Dienste: Kurskalender, Schwarzes Brett, etc.
Kommunikationsdienste: Chat, E-Mail, Foren
Lehrfunktionen: Folien, Referenzen zu Netzadressen, etc.
Evaluationsdienste: Tests, Selbstevaluation, etc.

Dokumente: Dokumente müssen Teil der Lernobjekte und der Dienste sein.

Gruppen: Kollaboratives Arbeiten muss möglich sein, wobei mehrere Benutzer gleichzeitig kommunizieren.

Institutionen: Die Lernumgebung ist an jede Institution anpassbar.

Sprache: Kurse in mehreren Sprachen müssen unterstützt sein.

Interface: Anpassungen der grafischen Schnittstelle an die Lernumgebung sind möglich.

Navigationsstruktur: Anpassung der Navigation an das Lernumfeld ist möglich.

Eigentlich erfüllt keine der heute verfügbaren Lernplattformen diesen Leistungskatalog und dennoch ist er nicht vollständig. Die Beschreibung am Anfang dieses Kapitels lässt erahnen, dass noch weitere Funktionen zum Umfang gehören. Es fehlt z.B. eine Kursverwaltung, die Wartung und Suche beinhaltet. Auch das Rollenmodell wird erst durch einen Authentifizierungs- und Autorisierungsmechanismus vollständig. Die Forderung nach anpassbaren grafischen Schnittstelle sollte in eine strikte Trennung von Inhalt und Darstellung überführt werden, um neben dem wandelbaren Aussehen verschiedene Ausgabeformate wie HTML und PDF zu ermöglichen. Zudem sind eine nähere Anbindung von Autorensystemen an Lernplattformen sowie die statistische Erfassung der Lernaktivitäten sinnvolle Ergänzungen für eine umfassende Definition.

Ausgehend von der gegebenen Definition, kennzeichnet Schulmeister in [Schulmeister03, S. 10] folgende Punkte als relevant, um eine Lernplattform von einer bloßen Kollektion von Skripten oder Hypertextsammlungen zu unterscheiden:

- Eine Benutzerverwaltung (Anmeldung mit Verschlüsselung)
- Eine Kursverwaltung (Kurse, Verwaltung der Inhalte, Dateiverwaltung)
- Eine Rollen- und Rechtevergabe mit differenzierten Rechten
- Kommunikationsmethoden (Chat, Foren) und Werkzeuge für das Lernen (Whiteboard, Notizbuch, Annotationen, Kalender etc.)
- Die Darstellung der Kursinhalte, Lernobjekte und Medien in einem netzwerkfähigen Browser

Durch diese Ergänzung fallen eine Reihe von Systemen heraus, die allgemein für die Gruppenarbeit bzw. den Datenaustausch konzipiert wurden, wie z.B. der BSCW-Server des Projekts *Basic Support for Cooperative Work* (BSCW) [Bentley95]. Lernplattformen zeichnen sich gegenüber solchen Systemen durch eine leistungsfähigere Administration, verschiedene Kommunikationsmethoden und einem größeren Repertoire an Werkzeugen für das Lernen aus.

Der funktionale Aufbau einer Lernplattform lässt sich auch grafisch darstellen, wie Abbildung 6.1 zeigt.

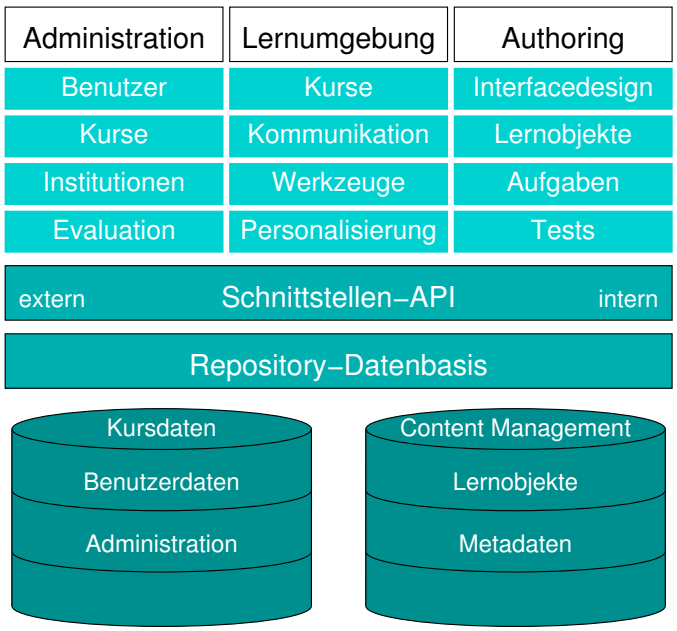


Abbildung 6.1: Idealtypische Architektur einer Lernplattform nach [Schulmeister03, S. 11]

Demnach besteht eine Lernplattform im Wesentlichen aus drei Schichten. Die erste Schicht hält alle Daten für den Betrieb vor, wie z.B. die Lernobjekte und Benutzerdaten. Darüber

liegt Schicht Zwei mit den Programmierschnittstellen (API) für den Zugriff auf die wesentlichen Funktionen. Auf oberster Ebene liegt Schicht Drei, die zur Visualisierung der Daten und zur Steuerung der Lernplattform dient. Bei normalem Betrieb arbeiten alle Rollen mit den Werkzeugen aus der Schicht Drei und nur bei Anpassungen oder Erweiterungen kann es vorkommen, dass auch auf die darunter liegenden Schichten zugegriffen werden muss.

Neben dem Begriff Lernplattform gibt es noch weitere, die sich meist nur in Nuancen unterscheiden. Der gängige Begriff im anglophonen Gebiet ist *Learning Management System* (LMS). Speziell in England wird aber auch oft von *Virtual Learning Environments* (VLE) gesprochen, wenn die didaktische Ausrichtung hervorgehoben werden soll [Britain00]. Ein Beispiel für eine Klassifizierung von VLEs findet sich in [Milligan00]. Hier werden klassische, lernzentrierte und kollaborative VLEs unterschieden sowie deren Erweiterungen. Die klassischen VLEs gliedern die Lerninhalte hierarchisch: Ein Kurs wird in Lektionen aufgeteilt, die wiederum aus Seiten bestehen und Übungen enthalten. Abschließend werden Tests angeboten, bevor es zum nächsten Thema geht, sodass die Sequenzierung überwiegend linear-sukzessiv ist (siehe Abschnitt 3.4). Lernzentrierte VLEs zeichnen sich durch die Unterstützung von Methoden der Projektarbeit aus [Schulmeister01]. Kollaborative VLEs erlauben mehreren Personen an gemeinsamen Projekten oder Objekten zu arbeiten. Es handelt sich um ein Netzwerk verschiedener Programme, die z.B. über das Internet miteinander verbunden sind [Schwabe01; Wessner00]. Als Erweiterungen werden alle Programme betrachtet, die VLEs um zusätzliche Spezialfunktionen ergänzen, z.B Editoren, Animationswerkzeuge und Programme für Videokonferenzen.

Eine andere Kategorisierung bringt die Definition von Brandon Hall [Hall00]. Demnach ist ein LMS hauptsächlich für die Administration der Lerninhalte und der Anwenderdaten zuständig. Können die Inhalte zusätzlich verändert und zusammengestellt werden, handelt es sich um ein *Integrated Learning Management System* (ILS). Ein ähnlicher Begriff ist das *Learning Content Management System* (LCMS), der eine Vermischung von *Content Management System* (CMS) und LMS ist. Schulmeister hält all diese Verfeinerungen jedoch nicht für sinnvoll:

> „Diese Begriffsunterscheidung zwischen LMS und ILS ist nicht wirklich trennscharf, selbst die Unterscheidung von Content Management System (CMS) oder Learning Content Management System (LCMS) und LMS ist nicht sehr hilfreich. Die Integration eines Editors für Autoren ist nicht konstitutiv für ein LMS, das Authoring kann auch mit einem externen Editor erfolgen.“ [Schulmeister03, S. 14,15]

6.2 Evaluation

Die Auswahl der geeigneten Lernplattform ist ein komplexes und zeitaufwendiges Unterfangen. Zuerst sollte der eigene Bedarf bestimmt werden, um die Kriterien für das benötigte System festzulegen. Hiernach erfolgt die Recherche nach den Lernplattformen und ihren Funktionen. Es sollten möglichst viele der aufgestellten Kriterien erfüllt sein, um etwaige Erweiterungen zu sparen. Aufgrund des riesigen Angebots an Lernplattformen, sollte auf bereits vorhandene Untersuchungen zurückgegriffen werden. Die ständige Weiterentwicklung, gelegentliche Übernahme durch andere Firmen oder die komplette Einstellung von Produkten bereitet auch Experten arge Probleme. Offensichtlich muss die Evaluation von Lernplattformen bis zur Kaufentscheidung im Auge behalten werden, denn eine falsche Wahl kann erhebliche finanzielle Nachteile in sich bergen.

Eine ältere Recherche von Schulmeister [Schulmeister00] hat 108 Software-Produkte untersucht, die in der Stichprobe „Evaluation von Lernplattformen“ (EVA:LERN) [Schulmeister03] auf 171 aufgestockt wurde. Brandon Hall führt in seiner Studie [Hall03] 72 Systeme an, von denen 44 auch in EVA:LERN vorkommen. Aus dem gleichen Hause gibt es zusätzlich eine Studie über LCMS [Chapman03]. In der Untersuchung von Baumgartner [Baumgartner02a] werden 133 Lernplattformen angegangen. Anhand dieser ausgewählten Arbeiten wird bereits

offensichtlich, wie kompliziert und subjektiv die Auswahl eines geeigneten Systems ist. Im folgenden werden die Lernplattformen **Blackboard 6**[1], **WebCT**[2] und **smartBLU**[3] vorgestellt, weil sie bei dieser Arbeit zur Verfügung standen. Der Installationsaufwand einer Lernplattform ist enorm und kann nur von Experten mit Erfahrungen auf diesem Gebiet durchgeführt werden.

6.2.1 Blackboard

Blackboard Inc. bietet die Lernplattform *Blackboard* an, die aus *CourseInfo* hervorgegangen ist. Bei der aktuellen Version *Blackboard 6* handelt es sich um verschiedene Komponenten, die nach den eigenen Bedürfnissen kombiniert werden können. An dieser Stelle wird die **Blackboard Academic Suite** näher betrachtet, die sich aus den Systemen **Blackboard Learning System**[4], **Blackboard Content System** und **Blackboard Portal System** zusammensetzt.

Das *Blackboard Learning System* ist ein Web-basiertes Werkzeug zur Verwaltung von Kursen, mit dem Lehrende eigene Lehrpläne erstellen und ihre zugehörigen Inhalte abspeichern. Für die Überprüfung des Lernerfolgs unterstützt das System die Erstellung und den Einsatz von Tests. Lernende profitieren von virtuellen Klassenräumen, in denen die Kommunikation und Zusammenarbeit gefördert werden. Sollte *Blackboard* eine gewünschte Funktion nicht anbieten, so kann es über die **Building Blocks** erweitert werden. Hierbei handelt es sich um eigene Anwendungen oder Ergänzungen, die mit einem speziellen *Software Development Kit* (SDK) entwickelt wurden. Über spezielle APIs erlangen die selbst entwickelten Module den vollen Zugriff auf *Blackboard*. Ein gutes Beispiel ist der *Building Block* zum Einlesen von SCORM-Dateien, durch den *Blackboard* erst standardkompatibel wird. Abbildung 6.2 zeigt einen typischen Screenshot des LMS.

Bei dem *Blackboard Content System* handelt es sich um eine Erweiterung, die Autoren/-innen bei ihrer Arbeit unterstützt. So lassen sich Inhalte versionieren, Änderungen überwachen und Arbeitsabläufe bestimmen. Einmal erstellt, können Dateien zentral gehalten und in verschiedenen Kursen eingesetzt werden. Einen ähnliche Funktionalität wird auch den Studierenden angeboten. In „virtuellen Speicherbereichen“ können sie eigene Daten halten, auf die sie über *Blackboard* zugreifen können.

Mit dem *Blackboard Portal System* lässt sich ein anpassbares Portal für Firmen und Universitäten aufbauen, das die Lernplattform mit anderen Diensten in einer einheitlichen Oberfläche vereint.

6.2.2 WebCT

WebCT Inc. bietet die beiden Ausführungen **WebCT Campus Edition** und **WebCT Vista** ihres E-Learning-Systems für die Hochschulausbildung an. Die **WebCT Campus Edition** ist ein System zur Erstellung, Verwaltung und Nutzung von Kursen. Abbildung 6.3 zeigt eine typische Präsentation von Inhalten.

Den Lehrenden wie Lernenden werden diverse Werkzeuge an die Hand gegeben, mit denen der Zugriff auf das System vereinfacht wird. So lassen sich alle gängigen Kursinhalte, wie z.B. Texte, Bilder, Videos und Quiz, per *Drag'n'Drop* einstellen. Lehrende können Lernpfade vorgeben, indem sie Leistungskriterien aufstellen, die mit Tests auf Basis von *Multiple Choice* und offenen Aufgaben überprüft werden. Der gesamt Lernfortschritt lässt sich festhalten, sodass sich auch Rückschlüsse auf die Qualität der Kurse ziehen lassen. So können die Inhal-

[1] `http://www.blackboard.com` (29.10.05) Mein besonderer Dank gilt Prof. Dr. Hans-Jürgen Appelrath und dem „Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und -Systeme“ (OFFIS) für die Nutzung der Installation.

[2] `http://www.webct.com` (29.10.05) Mein besonderer Dank gilt dem „Multimedia Kontor Hamburg“ (MMKH) für die Nutzung der Installation.

[3] `http://www.smartblu.de` (29.10.05)

[4] `http://www.blackboard.com` (29.10.05)

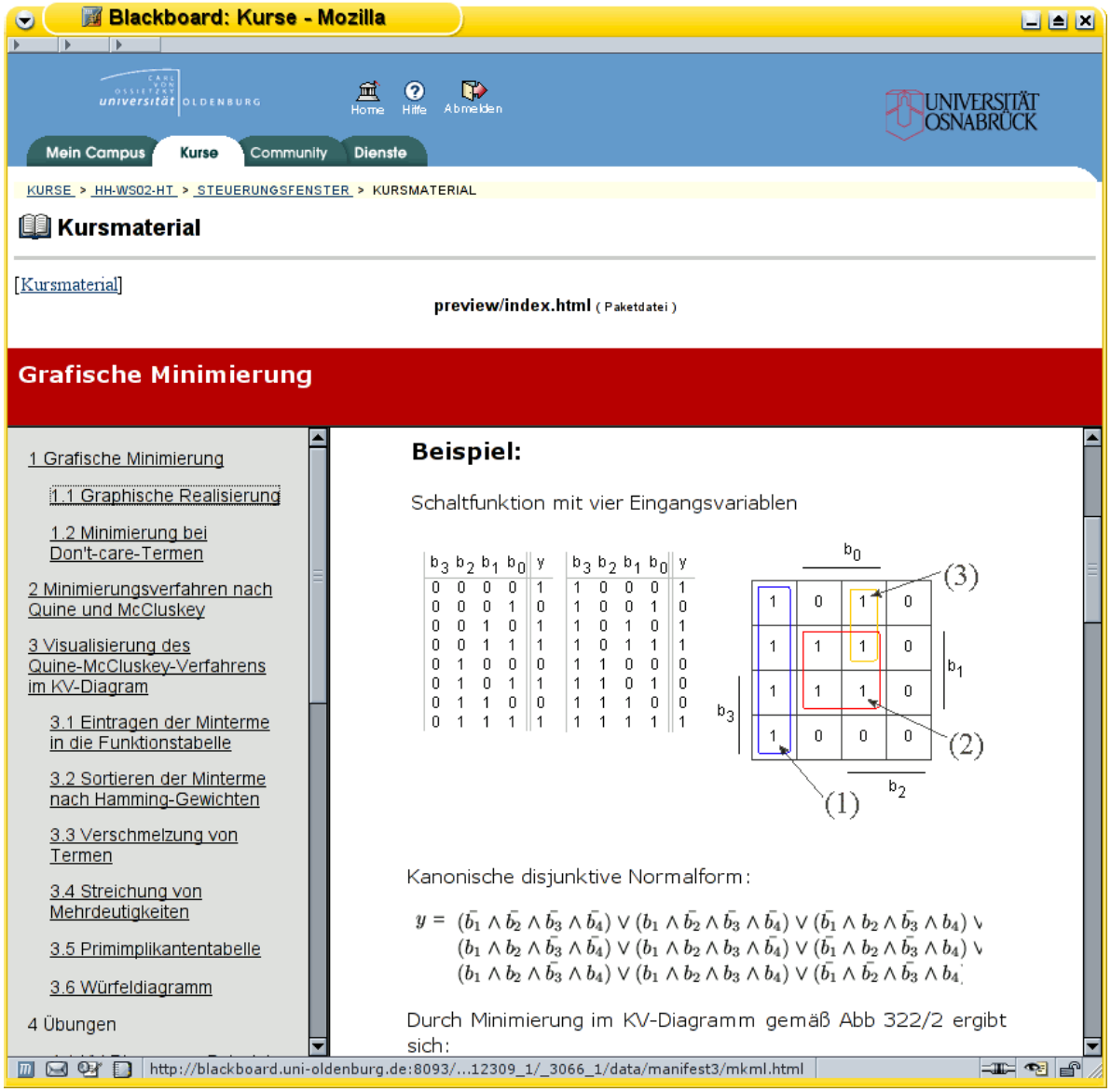


Abbildung 6.2: Screenshot von *Blackboard*

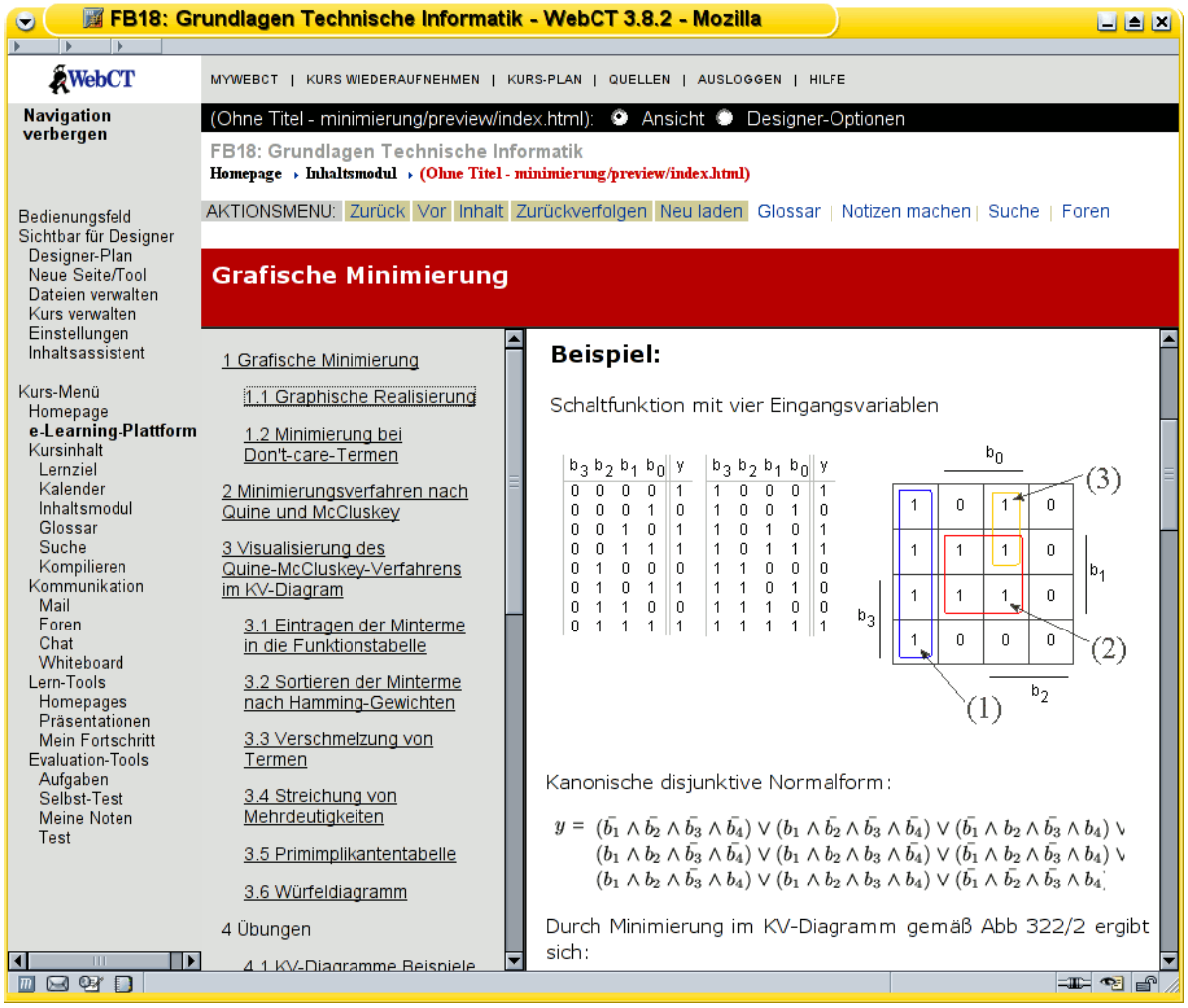


Abbildung 6.3: Screenshot von *WebCT*

te ständig verfeinert und verbessert werden, um optimale Lernerfolge zu erreichen. Für die Kommunikation der Lernenden werden *Chat*, *Whiteboard*, E-Mail, und Foren angeboten.

Mit dem WebDAV-Zugang haben Autoren/-innen einen flexiblen Zugang zu *WebCT*, über den Inhalte schnell eingespielt und verändert werden können. Moderne Betriebssysteme können WebDAV bereits nahtlos integrieren, sodass auch nicht unterstützte Autorenwerkzeuge direkten Zugriff haben. Mit einem sehr einfachen HTML-Editor können Seiten auch direkt in *WebCT* bearbeitet werden, wovon jedoch abzuraten ist, weil die Funktionen zu rudimentär sind.

WebCT Vista ist für größere Installationen ausgelegt, bei denen z.B. verschiedene Internetauftritte einer Universität auf einem System betrieben werden. Diese Ausführung von *WebCT* steht dieser Arbeit leider nicht zur Verfügung und kann nicht weiter betrachtet werden.

6.2.3 SmartBLU

Das LMS *SmartBLU* ist aus dem System *CMS-W3* hervorgegangen und wird seit 1996 vom *Fraunhofer Institut für Graphische Datenverarbeitung* (IGD) entwickelt. Im Gegensatz zu den anderen vorgestellten Systemen beschränkt sich *SmartBLU* auf die Präsentation von Lernmaterialien und nutzt *Chat* sowie Foren als Kommunikationsmittel. Obwohl Portale, *Organizer* sowie andere Funktionen nicht zum Leistungsumfang gehören, ist es dennoch eine interessante Alternative, da es von den hier vorgestellten LMS die beste Standardunterstützung anbietet. So kann *SmartBLU* selbständig eine Navigation aus einem Manifest generieren, was bei den Konkurrenzprodukten nicht ohne weiteres geht. Abbildung 6.4 zeigt einen exemplarischen Screenshot des Systems.

SmartBLU hat ein eigenes Rollenkonzept und unterscheidet zwischen **Lerner**, **Betreuer**, **Fachautor** sowie **Administrator**. Die Rolle Lerner nutzt das System zum lernen und kann verschiedene Kurse belegen. Eine Kontrolle des Lernerfolgs wird über Tests angeboten, von denen es Lückentexte, Zuordnungsaufgaben, Multiple- und Single-Choice-Aufgaben gibt. Der Betreuer begleitet die Lernenden und kann die Ergebnisse der Tests überprüfen. Für die Erstellung der Inhalte sind die Fachautoren zuständig, die sich um Konzeption, Gestaltung und Implementierung kümmern. Alle Verwaltungsaufgaben, wie z.B. die Benutzerverwaltung, werden von der Rolle Administrator durchgeführt.

Die Strukturierung der Kurse orientiert sich an der „Buchmetapher“, sodass sich Lerninhalte wie in einem Buch in Kapitel und Abschnitte aufteilen. Die kleinste Einheit sind Module, z.B. Texte, Grafiken sowie Animationen, und entsprechen im Sinne der Metapher einem Abschnitt. Ein Modul kann in mehreren Kursen eingesetzt werden, wodurch die Wiederverwendbarkeit gefördert wird.

6.3 Bewertung

Im Abschnitt 6.1 wurde bereits darauf hingewiesen, dass die heutigen Lernplattformen den Ansprüchen bzw. den Definitionen nicht vollends genügen. Sicherlich darf die Komplexität eines solchen Systems nicht unterschätzt werden, aber bereits bei der Basisfunktionalität, dem Präsentieren der Lernmaterialien, gibt es Gründe zur Kritik. Für einen besseren Vergleich listet Tabelle 6.1 einige Merkmale der drei vorgestellten Lernplattformen auf.

Es beginnt mit dem Datenaustausch zwischen Autorensystem und Lernplattform, wie in [Bungenstock03a; Bungenstock03b] genauer untersucht wurde. Nur *WebCT* ermöglicht mit WebDAV unter den vorgestellten Lernplattformen eine einfache Anbindung. Die anderen Produkte bieten lediglich umständliche Web-Oberflächen an, was bei vielen Änderungen und Anpassungen, wie sie besonders während des Erstellungsprozesses neuer Lernmaterialien auftreten, schnell lästig werden kann. Wird noch in Betracht gezogen, wie viele Dateien einem vollständigen Kurs in HTML angehören, zeigt sich schnell das Ausmaß der Arbeit. Auch Lösungen mit gepackten Dateien, wodurch lediglich eine Datei hochgeladen werden muss, können

Abbildung 6.4: Screenshot von *SmartBLU*

höchstens als „fauler Kompromiss“ gewertet werden. Die Verbindung zwischen Autorensystem und Lernplattform muss transparent erfolgen, daran führt kein Weg vorbei.

Grundsätzlich zeigt sich, dass Lernplattformen nur bedingt für den Erstellungsprozess von Lernmaterialien geeignet sind. Hier hebt sich *Blackboard* mit seinem *Content System* von den anderen Produkten ab, da es Versionierung und Verknüpfungen von Dateien anbietet. So muss ein Baustein mit einem Rechtschreibfehler, der bereits in vielen Kursen eingesetzt wurde, nur ein Mal angepasst werden. Durch die Verknüpfung „propagiert“ sich die Änderung durch.

Auch die Unterstützung der Standards könnte besser sein. Viele Lernplattformen beschränken sich auf das Entpacken und direkte Anzeigen von Dateien in den Formaten SCORM oder *IMS Content Packaging*. Nur *SmartBLU* ist in der Lage, aus den Strukturinformationen des enthaltenen Manifests eine Navigation zu generieren. Bei den anderen Lernplattformen muss dies von den Autoren/-innen erledigt werden, was nicht nur Mehrarbeit verursacht, sondern auch schlechtere Ergebnisse. Neben der eigenen Navigation des LMS muss parallel die des Kurses untergebracht werden, wie es deutlich in den Screenshots der Abbildungen 6.2 und 6.3 zu erkennen ist. Letztendlich bleibt für die eigentlichen Inhalte weniger Platz auf dem Bildschirm. Ein anderes Problem dieser Darstellung ergibt sich aus der Verwendung von *Frames* bei HTML, wodurch die *Bookmark*-Funktionalität des Browsers nicht genutzt werden kann.

Leider wird von keiner Lernplattform die Übersetzung von XML-Dateien in Formate wie beispielsweise HTML oder PDF angeboten. Es werden auch keine Alternativen zur Trennung von Inhalt und Layout angeboten, einer wichtigen Voraussetzung für die Wiederverwendung von Inhalten aus unterschiedlichen Quellen. Hier muss bei allen Systemen nachgebessert werden.

System	Anbindung	Standards	Kommunikation	Organization	Gestaltung
Blackboard	Web-Formular	SCORM, IMS	Chat, Foren, Whiteboard, E-Mail	Kalender, Terminplan, Adressbuch	Unterstützung von Fremdsprachen, Layout beschränkt anpassbar
WebCT	WebDAV, Web-Formular	SCORM, IMS, AICC	Chat, Foren, Whiteboard, E-Mail	Kalender, Aufgabenliste	Unterstützung von Fremdsprachen, Layout beschränkt anpassbar
SmartBLU	Web-Formular	SCORM, AICC	Chat, Foren, Info-Bord		Unterstützung von Fremdsprachen, festgelegtes Layout, Anpassung der Oberflächengröße

Tabelle 6.1: Übersicht der Lernplattformen

Die Kommunikations- und Organisationsmöglichkeiten sind bei allen Systemen sehr gut, da gibt es nicht viel zu beanstanden. Zudem hat sich gezeigt, dass sie alle ihre Stärken für bestimmte Aufgaben haben. Ließen sie sich zu einem System vereinen, was technisch leider nicht machbar ist, dann wäre das Ziel einer ausgereiften Lernplattform schon näher. Es bleibt bei den kommerziellen Systemen nichts anderes übrig, als auf entsprechende Erweiterungen der Hersteller zu warten.

Kapitel 7

Web-Technologie

Im vorherigen Kapitel 6 über Lernplattformen hat sich gezeigt, dass ihre Funktionalität auf die tägliche Lehre ausgerichtet ist. Für eine Archivierung von Lernmaterialien, gleich welcher Form, eignen sie sich hingegen nur bedingt, denn oft sind die Verwaltungs- und Suchmöglichkeiten auf einfache Operationen beschränkt. Daher wird ein spezielles Datenhaltungssystem für Lernobjekte benötigt, das in dieser Arbeit als **Repository** bezeichnet wird. Zu seinen wesentlichen Eigenschaften gehört eine direkte Anbindung an Lernplattformen sowie Autorensysteme. Das Repository koordiniert die Arbeit und den Datenaustausch in Teams, sodass es als zentrale Komponente zur Verfügung stehen muss.

Zur Zeit ist leider kein entsprechendes System für modulare E-Learning-Inhalte erhältlich, weshalb seine Entwicklung als Teil dieser Arbeit abzusehen ist. Weil Java als Programmiersprache bereits feststeht und das Repository über ein Rechnernetzwerk angesteuert werden soll, wird in diesem Kapitel die Verfügbarkeit existierender Server, Rahmenwerke sowie Libraries erörtert.

Grundlage für alle heutigen Netzwerk-Anwendungen ist das Protokoll TCP/IP [Stevens94; Wright95], welches jedes moderne Betriebssystem von Haus aus beherrscht. Für die Übertragung von HTML-Seiten setzt das Protokoll HTTP [Stevens96; Gourley02] auf TCP/IP auf und muss somit vom Repository unterstützt werden. Obwohl HTTP vom Aufbau her recht einfach ist und mit den Klassen der *Java*-Standard-Library leicht umzusetzen ist, soll auf fertige Lösungen zurückgegriffen werden. Im nächsten Abschnitt werden verschieden Produkte vorgestellt und ihre Vorteile sowie Schwächen herausgearbeitet. Wesentlicher Gegenstand der Betrachtung sind die verschiedenen etablierten Schichtenmodelle, die in Abschnitt 7.1 vorgestellt werden.

Die Steuerung des Repositories über einfache HTML-Seiten wird nicht möglich sein. Vielmehr wird eine **Web Application** benötigt, also ein richtiges Programm, das HTML-Seiten zur Repräsentation der Daten einsetzt und die so genannte Geschäftslogik in Komponenten auslagert. Für die Erstellung von *Web Applications* mit *Java* gibt es bereits eine Reihe von fertigen Rahmenwerken, die in Abschnitt 7.2 vorgestellt werden.

Für die Steuerung der *Web Application* über RPC bietet sich geradezu ein Web-Server an. Das auf HTTP aufsetzende Protokoll *Simple Object Access Protocol* (SOAP) ist ideal für diese Aufgabe und soll daher die Fernsteuerung ermöglichen. Neben der Spezifikation *Java API for XML Messaging* (JAXM) von Sun, die eine rudimentäre Ansteuerung dieses Protokolls ermöglicht, gibt es auch Programmpakete, die eine Nutzung dieser Technologie auf einem höheren Abstraktionsniveau erlauben. In Abschnitt 7.3 werden einige erhältliche Produkte vorgestellt.

Als letzte Funktion des Repositories soll kurz der Datenaustausch über *WebDAV* erläutert werden. Weil *WebDAV* ebenfalls auf HTTP aufbaut, soll wieder eine integrierte Lösung gefunden werden. Abhängig vom ausgewählten Web-Server stehen hier verschiedene Module zur Auswahl.

7.1 Infrastruktur

Eine Errungenschaft der Software-Technik ist die Wiederverwendung von Modulen bzw. Komponenten, wodurch die Entwicklungszeit reduziert und die Stabilität des Produkts erhöht wird. Bei einer komplexen Anwendung wie einer *Web Application* gibt es eine Vielzahl potentieller Kandidaten, die praktisch in jeder *Web Application* auftauchen. Die Firma Sun hat diesen Umstand zum Anlass genommen, neben der bekannten *Java 2 Platform, Standard Edition* (J2SE) für gängige Applikationen, die *Java 2 Platform, Enterprise Edition* (J2EE) zu veröffentlichen [Shannon04]. Sie definiert einen Standard für komponentenbasierte mehrschichtige Unternehmensanwendungen, die Web-Technologien einsetzen. Abbildung 7.1 zeigt die grundsätzlichen Schichten einer Anwendung, auf die folgend eingegangen wird.

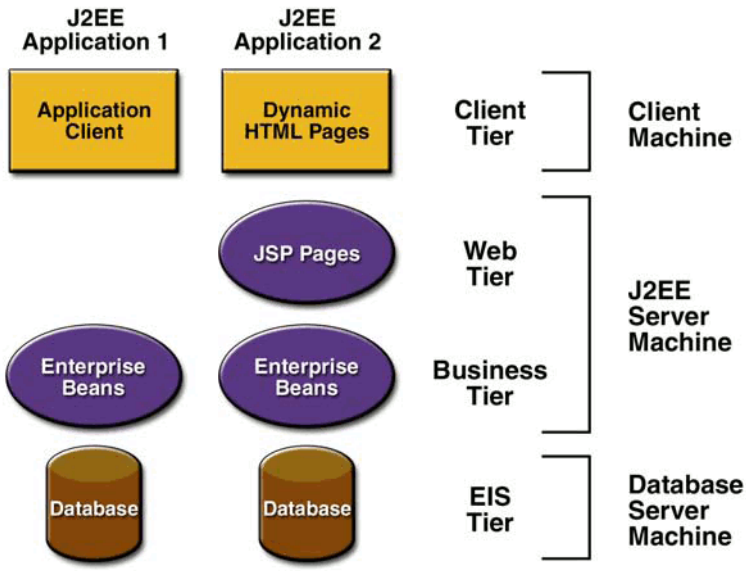


Abbildung 7.1: Schichten von J2EE-Anwendungen [Bodoff04, S. 3]

Ganz unten steht die Datenhaltung, die hier als Datenbank dargestellt ist. In den meisten Fällen wird es sich in der Tat um eine relationale Datenbank handeln, aber die Daten können auch im Dateisystem oder einer anderen Datenhaltungsform gespeichert sein. Wichtig ist lediglich die zur nächsten Schicht präsentierte Schnittstelle, die den Komponenten mit der Geschäftslogik, hier als *Enterprise Beans* bezeichnet, einen standardisierten Zugriff ermöglicht. Abhängig von der Art des Clients, entweder handelt es sich um eine Anwendung (in der Abbildung als *Application Client* bezeichnet) oder dynamische HTML-Seiten in einem Browser, sitzt über den Enterprise Beans eine Schicht mit *Java Server Pages* (JSP). Kurz umrissen ist eine JSP eine Schablone, die aus HTML-Fragmenten und *Java*-Befehlszeilen besteht. Auf diese Weise werden Inhalt und Logik der *Enterprise Beans* in die Darstellung integriert. Soll z.B. ein bestimmter Wert angezeigt werden, der in einer *Enterprise Bean* gespeichert ist, genügt ein kurzer Befehl in *Java* für das Auslesen und Umwandeln in HTML. Dieser Schritt entfällt bei eigenständigen Anwendungen, da sie sich um die Darstellung selbst kümmern müssen. Auf diese Weise soll z.B. das Autorensystem auf Lernobjekte zugreifen, indem es sie vom Server herunterlädt und in einer grafischen Oberfläche anzeigt. Zu Kontrollzwecken soll das Repository aber auch ohne Autorensystem genutzt werden, sodass beide J2EE-Architektur-Modelle für modulare E-Learning-Inhalte benötigt werden.

Die J2EE umfasst eine Vielzahl von Technologien, Modellen und APIs, auf die bisher noch nicht eingegangen wurde. Aufgrund der Komplexität kann dies im Rahmen dieser Arbeit auch nicht vollständig geschehen. Weil J2EE für eine große Zahl von Anwendungen konzipiert wurde, ist es sehr vielschichtig. In der Praxis wird jedoch häufig nur ein Bruchteil der

Funktionalität benötigt, was besonders bei unerfahrenen Entwicklern/-innen zu Konfusionen führt. Welche Schichten sind wichtig und mit welcher Technologie erfolgt die Umsetzung? Wer nicht den gesamten Umfang von J2EE kennt, läuft leicht Gefahr, vorhandene Lösungen selbst zu implementieren oder kompliziertere Wege als nötig einzuschlagen. Auch für das angestrebte Repository wird nicht der volle Funktionsumfang benötigt, sodass sich der Entwurf durch Weglassung einzelner Schichten vereinfachen lässt.

Bei den *Enterprise Beans* handelt es sich um ein Komponentenmodell, das für die Implementierung der gesamten Funktionalität genutzt wird. Alle technischen Details werden hierbei vom Komponenten-Container gekapselt, wodurch die Geschäftslogik in den Vordergrund rückt. Es gibt einige Merkmale bei Anwendungen, die den Einsatz von *Enterprise Beans* anzeigen. Muss das System mit der Anzahl von Benutzer/-innen skalieren, also über mehrere Server verteilt werden oder sollen Transaktionen unterstützt werden, dann sind *Enterprise Beans* die geeignete Wahl.

In Hinblick auf das Repository werden die Eigenschaften der *Enterprise Beans* wohl nicht benötigt. Abbildung 7.2 zeigt daher die beiden schematischen Schichten des J2EE-Servers an, die für die Umsetzung des Repositories relevant sind.

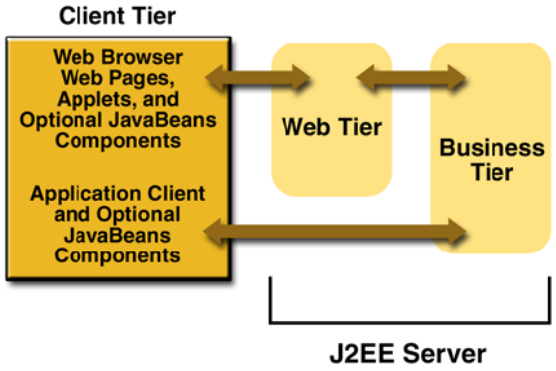


Abbildung 7.2: Client und Server [Bodoff04, S. 6]

Es gibt zwei Arten von Clients, die das Repository unterstützen soll: Web-Browser, die über die *Web Tier* auf die *Business Tier* zugreifen und selbst geschriebene Programme mit direktem Zugriff. Wie viel der Server leisten muss, hängt von den verwendeten Schichten ab. Ein Server für *Enterprise Beans* in der *Business Tier* ist wesentlich anspruchsvoller und umfangreicher als einer, der lediglich die *Web Tier* unterstützt. Da für die *Business Tier* die entwickelten Komponenten aus den vorherigen Kapiteln zum Einsatz kommen, genügt für das Repository ein Web- Server als J2EE-Umgebung. Für ein besseres Verständnis sind in Abbildung 7.3 sechs Schritte aufgeführt, die von der Infrastruktur zu leisten sind. Eine Anwendung als Client sieht im Prinzip gleich aus, nur kann hier der Schritt 3 entfallen.

Zunächst stellt der Client eine Anfrage an den Server über das Protokoll HTTP, den so genannten *HTTP Request*, der als Schritt 1 deklariert ist. Obwohl HTTP fünf unterschiedliche Methoden[1] kennt, treten in der Praxis überwiegend `GET`- und `POST`-Anfragen auf, z.B. wenn Daten über eine Formular-Seite gesammelt und übertragen werden. Auf der Server-Seite werden die Daten von einem Web-Server entgegen genommen, in ein *HTTPServletRequest* umgewandelt und in Schritt 2 an eine JSP oder ein *Servlet* weitergereicht. Ein *Servlet* ist eine *Java*-Klasse mit genau definierter Schnittstelle, die in einem *Container* eingebettet ist und die Anfragen des Clients verarbeitet. Neben den übertragenen Parametern werden dem *Servlet* zusätzlich eine Reihe von kontextabhängigen Daten, wie z.B. Cookies, IP-Adresse und User-Daten übergeben. JSPs und *Servlets* können in Schritt 3. die Daten auf zwei mögliche Weisen verarbeiten. Neue sowie veränderte Daten werden an die *Java Beans* übergeben und

[1] `HEAD`, `GET`, `POST`, `PUT` und `DELETE`

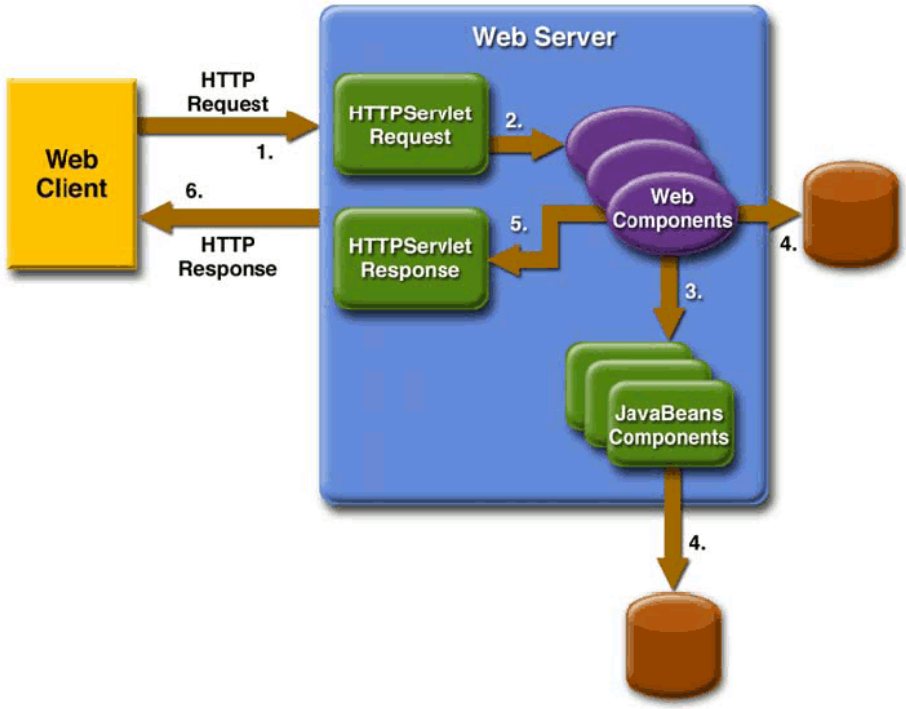


Abbildung 7.3: Sechs Schritte einer Anfrage [Bodoff04, S. 84]

anzuzeigende Daten ausgelesen. Bei *Java Beans* handelt es sich um Klassen, deren Schnittstelle einer genauen Definition unterliegt [Englander97]. Wie die Daten persistent gehalten werden, ist Bestandteil des Schritts 4, und kann z.B. über Datenbanken, XML-Dateien oder Serialisierung erfolgen. Der direkte Zugriff von den *Web Components* auf die Datenhaltungsschicht, ebenfalls als Schritt 4 bezeichnet, ist zwar theoretisch möglich, führt aber zu einer sehr engen Verzahnung die sich nachteilig auswirkt. Wenn z.B. SQL-Anweisungen direkt in eine JSP integriert sind, wird praktisch die Trennung zwischen Darstellung und Datenhaltung aufgehoben, sodass spätere Erweiterungen, Anpassungen, Fehlerbehebungen, etc. beeinträchtigt werden. Aus diesem Grund soll in dieser Arbeit die Kommunikation stets zwischen *Web Components* und *Java Beans* erfolgen. Nachdem alle Berechnungen und Operationen durchgeführt wurden, wird in Schritt 5 das Ergebnis in Form eines *HTTPServletResponse* aufbereitet und an den Web-Server übergeben. Der schickt dem Client in Schritt 6 per *HTTP Response* eine anzeigbare HTML-Seite.

Aus diesem Ablauf lässt sich gut ableiten, was für die Umsetzung des Repositories benötigt wird, nämlich ein Web-Server mit einem *Container* für *Web Components*. Um die Möglichkeiten der *Web Components* differenzierter darzustellen, ist diese Schicht in Abbildung 7.4 weiter aufgegliedert.

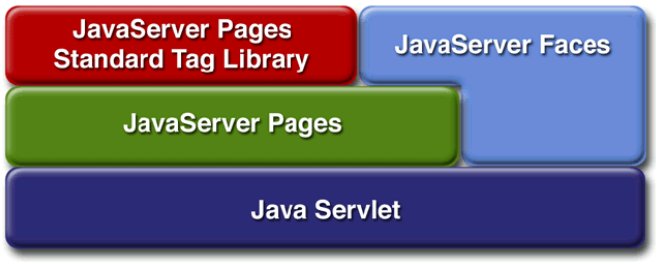


Abbildung 7.4: Schichten der Repräsentation [Bodoff04, S. 85]

Die Basis aller zur Verfügung stehenden Technologien sind die *Servlets*. Sie ermöglichen eine sehr genaue Steuerung der Vorgänge, setzen aber umfangreiche Kenntnisse voraus. Um die Arbeit zu vereinfachen, gibt es abstraktere Mechanismen wie die *Java Server Pages*, die eine Verquickung von HTML und *Java*-Code ermöglichen. Technisch gesehen, werden JSPs wiederum zu *Servlets* übersetzt. Weil auch JSPs meist nicht um eine Programmierung in *Java* umhin kommen, wenn z.B. die Kommunikation mit den *Java Beans* erfolgt, wurden die *Standard Tag Library* eingeführt. JSPs lassen sich nämlich um eigene *Tags* erweitern, sodass sich auch anspruchsvollere Operationen kapseln lassen. Im optimalen Fall kann eine Person ohne *Java*-Kenntnisse JSPs erzeugen und auf *Java Beans* zugreifen, ohne programmieren zu müssen. Die *Java Server Faces* sind eine recht neue Technologie und orientieren sich an der klassischen Programmierung grafischer Oberflächen. Es gibt einzelne Komponenten für Benutzerinteraktionen, die sich beliebig kombinieren lassen und auf *Events* reagieren. Praktisch gesehen, abstrahiert dieser Mechanismus die drei anderen Schichten, indem die eingesetzten Techniken weitestgehend verdeckt werden.

Stellt sich noch die Frage, wie eng die einzelnen Komponenten mit dem Web-Server verbunden sind. Für das Repository lässt sich bereits absehen, dass es aus vielen Klassen, JSP und anderen Dateien bestehen wird, die sich auf diverse Verzeichnisse verteilen. Über eine Konfiguration wird festgelegt, wie diese Dateien in Beziehung stehen und welche Aufgabe sie haben. Wie viel Einfluss nimmt aber die eingesetzte Software? Muss bei einem Wechsel des Web-Servers die gesamte Anordnung und Konfiguration angepasst werden? Die Antwort lautet „Nein“, denn die Spezifikation von J2EE sieht auch diesen Fall vor. Alle benötigten Dateien lassen sich in einem Paket zusammenfassen und in einen Web-Server *„deployen“*. Dieser Begriff hat sich durchgesetzt und wird auch in anderen Kontexten verwendet, in dem Komponenten oder Pakete integriert werden. Abbildung 7.5 zeigt die vorgeschriebene interne Struktur eines solchen Pakets.

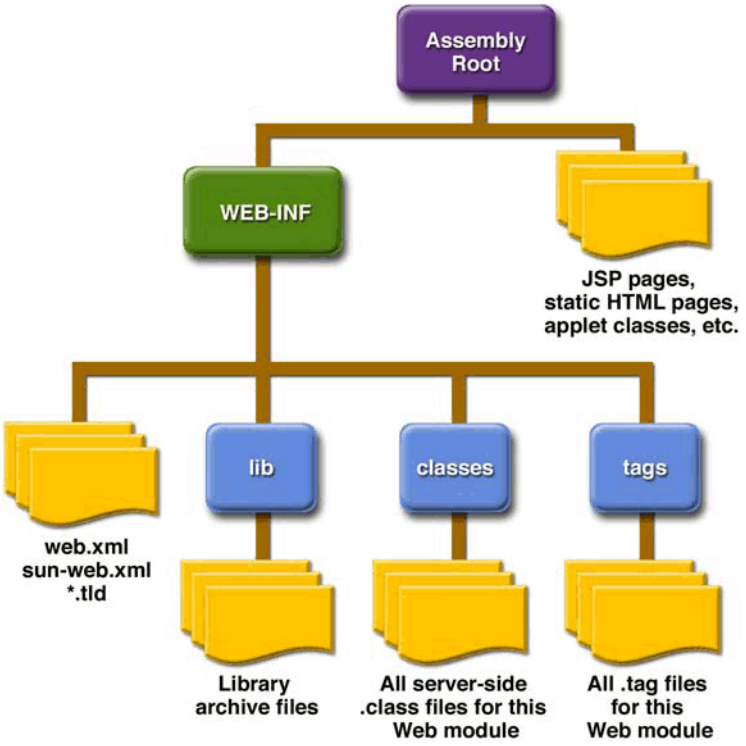


Abbildung 7.5: Interne Modulstruktur [Bodoff04, S. 90]

An oberster Stelle liegt das Hauptverzeichnis, das hier als *Assembly Root* bezeichnet ist. Der Verzeichnisname kann beliebig gewählt werden und wird bei manchen Web-Servern zum Bestandteil der späteren Aufruf-URL. Um als Paket zu gelten, muss es mindestens das Verzeichnis *WEB-INF*, das genau so geschrieben sein muss, und die Datei *web.xml* enthalten. Sie

wird oft auch *Deployment Descriptor* genannt, weil sie beschreibt, wie welche Komponenten verbunden und adressiert werden. Eigene Libraries werden im Verzeichnis *lib* abgelegt und lose Klassen in *classes*. Die enthaltenen Dateien werden automatisch in den Klassenpfad eingetragen, weshalb keine weitere Konfiguration notwendig ist. Im letzten Verzeichnis *tags* sind alle nötigen Dateien für die *Tag Libraries* enthalten, die ebenfalls automatisch eingelesen werden. Bleiben noch die JSP-Dateien, die sich innerhalb des Pakets beliebig positionieren lassen. Allerdings sollte das Verzeichnis *WEB-INF* ausgenommen werden, da nicht alle Web-Server diesen Ort für JSP unterstützen. Gepackt zu einer Datei, kann diese Struktur dann einfach in ein System integriert werden.

Bleibt zu klären, welche Produkte es überhaupt gibt. Neben vielen kommerziellen Anbietern, wie z.B. *WebSphere*[2] von IBM, *WebLogic*[3] von BEA und *WebObjects*[4] von Apple, gibt es auch einige frei erhältliche Produkte. Die Stärken der kommerziellen Systeme sind auf jeden Fall ihr hoher Leistungsumfang und die bessere Bedienbarkeit, die durch eine Reihe mitgelieferter Werkzeuge erreicht wird. Da nur die *Web Tier* von J2EE benötigt wird, genügt für das Projekt mαth-kit aber eine freie Lösung. Zur näheren Auswahl stehen hier *Tomcat*[5], *Jetty*[6] und *Resin*[7]. In Funktionsumfang und der Leistung stehen sich die Produkte im Großen und Ganzen in nichts nach.

7.2 Web Applications

Mit J2EE wird größeren Projekten eine Vielzahl von Techniken und Mechanismen angeboten, die eine strukturierte Gestaltung sowie Planung ermöglicht. Der generische Ansatz bereitet keine großen Einschränkungen, sodass nach eigenem Belieben vorgegangen werden kann. Durch Auslassen oder Hinzufügen bestimmter Teile sind der individuellen Umsetzung keine Grenzen gesetzt. Der Preis für diese Flexibilität ist die Suche nach dem eigenen geeigneten Vorgehen. Eine Möglichkeit ist die Verwendung des Entwurfsmusters MVC, das die Aufteilung des Systems in *Model*, *View* und *Controller* (MVC) vorgibt. Ursprünglich für grafische Anwendungen gedacht, hat es sich mit ein paar Anpassungen als Systemarchitektur für *Web Applications* durchgesetzt. Abbildung 7.6 verdeutlicht die Zusammenhänge zwischen den einzelnen Komponenten.

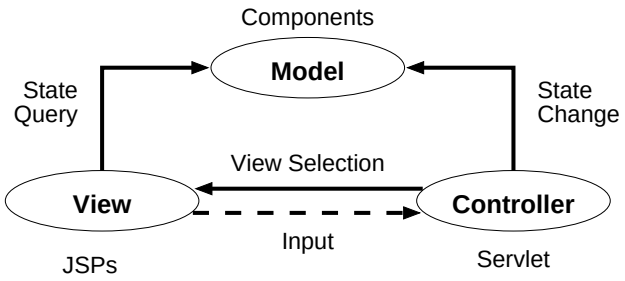


Abbildung 7.6: *Model*, *View* und *Controller* für *Web Applications*

In dieser Aufteilung sind *View* und *Controller* als Elemente der *Web Tier* realisiert, wobei von außen betrachtet lediglich der *Controller* angesprochen wird (als *Input* eingezeichnet). Jeder Aufruf einer URL geht somit direkt auf das *Servlet*, welches die Eingabe aufbereitet, die entsprechende Komponente des *Models* auswählt und den gewünschten Befehl aufruft. Die Komponente verarbeitet anschließend die übergebenen Daten und gibt einen Rückgabewert zurück, von dem der *Controller* seinen nächsten Arbeitsschritt abhängig macht. War z.B. eine

[2] `http://www-306.ibm.com/software/websphere` (29.10.05)
[3] `http://www.bea.com` (29.10.05)
[4] `http://www.apple.com/webobjects` (29.10.05)
[5] `http://jakarta.apache.org/tomcat` (29.10.05)
[6] `http://jetty.mortbay.org` (29.10.05)
[7] `http://caucho.com` (29.10.05)

Eingabe fehlerhaft oder konnte der Befehl aufgrund anderer Umstände nicht ordnungsgemäß durchgeführt werden, ruft das *Servlet* eine JSP für die erneute Eingabe oder eine Fehlerseite auf. Bei erfolgreicher Ausführung wird eine andere JSP ausgewählt, die über den weiteren Fortgang informiert. Unabhängig vom zurück gegebenen Status, benötigen annähernd alle JSP einen Zugriff auf die Daten des Modells, um den aktuellen Zustand anzuzeigen. Der Zugriff selbst ist nur lesend, weil andernfalls der *Controller* umgangen würde, was nicht gewünscht ist. Nachdem die JSP ausgeführt und als Ergebnis eine HTML-Seite produziert wurde, wird diese als View an den Client gesendet.

Das Entwurfsmuster MVC gibt einen Leitfaden, wie eine *Web Application* zu strukturieren ist, lässt die Implementierung aber offen. Auch das J2EE bietet keine Bordmittel an, die eine Entwicklung solcher Anwendungen vereinfacht. Als Lösungen dieses Problems bleiben somit entweder eine Eigenentwicklung oder der Einsatz eines existierenden Rahmenwerks übrig. Die Eigenentwicklung spielt ihren Vorteil bei kleinen Systemen aus, die sich nicht oft ändern. Hier kann eine kleine schnelle Lösung wesentlich effizienter sein als eine umfangreiche. Ist die Anwendung aber etwas größer und benötigt eine gewisse Flexibilität, dann sollte von diesem Ansatz Abstand genommen und stattdessen auf ein Rahmenwerk zurückgegriffen werden. Der Nachteil hierbei liegt in der Komplexität, die nicht zu unterschätzen ist. In dieser Arbeit werden kurz zwei Produkte vorgestellt: *Struts* und *Spring*.

Mit *Struts*[8] bietet die *Apache Group* ein Rahmenwerk an, dass sich perfekt in den ausgewählten Web-Server *Tomcat* integrieren lässt. Ein zentrales Konzept zur Konfiguration und Adaption sind die bereits erwähnten *Java Beans*. Ihre Schnittstelle ist so ausgelegt, dass sie zur Laufzeit ohne vorherige Bindung beim Kompilieren initialisiert und genutzt werden können. *Struts* nutzt diese Eigenschaften zur Konfiguration des *Controllers*, der durch eigene Klassen erweitert werden kann. Mit Hilfe einer XML-Datei werden die einzelnen Komponenten zur Laufzeit erzeugt und miteinander verknüpft. Aber auch zum Datenaustausch zwischen den Komponenten von *Struts* und den eigenen Klassen kommen *Java Beans* zum Einsatz. So werden Eingaben über Formulare in HTML Seiten automatisch in *Java Beans* umgewandelt und weitergereicht. Wenn gewünscht, überprüfen so genannte *Validatoren* die Werte auf Gültigkeit, indem sie vorher festgelegte Regeln anwenden. Weitere Informationen zu *Struts* finden sich z.B in [Turner03; Carnell03; Cavaness04].

Das Rahmenwerk *Spring*[9] geht indes einen wesentlichen Schritt weiter. Es ist selbst in Schichten eingeteilt, und schickt sich an, eine Alternative bzw. Ergänzung zu den Architekturen mit *Enterprise Beans* zu sein. Neben der Unterstützung des Entwurfsmusters MVC gibt es viele weitere Bausteine in *Spring*, mit denen sich *Web Applications* aufziehen lassen. Anstatt auf die umfangreichen aber technisch anspruchsvollen *Enterprise Beans* zuzugreifen, werden die Daten und die Geschäftslogik in einfachen *Beans* sowie POJOs[10] gehalten. Spezielle Klassen nach dem Entwurfsmuster *Factory* [Gamma95] ermöglichen die Trennung der Konfiguration von *Beans* und der Programmlogik. Auf diesem Prinzip beruht das gesamte Rahmenwerk. Dank der Flexibilität und Unabhängigkeit der einzelnen Komponenten können mit *Spring* von kleinen Web-Präsentationen bis zu Unternehmensanwendungen fast alle Projektformen realisiert werden. Mehr Informationen zu *Spring* finden sich z.B. in [Johnson04; Tate04].

7.3 Web Services

Web Services ermöglichen den Zugriff auf Daten und das Ausführen von Befehlen über etablierte Techniken. Mittlerweile sind die Spezifikationen, Standards und Implementierungen verschiedener Hersteller jedoch so vielfältig geworden, dass eine pauschale Aussage über die Fähigkeiten von *Web Services* schwer fällt. Grundsätzlich werden Daten über das *Simple Object Access Protocol* (SOAP) mit Hilfe des Internets übertragen. Obwohl es bei der physikalischen

[8] `http://struts.apache.org` (29.10.05)

[9] `http://www.springframework.org` (29.10.05)

[10] POJO steht für *Plain Old Java Object* und bezeichnet schlicht alle Klassen, die keine *Beans* sind.

Übermittlung der Daten keine Vorgaben gibt — der Einsatz per Mail wird z.B. von vielen Implementierungen unterstützt —, ist in der Praxis das Protokoll HTTP die erste Wahl. Zur Laufzeit werden bei SOAP interne Datenstrukturen zu XML übersetzt, übertragen und vom Kommunikationspartner zurück transformiert. Hierdurch ist SOAP unabhängig von jeglichen Programmiersprachen, benötigt aber spezielle Mechanismen, die eine Verbindung zwischen diesen beiden Welten herstellen. Denn Methodenaufrufe sollen auf Seiten der Clients möglichst transparent durchgeführt werden, sodass die Entwickler/-innen mit so wenig Details wie nötig belastet werden.

Zunächst muss die Geschäftslogik implementiert werden und die nach außen angebotene Schnittstelle auf möglichst wenig Klassen verteilt werden. Als so genanntes *Package* werden die zusammengefassten Dateien in einen speziellen Server integriert, der die nötige Infrastruktur zur Ausführung bereit hält. Hiernach können die Clients bestimmte Parameter zur Nutzung des *Web Service* abfragen, wie z.B. Kodierungen, Datenstrukturen und Protokolle. Die Antworten werden in der *Web Services Description Language* (WSDL) [Walsh02a] übermittelt, einer vom *WWW Consortium* (W3C) spezifizierten Sprache in XML. Mit diesen Informationen können in *Java* drei verschiedene Formen von Clients gebaut werden: *Static Stub*, *Dynamic Proxy* und *Dynamic Invocation Interface* (DII).

Die ersten beiden Varianten benötigen ein spezielles Werkzeug, mit dem die WSDL-Daten einmalig im Voraus in Klassen umgewandelt werden. Bei einem Client mit *Static Stub* werden alle Klassen erzeugt, die für die Serialisierung und Deserialisierung der Daten benötigt werden[11]. Ein Nachteil dieser Vorgehensweise ist der Aufwand bei Änderungen der WSDL-Daten, die immer eine vollständige Neuübersetzung nach sich ziehen. Dieses Problem wird bei Clients mit *Dynamic Proxy* teilweise umgangen, denn es werden lediglich Schnittstellen von den Werkzeugen erzeugt und keine Klassen mit einer Implementierung. Die wird erst zur Laufzeit automatisch erzeugt und eingebunden, wodurch kleine Änderungen Berücksichtigung finden. Der Preis für diesen „Komfort“ liegt in der verzögerten Ausführung des ersten Aufrufs, denn im Hintergrund laufen komplexe Prozesse ab, die den *Dynamic Proxy* erzeugen. Volle Kontrolle, weniger Ressourcen-Bedarf und volle Flexibilität lassen sich nur mit dem *Dynamic Invocation Interface* erreichen. Diese Schnittstelle ist Bestandteil der JAX-RPC-API, auf die gleich näher eingegangen wird. Die Auswertung der WSDL-Daten bleibt bei DII den Entwicklern/-innen überlassen, um z.B. einen sehr schlanken und optimierten Client zu schreiben, der aber nicht automatisch auf Änderungen des *Web Services* reagieren kann.

Um eine richtige Abwägung treffen zu können, müssen die Vor- sowie Nachteile von *Static Stub*, *Dynamic Proxy* und DII verglichen werden. Für die Belange des Projekts mαth-kit ist der *Static Stub* vollkommen ausreichend, denn es werden keine gravierenden Änderungen an den Schnittstellen der Komponenten erwartet. Diese Lösung ist somit schlanker und schneller in der Ausführung als der *Dynamic Proxy* und einfacher in der Umsetzung als die Ansteuerung über DII.

Die Steuerung der vorgestellten drei Methoden erfolgt über die *Java API for XML-Based RPC* (JAX-RPC). Abhängig vom gewählten Typ sind nur wenige Befehle nötig, bis ein *Web Service* initialisiert und angesteuert ist. Abbildung 7.7 zeigt ein typisches Szenario.

Der Client greift direkt auf den *Stub* zu, der intern wiederum JAX-RPC-Aufrufe nutzt. Über das Netz werden dann in beide Richtungen SOAP-Nachrichten verschickt. Auf der Seite des Servers ruft die JAX-RPC-Laufzeitumgebung mit Hilfe von so genannten *Ties* die Service-Methoden auf. Bei den *Ties* handelt es sich um den „Klebstoff“ zwischen den Service-Klassen, der von dem Server automatisch generiert wird.

Neben den kommerziellen Anbietern, wie z.B. *Systinet Server for Java*[12] von Systinet,

[11] Unter Serialisierung bzw. Deserialisierung wird die Umwandlung bzw. Rückumwandlung von Daten in einen Daten-*Stream* verstanden, der z.B. über ein Netzwerk transportiert wird.

[12] `http://www.systinet.com` (29.10.05)

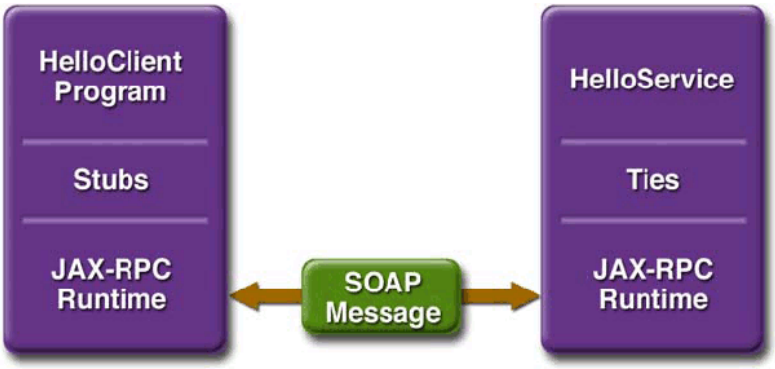


Abbildung 7.7: JAX-RPC-Aufruf [Bodoff04, S. 321]

Cape Clear 6[13] von Cape Clear und *Artix*[14] von Iona, gibt es leider nur das Projekt *Axis*[15] von der *Apache Group*, das eine ernst zu nehmende freie Alternative anbietet.

7.4 WebDAV

WebDAV steht für *Web-based Distributed Authoring and Versioning* und bezeichnet eine Erweiterung des HTTP-Protokolls. In erster Linie soll es die gemeinsame Arbeit in Gruppen ermöglichen und sich in die existierende Infrastruktur einbetten. Die genaue Funktion, die weder die Entwickler/-innen noch die Benutzer/-innen interessieren dürfte, ist in zwei Dokumenten [Slein98; Goland99] festgelegt. Im Grunde genommen wird HTTP um ein paar *Header* und Methoden erweitert, die das Auslesen von Dateistrukturen gestattet. Leider fehlt bei *WebDAV* die Versionierung, obwohl der Begriff Bestandteil des Namens ist, weshalb eine Ergänzung mit dem Namen *DeltaV* nötig war. Auch die Details dieser Spezifikation [Clemm99] sind unerheblich, weil auf fertige Lösungen zurückgegriffen werden soll.

Der Web-Server *Tomcat* wird beispielsweise mit einem eigenen *WebDAV*-Modul ausgeliefert, sodass auf der Server-Seite lediglich ein wenig Konfigurationsarbeit ansteht. Auf der Client-Seite ist die Auswahl freier Libraries leider wieder beschränkt. Mit *Jakarta Slide*[16] stellt die *Apache Group* eine vollständige Umsetzung von *WebDAV* und eine rudimentäre von DeltaV bereit.

[13] `http://www.capeclear.com` (23.10.05)
[14] `http://www.iona.com` (29.10.05)
[15] `http://ws.apache.org/axis` (29.10.05)
[16] `http://jakarta.apache.org/slide/` (29.10.05)

Kapitel 8

Metapher

Ein wichtiger Aspekt des Projekts mαth-kit ist der Einsatz der Baukasten-Metapher, um den Benutzern/-innen ein besseres Verständnis der Funktionalität zu vermitteln. In der Informatik spielen Metaphern seither eine bedeutende Rolle bei der Begriffsbildung, was bei einer so jungen Wissenschaft nicht weiter verwunderlich ist, da nicht für jeden neuen Gegenstand des Interesses ein neues Wort erfunden werden kann. Begriffe wie Mäuse, Schlangen, Bäume, Keller und viele mehr stammen aus dem alltäglichen Sprachgebrauch, werden aber in einer anderen Bedeutung eingesetzt, die ihrem ursprünglichen Kontext enthoben ist. Nur bestimmte Aspekte des Begriffs werden übernommen, andere hingegen ignoriert. Was zeichnet aber eine Metapher genau aus, wie ist sie definiert? In der Brockhaus-Enzyklopädie findet sich hierzu folgendes:

> „Ausdrucksmittel der uneigentlichen Rede; das eigentlich gemeinte Wort wird ersetzt durch ein anderes, das eine sachl. oder gedankl. Ähnlichkeit oder dieselbe Bildstruktur aufweist, z.B. »Quelle« für »Ursache«. Die Sprache springt dabei, im Unterschied zur Metonymie, gleichsam von einem Vorstellungsbereich in einen anderen.“ [Bro91, S. 521]

Wie die gegebene Definition vermuten lässt, gibt es eine Reihe anderer sprachliche Begriffe, die gewisse Eigenschaften mit der Metapher gemein haben, aber nicht mit ihr verwechselt werden sollten. Die Allegorie ist eine bildhafte Darstellung eines Begriffs (Frau mit verbundenen Augen für „Gerechtigkeit“), das Homonym ein gleich lautendes Wort mit anderer Bedeutung (der Gehalt/das Gehalt), die Katachrese ein bildlicher Ausdruck für eine fehlende Bezeichnung (Schlüssel„bart“) und die Metonymie eine Bedeutungsvertauschung („Stahl“ für „Schwert“), um nur einige Beispiele zu nennen.

Metaphern können aus mehreren Wörtern, so genannte Wortfelder, bestehen, die in einem größeren Bedeutungszusammenhang stehen. Die Flüssigkeitsmetapher in der Elektrotechnik sei stellvertretend als Beispiel genannt, bei der die Begriffe Strom, Kanal, Quelle, Kondensator usw. ein Wortfeld bilden. Es ist daher zweckmäßig, die metaphorische Definition nicht auf einzelne Begriffe zu reduzieren.

Als rein sprachliches Mittel ist die Metapher für den Einsatz in der Software-Technik freilich nicht hinreichend, sondern bedarf einer weitergehenden, umfassenderen Definition, die den Menschen und die Gegenstände des Interesses in einen Zusammenhang bringt. Werner Ingendahl führt den Begriff *metaphorischer Prozess* in [Ingendahl71] ein, der umfassend die verschiedenen Kontexte der Metaphorik beschreibt.

8.1 Metaphorischer Prozess

Der *Metaphorische Prozess* wird nun auf Basis von [Busch98] aus verschiedenen Theorien und Ansichten hergeleitet. Abbildung 8.1 gibt einen Überblick über die verschiedenen Begriffe, die mit ihm verbunden sind, und deren Relationen.

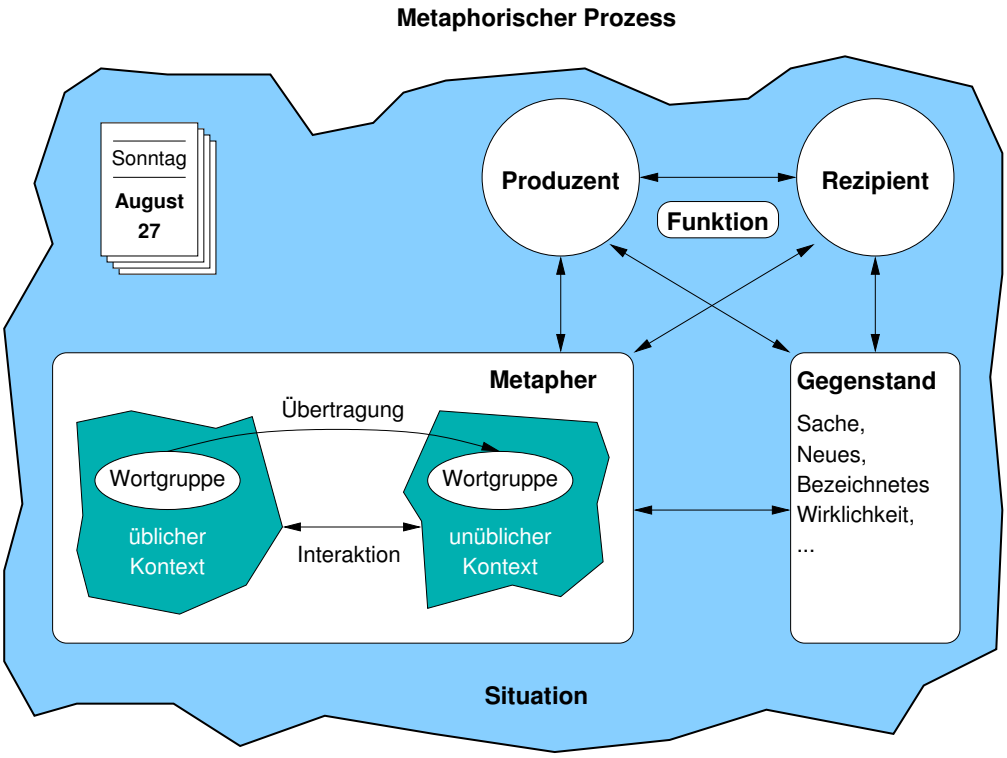


Abbildung 8.1: Metaphorischer Prozess nach [Busch98, S. 25]

Jedes Wort hat in Aristoteles' *Poetik* [Aristoteles82] genau eine „eigentliche" Bedeutung und ist außerhalb des Zusammenhangs „uneigentlich" verwendet. Die **Übertragung** (griech. $\mu\varepsilon\tau\alpha\varphi o\rho\alpha$) eines Wortes aus einem **üblichen Kontext** in einen **unüblichen** ist daher ein zentraler Aspekt des aristotelischen Metapher-Begriffs. Jedoch ist die Beschränkung auf einzelne Worte zu restriktiv, weshalb lieber die bereits definierte **Wortgruppe** genutzt werden soll. Max Black kritisierte die einseitig gerichtete Beziehung der Übertragung als unzureichend und hat daher die *Interaction View* entwickelt [Black62]. Diese beschreibt, wie sich die Wortgruppen aus dem üblichen und dem unüblichen Kontext gegenseitig beeinflussen. Es werden die implizierten Bedeutungselemente der Wortgruppe aus dem üblichen Kontext übernommen, die mit denen des **Gegenstandes** übereinstimmen. Hierdurch werden dessen übertragbare Charaktermerkmale verdeutlicht, wohingegen die nicht erfassten an Bedeutung verlieren. In den Worten von Max Black heißt es:

> „Die Metapher kommt dadurch zustande, daß auf den Hauptgegenstand[1] ein System von »assoziierten Implikationen« angewandt wird, das für den untergeordneten Gegenstand[2] charakteristisch ist." Black zitiert nach [Haverkamp83, S. 75]

> „Die Metapher selegiert, betont, unterdrückt und organisiert charakteristische Züge des Hauptgegenstands, indem sie Aussagen über ihn einbezieht, die normalerweise zum untergeordneten Gegenstand gehören" Black zitiert nach [Haverkamp83, S. 76]

Der Gegenstand im Schaubild 8.1 steht mit der Metapher ebenfalls in einer Wechselbeziehung, da der Zustand des Gegenstandes Einfluss auf die Metapher selbst ausübt. Carsten Busch nennt als Beispiel den Mond, bei dem die Bezeichnungen „Zitronenmond" und „Silbersichel" von dessen momentanen Eigenschaften geprägt sind [Busch98, S. 15–16]. Angemerkt sei noch, dass der Begriff „Gegenstand" im Zusammenhang mit Metaphern offensichtlich unglücklich gewählt ist, da unter dieser Kategorie auch Menschen und Tiere verstanden sind, und wird hier ausschließlich zur Konsistenzwahrung mit den referierten Arbeiten verwendet.

[1] Das ist der über die Wortgruppe im unüblichen Kontext beschriebene Gegenstand.
[2] Das ist der über die Wortgruppe im üblichen Kontext beschriebene Gegenstand.

Die Beziehungen zwischen den Wortgruppen, der Metapher und dem Gegenstand existieren selbstverständlich nicht um ihrer selbst willen, sondern nur indirekt über den Menschen, in seinem Denken, Sprechen und Fühlen. Der Mensch ist es, der durch kreative Leistungen eine sprachliche Handlung durchführt. Im Schaubild tritt er als **Produzent/-in** und **Rezipient/-in** auf, der/die jeweils mit der Metapher und dem Gegenstand in Wechselbeziehung steht. Im Grunde sind sich Produzent/-in und Rezipient/-in sehr ähnlich, da sie die gleiche geistige Leistung erbringen müssen. Der wesentliche Unterschied besteht darin, wie sie zu einer Metapher gekommen sind. Bei dem/der Produzent/-in kommt die Metapher, bewusst oder unbewusst, aus dem Inneren, hingegen empfängt der/die Rezipient/-in sie von außen. Auch wenn diese Unterscheidung oftmals verschwimmt bzw. umgekehrt wird, ist diese Betrachtung sinnvoll, um die pragmatische Dimension — wie also (Sprach)-Zeichen auf einen Menschen wirken — ausmachen zu können. Metaphern können demnach vom/von der Rezipient/-in „*nicht erkannt*“, „*erkannt aber nicht verstanden*“, „*verstanden aber abgelehnt*“ und „*völlig anders verstanden*“ werden. Die Kommunikation beschränkt sich hierbei nicht nur auf das gesprochene Wort, sondern umfasst den gesamten Habitus.

Der Gebrauch einer Metapher hat eine **Funktion**, eine Absicht, die mit ihm verbunden ist. Walter Seifert beschreibt in seinem Aufsatz [Seifert80] sieben Funktionen, von denen in dieser Arbeit eine als besonders wichtig für die Entwicklung und Nutzung eines Software-Systems erachtet wird: die *Prädikationsfunktion*. Sie dient dem Erfassen der Realität durch Modellbildung sowie Analogiebeziehungen und soll in diesem Kontext den Entwicklern/-innen beim Entwurf und der Kommunikation eine höhere Produktivität verleihen. Diese Funktion kann ebenfalls die Benutzer/-innen bei der späteren Gewöhnung an die Arbeit mit dem System unterstützen. Für diesen Personenkreis kann auch eine weitere Funktion, die **affektiv-emotionale**, beabsichtigt sein. Dabei geht es um die Vermittlung von Gefühlsnuancen, mit Ausrichtung auf intuitive Erfahrungen. So können z.B. Ängste minimiert werden, welche komplexe Software-Systeme in Laien auslösen können.

Die Funktion an sich ist abhängig von der **Situation**, in der eine Metapher produziert oder rezipiert wird. Es besteht schon ein Unterschied, ob eine Metapher in einem Software-Entwurf, im Schulunterricht oder einem Gedicht Verwendung findet. Somit umfasst die Situation alles, was bei der Metaphorik von Bedeutung ist. Als drei wesentliche Elemente zur Bestimmung oder Unterscheidung von Situationen lassen sich die *Örtlichkeit*, der *Zeitpunkt* bzw. die *Dauer* und die beteiligten *Personen* benennen.

Die *Zeit* spielt im metaphorischen Prozess eine besondere Rolle, und ist daher im Schaubild explizit aufgeführt. Sie bestimmt wie und ob eine Metapher verstanden wird. Der „Gully“ z.B. ist eine Metapher, die in den allgemeinen Sprachgebrauch übergegangen ist und wohl im 19. Jahrhundert aus dem englischen „gullet“ (Schlund) hervorgegangen ist. Solche nicht ohne weiteres identifizierbare Metaphern werden *tote* bzw. *lexikalisierte Metaphern* genannt. In [Wolff82] werden neben diesem drei weitere Typen von Metaphern unterschieden: *kreative Metaphern* (spontan, innovativ), *konventionelle Metaphern* (Klischees) und *Remetaphosierung* (Reaktivierung eines Bildes durch eine Abwandlung). Um welchen Typen einer Metapher es sich im Einzelnen handelt, ist abhängig von der jeweiligen Gruppe von Rezipienten/-innen und dem relativen Verwendungszeitraum innerhalb dieser „Sprachgemeinschaft“.

Zusammengefasst ist ein metaphorischer Prozess eine Übertragung einer Wortgruppe von einem üblichen in einen unüblichen Kontext, bei der es zu einer Interaktion kommt, die im günstigen Fall zu einem besseren Verständnis eines Gegenstandes führt. Die Metapher wird meist bewusst von Produzenten/-innen erzeugt und von Rezipienten/-innen nachvollzogen, wobei eine bestimmte Funktion beabsichtigt ist, deren Wirkung von der Situation und dem Zeitpunkt bzw. der Dauer abhängt.

8.2 Metaphern und Software-Technik

Metaphern sollen in dieser Arbeit zwei wesentliche Aufgaben erfüllen: Beim Entwurf erlauben sie einen selbstverständlichen Umgang mit abstrakten Begriffen und für die späteren Anwender/-innen beschleunigen sie den Zugang zum erstellten Programm. Besonders bei der Arbeit im Team und einem Austausch von Ideen können Metaphern die Kommunikation vereinfachen. Letztendlich werden Softwaresysteme aber nicht um ihrer selbst willen erstellt, sondern sollen Menschen bei der Erledigung ihrer Tätigkeit unterstützen.

In der Software-Technik spielen Metaphern bereits auf der Ebene der Programmiersprache eine wichtige Rolle. Genau genommen handelt es sich bereits bei dem Wort „Programmiersprache“ um eine Metapher. In Anlehnung an natürliche Sprachen wie Englisch, können Programmtexte auch von Menschen gelesen werden, die nicht mit den technischen Details vertraut sind [Louden94]. Abstrakt und formal beschriebene Kontroll- und Datenstrukturen erhalten durch Metaphern ,wie beispielsweise Schleifen, Bäume und Schlangen, sprechende Bezeichner, durch die wesentliche Merkmale erfasst werden. Auch die Vererbung in objektorientierten Sprachen ist eine Metapher, mit der die kontrollierte Weitergabe bestimmter Eigenschaften umschrieben wird. Um dem metaphorischen Prozess aus dem vorherigen Abschnitt gerecht zu werden, folgen Buschs Spezifika für Metaphern aus der Sichtweise der Programmierung [Busch98, S. 151]:

- Als Metaphern-**Produzenten/-innen** kommen in erster Linie Programmierer/-innen und Entwickler/-innen in Betracht.
- Die **Rezipienten/-innen** lassen sich weniger genau eingrenzen, aber in Frage kommen vor allem wiederum Programmierer/-innen und eventuell Benutzer/-innen.
- Metaphern auf dieser Ebenen nehmen vor allem eine Prädikations- und eine heuristische **Funktion** wahr.
- **Gegenstand** der Metapher sind das Programmieren, Programmiersprachen und alle enger damit zusammenhängende Aspekte.
- Als **übliche Kontexte** dienen eine Vielzahl verschiedenster Bereiche. Die vorherrschenden sind sicherlich nach wie vor die Bedeutungsfelder „Sprache“ und „Vorschrift“.
- Den **unüblichen Kontext** bildet wie so oft die Informatik.

Metaphern auf einem höheren Abstraktionsniveau dienen grundlegenden Sichtweisen des Entwurfs und der Programmierung von Software. Ein gutes Beispiel sind John Shores Software-Gebäude in [Shore85]. Er vergleicht den Entwurf und Bau eines Hauses mit dem von Programmen. Zuerst steht ein Idee von einem Haus, die in einer Reihe bekannter Schritte verfeinert wird, bis am Ende eine physikalische Struktur entsteht. Einen anderen Ansatz verfolgt die Metapher Werkzeug, Automat, Material (WAM) in [Züllighoven98]. Fachliche Gegenstände und Konzepte werden als Material betrachtet, das Anwendern/-innen mit Werkzeugen bearbeiten. Wiederkehrende Aufgaben, die ohne menschliches Zutun abgearbeitet werden können, lassen sich durch Automaten abarbeiten.

Auch Bertrand Meyer hält Metaphern für ein wichtiges Instrument bei der Entwicklung und Nutzung von Software. Besonders neue Ideen lassen sich mit ihnen überzeugend vermitteln:

> „Metaphors can be excellent teaching tools. The great scientist-expositors — the Einsteins, Feynmans, Sagans — are peerless in conveying difficult ideas by appealing to analogies with concepts from everyday's experience. This is the best.“ [Meyer97, S. 672]

Er warnt aber auch vor den Gefahren, die mit dem Gebrauch von Metaphern verbunden sind. Leicht können Dinge verwechselt oder falsche Schlüsse gezogen werden. Aus diesem Grund müssen Metaphern immer mit Bedacht gewählt werden.

Kapitel 9

Bewertung

In diesem Teil der Arbeit wurden die wichtigsten Themen und Aspekte für den Umgang mit modularen E-Learning-Inhalten vorgestellt. Ausgehend von der Zielsetzung dieser Arbeit in Abschnitt 1.2 soll nun eine Bewertung des aktuellen Stands der Wissenschaft erfolgen. Die genaue Benennung der bestehenden Lücken ergibt die Grundlage für das Vorgehen in den nächsten Teilen dieser Arbeit.

Die Lernobjekte in Kapitel 3 sind Container für Lerninhalte und haben durch ihre Größe (siehe Abschnitt 3.3) und Anordnung (siehe Abschnitt 3.4) Einfluss auf die Didaktik. Die zahlreichen Theorien und Definitionen der Lernobjekte verdeutlichen, dass mittlerweile der Begriff Lernobjekt im E-Learning verankert ist. Auch wenn es Differenzen bei dem einen oder anderen Detail gibt, scheint über die wesentlichen Funktionen Einigkeit zu herrschen. In Hinblick auf die Zielsetzung dieser Arbeit sind Lernobjekte somit die ideale Einheit für die Module. Sie sind ein zentrales Thema dieser Arbeit und Ausgangspunkt für die Bewertung der Autorensysteme sowie Lernplattformen. Mit den Standards *IMS Content Packaging* (siehe Abschnitt 3.5) und SCORM (siehe Abschnitt 3.6) stehen Kodierungen für Lernobjekte zur Verfügung, die weithin akzeptiert sind.

Eng verknüpft mit den Lernobjekten sind die Metadaten aus Kapitel 4. Sie sind unerlässlich für die Identifizierung von Lernmaterialien, besonders in großen Systemen. Es wurden einige Definitionen und Standards vorgestellt, bei denen teilweise die Meinungen stark auseinander liefen. Besonders schwierig scheint die Vereinigung von Technik und Didaktik zu sein. An der Standardisierung führt dennoch kein Weg vorbei und mit LOM (siehe Abschnitt 4.3) ist ein erster Schritt getan. Wenn diese Spezifikation nicht ausreichen sollte, gibt es noch diverse Spezialisierungen in Form von *Application Profiles*.

Auf der Seite der Lernobjekte und Metadaten stehen also die nötigen Mittel für modulare E-Learning-Inhalte bereit. Aus den Bewertungen für Autorenwerkzeuge (siehe Abschnitt 5.2) und Lernplattformen (siehe Abschnitt 6.3) lässt sich aber ersehen, dass diese Möglichkeiten nicht genutzt werden. Keines der vorgestellten Produkte benutzt den Begriff Lernobjekt, geschweige denn bietet die damit verbundene Funktionalität an. Meist lassen sich keine Module definieren, sondern nur Kurse. Wenn dann auch noch fremde Inhalte integriert werden sollen, ist schon aufgrund der verschiedenen Layouts und Notationen eine Inkonsistenz vorhanden. Mit der Metadatenunterstützung sieht es zwar ein wenig besser aus — immerhin bieten einige LOM an —, aber das vorhandene Potential der Standards wird nicht genutzt. Es reicht nicht aus, die Metadaten anzuzeigen und gegebenenfalls bearbeiten zu lassen. Hierdurch werden sie zum Selbstzweck degradiert. Lehrende und Lernende müssen durch Metadaten einen schnellen Zugang zu den Materialien erlangen, die sie für ihren Einsatz benötigen. Nur so lässt sich auf Seiten der Lehrenden die Mehrfachentwicklung bereits existierender Inhalte vermeiden. Wenn passende Lernobjekte, fremde wie eigene, über ein paar Stichworte in einer Datenbank gefunden werden, dann hat sich die Mühe der Metadateneingabe gelohnt. Für Lernende erschließt sich so das gesamte Angebot und im Selbststudium lassen sich einfacher individuelle Lücken schließen.

Von der Unterstützung der Lerntheorien aus Kapitel 2 kann bei den Autorenwerkzeugen und Lernplattformen keine Rede sein. Lediglich der Behaviorismus wird in Form von Quiz und Tests angeboten. Wenigstens gibt es keine technischen Hindernisse, eigene Lernmodelle zu integrieren, aber hier muss auf geeignete Programmen noch gewartet werden.

9.1 Resümee

Die Beschreibung des Standes der Wissenschaft hat deutlich gezeigt, dass die angestrebte Zielsetzung mit den heutigen Mitteln nicht in dem gewünschten Umfang realisierbar ist. Es muss insbesondere eine bessere Verbindung zwischen den Theorien und der Technik hergestellt werden. Mit den vorhandenen Theorien lässt sich ohne Probleme ein System für modulare E-Learning-Inhalte modellieren, aber die Umsetzung ist kompliziert. Das liegt unter anderem an den abstrakten Funktionsbeschreibungen. Gewiss ist es einfach, die Wiederverwendbarkeit von Lernobjekten einzufordern, doch impliziert dies eine Reihe von Hürden, die bereits in Kapitel 3 über Lernobjekt angerissen wurden. So ist es nicht weiter verwunderlich, dass jedes Produkt für sich genommen, sei es ein Autorenwerkzeug oder eine Lernplattform, für seine Aufgabe einen guten Dienst erweist. Hierbei wird jedoch nur ein spezielles Problem aufgegriffen und der Blick für das Gesamte fehlt. Als einziger Ausweg bietet sich ein ganzheitliches Konzept an, das von der Herstellung bis zur Präsentation von E-Learning-Inhalten alle Aspekte abdeckt. Dieses Konzept ist der rote Faden, der sich durch diese Arbeit ziehen soll. Im folgenden wird eine Vision des Systems skizziert, die etwas detaillierter als die Zielsetzung ist und die bereits vorgestellte Funktionen als Anregung nimmt.

Konkret soll ein System angeboten werden, dessen Kern modulare E-Learning-Inhalte sind und von dem Autoren/-innen, Lehrende sowie Lernende profitieren. Bei der Erstellung neuer Inhalte sollen Autoren/-innen alle technischen Möglichkeiten an die Hand bekommen, mit denen sie moderne Lernobjekte produzieren können. Hierzu gehören unter anderem eine Einbettung multimedialer Komponenten oder die Darstellung eines Dokuments in verschiedenen Layouts. Aber auch die Integration existierender Inhalte in proprietären Formate — quasi eine Umwandlung in ein allgemeines Format — und die Kombination mit fremden Lernobjekten müssen möglich sein. Im Interesse einer vielseitigen Nutzung auch mit anderen Systemen sollen international anerkannte Standards verwendet werden. Für die Autoren/-innen dürfen sich hieraus aber keine Einschränkungen in der Gestaltung ergeben und die Kodierung sollte transparent ablaufen. Bei der Entwicklung von Inhalten in Teams, womöglich an verschiedenen Orten, muss es eine zentrale Datenhaltung geben, die eine synchronisierte Entwicklung erlaubt. Unnötige Mehrarbeit oder der totale Verlust von Änderungen, z.B. durch das gleichzeitige Arbeiten an einer Datei, wobei der/die letzte Schreibende „gewinnt“, dürfen nicht auftreten.

Für Lehrende soll das System einen Pool verschiedener E-Learning-Inhalte bereithalten, die sie für Präsenzveranstaltungen, Fernveranstaltungen und zum Selbststudium anbieten. Sollte ein Thema nur unvollständig oder nicht vorhanden sein, können Lehrende selbst als Autoren/-innen auftreten bzw. diese Aufgabe delegieren. Die neuen Lernobjekte werden ebenfalls in den Pool gestellt und stehen damit allen zur Verfügung. Hier können sich auch die Lernenden bedienen, und über Suchmasken ihre Materialien finden. Die Vision sind thematisch verknüpfte Lernobjekte, die manuell oder automatisch Exkurse zu einem Thema ermöglichen. Diesen Mechanismus soll das folgende Beispiel verdeutlichen: Ein Lernobjekt beschreibt das Ohmsche Gesetz, in dem die Beziehung zwischen Strom, Spannung und Widerstand erklärt wird, die Begriffe selbst aber nicht. Ohne vorherige direkte Verknüpfung kann das System über die Metadaten verwandte Lernobjekte anbieten, in denen diese Begriffe erklärt werden. Da der Mechanismus auch für die herangezogenen Lernobjekt zählt, kann er beliebig oft wiederholt werden. Auf diese Weise entsteht regelrecht ein Netzwerk unter den Lernobjekten, das nicht statisch sein muss, sondern je nach Bedarf neu berechnet werden kann.

Diese Abstrakte Sicht auf die Zielsetzung muss freilich verfeinert werden. Ein Mittel zur besseren Erschließung der wesentlichen Merkmale ist die Metapher des Baukastens, die Entwicklern/-innen wie Benutzer/-innen ein intuitives Verständnis gibt.

Teil II

Entwurf

Kapitel 10

System-Vision

Die Bewertung im vorherigen Kapitel hat eindeutig gezeigt, dass die Zielsetzung dieser Arbeit nicht mit den heute verfügbaren Mitteln zu realisieren ist. Deshalb soll nun eine verfeinerte, eine technischere Sicht auf das angestrebte System erstellt werden. Diese **System-Vision** gibt den Bauplan für die Implementierung vor und ist eine verbindliche Vorgabe. Sie wird für die Entscheidung herangezogen, welche existierenden Programme, Libraries[1] und Standards zum Einsatz kommen. Fehlende Komponenten werden benannt und so modelliert, dass sie einfach implementiert werden können. Einige Rahmenbedingungen, die alle Projektbeteiligten von mαth-kit als sinnvoll erachten, wurden bereits im Abschnitt 1.3 über die Methodik erwähnt. Wesentliche Entscheidungen für die System-Vision sind die objektorientierte Modellierung und der Einsatz der Programmiersprache *Java*.

Zuerst muss ein fachliches Modell erstellt werden, aus dem das technische abgeleitet werden kann. Hierbei gilt zu beachten, dass es vollständig ist und alle gewünschten Funktionen beinhaltet. Eine Überprüfung zwischen fachlichem Modell und der realen Welt, auch **Verifizierung** genannt, gibt Aufschluss hierüber. Danach kann das technische Modell hergestellt werden, wobei es sich um eine Abbildung des fachlichen handelt. Die abschließende Überprüfung zwischen technischem und fachlichem Modell heißt **Validierung**. Hieraus wird auch ersichtlich, warum so sorgfältig bei der fachlichen Modellierung gearbeitet werden muss. Fehler und Lücken, die sich an dieser Stelle eingeschlichen haben, wirken sich möglicherweise erst bei der Implementierung oder bei der Arbeit aus. Über die Validierung sind sie nicht zu erfassen und eine Verifizierung ist zu diesem Zeitpunkt zu spät. Jede nachträgliche Änderung im fachlichen Modell kann schwerwiegende Konsequenzen nach sich ziehen.

Steht dieses Vorgehen aber nicht im Widerspruch zu dem iterativen Prototyping, wie es eingangs bei der Methodik festgelegt wurde? Diese Frage ist wichtig, da ihre Antwort einige Missverständnisse ausräumt. Bei der iterativen Vorgehensweise werden bewusst bestimmte Bereiche nicht sofort modelliert, um schnellst möglich vorzeigbare Ergebnisse zu haben. Damit ist aber nicht gemeint, dass in irgendeiner Form schlampig gearbeitet werden darf und Lücken im Modell erlaubt sind. Das Gegenteil ist der Fall. Es muss eine genaue Vorstellung vom Aufgabenbereich geben, um abschätzen zu können, welche Funktionen sich nachträglich hinzufügen lassen. Bei dem iterativen Vorgehen werden somit bewusst Abwägungen und Prioritäten gesetzt, die ein genaues Wissen über das Zielsystem voraussetzen. Dennoch bleibt bei der Implementierung genügend Spielraum, um unvorhergesehene Schwierigkeiten oder Änderungswünsche zu berücksichtigen. Auf keinen Fall sollen hier Vorgehensmodelle wie das Wasserfallmodell proklamiert werden.

Bei der fachlichen Modellierung stellt sich immer wieder die Frage, wie die Fakten aus der Realität gezogen und formal festgehalten werden. Eine beliebte Vorgehensweise sind Interviews. Mit ihnen wird versucht, sich von den Benutzern/-innen die Prozesse erklären zu lassen, die vom System unterstützt werden sollen. Hieraus werden Prosatexte entwickelt, die eine um-

[1] In dieser Arbeit wird der Begriff Library für Programmbibliotheken, wie z.B. von *Java* oder *C++*, verwendet.

gangssprachliche Beschreibung der Funktionalität bilden. Im Fall des Projekts mαth-kit wurde bereits im Vorwege festgehalten, welche Vorstellung von dem System existiert. Als Teil des Antrags und der Projektbeschreibung wurde quasi eine fachliche Beschreibung vorgelegt, die es genauer zu untersuchen gilt. In mehreren Projekttreffen wurde dann weiter herausgearbeitet, was einen multimedialen Baukasten ausmacht und welche Funktionen wünschenswert sind. Aus diesem Grund waren bei dem Projekt mαth-kit keine Interviews für eine Beschreibung in Prosa nötig.

Das Resümee (siehe Abschnitt 9.1) des Standes der Wissenschaft ist bereits eine Verfeinerung der Zielsetzung, angeregt durch die neuen Erkenntnisse der Untersuchung. Dennoch lässt sie einige Interpretationsfreiräume zu, die durch eine formalere Beschreibung eingeschränkt werden sollen. Hierzu werden die Fakten bestimmt, die bereits bekannt sind. Ein wichtiger Anhaltspunkt sind die verschiedenen Personen, die mit dem System interagieren. Danach werden Feststellungen abgeleitet, aus denen optimale Lösungen hervorgehen. So soll z.B. die bestmögliche Aufteilung der gesamten Architektur gefunden werden. Ein gesundes Gleichgewicht zwischen existierenden Lösungen und Eigenentwicklungen ist freilich erstrebenswert. Folglich muss ein Kompromiss zwischen den beiden Extremen „alles neu entwickeln“, mit einer optimalen Erfüllung der Ansprüche, und „alles zusammensetzen“, dafür aber Abstriche in der Funktionalität, gefunden werden.

Nach der Fertigstellung des fachlichen Modells erfolgt die Überführung in das technische. Zuerst wird eine grobe Architektur aufgestellt, die einzelne Programme und die Beziehungen untereinander verdeutlicht. Dann folgen die Komponenten, aus denen sich die Programme zusammensetzen, die wiederum in Klassen zerlegt werden. Neben diesen starren Relationen gibt es auch Algorithmen und Interaktionen, die näher modelliert werden sollen. Dies erfolgt über Sequenz- und Ablaufdiagramme, mit denen sich dynamische Zusammenhänge beschreiben lassen. Letztendlich wird die Modellierung so weit getrieben werden, dass eine präzise Übersetzung in eine objektorientierte Programmiersprache möglich ist.

Um den Begriff Komponente im Kontext der Software-Technik richtig zu verwenden, sollte eine ergänzende bzw. erklärende Erläuterung angeführt sein. Denn im Gegensatz zur Objektorientierung divergieren die Meinungen der Gelehrten sowie die technischen Umsetzungen bei diesem Begriff [Szyperski98; Griffel98]. Die folgende Definition soll weitestgehend für diese Arbeit gelten:

> „A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“ [Szyperski98, S. 164]

10.1 Rollen und Anwendungsfälle

Wie erwähnt, soll das fachliche Modell über die Tätigkeiten der handelnden Personen, manchmal auch Akteure genannt, entwickelt werden. Da manche Aktivitäten nicht von Einzelnen sondern von Mehreren ausgeführt werden können, ist das Konzept **Rollen** eine wichtige Abstraktion. Es handelt sich um eine Gruppierung, bei der alle für eine Aktion in Frage kommenden Personen durch die stellvertretende Rolle beschrieben werden. Dieser Ansatz bietet eine Reihe von Vorteilen. Mit einer Rolle ist immer ein genau definierter Aufgabenbereich verbunden, der eine bestimmte Anzahl von Tätigkeiten umfasst. Somit ist eine Person nicht an eine Rolle gebunden, sondern kann je nach durchzuführender Tätigkeit eine andere annehmen.

Die einzelnen Aktivitäten werden in dieser Arbeit als **Anwendungsfälle** beschrieben. Von Ivar Jacobsen in den späten 60er Jahren entwickelt, fanden sie Ende der 80er Jahre ihren Einzug in die objektorientierte Analyse. Um eine falsche Annahme gleich vorweg auszuräumen: In erster Linie handelt es sich um Beschreibungen in Form von Texten und nicht um Strichmännchen und Ellipsen, wie sie von vielen *CASE-Tools* angeboten werden. Die Diagramme der UML sind kein echter Ersatz für die schriftliche Form, weil sie keine Abläufe beschreiben können. Sie haben aber trotzdem ihre Daseinsberechtigung, denn es ist für diese Arbeit nicht

sinnvoll, in den folgenden Abschnitten alle Anwendungsfälle in ihrer vollen Länge aufzulisten. Für das Verständnis dieser Arbeit ist dieses Detailwissen nicht von Belang und würde den Rahmen sprengen. Deshalb werden die UML-Diagramme als Inhaltsverzeichnisse genutzt, die eine Übersicht der Beziehungen zwischen den Anwendungsfällen untereinander und zu den Rollen geben. Grundsätzlich wird für Anwendungsfälle keine Form vorgeschrieben. Der Einsatz formloser Anwendungsfälle ist genauso legitim wie eine Vorgabe durch Schablonen, in denen definierte Felder ausgefüllt werden müssen.

Im Stand der Wissenschaft werden bereits alle Personen genannt, die bei der Arbeit mit dem System auftreten. Zur eindeutigen Identifizierung erhält jede Rolle einen Bezeichner, der aus Prägnanz und Kürze in Englisch angegeben wird. Im folgenden stehen sie jeweils in Klammern hinter der genannten Personengruppe.

Die zentralen Gruppen sind die Lehrenden (**Professor**) und Lernenden (**Students**), da letztendlich alle Bemühungen dieser Arbeit ihr Streben unterstützen soll. Allgemeiner definiert handelt es sich um Benutzer/-innen (**User**) der Lernplattform, über die sie alle Aktivitäten durchführen. Im Zusammenhang mit den Autorensystemen treten die Autoren/-innen (**Author**) in Erscheinung, deren Aufgaben so vielfältig sind, dass eine Spezialisierung dieser Rolle, wie in [Bungenstock02; Baudry02b] vorgeschlagen, sinnvoll ist. Zwischen der Erstellung von Lernobjekten (**Developer**), deren Kombinierung zu höheren Strukturen (**Composer**) und der Veröffentlichung auf einem Server (**Publisher**) soll durch drei Rollen unterschieden werden. Die Tätigkeiten der Administratoren/-innen (**Administrator**) sind so vielfältig, dass sie sich nur schwer vollständig beschreiben lassen. Da sie neben den Verwaltungsaufgaben die anderen Rollen bei der Arbeit unterstützen, werden sie häufiger mit unvorhersagbaren Schwierigkeiten konfrontiert. Denn, wann immer ein technisches Problem auftritt, ist dessen Beseitigung Aufgabe der Rolle *Administrator*.

Bevor nun die Anwendungsfälle der einzelnen Rollen beschrieben werden, gibt Abbildung 10.1 eine Übersicht aller definierten Rollen. Die Pfeile zwischen ihnen drücken die Spezialisierung aus.

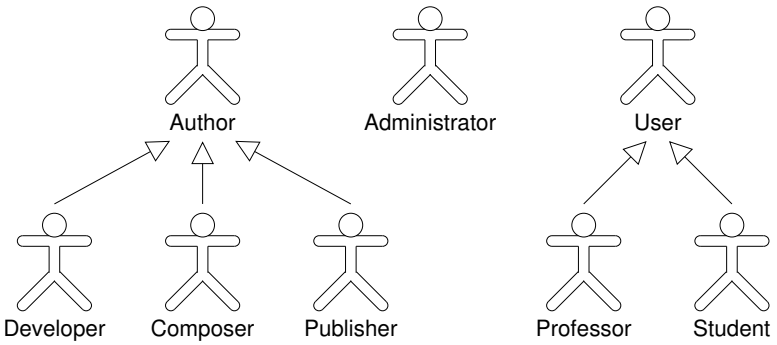


Abbildung 10.1: Übersicht der Rollen

10.1.1 Author

Die Rolle *Author* dient zur Abstraktion allgemeiner Aufgaben der Autoren/-innen und kann in ihrer Funktion etwa mit einer abstrakten Klasse verglichen werden. Aus diesem Grund wird in dieser Rolle kein Anwendungsfall für die Erstellung von E-Learning-Materialien eingeführt. Dies geschieht erst in den drei Spezialisierungen *Developer*, *Composer* und *Publisher*, die alle Anwendungsfälle „erben“ und manche auch erweitern.

Die Rolle *Author* beschreibt alle Tätigkeiten, die auf dem Weg zur Produktion anfallen. Hierzu gehört der Umgang mit Dateien, der das Erstellen, Bearbeiten und Löschen einschließt. Weil jede spezialisierte Rolle mit den Dateiformaten vertraut ist, mit denen sie täglich zu tun hat, ist dies ein gutes Beispiel für die Erweiterung von Anwendungsfällen. Ein potentieller Anwendungsfall ist „Datei bearbeiten“, der in der Rolle *Author* allgemein gültig beschrieben

ist. Erst in der Spezialisierung lassen sich dann die Abläufe für ein spezielles Format, z.B. ein Lernobjekt, genau angeben. Zu den weiteren Operationen auf Dateiebene gehören die Versionierung und das Sperren bzw. Freigeben von Dateien. Bei der Versionierung werden die Änderungen zwischen zwei Zeitpunkten protokolliert, sodass sich alle Arbeitsschritte jederzeit nachvollziehen oder rückgängig machen lassen.

> „Versioning is the management of multiple copies of the same evolving resource, captured at different stages of its evolution.“ [Vitali99]

Mit dem Sperren von Dateien lässt sich die parallele Bearbeitung verhindern. Weil diese Anwendungsfälle für alle Formate gleich sind, werden keine Erweiterungen in den abgeleiteten Rollen erwartet.

Bis jetzt sind die Aufgaben der Rolle *Author* so allgemein gefasst, dass sie bei jeder Tätigkeit mit dem Rechner auftreten können. Zu der Rolle gehören aber auch spezifischere Anwendungsfälle, wie z.B. der Umgang mit Metadaten, deren Erstellung bzw. Bearbeitung Aufgabe aller Autoren/-innen ist. Zwar werden bei der Erstellung von Lernobjekten inhaltlich andere Metadaten vergeben als bei der Festlegung des Layouts und Formats, aber durch den Einsatz von Standards werden sich die Eingabemasken wenig unterscheiden. Ähnlich sieht es mit einer Voransicht auf die geleistete Arbeit aus. So unterschiedlich die erstellten Materialien auch sein mögen, die auszuführenden Schritte zur Kontrolle des Resultats sind die gleichen. Abbildung 10.2 zeigt eine Übersicht aller Anwendungsfälle in UML-Notation.

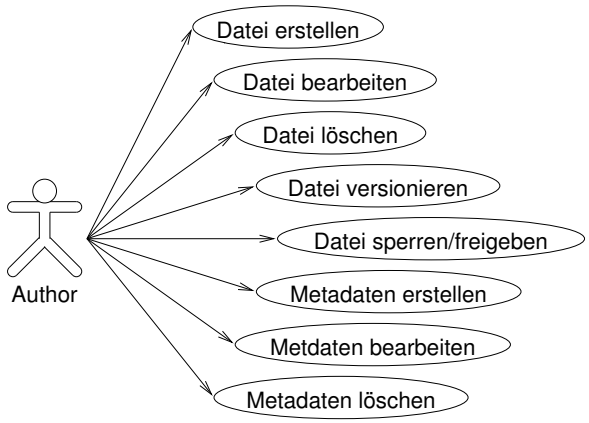


Abbildung 10.2: Anwendungsfälle der Rolle *Author*

10.1.2 Developer

Die Erstellung und Wartung von Lernobjekten ist die wesentliche Aufgabe der Rolle *Developer*. Weil sie mit vielen verschiedenen Theorien und Techniken umgehen muss, sind die Ansprüche an ihre Fertigkeiten sehr hoch. Normalerweise läuft die Arbeit dieser Rolle wie folgt ab: Die Rolle *Professor* hat ein didaktisches Konzept entwickelt und eine grobe Vorstellung von den benötigten Inhalten. Da ihr kein hohes technisches Wissen abverlangt werden darf, hilft die Rolle *Developer* bei der Umsetzung der Gedanken, wobei an dieser Stelle ausdrücklich auf die Trennung der Aufgaben dieser beiden Rollen eingegangen werden soll. So könnte die Erstellung von Texten und Abbildungen gewiss der Rolle *Professor* zugeschrieben werden, da sie eher fachlichen als technischen Sachverstand voraussetzt. Dennoch soll aus Gründen der Konsistenz dieser Prozess des „Kodierens“ der Rolle *Developer* zugeschrieben werden, denn auf diese Weise wird eine Vermengung ähnlicher Tätigkeiten vermieden. Dies ist ohne Weiteres möglich, da Personen nicht an Rollen gebunden sind und es sich jeweils nur um eine Sicht auf das System handelt. Eine Dozentin kann also ihre Texte und Abbildungen selbst erstellen, ohne sich an eine reale Technikerin wenden zu müssen.

Neben diesen einfachen Umsetzungen gibt es aber auch komplexere Aufgaben, wie z.B. die Erstellung von Animationen und Videos oder die Programmierung von interaktiven Komponenten und Simulationen. Obwohl die professionellen Autorenwerkzeuge aus Abschnitt 5.1.1 komplexe Details verdecken, braucht es Erfahrung, um die technischen Möglichkeiten zu nutzen. Noch anspruchsvoller ist die Entwicklung von *Java Applets*, bei der wirklich nur noch Experten Ergebnisse in akzeptabler Zeit erreichen. Hier zeigt sich, wie unterschiedlich die Ansprüche an die Rolle *Developer* sind, denn das Spektrum reicht von *Drag'n'Drop* mit WYSIWYG bis zur Programmierung in Entwicklungsumgebungen.

Ein weiterer Anwendungsfall ist die Umwandlung existierender Inhalte in das Hauptformat des Systems. Oft hat die Rolle *Professor* Skripte, Übungsaufgaben und Vorträge aus bereits gehaltenen Veranstaltungen, die wiederverwendet werden sollen. Da es sich um umfangreichere Dokumente handelt, können sie nicht 1:1 in Lernobjekte umgewandelt werden und müssen per Hand verkleinert werden. Neben den technischen Problemen, wie einzelne Daten extrahiert und konvertiert werden, sollten auch die Belange der Granularität und Sequenzierung aus den Abschnitten 3.3 und 3.4 bedacht werden. Abbildung 10.3 zeigt die Rolle *Developer*, welche aus der Rolle *Author* inklusive der genannten Anwendungsfälle abgeleitet ist.

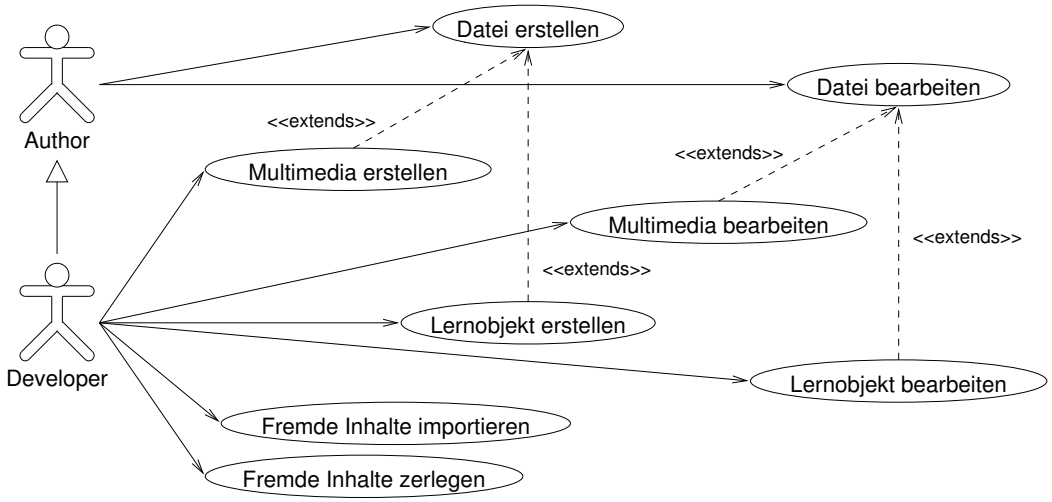


Abbildung 10.3: Anwendungsfälle der Rolle *Developer*

10.1.3 Composer

Die fertigen Lernobjekte werden von der Rolle *Composer* zu höheren Strukturen zusammengesetzt, wobei keine Aufteilungen vorgegeben sind. Es kann sich z.B. um Abschnitte, Kapitel und Kurse handeln, aber auch andere Strukturen wie Übungen, Tests und Explorationsbereiche sind denkbar. Im Gegensatz zu der Rolle *Developer* stehen die technischen Fragen eher im Hintergrund, denn die Rolle *Composer* muss einen fachlichen Gesamtüberblick haben — möglicherweise von der Rolle *Professor* angeleitet —, um bei der Sequenzierung vorteilhafte Lernpfade auszuarbeiten.

Richtig effizient kann die Rolle *Composer* nur arbeiten, wenn sie gezielt aus möglichst vielen Inhalten die passenden herausnehmen kann. Hierfür muss das System raffinierte Suchmechanismen über die Metadaten anbieten, die allgemeine und detaillierte Anfragen anbieten. Nach einer Suche muss die Rolle *Composer* das Suchergebnis analysieren und entscheiden, ob passende Inhalte enthalten sind. Stellt sich heraus, dass ein Lernobjekt nur bedingt geeignet ist, kann die Rolle *Developer* entsprechende Anpassungen vornehmen. Lässt sich zu einem Thema gar kein Material finden, müssen die Rollen *Professor* und *Developer* es erst entwickeln.

Bei der Zusammenstellung der Lernobjekte können zwei Anwendungsfälle unterschieden werden, die sich auf die Wartung und Aktualität der erzeugten Struktur auswirken. Entweder wird eine Kopie eines Lernobjekts oder ein Verweis erzeugt. Bei der Kopie wird das Lernobjekt

vervielfältigt und direkt im Kurs abgespeichert, wobei etwaige Korrekturen im Original keine Auswirkung auf die Kopien haben. Da dies auch umgekehrt gilt, Anpassungen der Kopie nur lokal wirken, kann dies aber ein gewünschter Effekt sein, denn auf diese Weise lassen sich Inhalte nach eigenen Wünschen ändern, ohne andere Kurse zu beeinflussen. Kann ein Lernobjekt wie gefunden übernommen werden, sollten lieber Verweise eingesetzt werden. Sie verbrauchen weniger Platz, da jedes Lernobjekt nur ein Mal vorliegt, und die Inhalte sind immer auf dem neuesten Stand. Ist der letzte Effekt nicht gewünscht, kann dieser bei eingeschalteter Versionierung durch einen Verweis auf eine bestimmte Version vermieden werden.

Fertige Strukturen können selbstverständlich wieder in ihre Bestandteile zerlegt werden, um z.B. den Pool an Lernobjekten aufzufüllen. Dies ist besonders wichtig für die Wiederverwendbarkeit fremder Materialien (siehe Abschnitt 3.1).

Auch die Rolle *Composer* kann wie die Rolle *Developer* fremde Inhalte importieren. Da es sich aber um einen automatisierten Prozess handelt, ist das Ergebnis von der Qualität des Ursprungsdokuments abhängig. Enthält es nur unzureichende Strukturinformationen, müssen die Lernobjekte möglicherweise von der Rolle *Developer* zerkleinert werden. Abbildung 10.4 fasst die beschriebenen Anwendungsfälle zusammen.

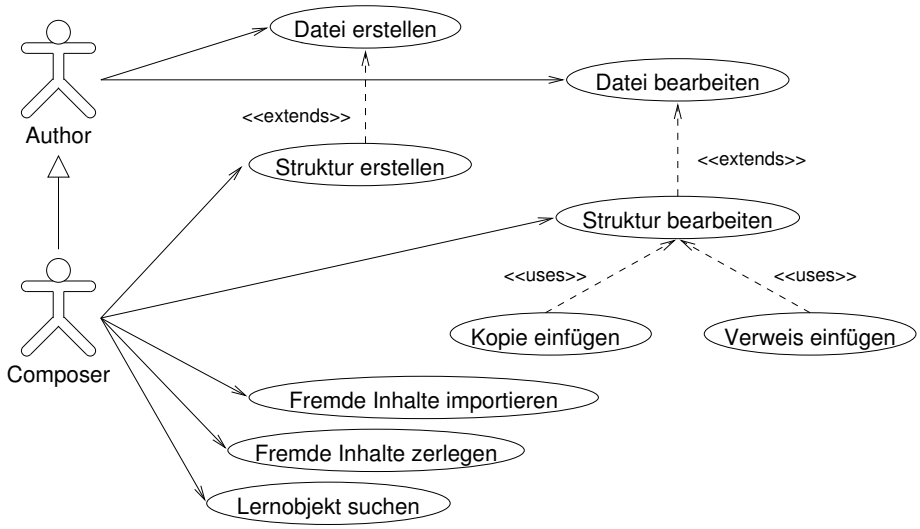


Abbildung 10.4: Anwendungsfälle der Rolle *Composer*

10.1.4 Publisher

Für die ästhetischen und praktischen Belange der Präsentation modularer E-Learning-Inhalte ist die Rolle *Publisher* zuständig. Sie kümmert sich um die Integration der Materialien in die Lernplattform, sodass die Rollen *Professor* und *Student* sie mit ihren Anzeigeprogrammen nutzen können. Hierbei muss die Rolle *Publisher* einige Punkte beachten, denn von der Rolle *Composer* erhält sie in der Regel einen Kurs, dessen Inhalte lediglich semantisch beschrieben sind. Für die Darstellung muss zuerst ein Layout erstellt bzw. ausgewählt werden, in dem der Kurs erscheinen soll. Dann muss das Ausgabeformat, z.B. HTML oder PDF, auf die Anzeigeprogramme und die Lernplattform abgestimmt werden, damit es zu keinen Inkompatibilitäten kommt. Im Idealfall unterstützt die Lernplattform einen der genannten *Content Packaging* Standards (siehe Abschnitt 3.5 und 3.6), sodass die Rolle *Publisher* die Ausgabe parametrisieren kann, ob z.B. Submanifeste generiert werden oder welche Metadaten enthalten sind.

Nach der Übersetzung des Kurses muss das Resultat auf die Lernplattform hochgeladen werden. Die einzelnen Schritte dieses Anwendungsfalls hängen ausschließlich von der Lernplattform ab und variieren von System zu System. Bei einem gemeinsam nutzbaren Speicherbereich, z.B. in Form eines Netzlaufwerks, ist dieser Schritt obsolet. Abbildung 10.5 zeigt zusammenfassend die genannten Anwendungsfälle.

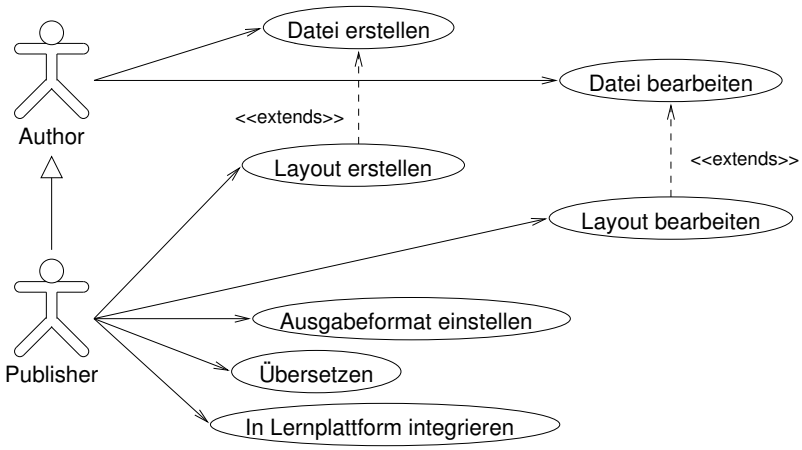


Abbildung 10.5: Anwendungsfälle der Rolle *Publisher*

10.1.5 User

Die Rolle *User* beschreibt die Anwendungsfälle aller Personen, die Dienstleistungen und Inhalte einer Lernplattform nutzen. Hierbei geht es in erster Linie um die Mechanismen des Zugriffs und weniger um die Verwendung des Angebots. Diese Details werden als Anwendungsfälle in den beiden Spezialisierungen *Student* und *Professor* beschrieben. Über bestimmte Anzeigeprogramme, wie z.B. Webbrowser oder Media-Player, werden die Inhalte von der Lernplattform heruntergeladen und angezeigt. Mit den angebotenen Kommunikations- und Organisationsdiensten können die Abläufe mit anderen Personen besser abgestimmt werden.

Die Rolle *User* dient der Kapselung administrativer Anwendungsfälle und tritt nie real in Erscheinung. Durch diesen Ansatz lassen sich die beiden Spezialisierungen *Student* und *Professor* übersichtlicher gestalten, da sich ihre Beschreibung auf die wesentlichen Aufgaben beschränkt. Zu den Anwendungsfällen der Rolle *User* gehören beispielsweise das An- bzw. Abmelden am System, Lesen von E-Mails und der Einsatz eines Terminkalenders. Es handelt sich um wiederkehrende Tätigkeiten, die wenig mit den eigentlichen Zielen Lehren und Lernen zu tun haben, aber dennoch modelliert werden müssen. Abbildung 10.6 listet die administrativen Anwendungsfälle der Übersicht halber auf.

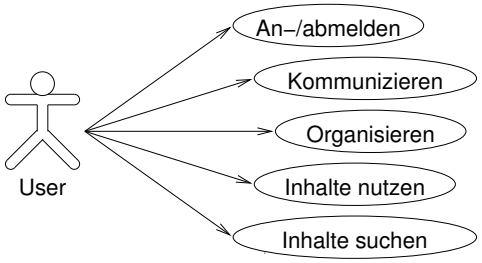


Abbildung 10.6: Anwendungsfälle der Rolle *User*

10.1.6 Student

Alle Aktivitäten der Lernenden werden durch die Rolle *Student* beschrieben, die das Angebot entweder im Selbststudium oder in Präsenz- bzw. Fernveranstaltungen nutzt. Ziel dieser Rolle ist gewiss, so schnell und viel wie möglich zu lernen. Neben dem Zugriff auf Materialien und deren Betrachtung — bereits durch die Rolle *User* angegeben — gehört das Absolvieren von Übungen und Tests zu den Anwendungsfällen.

Obwohl von der geplanten Infrastruktur kein direktes Lernparadigma vorgegeben wird, soll an dieser Stelle auf die Vorgehensweise im Projekt mαth-kit eingegangen werden. Ein zentra-

ler Aspekt von mαth-kit sind die Explorationsumgebungen, die konstruktivistisch organisiert sind. Anstatt starre Lernpfade vorzugeben, kann die Rolle *Student* eine individuelle Lernerfahrung machen. Neben der Vermittlung theoretischer Grundlagen, die jederzeit auch nachgeschlagen werden können, gibt es einen multimedialen Explorationsbereich, der interaktiv bedient wird. Zur Kontrolle des erreichten Lernerfolgs beinhaltet die Explorationsumgebung einen Übungsbereich, in dem Quiz, Puzzles und Multiple-Choice-Aufgaben Aufschluss über den Wissensstand geben.

In Chats und Foren können mit anderen Lernenden Arbeitsgruppen gegründet werden, die gemeinsam Aufgaben und Probleme lösen. Wenn angeboten, können Verständnisfragen auch direkt an die Rolle *Professor* gerichtet werden, die den Lernvorgang betreut. Abbildung 10.7 stellt den Lernprozess als Verfeinerung des Anwendungsfalls „Inhalte nutzen“ dar.

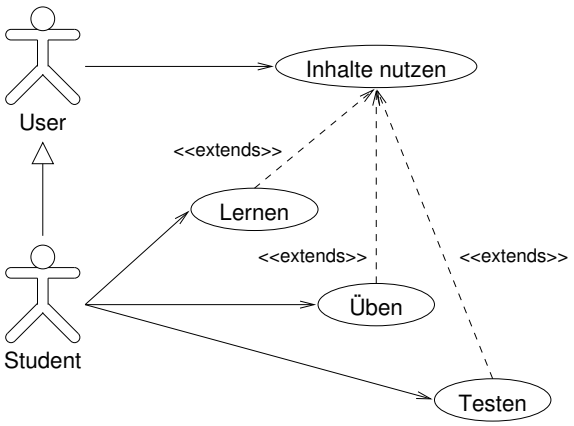


Abbildung 10.7: Anwendungsfälle der Rolle *Student*

10.1.7 Professor

Für die fachlichen bzw. inhaltlichen Anwendungsfälle ist die Rolle *Professor* zuständig, die Personen in der Rolle *Student* begleitet und den Rollen *Developer*, *Composer* sowie *Publisher* bei didaktischen Fragen hilft. In anderen Worten koordiniert die Rolle *Professor* die gesamte Lehre. Einen wesentlichen Teil der Arbeit machen Vorbereitungen aus, wobei neben den Inhalten passende Übungen und Tests entwickelt werden müssen. Hierbei kann die Rolle *Professor* im Vorwege Ideen entwickeln, wie bestimmte Themen multimedial aufbereitet werden, um sie gemeinsam mit der Rolle *Developer* zu realisieren.

Neben der Vermittlung von Wissen muss die Rolle *Professor* den Lernfortschritt der Lernenden überprüfen und bewerten. Auf diese Weise kann gegebenenfalls auftretenden Defiziten entgegen gewirkt werden. Ein Instrument sind die Quiz und Tests, deren Positionierung Bestandteil der Lernpfadgestaltung ist. Bei einer automatischen Auswertung können zusätzlich Statistiken angeboten werden, mit denen sich die Begutachtung, z.B. auf ganze Gruppen, ausdehnen lässt.

Weil die Rolle *Professor* auch die Funktionen der Lernplattform nutzt, erweitert sie die Rolle *User*, sodass sie beispielsweise in der Präsenzlehre eine Vorlesung mit multimedialen Inhalten bereichert oder Fragen in Foren beantwortet. Abbildung 10.8 zeigt eine Auflistung der vorgestellten Anwendungsfälle.

10.1.8 Administrator

Die Rolle *Administrator* unterstützt alle anderen Rollen bei der Arbeit, indem sie für für einen einwandfreien Zustand des Systems sorgt und Ansprechpartner/-in bei technischen Schwierigkeiten ist. Besonders bei unvorhersagbaren Problemen, wie z.B. falsch konfigurierten Programmen oder wiederkehrenden Abstürzen, ist die Rolle *Administrator* eine Anlaufstelle. Meist hin-

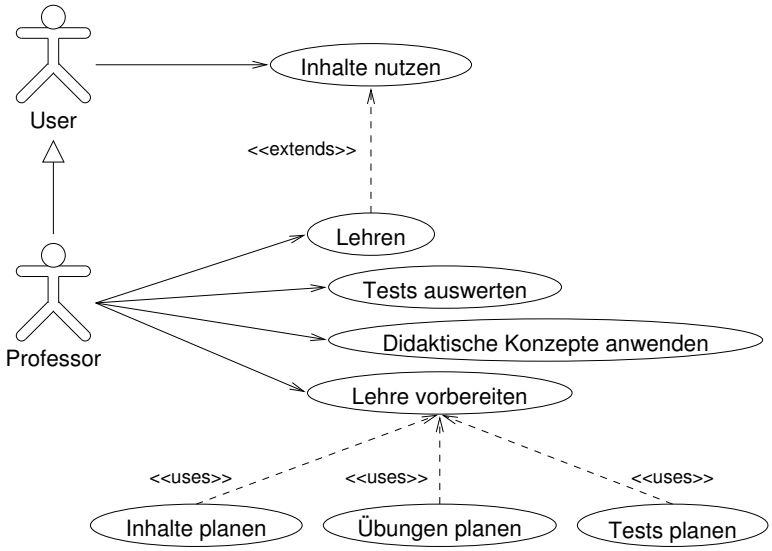


Abbildung 10.8: Anwendungsfälle der Rolle *Professor*

ter den Kulissen werden alle Verwaltungstätigkeiten von dieser Rolle durchgeführt. Für einen geordneten Ablauf legt sie Zugänge für Benutzer/-innen an und vergibt Zugriffsrechte, die für Dateien, Kurse, Foren, Chats, etc. gelten. Bei Bedarf installiert sie neue Programme und führt Updates installierter Software durch. Abbildung 10.9 fasst die Anwendungsfälle zusammen.

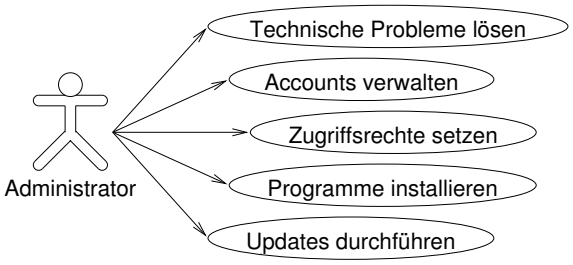


Abbildung 10.9: Anwendungsfälle der Rolle *Administrator*

10.2 Komponenten

Mit den Rollen und Anwendungsfällen wurde ein externer Blick auf das Verhalten des Systems gegeben. Für die detaillierte Beschreibung der internen Vorgänge werden in der Regel aus den Anwendungsfällen Aktivitätsdiagramme hergeleitet. Dieser Diagrammtyp ist Bestandteil der UML mit vorgegebener Notation und ist z.B mit Flussdiagrammen vergleichbar, in denen mit Bedingungen, Verzweigungen, Schleifen, Zuständen, Synchronisationen, etc. Abläufe modelliert werden. Abhängig vom gewählten Abstraktionsniveau können sogar Algorithmen genauestens beschrieben werden. Für den Entwurf der geplanten Systemarchitektur ist dieses Vorgehensmodell aber ungeeignet, denn wenn möglich, sollen vorhandene Programme und Komponenten eingesetzt werden, weshalb die Anwendungsfälle mit ihrer externen Sicht als Entscheidungskriterien hinreichend sind. Lediglich für die Eigenentwicklungen ist dieser Schritt sinnvoll, weil die Ergebnisse weiterverwendet werden können. Da aber noch nicht beschlossen wurde, welche Teile neu entwickelt werden, wird in dieser Arbeit ein abgeänderter Weg eingeschlagen. Die Komponenten werden nicht aus den Aktivitätsdiagrammen gebildet, sondern direkt aus den Anwendungsfällen.

Mit dieser Herangehensweise lassen sich offensichtlich nicht sehr kleine Komponenten bilden, aber sie hilft bei der Aufteilung auf Programmebene. Manche Anwendungsfälle lassen

sich über eine Komponente bearbeiten, sodass sich die Hauptkomponente für eine Rolle in Teilkomponenten aufteilen lässt. Auch die Hauptkomponenten setzen sich aus kleineren Bestandteilen zusammen, wodurch eine Sicht mit verschiedenen Abstraktionsniveaus entsteht. Auf diese Weise lassen sich exakt die existierenden und fehlenden Komponenten bestimmen.

10.2.1 Basis

Es wird mit den Anwendungsfällen der Rolle *Author* begonnen, die eine Art Basis für die Spezialisierungen darstellt. Dementsprechend werden die Teilkomponenten so gestaltet, dass sie sehr allgemeine Dienste anbieten und leicht ansteuerbar sind. Bei der Rolle *Author* lassen sich zwei wesentliche Bereiche ausmachen: der Umgang mit Dateien und die Verwendung von Metadaten. Die sich ergebenden Komponenten sind in Abbildung 10.10 dargestellt.

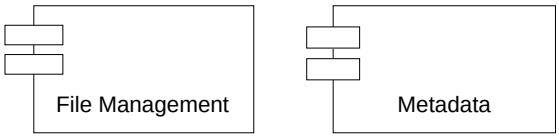


Abbildung 10.10: Komponenten für die Rolle *Author*

Mit der Komponente **File Management** wird der Zugriff auf das Dateisystem des Betriebssystems abstrahiert. Anstatt einer Reihe primitiver Systemaufrufe, mit denen sich komplexere Operationen zusammensetzen lassen, werden einfache Befehle für oft genutzte Funktionen angeboten. Hierzu gehören unter anderem das Suchen von Dateien mit regulären Ausdrücken, das Entpacken komprimierter Dateien und das Kopieren bzw. Verschieben von Verzeichnissen sowie Dateien. In Hinblick auf die *Content Packages* (siehe Abschnitt 3.5) und Lernplattformen mit WebDAV (siehe WebCT in Abschnitt 6.2.2) wird zusätzlich die Abstraktion des Zugriffs zum Wunsch. So unterschiedlich die zugrunde liegenden Techniken auch sein mögen, die Operationen sind fast immer gleich. Zusammengefasst ist die Dienstleistung der Komponente *File Management* eine transparente Behandlung unterschiedlichster Medien, Formate und Protokolle verbunden mit umfassenden Operationen.

Prinzipiell kann jedes Datum mit Metadaten versehen werden. Die hierfür nötige Funktionalität stellt die Komponente **Metadata** bereit, durch die sich einzelne Werte bis hin zu komplex verschachtelte Strukturen mit den benötigten Metadaten versehen lassen. Dabei müssen die gängigen Standards und Kodierungen unterstützt werden bzw. Änderungen und neue Versionen leicht integrierbar sein. Hierfür ist intern eine modulare Struktur nötig, bei der Module ergänzt und neue hinzugefügt werden können. Zusätzlich müssen generische Funktionen angeboten werden, wie z.B. die Generierung objektiver Metadaten, eine Umrechnung von Zeiteinheiten oder komplexe Operationen wie das Zusammenfügen von Metadaten aus verschiedenen Quellen. Insgesamt erledigt diese Komponente alle Aufgaben rund um die Metadaten, von der Eingabe über die Speicherung bis hin zur kompletten Umwandlung in diverse Formate der Metadatenstandards.

10.2.2 Learning Object Development

Die Anwendungsfälle der Rolle *Developer* beschreiben drei wesentliche Aufgaben, die sich in Teilkomponenten aufteilen lassen: die Erstellung von Multimedia und Lernobjekten sowie die Integration fremder Inhalte. Abbildung 10.11 zeigt die zusammengesetzte Komponente **Learning Object Development**.

Mit der Komponente **Multimedia Environment** soll eine Entwicklungsumgebung für multimediale Inhalte angeboten werden. Weil es für alle Formen von Multimedia bereits Programme gibt, soll an dieser Stelle nicht das Rad neu erfunden werden. Genauer betrachtet ist es auch nicht leistbar, für die vielen Möglichkeiten und Anwendungen eigene Werkzeuge zu entwickeln. Daher wird in dieser Arbeit der Einsatz bewährter Programme für diese Aufgabe bevorzugt, weil sich hieraus wesentliche Vorteile ergeben: Im idealen Fall werden optimale

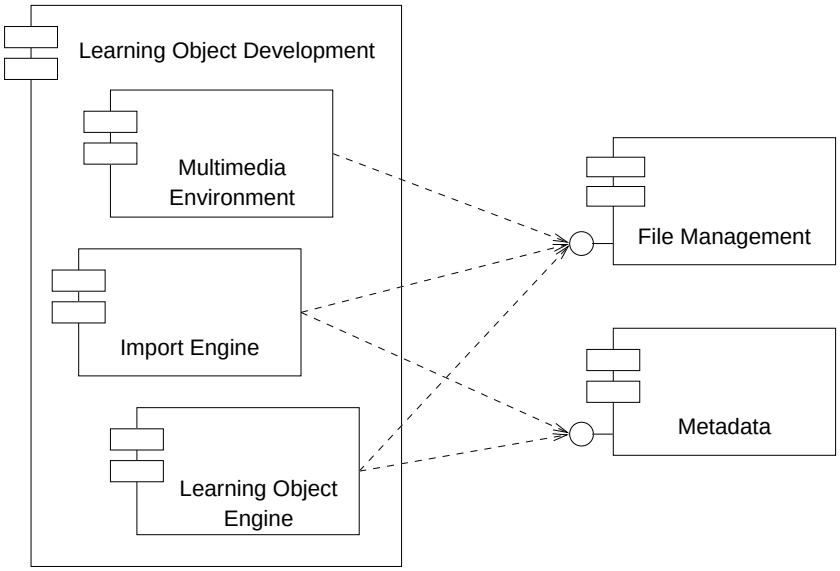


Abbildung 10.11: Komponente für die Rolle *Developer*

Ergebnisse mit geringer Einarbeitungszeit erzielt. Für die Realisierung wird ein Mechanismus benötigt, mit dem sich individuelle Verbindungen zwischen den Multimedia-Dateien und -Programmen herstellen lassen. Ähnlich wie im Explorer von Windows werden Dateiformaten unterschiedliche Medientypen zugewiesen, die wiederum über ein Verb eine Verknüpfung zu einem Programm besitzen. Bei dem Verb handelt es sich um Anweisungen wie z.B. „drucke“, „bearbeite“ und „sende an“. Der Aufruf ist dann ein Tupel, das aus einem Dateinamen und dem Verb besteht. Beispielsweise überprüft die Komponente für ein übergebenes Wertepaar `(test.avi, edit)` die Verbindung zu dem eingetragenen Werkzeug und ruft in diesem Fall das zugewiesene Videobearbeitungsprogramm auf.

Die Komponente **Import Engine** liest fremde Inhalte ein und konvertiert sie zu dem intern genutzten Format. Um möglichst viele Formate unterstützen zu können, muss die interne Architektur sehr flexibel sein, doch leider gibt es keinen generischen Mechanismus wie XSLT, mit dem sich Übersetzungsregeln beschreiben lassen. Weil die Kodierungen der einzulesenden Formate sehr verschieden sind, von LaTeX-Dateien in ASCII bis hin zu proprietären *MS Word*-Dateien, ist eine automatisierte Interpretation unmöglich. Hieraus folgt, dass für jedes Format ein eigens entwickelter *Reader* benötigt wird. Um die Programmierung einfacher zu gestalten, bietet die Komponente für das Zielformat eine übersichtliche Schnittstelle, die Inhalte in mehrere Elemente aufteilt, wie z.B. Überschrift, Abbildung und Tabelle. Hierdurch beschränkt sich die Entwicklung des *Readers* auf das Auslesen der Elemente mit anschließender Abbildung in das Zielformat. Gebündelt in einem Paket, werden *Reader* nur ein Mal implementiert und anderen Orts installiert. Teilweise enthalten die fremden Inhalte zusätzliche Metadaten, deren Konvertierung und Aufbereitung über die Komponenten *Metadata* erfolgt.

Die eigentlichen Lernobjekte werden mit der Komponente **Learning Object Engine** verwaltet, die alle gängigen Standards unterstützt und intern eine festgelegte Struktur von physikalischen Dateien, Einstiegspunkten sowie Metadaten einsetzt. Dank einer Konsistenzprüfung mit Fehlerbehebung ist garantiert, dass nicht ungewollt defekte Lernobjekte erzeugt werden. Der Speicher- und Lademechanismus ist modular aufgebaut und unterstützt Dateisysteme, Datenbanken sowie Datei-*Server*. Weil alle Speicher- und Ladevorgänge über die interne Struktur ablaufen, ergibt sich ein optimales Werkzeug für die Konvertierung verschiedener Standards. Für den Einsatz dieser Komponente ist freilich ein grafisches Frontend (*View*) sinnvoll, mit dem sich die Manipulationen per Maus steuern lassen. Eine Aufteilung nach dem MVC-Muster [Gamma95] soll daher als übliche Vorgehensweise angenommen werden, die durch einen speziellen Nachrichtendienst für *Views* unterstützt wird.

10.2.3 Structure Development

Für die Erstellung höherer Strukturen oder Kurse aus Lernobjekten und deren Kompositionen steht der Rolle *Composer* die Komponente **Structure Development** zur Verfügung. Ihre Teilkomponenten entstehen wieder aus der Zusammenfassung der Anwendungsfälle, die sich aus dem Importieren fremder Kurse, dem Suchen von Inhalten und dem eigentlichen Aufbau der Struktur ergeben. Abbildung 10.12 illustriert die resultierende Aufteilung.

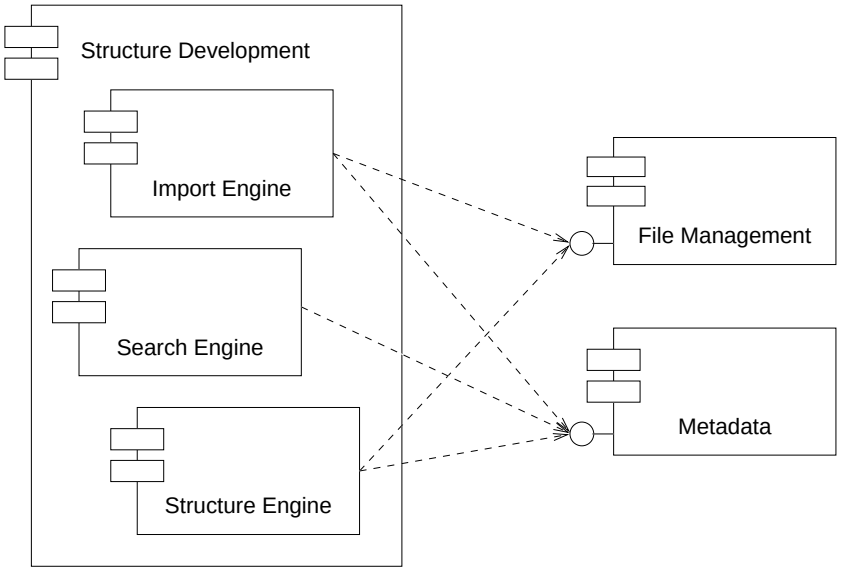


Abbildung 10.12: Komponente für die Rolle *Composer*

Die gleiche Namensgebung der Komponente **Import Engine** wie in der *Learning Object Development* ist nicht zufällig gewählt. Da die Aufgaben sehr ähnlich sind, aber die Anwendungsfälle im Detail anders gehalten sind, ist aus konzeptioneller Sicht diese Trennung einzuhalten. Bei der späteren Implementierung kann auf sie wahrscheinlich verzichtet bzw. über den Zugriff durch zwei Schnittstellen nachgebildet werden. An dieser Stelle wird die Aufteilung eines fremden Kurses in mehrere Lernobjekte plus einer Strukturierung unter dem Importieren fremder Inhalte verstanden. Im Gegensatz hierzu können auf Ebene der Lernobjektentwicklung nur einzelne Lernobjekte eingelesen werden, weil sich die Rolle *Developer* sonst in den Aufgabenbereich der Rolle *Composer* begibt. Anders herum kann die Rolle *Developer* aber ein Lernobjekt in beliebig viele andere aufbrechen, was wiederum der Rolle *Composer* nicht gestattet ist.

Mit Hilfe der Komponente **Search Engines** können Suchanfragen gestellt werden, die entweder auf lokale Datenbestände angewendet oder an spezielle Server delegiert werden. Als Resultat werden Listen von Referenzen auf Lernobjekte und Kurse geliefert, die sich in die eigene Struktur integrieren lassen. Diese Komponente implementiert selbst keine Suchalgorithmen, sondern benutzt Module für die verschiedenen Anfragen. So kann ein Suchmodul durch ein effizienteres ausgetauscht werden, ohne dass es Auswirkungen auf die Darstellung der Suchmaske hat. Bei den Umsetzungen der Module gibt es keine Einschränkungen: Von der einfachen Suche über Dateien, bei der jedes *Content Package* für die Metadaten geöffnet wird, bis hin zu verteilten Datenbanken, in denen über Indizes optimale Zugriffszeiten erreicht werden, sind alle Ansätze denkbar. Zu berücksichtigen sind nur die Vorgaben der Komponente *Metadata*, damit alle Felder der Metadatenstandards abgefragt werden können. Da die Syntax der Abfragesprache für die Architektur von geringer Priorität ist, soll diese Entscheidung auf den Klassenentwurf bzw. die Implementierung verschoben werden.

Der Aufbau der Struktur läuft über die Komponente **Structure Engine**, mit der standardkompatible *Content Packages* erzeugt werden. Angelehnt an die Struktur eines Manifests (siehe Abschnitt 3.5) können verschiedene Organisationen für die Lerninhalte erzeugt und bearbei-

tet werden, wodurch sich unterschiedliche Lernpfade definieren lassen, die sogar Bedingungen und Verzweigungen ermöglichen. Über Referenzen sind die einzelnen Elemente der Struktur mit physikalischen Ressourcen verbunden, die entweder Bestandteil des *Content Package* sind oder über eine URL auf einen externen Bereich verweisen. Bei Bedarf wandelt die Komponente solche externen Referenzen in physikalische Dateien um. Da neben Lernobjekten auch zusammengesetzten Strukturen als Ressourcen erlaubt sind, unterstützt die Komponente die Aggregation sowie Disaggregation der Organisationen. Weiter gedacht sind auch Operationen zur Umstrukturierung von Submanifesten oder zur Reduzierung mehrerer *Content Packages* zu einem nützliche Funktionen. Wie bei der Komponente *Learning Object Environment* sind Lade- und Speichermechanismus modular gehalten und interne Veränderungen werden über einen eigenen Nachrichtendienst propagiert.

10.2.4 Publishing Environment

Mit der Veröffentlichung der erzeugten E-Learning-Inhalte wird der letzte Schritt im Erstellungsprozess vollzogen. Die Rolle *Publisher* nutzt bei ihrer Arbeit die Komponente **Publishing Environment**, deren Teilkomponenten wieder aus den Anwendungsfällen hergeleitet sind. Abbildung 10.13 zeigt, wie die Gruppierung der Einzeltätigkeiten zur Erstellung von Layout- sowie Formateinstellung und Übersetzung die interne Struktur prägt.

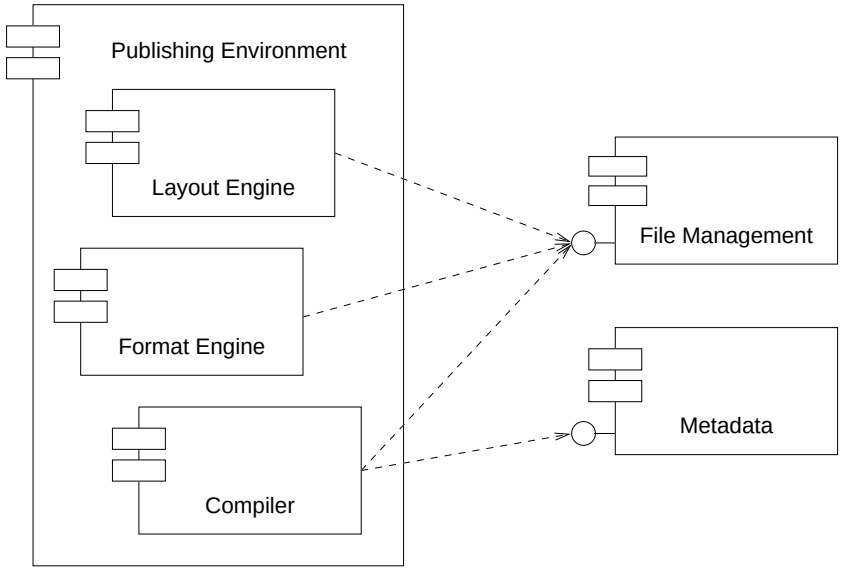


Abbildung 10.13: Komponente für die Rolle *Publisher*

Bei einem Layout handelt es sich um eine Beschreibung von Farbgestaltungen, Schriften, Seitenaufteilungen, Abständen und Navigationen, dessen Gestaltung über die Komponente **Layout Engine** erfolgt. Es werden quasi Schablonen erstellt, die bei der Übersetzung nur noch mit den Inhalten der Lernobjekte ausgefüllt werden. Bis auf den Punkt Navigation sollten die beschriebenen Parameter selbsterklärend sein. Weil nicht alle Lernplattformen in der Lage sind, aus den Manifesten eine geeignete Navigation abzuleiten, kann diese Funktion über das Layout gesteuert werden. Die Komponente zur Übersetzung wird durch die erzeugte Beschreibung angewiesen, zusätzlich eine entsprechende Navigation zu erstellen. Ein anderer Punkt sind Elemente wie Fuß- und Kopfzeilen, Logos, Verweise auf das Impressum, etc., die automatisch in alle Seiten generiert werden. Streng genommen wird hierdurch die Trennung von Darstellung und Inhalt verletzt, weil „Inhalte“ zum Bestandteil der Layout-Beschreibung werden. Technisch gesehen tritt diese Vermischung aber nicht ein, sodass sie auf fachlicher Ebene toleriert werden darf.

Mit den Kodierungsanweisungen trägt die Komponente **Format Engine** die zweite Beschreibungsform bei, die zusammen mit dem Layout die Übersetzung steuert. Sie wandelt ein

bekanntes Quellformat in ein beliebiges Zielformat um, auch in ein binäres oder proprietäres. Aus dieser Funktionsbeschreibung leitet sich ab, dass eine Übersetzung nur mit XSLT nicht hinreichend ist. Zwar gibt es Mechanismen wie die *XSL Formatting Objects* (XSL-FO) [Pawson02], deren Prozessoren aus XML PDF- und PS-Dateien generieren, aber andere wichtige Formate fehlen. Deshalb kann die Komponente um selbst entwickelte Module erweitert werden, mit denen sich einzelne Aufgaben oder komplette Umsetzungen realisieren lassen. Beispielsweise kann ein Modul MathML-Formeln[2] in PNG-Abbildungen umwandeln, um ältere Browser bei der HTML-Darstellung nicht auszuschließen. Wie bei anderen Komponenten zuvor, erfolgt die Speicherung der XML-Formatierungen und Module in eigenen Paketen, die mit anderen Systemen austauschbar sind.

Die eigentliche Übersetzung führt die Komponente **Compiler** durch. Ziel ist die Erzeugung einer oder mehrerer Dateien in einem Format, das mit einem Anzeigeprogramm dargestellt werden kann. Hierfür wird der Inhalt aus den Lernobjekten entnommen, nach der Layout-Vorlage arrangiert und anschließend über die Formatbeschreibung ausgegeben. Mit einem XSLT-Parser, einem XSL-FO-Renderer und einer Laufzeitumgebung für Module stellt diese Komponente die nötige Infrastruktur bereit.

10.2.5 User Environment

Aus der Definition der Rolle *User* als Stellvertreter für den generellen Umgang mit einer Lernplattform folgt unausweichlich, dass ihre Hauptkomponente eine Lernplattform ist. Weil die Anwendungsfälle dieser Rolle aus der Funktionsbeschreibung von Lernplattformen in Abschnitt 6.1 abgeleitet sind, könnten sie als bereits gegeben angesehen werden. Um die Konsistenz zu wahren, wird wie bei den anderen Komponenten verfahren, indem die Teilkomponenten durch Gruppierung der Anwendungsfälle entstehen. Im Wesentlichen geht es bei den Aufgaben um die Organisation und Kommunikation mit anderen Personen und die Nutzung der Inhalte. In Abbildung 10.14 wird neben diesen Teilkomponenten vollständigkeitshalber auch eine Verwaltungskomponente `Administration` definiert, die grundlegender Bestandteil einer Lernplattform ist.

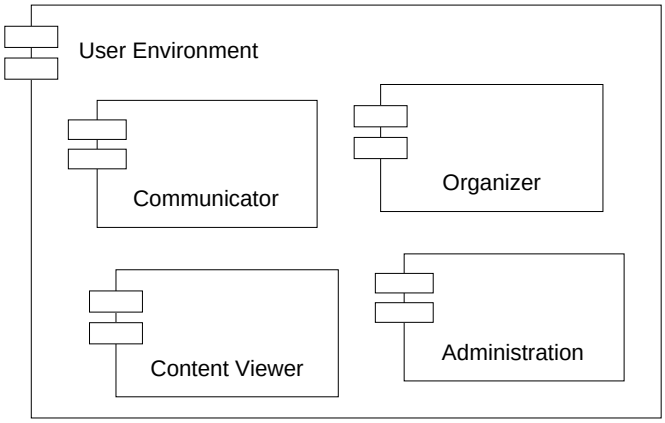


Abbildung 10.14: Komponente für die Rolle *User*

Bei der Komponente **Communicator** handelt es sich um eine Zusammenfassung aller gängigen Kommunikationsmethoden, wie z.B. E-Mail, Chat, Foren, *Whiteboard* und *Video-Conferencing*. Obwohl diese Mittel sich grundsätzlich in Bedienung und Technik unterscheiden, ist die Verallgemeinerung in einer Komponente für die angestrebte Architektur sinnvoll. In der Bewertung des Standes der Wissenschaft wurde bereits angedeutet, dass die heutigen Lernplattformen im Bereich Kommunikation gut ausgestattet sind, sodass es auf den Einsatz einer existierenden Lösung hinauslaufen wird.

[2]MathML ist eine Auszeichnungssprache für mathematische Formeln

Ähnlich verhält es sich mit der Komponente **Organizer**, die alle Mittel zur Planung der Arbeitsabläufe einschließt. Die Kalender, To-Do-Listen und Notizblätter der heutigen Lernplattformen sind bereits auf einem Stand, der eine Eigenentwicklung praktisch ausschließt.

Wenn die Lernplattformen bereits die gesamte Funktionalität liefern, die von der Rolle *User* benötigt wird, stellt sich die Frage, wozu eine Aufteilung in Komponenten erfolgt? Eine Antwort ist die Komponente **Content Viewer**, mit der die Inhalte recherchiert und angezeigt werden. Das Ziel dieser Arbeit im Hinterkopf wird schnell die Schwäche aktueller Lernplattformen deutlich, denn sie sind für die Präsentation „monolithischer“ Inhalte ausgelegt, die starre Strukturen vorgeben. Adaptives bzw. personalisiertes Lernen wird nicht unterstützt und überspitzt ausgedrückt, leisten Lernplattformen beim Suchen oder Anzeigen nicht mehr als gewöhnliche Web-Server. Die Komponente *Content Viewer* hingegen kennt modulare Inhalte, kann für mehrere Lernpfade Navigationen erzeugen und unterstützt eine kontextabhängige Suche in nicht explizit zugewiesenen Lernobjekten. Hierdurch sind beispielsweise Exkurse zu bestimmten Themen möglich, da die Komponente aus den Metadaten „weiß“, welche Lernobjekte inhaltlich zusammen passen.

Die Komponente **Administration** unterstützt die Rolle *User* bei Einstellungen und wiederkehrenden Tätigkeiten. Funktionsumfang und Benutzung hängen wieder stark vom eingesetzten System ab, sodass keine pauschale Beschreibung möglich ist. Nur wenige grundlegende Funktionen, wie die Bearbeitung der persönlichen Daten, z.B. das Zugangspasswort, sind obligatorisch.

Bei den Anwendungsfällen dient die Rolle *User* zur Zusammenfassung ähnlicher Tätigkeiten der Spezialisierungen *Professor* und *Student*. Aus der Komponentenbildung für die Rolle *Author* ließe sich nun schließen, dass auch die Komponente *User Environment* allgemeine Dienste anbietet, die von den Komponenten der spezialisierten Rollen aufgerufen werden. Dies ist aber nicht der Fall, weil für die Rollen *Professor* und *Student* keine eigenen Komponenten definiert werden. Der Grund liegt in den erweiterten Anwendungsfällen, die zwar eine semantische Differenzierung bringen, technisch aber keine Auswirkung haben. Ob aus dem Fall „Inhalte Nutzen“ nun „Lernen“ oder „Lehren“ wird, ist für das System irrelevant. Durch diese Trennung zwischen fachlicher und technischer Bedeutung vereinfacht sich die Architektur, ohne dass es zu fachlichen Lücken im System kommt. Alle definierten Anwendungsfälle, auch die der Rollen *Professor* und *Student*, lassen sich ohne Einschränkungen abarbeiten.

10.2.6 Administration

Die Komponentenbildung für die Rolle *Administrator* gestaltet sich schwierig, weil sie neben den Verwaltungstätigkeiten den anderen Rollen bei technischen Problemen zur Seite steht. Dementsprechend vage sind auch die Teilkomponenten in Abbildung 10.15.

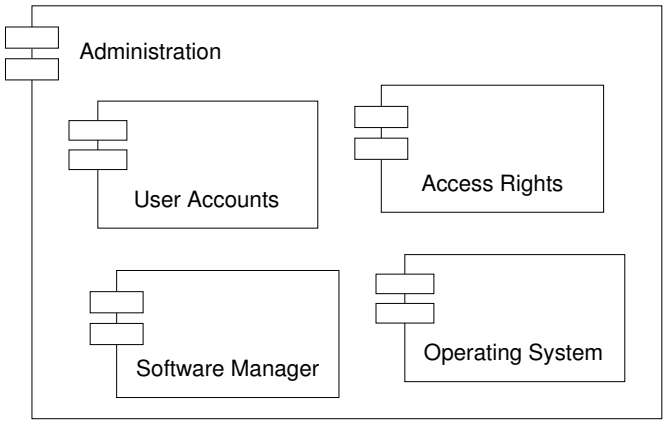


Abbildung 10.15: Komponente für die Rolle *Administrator*

Als Teil der Lernplattform verwaltet die Komponente **User Accounts** die Personen- und Zugangsdaten sowie etwaige Gruppenzugehörigkeiten. Die Einstellung der Zugriffsrechte für Ressourcen und Bereiche der Lernplattform erfolgt über die separate Komponente *Access Rights*.

In den Aufgabenbereich des Betriebssystems geht die Komponente *Software Manager*, über die alle Installationen und Updates verwaltet werden. Da sich die gesamte Architektur aus verschiedenen Programmen zusammenstellt, hilft eine zentrale Verwaltung, wie z.B. die *Package*-Mechanismen der Linux Distributionen *Debian* und *Red Hat*, den Überblick zu wahren. In einer Datenbank werden alle Dateipfade, Versionen, Abhängigkeiten, etc. gespeichert, um bei Änderungen der Programmkonstellation mögliche Konflikte aufzudecken. Diese Werkzeuge sind so weit fortgeschritten, dass von einer Eigenentwicklung abzusehen ist.

Gleiches gilt für die Komponente *Operating System*, die stellvertretend für alle Überwachungs-, Analyse- und Konfigurationswerkzeuge der Betriebssysteme steht. Wenn technische Probleme auftreten, kann die Rolle *Administrator* mit diesen Programmen interne Vorgänge nachvollziehen und Konfigurationen verändern. Die Möglichkeiten sind dermaßen vielfältig, dass sie sich nicht vollständig spezifizieren lassen. Auch wenn der Begriff Komponente in diesem Fall überstrapaziert wird, kann durch diese vereinfachte Sicht ein Bereich der Architektur berücksichtigt werden, der absolut unvorhersehbar ist.

10.3 Architektur

Nach der Komponentenbildung kann nun die Architektur des gesamten Systems für die Bearbeitung modularer E-Learning-Inhalte erstellt werden. Zuerst erfolgt eine Zusammenfassung der verschiedenen Komponenten zu Programmen, die fachliche und technische Bezüge haben, denn nicht jede Rolle bekommt ein eigenes Programm. Durch geschickte Kombination lassen sich manche Komponenten zu einer zusammensetzen, wodurch bei der späteren Implementierung weniger Arbeit anfällt.

Die Übersicht der Rollen in Abbildung 10.1 enthält bereits eine Aufteilung in drei Gruppen, die folgend näher analysiert wird: Bearbeitung von E-Learning-Inhalten, verkörpert durch die Rolle *Author*, Nutzung von E-Learning-Inhalten, vertreten durch die Rolle *User*, und die Administration des Systems. Pauschal jedem Aufgabenbereich ein Programm zur Seite zu stellen, wird einer vernünftigen Architektur leider nicht gerecht. Auf jeden Fall stehen die Komponenten der Rolle *Administrator* außen vor, denn die sind entweder Bestandteil der anderen Programme oder Werkzeuge des Betriebssystems. Die grobe Trennung in Erstellung und Nutzung von Inhalten bleibt somit als erster Anhaltspunkt übrig.

Für die Erstellung modularer E-Learning-Inhalte werden die Komponenten der Rollen *Developer*, *Composer* und *Publisher* in einer Komponente vereint. Dieses Vorgehen hat mehrere Vorteile, weil es bei der Arbeit in einer Rolle häufiger vorkommt, dass für bestimmte Tätigkeiten ein Wechsel in eine andere Rolle nötig ist. Andeutungen für enge Kooperationen haben sich schon bei den Rollenbeschreibungen abgezeichnet. Ein gutes Beispiel ist die Rolle *Composer*, die bei einer vollständigen Produktion im Mittelpunkt steht. In ihrer Haupttätigkeit kombiniert sie fremde und eigene Lernobjekte bzw. Strukturen zu Kursen. Was soll aber geschehen, wenn ein fremdes Lernobjekt nicht zu 100% passt und eine leichte Modifikation benötigt? Es entnehmen und unter anderer Rolle in einem neuen Programm öffnen? Das ist gewiss sehr umständlich und sicher nicht erwünscht. Eine bessere Lösung ist die direkte Bearbeitung des Lernobjekts in der Struktur, wodurch zwar beide Rollen (*Developer* und *Publisher*) Einblick in die Arbeit der jeweils anderen Rolle bekommen, aber konzeptionell ergibt sich hieraus kein Problem. An dieser Stelle sei noch einmal explizit betont, dass ein und die selbe Person in verschiedenen Rollen auftreten darf. Durch die Zusammenlegung der Komponenten werden die Übergänge unschärfer, was bei der täglichen Arbeit aber durchaus wünschenswert ist, denn ein schnelles Umschalten zwischen den Rollen erlaubt einen ungestörten Arbeitsfluss. Gleiches gilt auch für die Rolle *Publisher*, die beispielsweise zu Kontrollzwecken bei der Bearbeitung

von Lernobjekten oder Kursen eingenommen wird. Abbildung 10.16 zeigt die resultierende Komponente **Authoring Environment**.

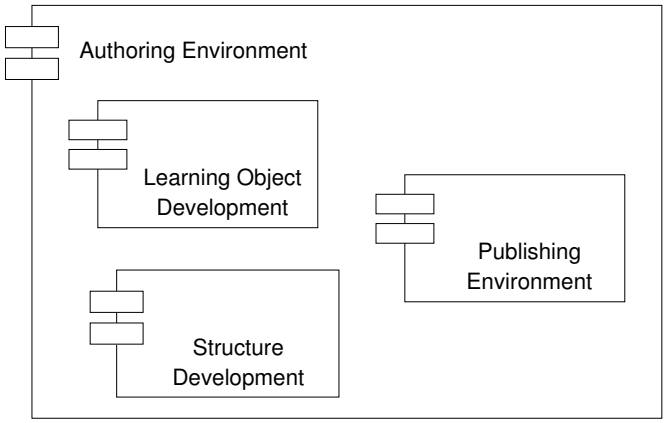


Abbildung 10.16: Funktionale Komponente des Autorensystems

Aufgrund der engen Zusammenhänge und Arbeitsabläufe stellt diese Komponente das ideale Werkzeug für die Bearbeitung modularer E-Learning-Inhalte dar. Doch bis jetzt handelt es sich um eine funktionale Komponente, die keine grafische Darstellung oder direkte Ansteuerung hat. Um ein vollwertiges Autorenwerkzeug zu erhalten, wird die Komponente *Authoring Environment* mit den Komponenten **Graphical User Interface** und **Scripting Environment** verbunden. Abbildung 10.17 verdeutlicht diese Erweiterung.

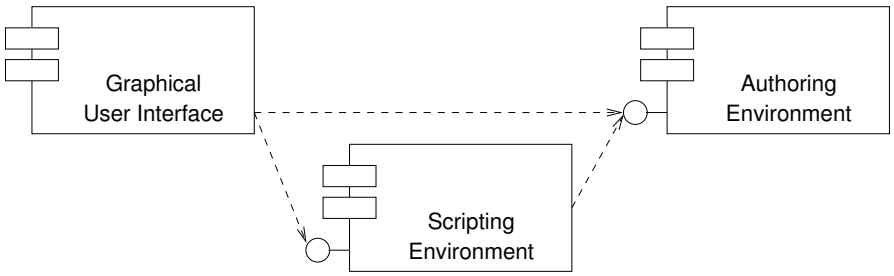


Abbildung 10.17: Zwei Komponenten zur Steuerung des Autorensystems

Hauptaufgabe beider Komponenten ist die Ansteuerung der Funktionen, entweder über eine grafische Oberfläche oder durch *Scripts*. Die Komponente *Graphical User Interface* ist beliebig und gibt keine konkrete Darstellung vor. Von der statischen Repräsentation eines Lernobjekts bis hin zu Modifikationen mit Drag'n'Drop in verschachtelten Strukturen ist alles möglich. Weil aber die Akzeptanz eines Programms mit der Leistung der Oberfläche steht und fällt, sei eine benutzerfreundliche Bedienung angemahnt, die alle Möglichkeiten der Komponente *Authoring Environment* offenbart. Es wäre äußerst unglücklich, wegen der eingeräumten Gestaltungsmöglichkeiten das Potential des Systems einzuschränken. Dennoch soll auf spezifischere Angaben verzichtet werden, um nicht von vornherein bestimmte Ansätze auszunehmen. Ähnlich verhält es sich mit den zu unterstützenden *Scripts*. Lediglich die Möglichkeit zur Stapelverarbeitung wird eingefordert, um einen Mechanismus für wiederkehrende Tätigkeiten anbieten zu können. Welche Sprache letztendlich eingesetzt wird, bleibt der Implementierung überlassen. Interessant ist auch die Verbindung der beiden Steuerungskomponenten, durch die sich das grafische Autorenwerkzeug mit Hilfe von *Scripts* beliebig erweitern lässt.

Das zusammengesetzte Autorenwerkzeug ist für sich genommen vollständig, genügt aber nicht der festgelegten Zielsetzung in Abschnitt 1.2 bzw. des verfeinerten Resümees in Abschnitt 9.1. Da ist von zentraler Datenhaltung und der Unterstützung von Teamarbeit die Rede. Wie lassen sich diese Anforderungen mit den bisherigen Komponenten vereinen? Warum tauchen sie bei keiner Rolle als Anwendungsfall auf? Die Antwort ist einfach: Diese Aspekte sind für

die Rollen unwichtig. Es ist bei der Arbeit egal, ob eine Ressource im eigenen Dateisystem liegt, oder Bestandteil einer komplexen Datenbank ist. Synchronisation und nahtloser Zugriff sind Aufgabe des Systems und dürfen die Rollen bei ihrer Arbeit nicht belasten. Für die Implementierung sind die Themen Teamarbeit und zentrale Datenhaltung umso wichtiger, weil technische Details hinter einem allgemeinen Mechanismus versteckt werden müssen. Dieser kann nur über die Teilkomponente *File Management* aus Unterabschnitt 10.2.1 laufen, die auch Funktionen wie Sperren und Versionierung anbietet. Sie ist die ohnehin geplante Schnittstelle zur Ressourcenverwaltung und muss intern die nötigen Dienste bereitstellen.

Wenn die zentrale Datenhaltung für die Realisierung eine so hohe Bedeutung besitzt, muss sie auch in der Architektur berücksichtigt werden. Hierfür wird eine neue Komponente eingeführt, die keinen direkten Kontakt zu den Rollen hat, aber mit dem Autorenwerkzeug kommuniziert. Abbildung 10.18 zeigt sie mit ihren bekannten Teilkomponenten, die alle Bestandteil bereits vorgestellter Komponenten sind.

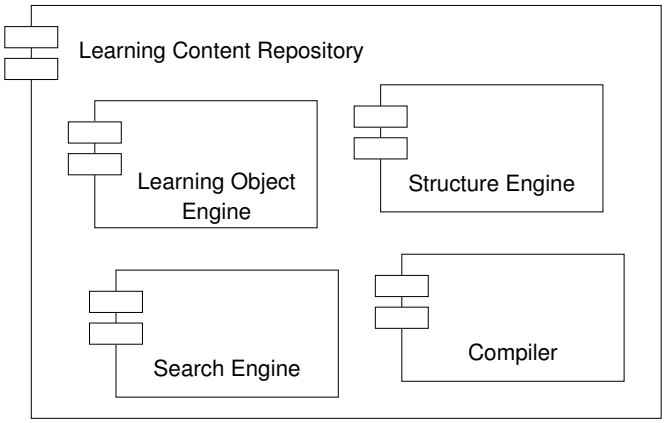


Abbildung 10.18: Komponente für die zentrale Datenhaltung (Repository)

Wie der Name **Learning Content Repository** andeutet, ist diese Komponente für die Haltung modularer E-Learning-Inhalte zuständig. Da sie nur aus bekannten Teilkomponenten besteht, hält sich der zusätzliche Entwicklungsaufwand in Grenzen. Aus Gründen der Übersicht ist die Komponente *Authoring Environment* bzw. deren Teilkomponente *File Management* in der Abbildung nicht eingezeichnet. Implizit wird diese Funktionalität über die Komponenten *Learning Object Engine* und *Structure Engine* angeboten, die einen transparenten Zugriff ermöglichen, ihn selbst aber nicht implementieren. Neben der Datenhaltung von E-Learning-Inhalten ist auch ihr Auffinden ein zentrales Anliegen. Hierfür ist die Komponente *Search Engine* zuständig, die Suchanfragen entgegennimmt und entsprechend delegiert. Für die serverseitige Kontrolle der Arbeit ist auch die Komponente *Compiler* integriert, um direkt Kurse übersetzen zu können, ohne den Umweg über ein Autorensystem als Client gehen zu müssen.

Genau wie beim Autorensystem bietet die Komponente *Learning Content Repository* ihre Funktionalität an und besitzt keine direkte Ansteuerung durch die Anwender/-innen. Eine grafische Oberfläche oder Steuerung über Skripten ist nur bedingt sinnvoll, darf aber nicht ausgeschlossen werden. Besonders wichtig ist die Verfügbarkeit über ein Netzwerk und der simultane Zugriff mehrerer Autorenwerkzeuge. Aus diesem Grund wird die Komponente **Web Server** als Schnittstelle eingeführt, die über die vorhandene Internet-Infrastruktur angesprochen wird. Ob nun Dienste als *Remote Procedure Calls* (RPC) oder über eine Web-Oberfläche aufgerufen werden, ist zunächst von untergeordneter Wichtigkeit, denn die verschiedenen Zugänge schließen sich nicht aus und können jederzeit um neue erweitert werden. Abbildung 10.19 verdeutlicht diese Konstellation der Komponenten.

Nun ist die Entwicklungsumgebung modularer E-Learning-Inhalte vollständig, jedoch auf einem hohen Abstraktionsniveau. Die Verbindungen zwischen den einzelnen Programmen und

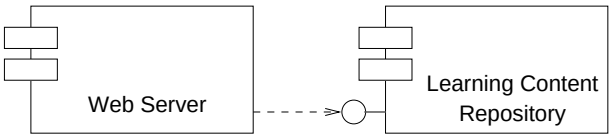


Abbildung 10.19: Komponente für den Web-basierten Zugriff auf das *Repository*

Rollen ist noch nicht in allen Details beschrieben und lässt Raum für Interpretationen. Um diese einzuschränken, wird in Abbildung 10.20 eine vereinfachte Ansicht auf die Architektur gegeben, bei der die Beziehungen zwischen den Rollen, Programmen und Daten offensichtlich ist.

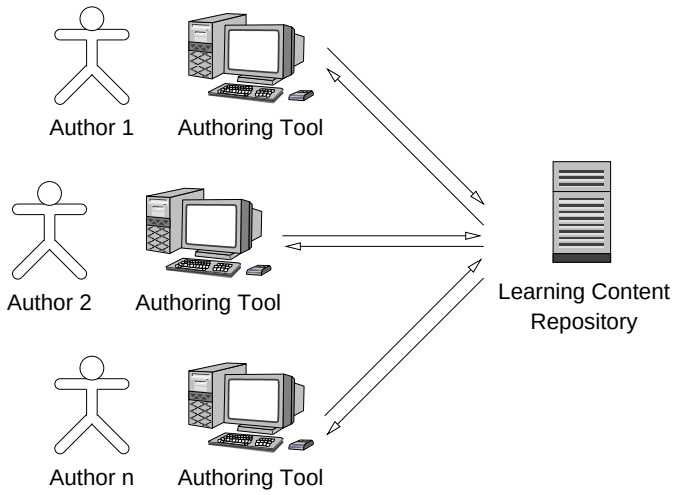


Abbildung 10.20: Zugriff der Autoren/-innen auf das *Repository*

Stellvertretend für alle Spezialisierungen nutzt die Rolle *Author* das Autorenwerkzeug. Zur Verdeutlichung der zentrale Datenhaltung und parallelen Bearbeitung sind n Personen eingezeichnet, die sich an getrennten Orten befinden. Die Pfeile in beide Richtungen stehen für den Datenfluss der Lernobjekte und Kurse. Existierende und neue Inhalte lassen sich transparent auf dem *Learning Content Repository* bearbeiten bzw. einspielen. Solche Änderungen müssen keine direkten Auswirkungen auf die Arbeit der anderen Personen haben. Sogar wenn das selbe Lernobjekt in Arbeit ist, muss es nicht zwangsläufig zu Konflikten kommen. Wann immer möglich und nötig, verdeckt die Infrastruktur die „technische“ Präsenz anderer Entwickler/-innen. Nur beim ausdrückliche Wunsch auf Synchronisation bzw. Absprache bei der gemeinsamen Arbeit, erlaubt das *Learning Content Repository* eine Aufhebung dieser Transparenz. Um Missverständnissen entgegen zu wirken, sei darauf hingewiesen, dass es nicht um die Verheimlichung der fachlichen Zusammenarbeit geht. Die Koordination läuft nur nicht über das Autorensystem, z.B. mit Kommunikationsmitteln wie Chat oder E-Mail, sondern muss extern geleistet werden. Auf der Seite der Entwicklungsumgebung fallen nur keine zusätzlichen Absprachen an, die in der technischen Umsetzung begründet liegen.

Mit der vollständigen Entwicklungsumgebung für modulare E-Learning-Inhalte steht die erste Säule der Architektur. Nun werden mit Hilfe der Rolle *User* und ihren beiden Spezialisierungen die Programme für die Nutzung der Inhalte auf gleiche Weise hergeleitet. Wie bereits angedeutet, soll diese Aufgabe von einer Lernplattform übernommen werden, die über einen gängigen Webbrowser angesprochen wird. Ähnlich der Komponente *Learning Content Repository* bietet *User Environment* lediglich ihre Funktionalität an und benötigt für die Ansteuerung eine separate Komponente. Weil über die internen Abläufe der eingesetzten Lernplattform zu diesem Zeitpunkt nichts bekannt sein kann, ist die Komponente **Web Server** in Abbildung 10.21 schematischer Natur.

Im Unterabschnitt 10.2.5 über die Komponente *User Environment* wurde die Schwäche der Lernplattformen im Umgang mit modularen Inhalten diskutiert. Die Leistungen auf den

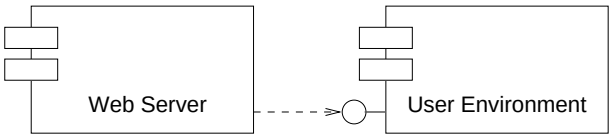


Abbildung 10.21: Komponente für den Web-basierten Zugriff auf die Lernplattform

Gebieten Suche und Darstellung sind so unzureichend, dass eine zusätzliche Lösung unumgänglich ist. In allen Lernplattformen ist es möglich, auch externe Inhalte zu referenzieren bzw. über bestimmte Mechanismen direkt durch das System darstellen zu lassen. Auf diese Weise soll das *Learning Content Repository* eingebunden werden, das die geforderten Funktionen anbietet. Diese Komponente erlaubt die Kombination von Lernobjekten und höheren Strukturen zur Laufzeit, sodass sich beim Lernen individuelle Pfade einschlagen lassen. Dank der Übersetzungsfähigkeit erscheinen alle Inhalte im gleichen Design und die redundante Datenhaltung mehrerer Kopien des selben Lernobjekts in verschiedenen Kursen entfällt. Auf diese Weise kann selbst eine leistungsschwache Lernplattform mit geringem Aufwand in ein modernes E-Learning-Portal verwandelt werden. Anwender/-innen in der Rolle *Professor* oder *Student* benötigen für ihre Tätigkeiten nicht den vollen Leistungsumfang des *Learning Content Repositories*, der bei der Entwicklung von Inhalten benötigt wird. Ihnen genügt eine eingeschränkte Nutzung, bei der sie lesend auf alle Inhalte zugreifen können.

Jetzt sind alle Programme und Komponenten benannt, die für die Nutzung der Inhalte notwendig sind. Über eine vereinfachte Darstellung sollen sie in Relation gebracht werden, um eine genauere Vorstellung über ihren Einsatz zu geben. In Abbildung 10.22 greift die Rolle *User* stellvertretend für die Spezialisierungen mit einem Browser auf das System zu.

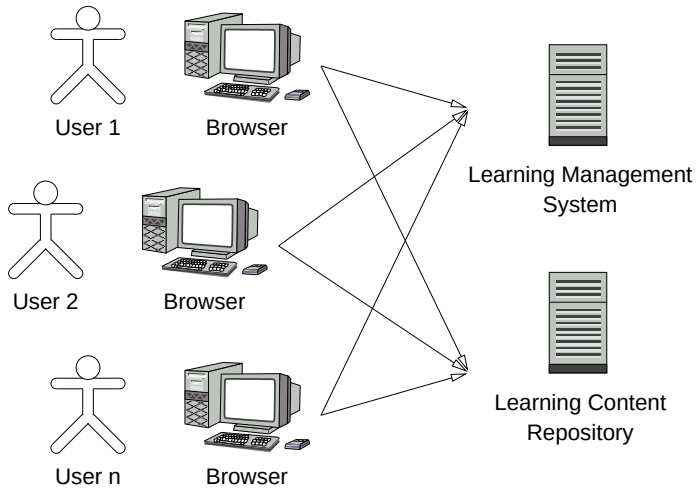


Abbildung 10.22: Zugriff der Benutzer/-innen auf die Lernplattform und das *Repository*

Im Gegensatz zu Abbildung 10.20 geht der Datenfluss der Inhalte, erkennbar an den Pfeilen, nur von der Lernplattform und dem *Learning Content Repository* zu den Browsern. In umgekehrter Richtung werden keine Inhalte transportiert. Gleichwohl es einen bidirektionalen Datenfluss gibt, wenn beispielsweise über ein Chat kommuniziert wird, ist er für das Verständnis unbedeutend und wird vernachlässigt. Die n Personen nutzen gleichzeitig die üblichen Dienste der Lernplattform und beziehen die modularen Inhalte vom *Learning Content Repository*. Im idealen Fall bekommen die Anwender/-innen von der Teilung auf zwei Server nichts mit.

Da in diesem Entwurf für die Rolle *Administrator* keine eigenen Programme modelliert werden, sind nun alle Rollen mit Werkzeugen versorgt. Für eine Übersicht der gesamten Architektur werden die Abbildungen 10.20 und 10.22 zusammengefasst. Es werden allerdings die spezialisierten Rollen anstatt der allgemeinen angegeben, um alle Facetten der Infrastruktur

zu berücksichtigen. Zusätzlich sind die Pfeile für den Datenfluss mit den Formaten und Typen beschrieben. Als Ergebnis ergibt sich eine Architektur, die in Abbildung 10.23 dargestellt ist.

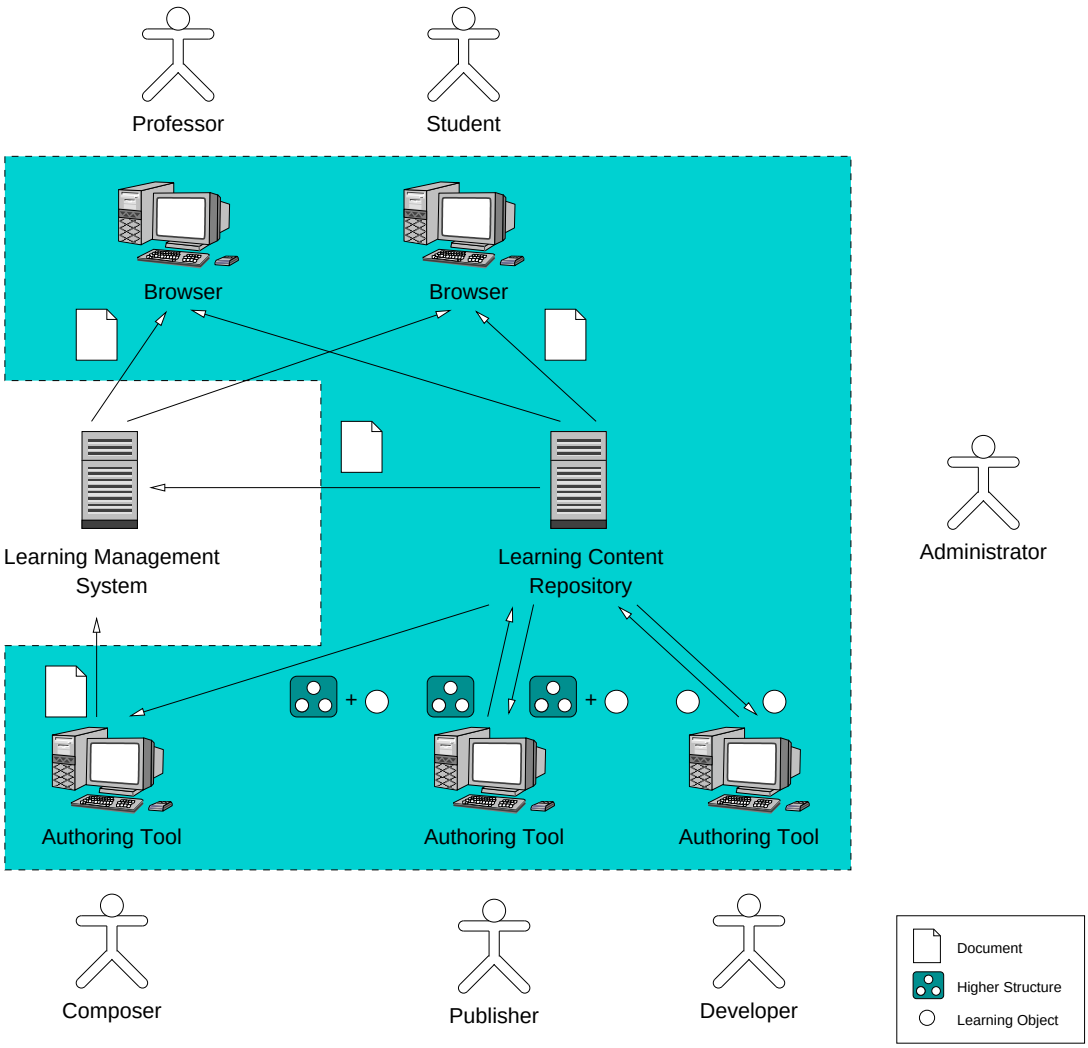


Abbildung 10.23: Vollständige Architektur des Systems

Unten befinden sich alle Rollen, die mit der Entwicklung von modularen E-Learning-Inhalten beschäftigt sind. Die Rolle *Developer* arbeitet mit Lernobjekten, die sie auf dem Server speichern bzw. von ihm herunterladen kann. Diese Lernobjekte kombiniert die Rolle *Composer* mit anderen Inhalten zu neuen Aggregationen. Durch die Wiederverwendung von Strukturen kann eine beliebige Aufteilung der Inhalte erfolgen, sodass es von der technischen Seite aus keine Einschränkungen für das didaktische Konzept gibt. Lernobjekte sowie Kurse werden anschließend von der Rolle *Publisher* mit einem individuellen Layout versehen und in ein unterstütztes Zielformat übersetzt. Das Aufspielen des Resultats auf die Lernplattform kann auf zwei Wegen erfolgen. Entweder importiert das Autorenwerkzeug die Daten direkt oder steuert die Übersetzung auf dem*Learning Content Repository*. Zwischen den beiden Server wurde nämlich eine neue Verbindung eingetragen, die in Abbildung 10.22 noch nicht vorhanden ist. Weil die Komponente zur Übersetzung im Autorensystem und *Learning Content Repository* identisch ist, ergibt sich hieraus technisch keine Neuerung. Diese Verbindung ist auch sehr praktisch für die transparente Integration der Inhalte auf der Lernplattform. Entweder erhalten die Rollen *Professor* und *Student* den direkten Zugang zu den Lernmaterialien über die Lernplattform oder, wenn ein Datenaustausch zwischen den Servern nicht unterstützt ist, werden über Links auf das *Learning Content Repository* weitergeleitet. Welche technische

Lösung letztendlich zum Einsatz kommt, ist beiden Rollen gleich. Die Rolle *Administrator* ist vollständigkeitshalber am Rand aufgeführt.

Der gestrichelte Rahmen schließt die Systeme und Programme ein, die bei der Entwicklung des Systems berücksichtigt werden. Nur die Lernplattform wird als gegeben angesehen und soll nicht verändert bzw. neu entwickelt werden. Um möglichst flexibel bei der Auswahl der Lernplattform zu bleiben, müssen alle anderen Programme so ausgelegt sein, dass sie mit dem eingesetzten System kooperieren können. Das Autorenwerkzeug muss auf jeden Fall vollständig neu entwickelt werden. Modulare E-Learning-Inhalte, die den Theorien über Lernobjekte entsprechen, können mit heutigen Autorenwerkzeugen nicht erstellt werden. Hieraus folgt auch die vollständige Neuentwicklung des *Learning Content Repository*, das modulare Inhalte analysieren und aufbereiten soll. Da auch neue Dienste über die Browser aufgerufen werden, sind sie Bestandteil des ausgezeichneten Bereichs.

10.4 Baukasten-Metapher

Der Entwurf des Systems soll mit Hilfe von Metaphern effizienter entwickelt werden und späteren Benutzern/-innen die Gewöhnung erleichtern. Dies gelingt aber nur mit geeigneten Metaphern, deren Bedeutung allgemein bekannt ist und die einen wirklichen Bezug zu Teilaspekten des Systems haben. Aus diesem Grund ist es wichtig, deren Sinnhaftigkeit im Vorfeld zu überprüfen, denn eine falsche Auswahl kann sich nachteilig auswirken.

Als Grundlage der folgenden Erörterungen dient der metaphorische Prozess aus Kapitel 8, der die Rahmenbedingungen der Begriffsbildung vorgibt. Die Idee dieser Arbeit, Lerneinheiten als einzelne Module zu betrachten, die flexibel erstellt, kombiniert, gewartet, gespeichert und wiederverwendet werden, assoziiert ein System mit gewissen Eigenschaften, die über die Metapher des **Baukastens** verstärkt werden sollen. Für eine differenziertere Betrachtung der Details wird sie als Wortgruppe aufgefasst, die des Weiteren die Metaphern **Baustein**, und **Bauplan** beinhaltet. Die Bildung dieser Wortgruppe bedeutet eine Verteilung von Eigenschaften des Baukastens auf mehrere Begriffe, sodass gewisse Aspekte des Systems in kleineren, in sich abgeschlossenen Metaphern, betrachtet werden können. Jede von ihnen vereinfacht die Analyse und Bewertung von Anwendungsfällen, indem die wichtigen Eigenschaften der beteiligten Objekte und Handlungen hervorgehoben werden.

Es gibt eine Reihe weiterer Metaphern, wie z.B. „Werkzeug“, „Werkbank“ und „Stecksystem“, die sich auf den ersten Blick in die vorgeschlagene Wortgruppe einreihen könnten, jedoch hat sich in der praktischen Arbeit gezeigt, dass die vier Substantive für die Beschreibung des Systems völlig ausreichend sind [Bungenstock02; Baudry02b]. Detaillierte Metaphern können den Entwurf sogar erschweren, wenn sie nicht benötigte Eigenschaften implizieren.

10.4.1 Metaphorischer Prozess

Der Gegenstand dieses metaphorischen Prozesses ist das Software-System, mit dessen Hilfe modulare E-Learning-Inhalte genutzt werden sollen. Die Übertragung der Wortgruppe Baukasten, Baustein und Bauplan vollzieht sich aus dem üblichen Kontext des Kinderspielzeugs in den unüblichen der Software. Es werden nun die einzelnen Aspekte des metaphorischen Prozesses beschrieben:

- Die Metaphern-**Produzenten/-innen** für diese Wortgruppe lassen sich in der Literatur nicht genau ausmachen. Lernmaterialien als Bausteine zu betrachten ist seit Hodgins (siehe Abschnitt 3.2.2) sehr beliebt, aber er ist gewiss nicht der Urheber dieser Metapher. Auch wenn ihm die Idee beim Betrachten seiner spielenden Kinder kam, wird die Metapher seit längerem im Kontext modularer Aufteilungen genutzt. Die anderen Metaphern Baukasten und Bauplan leiten sich konsequenterweise vom Baustein ab, lassen sich in der Literatur im Kontext modularer E-Learning-Inhalte aber nicht belegen. Dennoch soll Hodgins in dieser Arbeit als Produzent betrachtet werden, weil so eine sehr interessante

und hilfreiche Perspektive auf diese Wortgruppe entsteht. Da er nicht aus der Informatik kommt, sind die Rezipienten/-innen auf der Anwendungsebene zu sehen.

- Die **Rezipienten/-innen** sind alle Personen, die mit modularen E-Learning-Inhalten arbeiten. Aus Sicht der Rollen gehören hierzu *Developer*, *Composer*, *Publisher*, *Professor* und *Student*. Sie alle profitieren mehr oder weniger von den Metaphern. Würde die Rolle *Developer* auch gut ohne die Metapher Baustein auskommen, passt sie im Gegenzug ideal zu den Aufgaben der Rolle *Composer*. Zudem ist es ein leichtes, diese Metaphern auf die Entwickler/-innen auszuweiten, um ihnen beim Entwurf ein Leitbild an die Hand zu geben. Neben den bisherigen Komponenten helfen die Metaphern zusätzlich bei der Strukturierung und Umsetzung.

- Die **Funktionen** der Metaphern hängen von der Intention der Produzenten/-innen ab. Sehr wichtig ist die affektiv-emotionale Funktion des Begriffs „Baukasten“: Es wirkt vertraut und suggeriert einen kindlich einfachen Umgang. Auf diese Weise sollen Berührungsängste genommen und Interesse geweckt werden. Diese Metapher lädt einfach zum „Spielen“ mit dem System ein. In die gleiche Richtung wirkt die Prädikationsfunktion. Die gewollte Analogie zielt besonders auf die Bedienung des Systems ab, indem modulare E-Learning-Inhalte wie Bausteine einfach zusammengesetzt werden.

- Das System selbst bzw. Teile von ihm und die E-Learning-Inhalte sind **Gegenstand** der Metaphern.

- Der **übliche Kontext** erstreckt sich über verschiedene Bereiche, wobei der beherrschende sicherlich Spielen bzw. Spielzeug ist.

- Den **unüblichen Kontext** bildet in diesem Fall die Informatik.

10.5 Aufteilung

In diesem Kapitel sind so viele Komponenten entstanden, dass in dieser Arbeit nicht der Platz ist, ihre Entwicklung in allen Details zu beschreiben. Zudem ist die Entwicklung komplexer Anwendungen eine Aufgabe für Teams, so auch hier. Fast alle Ergebnisse des Projekts mαthkit wurden von mehreren Personen entwickelt, diskutiert und veröffentlicht. Für eine bessere Koordination wurden die Verantwortungen für die jeweiligen Komponenten auf Einzelne übertragen. In diesem Abschnitt werden nun die Arbeiten der Beteiligten vorgestellt, die maßgeblich mitgewirkt haben. Die resultierenden Komponenten werden dann mit den Ergebnissen dieser Arbeit im Teil „Implementierung“ zu einem System zusammengesetzt.

Im Rahmen seiner Dissertation übernimmt Andreas Baudry neben der Komponenten *Import Engine* — es handelt sich um die Teilkomponente für die Rollen *Developer* sowie *Composer* aus den Abbildungen 10.11 und 10.12 — alle Komponenten für die Rolle *Publisher*. Die bisherigen Ergebnisse dieser Arbeit finden sich in [Baudry04b; Baudry04a; Baudry03; Baudry02a] und entsprechen den Anforderungsbeschreibungen aus Unterabschnitt 10.2.4.

In der Diplomarbeit von Marc Vollmann [Vollmann04] werden verschiedene Suchalgorithmen auf Basis des fallbasierten Schließens untersucht, die in die Komponente *Search Engine* aus Abbildung 10.12 einflossen. Die Ansteuerung erfolgt über das *Simple Object Access Protocol* (SOAP), sodass sich eine Integration in das eigene Programm problemlos vollzieht.

Mit der Diplomarbeit [Turan04] wurde die Implementierung der bereits genannten Komponenten *Import Engine* und der zur *Authoring Foundation* gehörenden Teilkomponente *Metadata* aus Abbildung 10.10 unterstützt.

Alle bis jetzt nicht aufgeführten Komponenten, die für die Umsetzung des Autorenwerkzeugs und des *Learning Content Repository's* von Bedeutung sind, werden nun in den nachfolgenden Kapiteln hergeleitet und zu einem System für modulare E-Learning-Inhalte zusammengesetzt.

Komponente	Typ	Kapitel	Verantwortung	Kontext
File Management	Basis	11	MB	Dissertation
Metadata	Basis	11	MB	Dissertation
Multimedia Environment	Basis	11	MB	Dissertation
Import Engine	Basis	-	AB	Dissertation
LOB Engine	Basis	12	MB	Dissertation
LOB Development	Aggregation	13	MB	Dissertation
Search Engine	Basis	-	MV	Diplomarbeit
Import Engine	Basis	-	AB	Dissertation
Structure Engine	Basis	12	MB	Dissertation
Structure Development	Aggregation	13	MB	Dissertation
Layout Engine	Basis	-	AB	Dissertation
Format Engine	Basis	-	AB	Dissertation
Compiler	Basis	-	AB	Dissertation
Publishing Environment	Aggregation	-	AB	Dissertation
Authoring Environment	Aggregation	13	MB	Dissertation

MB: Michael Bungenstock AB: Andreas Baudry MV: Marc Vollmann

Tabelle 10.1: Arbeitsteilung für systemunabhängige Komponenten

Komponente	Typ	Kapitel	Verantwortung	Kontext
User Environment	Extern	-		
Administration	Extern	-		
Graphical User Interface	Basis	14	MB	Dissertation
Scripting Environment	Basis	14	MB	Dissertation
Learning Content Repository	Aggregation	15	MB	Dissertation
Web-Server	Basis	15	MB	Dissertation

MB: Michael Bungenstock

Tabelle 10.2: Arbeitsteilung für proprietäre Komponenten

Kapitel 11

Basiskomponenten

Aus den Komponenten und der Architektur des vorherigen Kapitels soll nun ein objektorientiertes Modell erstellt werden. Um möglichst effizient das angestrebte Ziel zu erreichen, werden die einzelnen Aufgaben der Komponenten genauer betrachtet und als Klassen dargestellt. Hierbei kann es durchaus vorkommen, dass sich ganz unterschiedliche Komponenten die gleichen Klassen teilen. Aus diesem Grund ist es nicht sinnvoll, stur die einzelnen Komponenten in Klassen herunterzubrechen, weil es so möglicherweise zu Mehrfachentwicklungen kommt. Zusammenhängende Klassen werden daher in Libraries oder nach der UML Notation in Paketen zusammengefasst, wobei der Unterschied zu einer Komponente in der Benutzung liegt. Ist die Komponente durch ihr Funktionsangebot eher auf der fachlichen Ebene definiert, geht es bei der Bildung von Paketen um die physikalische Zusammenfassung korrelierender Dateien. Dieser modulare Ansatz vereinfacht auch die spätere Implementierung, weil jeweils eine in sich geschlossen Klasse Gegenstand der Programmierung ist. Eine lose Kopplung zwischen Klassen unterschiedlicher Pakete ist daher Voraussetzung für diese Vorgehensweise.

Es sollen aber nicht die vermeintlichen Nachteile verschwiegen werden. Freilich ist es leichter, die Funktionen der Klassen auf eine Komponente zu beschränken. Alle benötigten Funktionen lassen sich bei sorgfältiger Planung sicher benennen und es dürfte selten zu bösen Überraschungen kommen. Dem steht der generischere Ansatz der allgemein gültigen Pakete gegenüber, die sich vielfältiger einsetzen lassen. Hier muss von wesentlich mehr Eventualitäten und Möglichkeiten ausgegangen werden, weil die Funktionalität in mehreren Kontexten benötigt wird. Dies führt nicht nur zu unspezifischeren Operationen, sondern verlangt auch eine stabilere Umsetzung. Innerhalb einer Komponente könne Annahmen getroffen werden, wodurch kritische Situationen nicht entstehen oder besonders behandelt werden können. Als Beispiel sei eine Verbindung zu einer Datenbank angegeben, die innerhalb einer Komponente benötigt wird. Alle Klassen können „wissen“, wann die Verbindung aufgebaut wird und fallen entsprechend schlanker im Überprüfungsteil aus. Bei einem generischen Paket darf von solchen Voraussetzungen nicht ausgegangen werden und jede Operation muss so stabil implementiert sein, dass sie ordentlich abgeschlossen wird. Hierdurch ist die Entwicklung von Paketen aufwändiger. Die Entwicklung einer allgemeinen Schnittstelle verbunden mit den nötigen Überprüfungen kostet Zeit und macht den Code nicht kürzer. Letztendlich führt dieses Vorgehen aber zu stabileren Programmen, die dank der Wiederverwendung kleiner sind. Zudem werden Fehler durch die intensivere Nutzung schneller entdeckt und behoben.

Mit der Komponente *Authoring Foundation* in Abbildung 10.10 ist bereits eine wichtige Vorbereitung für die Aufteilung in Pakete geleistet. Die Teilkomponenten sind bereits für bestimmte Aufgaben ausgelegt und entsprechen schon ungefähr der geplanten Aufteilung in Klassen. Alle Zugriffe auf Dateien, von binären Multimediadateien über Lernobjekte bis hin zu den übersetzten Kursen, erfolgt über die Komponente *File Management*. Durch die Architektur bedingt, beschränkt sich der Zugriff nicht nur auf lokale Dateien, sondern schließt auch verteilte Ressourcen ein, die auf unterschiedlichen Rechnern liegen. Auch die Betrachtung verschachtelter Bausteine und Modelle lässt erahnen, dass eine intelligente Realisierung

wichtig ist. Zusammengefasst ist die Aufgabe dieser Komponente die Abstraktion jeglicher Ressourcenzugriffe über einen einheitlichen Mechanismus bzw. eine Schnittstelle. Alle hierfür nötigen Klassen ergeben ein Paket.

Die Komponente *Metadata* ist wegen der verschiedenen Standards mit ihren vielen Elementen und Attributen in ihrer Struktur sehr umfangreich. Zusätzlich müssen verschiedene Spezifikationen, Kodierungen und Speichertechniken unterstützt werden, die möglichst ineinander überführbar sein sollen. Die Herausforderung bei der Umsetzung dieser Komponente liegt in der Findung eines Kompromisses zwischen generischer und einfacher Nutzung. Weil die Metadaten einem ständigen Wandel unterlegen sind, soll durch geschickte Wahl der Schnittstellen ein Paket geschaffen werden, das auch zukünftigen Entwicklungen gerecht wird.

11.1 Dateizugriff

Die bereitgestellten Funktionen der Komponente *File Management* werden eigentlich in fast allen Programmen benötigt, die auf Dateien zugreifen. In der Regel übernimmt diese Aufgabe das Betriebssystem, das über spezielle Libraries angesprochen wird. Nun stellt sich die Frage, warum für ein E-Learning-System eine spezielle Komponente benötigt wird? Schließlich bieten moderne Betriebssysteme viele Dienste an, die auch die Einbindung verteilter Daten einschließt. In die Dateisysteme lassen sich ohne weiteres WebDAV, SMB und NFS einbinden. Warum soll dieser zusätzliche Aufwand geleistet werden?

Die Antwort liegt in der Heterogenität der Betriebssysteme und der angestrebten Realisierung in der Programmiersprache *Java*. Für lediglich eine bestimmte Zielplattform wäre der Einsatz nativer Mechanismen die erste Wahl. Die Implementierung könnte alle Stärken des Betriebssystems nutzen und die Schwächen umgehen bzw. ausgleichen. Dies beschränkt sich nicht auf den Dateizugriff an sich. Bei manchen Systemen erstrecken sich die Funktionen bis hin zu Verknüpfungen mit Programmen durch *Mime Types* oder sogar in die grafische Oberfläche beim Drag'n'Drop. Muss auf all diese Vorzüge zu Gunsten einer Plattformunabhängigkeit verzichtet werden? Dank des Aufbaus der **Virtuellen Maschine** (VM) von *Java* wird letztendlich doch das Betriebssystem für den Zugriff auf Dateien genutzt, nur über eine einheitliche Schnittstelle. Die Methoden der *Java*-Klassen rufen intern Betriebssystemfunktionen auf, weshalb die mitgelieferten Libraries der Virtuellen Maschinen selbst nicht plattformunabhängig sind. Sie müssen für jedes Betriebssystem neu erstellt werden.

Java bringt also eine Schnittstelle für den Zugriff auf Dateien von sich aus mit, sodass keine neuen Klassen für die Komponente *File Manager* entwickelt werden müssen. Dieser voreilige Schluss wird jedoch durch eine genauere Betrachtung der aktuellen *Java™2 Platform, Standard Edition* (J2SE™) widerlegt. Alle Zugriffe auf das Dateisystem finden über die Klasse `java.io.File` statt. Laut der zugehörigen Dokumentation [Sun01] repräsentiert diese Klasse einen plattformunabhängigen, abstrakten Pfadnamen, der aus zwei Komponenten besteht: einer systemabhängigen Präfixzeichenkette (z.B. „/“ bei UNIX) und einer Sequenz von Null oder mehr Namen. Jeder Name, bis auf den letzten, repräsentiert hierbei ein Verzeichnis.

Die Umwandlung des abstrakten Pfadnamens in einen konkreten Systempfad des Dateisystems ist abhängig vom Betriebssystem, oder genauer betrachtet, vom Separationszeichen innerhalb der Pfade. Diese Umwandlung und das systemabhängige Präfix erschweren die Entwicklung portabler Anwendungen, die z.B. auf *Microsoft Windows* und parallel auf UNIX-Derivaten ohne Neuübersetzung laufen.

Neben dem Problem der Portierbarkeit, ist der Funktionsumfang der Klasse `java.io.File` sehr spartanisch. Es können gewisse Attribute von Dateien und Verzeichnissen abgefragt werden, wie z.B. Datum der letzten Bearbeitung, Länge der Datei in Bytes und die Rechtevergabe zum Schreiben oder Lesen. Die Manipulationsmöglichkeiten beschränken sich lediglich auf das Erstellen, Löschen und Umbenennen. Eine Operation wie Kopieren oder die Unterstützung Regulärer Ausdrücke, wie sie in diversen Kommandozeilen Verwendung findet, muss bereits selbst implementiert werden.

Die Definition der Komponente *File Management* sieht solche komfortablen Operationen und noch speziellere vor. Besonders die gepackten sowie verschachtelten Dateien und der nahtlose Netzwerkzugriff müssen unterstützt werden. Bei der Realisierung der Komponente müssen somit zumindest ergänzende Klassen, wenn nicht sogar ein komplett neuer Zugriff auf Dateien eingeplant werden. Auch Sun als Urheber von *Java* hat bemerkt, dass die Funktionalität der Klasse `java.io.File` nicht ausreicht und hat daher das *WebNFS™ Client SDK* entwickelt. Mit diesem SDK wird der Zugriff auf Dateien und Verzeichnisse verschiedener Dateisysteme über eine Schnittstelle realisiert. Die Referenzierung erfolgt über URLs [Berners-Lee94] und ermöglicht so die Portierbarkeit von Programmen ohne Neuübersetzung. Kernstück dieses Ansatzes ist die *XFile-Schicht*, die durch beliebige Dateisysteme erweitert werden kann. Abbildung 11.1 zeigt das Schichtenmodell von *WebNFS*.

Abbildung 11.1: *Extended Filesystem Architecture* [Sun99]

Der Dateizugriff über das Netzwerk (TCP/IP) ist das Hauptmerkmal von *WebNFS*. Es wundert daher nicht, dass die Schicht `com.sun.xfile.*` bezüglich der Attribute und Dateioperationen nicht mehr Funktionalität als die Klasse `java.io.File` anbietet und damit den Ansprüchen nicht vollends genügt.

Von anderen Herstellern bzw. Entwicklergruppen werden eine Reihe an weiteren kleinen APIs und Erweiterungen der Standardklassen angeboten, die meist einen erweiterten Zugriff auf betriebssystemspezifische Daten geben. Jedoch gibt es zu diesem Zeitpunkt kein Produkt, das einen sauberen Entwurf, bei dem allzu technische Details verborgen sind, in der geforderten Form ermöglicht. Um diesem Wunsch dennoch Rechnung zu tragen, wird daher eine eigene Dateisystem-API für *Java* entwickelt.

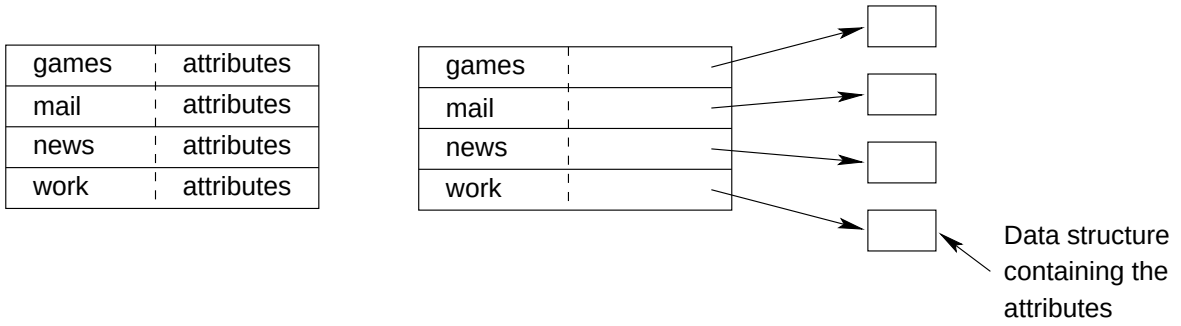
11.1.1 Dateisystem Grundlagen

Zuerst muss eine Idee formuliert werden, wie das zu entwickelnde Dateisystem aufgebaut sein soll. Hieraus ergibt sich automatisch die Schnittstelle nach außen, die zur Ansteuerung von anderen Komponenten genutzt wird. Nun soll das Rad nicht neu erfunden werden, weshalb kurz festgehalten wird, was an Libraries und Funktionen bereits zur Verfügung steht. Als erstes ist die mitgelieferte Klasse `java.io.File` zu nennen, die den Zugriff auf das lokale Dateisystem ermöglicht. Freilich soll ihre Funktion nicht nachimplementiert werden, denn die Umsetzung dieser Klasse ist proprietär und eine Eigenentwicklung wäre nicht mehr plattformunabhängig. Die von *Java* angebotene Klasse soll das Fundament für den Dateizugriff auf das lokale Dateisystem sein. Im Falle der verteilten Daten müssen Protokolle wie z.B. WebDAV, NFS oder SMB implementiert werden. Dies kann und soll auch nicht geleistet werden, weil Aufwand und Nutzen in keiner Relation stehen. Besser ist der Einsatz existierender Libraries, die wesentlich zuverlässiger sind. Damit wird zwar die Einbindung vieler verschiedener Schnittstellen als Nachteil in Kauf genommen, aber dieses Vorgehen sichert die höchstmögliche Flexibilität. Um dieser Vielfältigkeit zu begegnen, muss das Dateisystem eine abstrahierende Schnittstelle anbieten, die mit Hilfe von Adaptern auf die eigentliche Funktionalität der Libraries zugreift. So bleibt das Dateisystem auch für zukünftige Techniken erweiterbar.

Der Begriff Dateisystem an sich ist geprägt vom Einsatz in Betriebssystemen. Insofern sollen die Erfahrungen und Theorien aus diesem Bereich bei der Entwicklung des eigenen Dateisystems berücksichtigt werden. Im Grunde genommen handelt es sich bei dem Dateisystem um eine Verwaltung anderer Dateisysteme, sodass es beispielsweise dem *Virtual File System* (VFS) von Linux ähnlich ist. Beim VFS geht es zwar in erster Linie um die Abstraktion von physikalischen Medien und Gerätetreibern, aber das Prinzip ist sehr ähnlich.

Wegen der existierenden Libraries kann das Abstraktionsniveau sehr hoch gewählt werden, schließlich muss sich das *Java*-Dateisystem nicht um die physikalische Repräsentation kümmern. Die Aufteilung der Festplatten und die Adressierung übernehmen die jeweiligen Betriebssysteme. Aufgabe des Dateisystems ist daher die einheitliche Adressierung von Ressourcen und die Delegation von Operationen über Library-Grenzen hinweg.

Moderne Dateisystem ordnen ihre Dateien in Verzeichnissen an, die technisch gesehen meist selbst Dateien sind. Sie enthalten eine bestimmte Anzahl an Einträgen, wobei jeder für eine Datei steht. Abbildung 11.2(a) zeigt eine Form der Speicherung, bei der Dateiname, Attribute und die Adresse in einem Eintrag gespeichert sind.



(a) Attribute im Verzeichniseintrag

(b) Externe Attribute

Abbildung 11.2: Aufbau von Verzeichniseinträgen aus [Tanenbaum97, S. 411]

Zu den Attributen zählen diverse Daten (Erstellungsdatum, Bearbeitungsdatum, etc.), Zugriffsrechte, Eigentümer, Dateilänge und viele weitere systemspezifische Felder. Die Adresse hängt im Falle des *Java*-Dateisystems vom eingebundenen Dateisystem ab, also wie das Quellsystem das Zielsystem ansteuern muss. Handelt es sich beispielsweise um ein WebDAV-Dateisystem, ist die Adresse, die das *Java-Dateisystem* speichert, eine URL.

Eine andere Form des Verzeichniseintrags ist die Referenzierung einer externen Struktur, wie in Abbildung 11.2(b) zu sehen ist. Sie ist nur der Vollständigkeit halber aufgeführt und hat keinen weiteren Einfluss auf den Entwurf. Historisch gesehen lassen sich „echte“ Dateisysteme auf diese Weise besser erweitern. Aufgrund des objektorientierten Ansatzes des *Java*-Dateisystems ist dieses Argument aber hinfällig.

Weil auch Verzeichnisse normale Dateien sind, können sie problemlos verschachtelt werden. Zyklen sind aber nicht erlaubt, weshalb die resultierende Struktur ein Baum ist. Aus heutiger Sicht mag dieser Aufbau einer Dateistruktur selbstverständlich erscheinen, aber in den Anfängen der Betriebssysteme gab es entweder ein Hauptverzeichnis oder Unterverzeichnisse in sehr geringer Zahl.

Für den Zugriff auf die Dateien wird eine Adressierung benötigt, die jeden Knoten im Baum eindeutig identifiziert. Jeder Baum hat genau ein Wurzelverzeichnis das alle Unterverzeichnisse direkt oder indirekt beinhaltet. Wenn jeder Dateiname innerhalb eines Verzeichnisses nur ein Mal verwendet werden kann, dann lässt sich diese Datei über die Namen der höher liegenden Verzeichnisse eindeutig ansprechen. Diese Adressierung wird **Pfad** genannt und kann in zwei Varianten notiert werden. Bei einem **absoluten Pfad** werden alle Namen vom Wurzelverzeichnis bis zum Dateinamen angegeben, getrennt durch einen Separator. Bei *Windows* übernimmt das Zeichen „\“ diese Aufgabe und bei Unix „/“. Ein Beispiel für einen absoluten Pfad ist `/usr/jim/mails`. Das Wurzelverzeichnis „/“ enthält das Unterverzeichnis `usr`, das wiederum das Unterverzeichnis `jim` und schließlich folgt die Datei `mails`. Abbildung 11.3 zeigt eine passende Dateistruktur.

Die andere Schreibweise für einen Pfad nennt sich **relativer Pfad** und steht in Verbindung mit dem Konzept des **Arbeitsverzeichnisses**. Dahinter verbirgt sich die Möglichkeit, anstatt des Wurzelverzeichnisses ein beliebiges Verzeichnis anzugeben, von dem aus alle Pfade

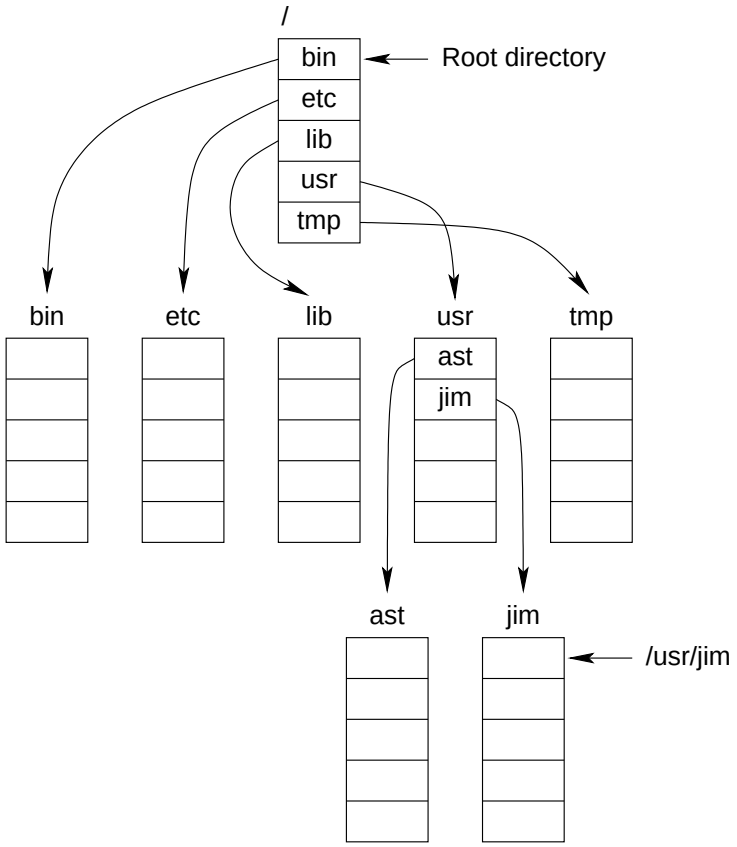


Abbildung 11.3: Ein UNIX Verzeichnisbaum aus [Tanenbaum97, S. 414]

beginnen. Wenn z.B. das Verzeichnis `/usr/jim` als aktuelles Arbeitsverzeichnis ausgewählt ist, kann die Datei `/usr/jim/mails` einfach über den Pfad `mails` aufgerufen werden. Relative Pfade vereinfachen somit den Zugriff auf Dateien, besonders wenn auf viele Dateien eines Unterverzeichnisses zugegriffen wird. Es wird noch angemerkt, dass absolute Pfade in keiner Weise vom Arbeitsverzeichnis beeinflusst werden. Bei Betriebssystemen hat jeder Prozess ein eigenes Arbeitsverzeichnis, was für das *Java*-Dateisystem nur bedingt gelten soll. Um flexibler zu sein, soll jedes Objekt der Dateisystemklasse ein eigenes erhalten können. Wie viele Objekte es geben kann, hängt von der jeweiligen Implementation ab.

Es gibt noch zwei spezielle Verzeichniseinträge, die eigentlich von allen hierarchischen Dateisystemen unterstützt werden. Mit „.“ wird auf das Verzeichnis selbst zugegriffen und mit „..“ auf das höher liegende, dem Elternverzeichnis. Der absolute Pfad `/usr/jim/.` adressiert also `/usr/jim` und `/usr/jim/..` das höher liegende Verzeichnis `/usr`.

Nachdem sich die Dateien adressieren lassen, sollen nun einige wichtige Operationen beschrieben werden. Obwohl Verzeichnisse reguläre Dateien sind, gibt es bei den Operationen dennoch Unterschiede. Hierzu gehört der Zugriff auf den Inhalt einer Datei, der bei einem Verzeichnis nicht möglich ist. Von den gängigen Betriebssystemen werden zwei verschiedene Methoden für normale Dateien unterstützt, die sich in den unterschiedlichen Medien, wie z.B. Bändern und Festplatten, begründen. Beim **sequentiellen Zugriff** können die Daten der Reihe nach vom Anfang bis zum Ende ausgelesen oder geschrieben werden, wobei ein Zugriff in beliebiger Reihenfolge nicht möglich ist. Die Klasse `java.io.File` bietet wie die erhältlichen Libraries den sequentiellen Zugriff über Ein- und Ausgabeströme zu einer Datei an.

Lässt sich auf Bereiche der Datei direkt zugreifen, heißt diese Form **beliebiger Zugriff**. Für viele Anwendungen, wie z.B. Datenbanken, ist dieser Zugriff sehr wichtig, denn besonders bei vielen Daten kann nicht jedes Mal die Datei von Anfang an durchgegangen werden, wenn am Ende ein kleines Datum benötigt wird. Leider sieht es bei der Unterstützung in *Java* eher schlecht aus. Zwar gibt es die Klasse `java.io.RandomAccessFile`, aber besonders die

zusätzlichen Libraries für den Netzwerkzugriff haben ihre Probleme. Manche Protokolle, wie z.B. WebDAV, sehen den beliebigen Zugriff nicht einmal vor. Um unnötige Komplikationen im Vorwege zu vermeiden, soll das *Java*-Filesystem diese Form des Zugriffs nicht unterstützen. Dies ist nicht weiter schlimm, da E-Learning-Inhalte in der Regel recht klein sind und sich komplett im Speicher halten lassen.

Zu den Operationen, die für Dateien wie Verzeichnisse gültig sind, gehören unter anderem Erstellen, Löschen und Umbenennen. Es gibt noch eine Reihe weiterer Funktionen, wie z.B. das Auflisten aller Verzeichniseinträge oder das Anhängen von Daten, die entweder für Verzeichnisse oder Dateien gelten, aber auf sie soll nicht weiter eingegangen werden. Wenn solche Funktionen von Bedeutung sind, werden sie als Methoden in die Klassen aufgenommen.

11.1.2 Virtuelles Dateisystem

In Anlehnung an das abstrahierende Dateisystem von Linux sollen die zentralen Klassen als **virtuelles Dateisystem** betrachtet werden. Hierunter wird im Kontext dieser Arbeit ein Mechanismus verstanden, der verschiedene existierende Dateisysteme in einer neuen Dateistruktur zusammenfasst. Das virtuelle Dateisystem bietet somit eine einheitliche Adressierung für Dateien an und die Delegation von Operationen. Aus diesem Grund werden die angesteuerten Dateisysteme auch **Zieldateisysteme** genannt.

Im Gegensatz zu den Dateisystemen der Betriebssysteme, wird die Dateistruktur beim virtuellen Dateisystem komplett im Speicher gehalten und nicht direkt auf dem Medium gespeichert. Jeder Zugriff auf Verzeichnisse und Dateien hat somit zur Folge, dass eine entsprechende Struktur im Speicher erzeugt wird, die von der physikalischen Repräsentation abstrahiert. Um die Struktur eines virtuellen Dateisystems persistent zu machen, müssen sie an anderer, frei wählbarer Stelle gespeichert werden. Die Dateien selbst werden freilich in ihrem jeweiligen Zieldateisystem gehalten.

Im *Java*-Dateisystem wird es daher eine Klasse geben, mit der entsprechende Strukturen im Speicher aufgebaut werden können. Anhand des Beispiels in Abbildung 11.4, sollen die Zusammenhänge für ein stellvertretendes WebDAV-Dateisystem erläutert werden.

Abbildung 11.4: Interne Abbildungen im VFS

Ein Server mit der Internetadresse `get1.upb.de` stellt das auf der rechten Seite liegende Zieldateisystem zur Verfügung. Wie und wo die Dateistruktur gespeichert ist, soll an dieser Stelle nicht interessieren. Wichtiger ist die interne Struktur des virtuellen Dateisystems auf der linken Seite, die sich auf einem anderen Client-Rechner befindet. Verzeichnisknoten sind als Quadrate dargestellt und Dateien als Kreise. Ein Pfeil in den Knoten weist auf die Adressierung einer anderen Ressource hin. In dem Beispiel werden URLs genutzt, die als zusätzliche Kantenbeschriftung angegeben sind. Das oberste Verzeichnis hat keinen Namen und deutet die Beliebigkeit der Position im Dateisystem an. Wichtiger ist das Unterverzeichnis `repository`, das eine Referenz auf das WebDAV-Verzeichnis „/“ besitzt. Die unterschiedlichen Namen sind kein Problem, da die absoluten Pfade sowieso nicht identisch sein können.

Weil der Knoten `repository` auf ein Wurzelverzeichnis eines anderen Dateisystems verweist, wird er als **Mount-Punkt** bezeichnet. Es ist quasi der Einstiegspunkt, der „manuell“ gesetzt werden muss. Alle anderen Knoten, im Beispiel sind es `start.xml` und `title.xml`, werden automatisch abgefragt und im virtuellen Dateisystem neu erzeugt. Nun können solche eingehängten Dateisysteme mit tausenden von Unterverzeichnissen recht umfangreich sein und es wäre ein erheblicher Aufwand, die gesamte Struktur auf ein Mal zu übertragen. Viele Verzeichnisse werden möglicherweise nie aufgerufen, sodass sich diese Herangehensweise nicht lohnt. Daher soll die interne Struktur nur für Dateien aufgebaut werden, die tatsächlich benötigt werden. Der Struktur in Abbildung 11.4 sind ein paar Operationen voraus gegangen, wie sie in Abbildung 11.5 dargestellt sind.

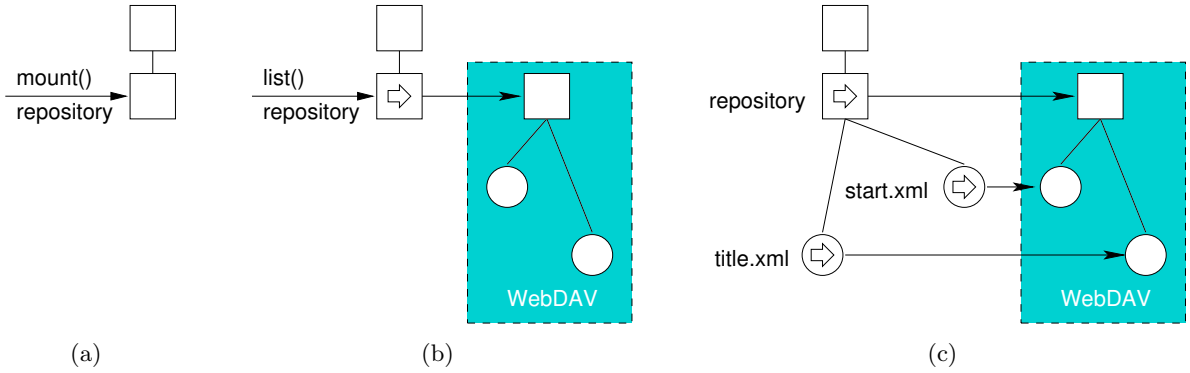


Abbildung 11.5: Aufbau der Dateistruktur in zwei Schritten

Der Ausgangszustand des virtuellen Dateisystems ist in Abbildung 11.5(a) dargestellt, der aus zwei Knoten besteht, die keine externen Ressourcen referenzieren. Mit dem Befehl `mount()` an den Knoten `repository` wird ihm die Adresse einer WebDAV-Datei zugewiesen. Das Resultat ist in Abbildung 11.5(b) zu sehen. Zu diesem Zeitpunkt hat sich an der Struktur des virtuellen Dateisystems nichts geändert. Dies geschieht erst durch den Aufruf des Befehls `list()`, der die Namen aller Dateien des Verzeichnisses ausliest. Hierfür delegiert der Knoten `repository` den Befehl an die Zieldatei und generiert aus dem Resultat alle Knoten mit ihren Attributen, einschließlich der Adressen. Erst jetzt entspricht der Zustand in Abbildung 11.5(c) dem vorherigen Beispiel.

Selbstverständlich lassen sich beliebig viele Zieldateisysteme verschiedenen Typs in das virtuelle Dateisystem einhängen, wie in Abbildung 11.6 dargestellt. Der Übersicht halber sind nicht alle Verbindungen zwischen den Knoten und Dateien eingezeichnet.

Auch verschiedene Laufwerke eines *Windows* Systems lassen sich problemlos hinzufügen und durch die direkte Einbindung in die Dateistruktur werden keine Laufwerksbuchstaben mehr benötigt. Beim Zieldateisystem WebDAV wird zudem gezeigt, dass sich jedes beliebige Verzeichnis mit einem Mount-Punkt verbinden lässt. Lediglich die darüber liegenden Verzeichnisse bzw. die Geschwister auf gleicher Höhe lassen sich dann nicht über einen Pfad im virtuellen Dateisystem ansprechen.

Aus den geschilderten Funktionen des virtuellen Dateisystems lässt sich nun die gewünschte Klasse zur Strukturierung ableiten. Da es sich bei der Dateistruktur um einen Baum handelt, der aus vielen Knoten besteht, ergibt sich die Klasse `VFSNode`, in der alle Attribute gespeichert sind. Abbildung 11.7 zeigt sie mit ihren wichtigen Methoden, die aufgrund der vorherigen Beschreibungen in Abschnitt 11.1.1 keiner weiteren Erklärungen bedürfen.

Offensichtlich bietet diese Klasse aber nur rudimentäre Operationen an, obwohl in den vorherigen Ausführungen Funktionen wie das Kopieren oder Entpacken von Dateien eingefordert wurden. Auch das beschriebene Einhängen anderer Dateisystem ist anscheinend nicht auf dieser Ebene realisiert.

Der Grund hierfür ist einfach. Solche Operationen schließen immer mehrere Knoten ein und benötigen eine zusätzliche Verwaltung. Als Beispiel sei nochmals das Einhängen eines

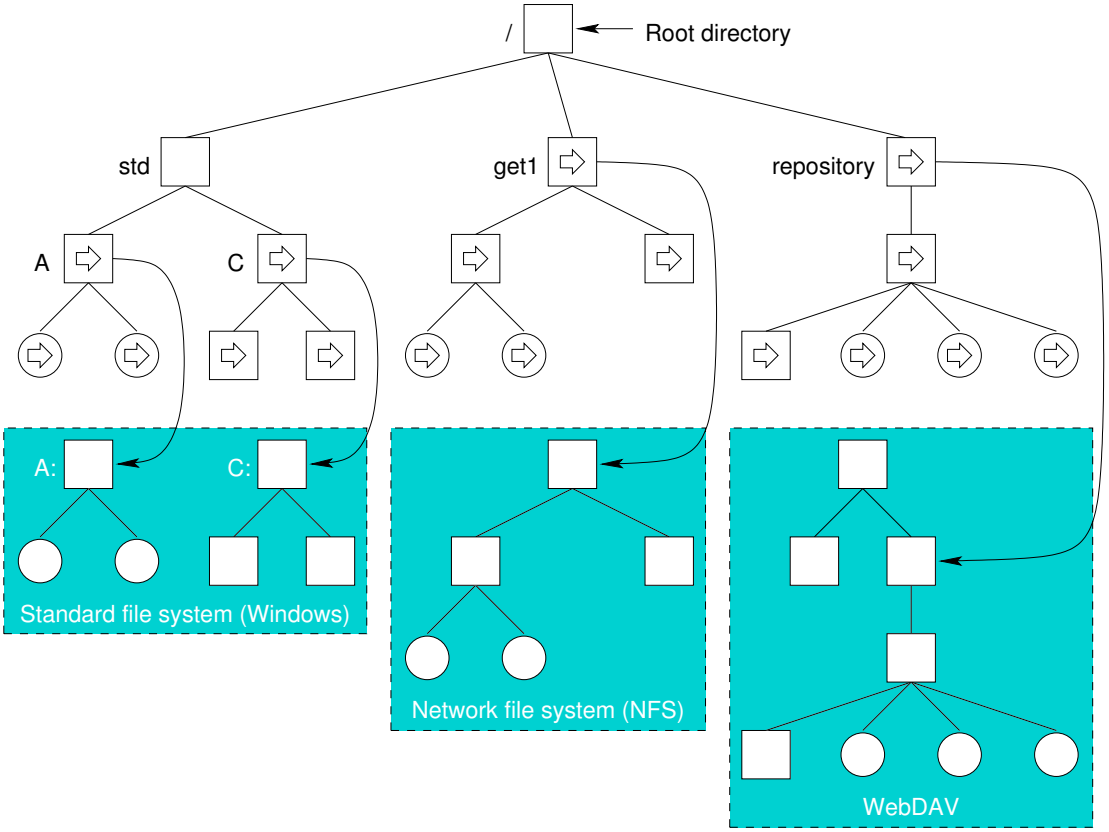


Abbildung 11.6: Beispiel für den Aufbau des VFS

Zieldateisystems genannt. Die Operation `mount()` bedeutete für einen Knoten, alle Attribute und Eigenschaften des neuen Zieldateisystems zu kennen. Hierdurch würden aber die Vorteile des objektorientierten Entwurfs leichtfertig aufgegeben, da die Klasse `VFSNode` als universeller Knoten auftritt. Um möglichst flexibel zu bleiben, sollte sie lieber als Schnittstelle für spezielle Implementationen dienen, was in Abbildung 11.7 durch den Stereotyp `abstract` angezeigt ist. Ein paar erbende Klassen für jeweils ein Zieldateisystem sind in Abbildung 11.8 zu sehen.

Die Verwaltung des Einhängens und anderer komplexer Operationen wird in einer eigenen Klasse Namens `VFS` implementiert. Neben einigen Vereinfachungen bietet diese Klasse auch Dienstleistungen an, die alle Knoten eines Dateisystems betreffen. Es ist beispielsweise für eine grafische Darstellung eines Dateisystems notwendig, nach Änderungen den aktuellen Zustand neu zu zeichnen. Um nicht ständig alle Knoten überprüfen zu müssen, bietet die Klasse `VFS` einen Benachrichtigungsmechanismus an, der angemeldete Zuhörer über alle Änderungen informiert. Jede Klasse kann zum Zuhörer werden, indem sie eine einfache Schnittstelle implementiert, über die sie die Benachrichtigungen empfangen kann. Bei manchen Benachrichtigungen können die Zuhörer sogar Einfluss auf das Geschehen nehmen. Wenn z.B. eine Datei gelöscht werden soll, werden die Zuhörer über dieses Vorhaben benachrichtigt. Nach dem Empfang können sie den/die Anwender/-in mit Hilfe eines Dialogs hierüber informieren und das Einverständnis abfragen. Sollte das Löschen doch nicht gewünscht sein, wird der Abbruch dieser Operation an das Dateisystem zurückgeschickt. Die wichtigen Methoden der Klasse `VFS` sind in Abbildung 11.9 dargestellt.

Auch bei dieser Klasse sind die Methoden weitestgehend selbsterklärend. Mit der Methode `zip` lassen sich mehrere Dateien zu einem Archiv zusammenfassen und komprimieren. Um wieder an die einzelnen Dateien zu kommen, lassen sich die Archive mit `unzip` entpacken. Interessant ist die Klasse des Parameters, den beide als Argument akzeptieren. Bei `VirtualFile` handelt es sich um eine Pfadbeschreibung im virtuellen Dateisystem. Da sie intern den Zugriff auf das Dateisystem benötigt, wird sie erst anschließend näher erläutert. Neben

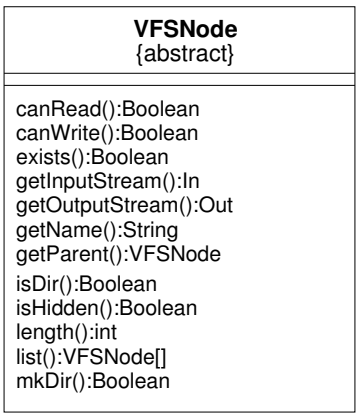


Abbildung 11.7: Klasse `VFSNode`

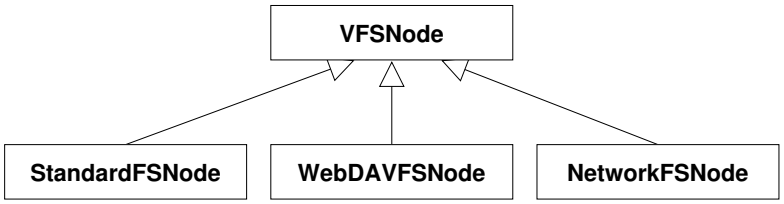


Abbildung 11.8: Verschiedene Unterklassen der Klasse `VFSNode`

einzelnen Knoten lassen sich auch Zieldateisysteme direkt in das Dateisystem einhängen. Dies ist besonders praktisch bei Dateistrukturen, wie sie von *Windows* angeboten werden. Auf diese Weise müssen nicht alle Laufwerke einzeln eingehängt werden.

Wie bei den Knoten die Klasse `VFSNode`, ist auch die Klasse `VFS` abstrakt definiert und eine Schnittstelle für die unterschiedlichen Implementationen. Abbildung 11.10 zeigt eine mögliche Vererbungshierarchie.

Eine Konsequenz aus diesem Ansatz ist, dass jedes Zieldateisystem gleichzeitig als virtuelles Dateisystem auftritt. Dies führt zwangsläufig zu einem Problem mit *Windows*, da im vorherigen Abschnitt11.1.1 immer von einem Wurzelverzeichnis ausgegangen worden ist, über das alle absoluten Pfade definiert sind. Unter *Windows* gibt es nun historisch bedingt bis zu 26 Wurzelverzeichnisse (Laufwerk `A:`–`Z:`), sodass die Pfadbeschreibung ein wenig anders ist. Das virtuelle Dateisystem soll aber mit nur einem Wurzelverzeichnis auskommen, weshalb das bestehende Konzept um eine Ebene erweitert werden muss. Neben dem Wurzelverzeichnis, das über die Methode `getRootNode` erreichbar ist, werden deshalb die **Volumes** eingeführt. Dieser Name ist absichtlich aus der *Windows* Welt entnommen, weil es das einzige verbreitete Betriebssystem mit dieser Aufteilung ist.

Volumes liegen direkt unter dem Wurzelverzeichnis und sind eine Art privilegiertes Verzeichnis. Bei der Pfadangabe muss nicht der echte absolute Pfad angegeben werden, sondern kann mit dem Namen eines Volumes beginnen. Mit diesem Verfahren lassen sich die Laufwerke simulieren. Über die Methode `getVolumes` gibt ein Dateisystem eine Liste aller Volumes zurück. Bei einem UNIX-Dateisystem wird eine leere Zeichenkette zurückgegeben, sodass der absolute Pfad immer mit „/“ anfängt.

Die Methode `getRootNode` deutet bereits an, dass der Zugriff auf die Knoten über das Dateisystem läuft. Nun hat der interne Baum, wie er bis jetzt dargestellt wurde, einen gravierenden Nachteil. Änderungen auf Seiten des Zieldateisystems werden nicht weitergereicht, was zu Inkonsistenzen führen kann. Wird z.B. der Inhalt eines Verzeichnisses mit `list` ausgelesen und nachfolgend direkt im Zieldateisystem eine der aufgelisteten Dateien gelöscht, so bleibt der Knoten im virtuellen Dateisystem bestehen. Erst beim Zugriff auf die Datei offenbart sich dieser Zustandsunterschied. Abhängig von der Implementation der Knoten kann dieses Problem früher oder später auftreten. Bei einer einfachen Umsetzung mit einer direk-

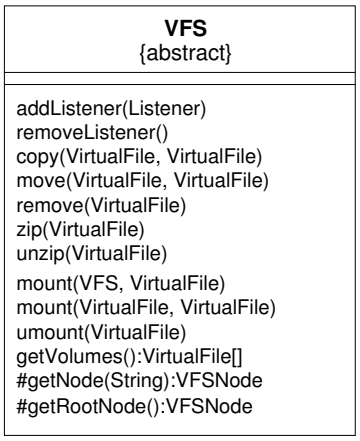


Abbildung 11.9: Klasse `VFS`

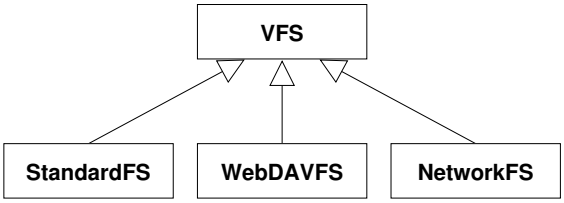


Abbildung 11.10: Verschiedene Unterklassen der Klasse `VFS`

ten Referenzierung der Knoten kann es sogar vorkommen, dass auch wiederholte Aufrufe von `list` den alten Zustand anzeigen. Daher sollte bei jeder Operation überprüft werden, ob die Dateistruktur des virtuellen Dateisystems mit dem des Zielsystems übereinstimmt. Für eine grafische Darstellung wäre es sogar sinnvoll, in regelmäßigen Zeitabständen diese Überprüfung durchzuführen. Leider kann diese Abfrage bei komplexen Dateisystemen sehr teuer sein, zumal ein Zugriff über das Netzwerk zusätzlich Zeit benötigt. Auf der anderen Seite ist die Pflege eines Baums mit den verschiedenen Dateisystemoperationen sehr umständlich und kann bei sehr großen Dateisystemen immer noch viel Platz in Anspruch nehmen.

Die Lösung liegt in einer „flachen“ Organisation der Knoten. Anstatt die Dateistruktur als Ganzes zu betrachten, kann jeder Knoten eindeutig über einen Pfad identifiziert werden. Um einen angemessenen Kompromiss zwischen Speicherbedarf und Laufzeit zu finden, werden lediglich n Knoten gleichzeitig im Speicher gehalten. Abbildung 11.11 zeigt die angestrebte Variante.

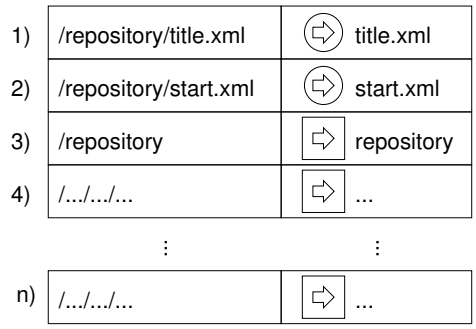


Abbildung 11.11: Dateistruktur im Arbeitsspeicher

Es handelt sich quasi um eine Hash-Map mit einer internen Ordnung. Jeder der n Einträge repräsentiert einen Knoten mit zugehörigem Pfad. Der zuletzt genutzte Knoten ist auf Position 1, der vorherige auf 2 und der letzte auf n. Wird nun auf einen Knoten zugegriffen, der zuvor nicht in der Liste aufgeführt war, wird Knoten n herausgenommen und alle $n - 1$ Knoten

um eine Position nach hinten verschoben. An Position 1 wird abschließend der neue Knoten eingetragen. Die ganze Anordnung kann als **Cache** betrachtet werden, auch wenn er nicht so leistungsfähig ist, wie ihn manche Dateisysteme anbieten. So könnte beispielsweise eine zweite Liste angeschlossen werden, in der die Häufigkeit der Zugriffe berücksichtigt werden. Sollte die Leistungsfähigkeit des gewählten Ansatzes wider Erwarten nicht ausreichen, könnten solche Funktionen immer noch nachträglich für eine Optimierung sorgen.

Wie angekündigt, soll nun die letzte wichtige Klasse des virtuellen Dateisystems näher vorgestellt werden. Bei dem **VirtualFile** handelt es sich sich um eine Aggregation von Pfad und Referenz auf ein Dateisystem, um den Zugriff zu vereinfachen. Abbildung 11.12 zeigt das zugehörige Klassendiagramm.

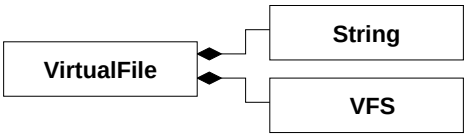


Abbildung 11.12: Aggregation von `String` und `VFS`

Diese Klasse bietet alle Methoden der Klasse `VFSNode` an, weshalb kein detaillierteres Diagramm angegeben werden muss. Intern werden die Methodenaufrufe an die Knoten mit dem passenden Pfad weitergegeben. Die Programmierer/-innen nutzen folglich die Knoten-Klassen nie direkt, sondern bedienen sich des vereinfachten Zugriffs. Bei einfachen Operationen ist kein Unterschied in der Bedienung zu der Klasse `java.io.File` auszumachen, sodass einer gewohnten Verwendung bzw. der Umstellung existierenden Programm-Codes nichts im Wege steht.

Wie es nach der UML vorgesehen ist, kann nun, da die Klassen vollständig für den Umgang mit Dateien erschlossen sind, die erste Komponente `File Management` gebildet werden, wie sie in Abbildung 10.10 zu sehen ist. Hierfür werden einfach alle vorgestellten Klassen zusammengefasst und als eine Einheit ausgegeben. Der Prozess dieser Kombination ist in Abbildung 11.13 schematisch dargestellt und enthält freilich nur ein paar ausgewählte Klassen.

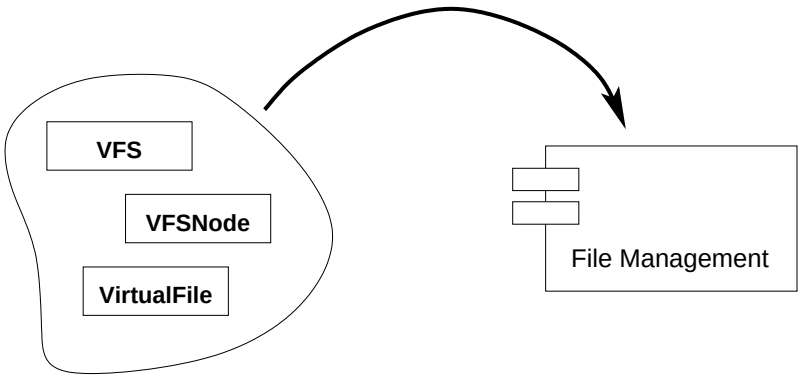


Abbildung 11.13: Bildung der Komponente `File Management`

Im nächsten Abschnitt wird die nächste Komponente für die Rolle *Author* hergeleitet, um Abbildung 10.10 zu vervollständigen.

11.2 Metadaten

Das Kapitel 4 mit seinen theoretischen Ausführungen und detaillierten Vorstellungen der gängigen Standards bildet die Basis für die folgenden Überlegungen. Die zugehörige Komponente für die Metadaten lässt sich, ähnlich der Komponente *File Management*, vollständig in Form einer allgemeinen Library aufziehen. Hierdurch wird die benötigte Flexibilität erreicht, die für eine solch zentrale Funktion unabdinglich ist.

Grundlegend wird eine allgemeine Architektur für die Erstellung, Bearbeitung und den Austausch von Metadaten entwickelt. Eine besondere Schwierigkeit dieses Unternehmens ist die Vielfältigkeit der unterschiedlichen Standards und deren Kodierungen. Das angestrebte Ziel kann folglich nur eine interne Repräsentation sein, die allen Belangen gerecht wird. Hierfür müssen die zu unterstützenden Standards festgelegt und ihre Gemeinsamkeiten erkannt werden. Auf jeden Fall sollen die im Teil „Stand der Wissenschaft“ vorgestellten Spezifikationen von *Dublin Core* und *IEEE LOM* in verschieden Kodierungen unterstützt werden. Neben dem definierten Format in XML soll auch die direkte Unterstützung von Datenbanken angeboten werden. Hieraus ergibt sich eine Architektur, die schematisch in Abbildung 11.14 dargestellt ist.

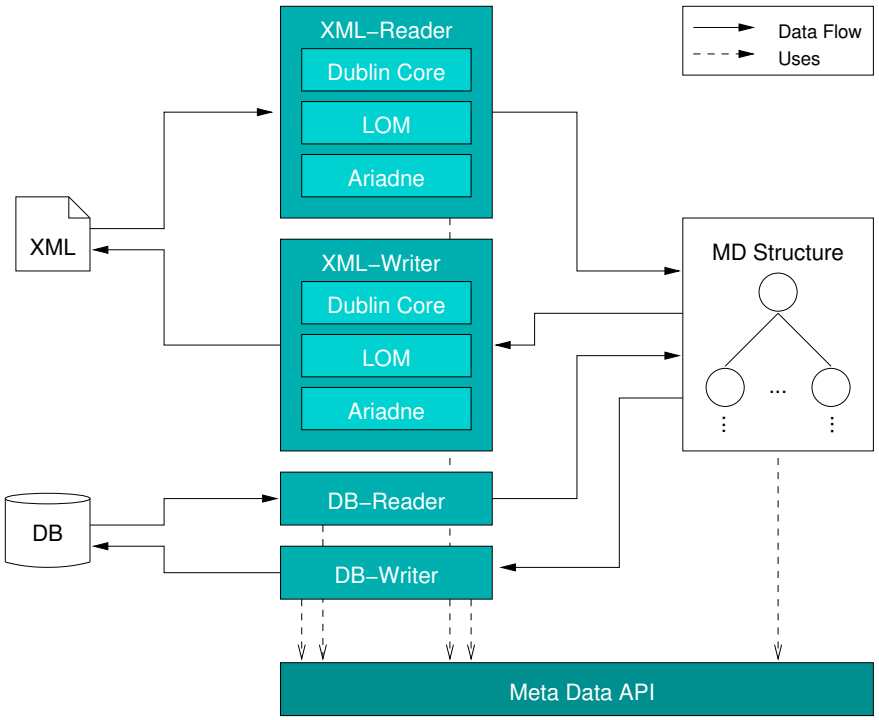


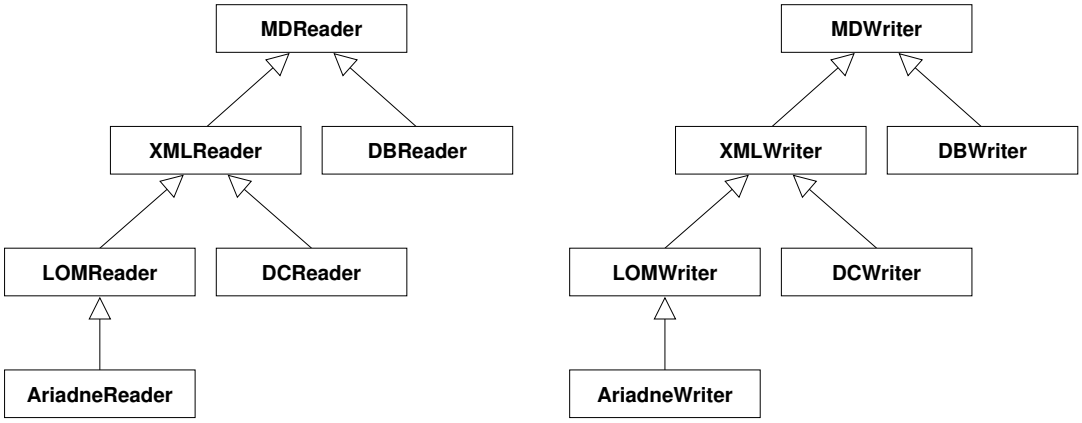
Abbildung 11.14: Architektur für heterogene Metadatenformate

Aus den verschiedenen Datenquellen, XML-Datei und Datenbank, werden die Metadaten über die **Reader** eingelesen und unter Zuhilfenahme der unten stehenden API zu einer hierarchischen Struktur zusammengesetzt. Eine der Gemeinsamkeiten ist nämlich der Aufbau als Baum, auf den sich alle wichtigen Metadatenspezifikationen abbilden lassen. Über die **Writer** kann die Metadatenstruktur wieder in die Formate der Datenquellen umgewandelt werden. Neben dem Aufbau der Metadatenrepräsentation im Speicher, ermöglicht die Metadaten-API auch die Manipulation der Struktur.

Die gesamte Flexibilität dieses Entwurfs begründet sich in den unterschiedlichen *Reader*- und *Writer*-Modulen, deren Aufbau näher betrachtet werden soll. Da es für Datenbanken keine verbindlichen Anordnungen bzw. Datenbankschemata gibt, kann der Zugriff sehr individuell gestaltet werden. Eine denkbare Lösung ist eine Klasse, die aus einer festgelegten Quelle die nötigen SQL-Skripte auslesen kann. Hierdurch wird zwar Programmlogik außerhalb der Applikation angesiedelt, was im Normalfall zu vermeiden ist, aber die gewonnene Flexibilität rechtfertigt möglicherweise diese Vorgehensweise. Alternativ kann auch eine Klasse mit der vollständigen Datenbankansteuerung erstellt werden und in frei definierbaren Unterklassen erfolgt die Zuordnung zu den Tabellen. Welcher Ansatz letztendlich gewählt wird, soll erst in der Implementierungsphase entschieden werden.

Anders sieht es bei der Kodierung in XML aus. Weil alle relevanten Spezifikationen ein *XML-Binding* beinhalten, muss dieser Umstand bereits beim Entwurf berücksichtigt werden. In Abbildung 11.14 sind die Standards eingetragen, die von der angestrebten Implementierung

unterstützt werden sollen. Zu den gestellten Anforderungen lässt sich elegant eine passende Klassenstruktur bilden, deren Diagramme für *Reader* und *Writer* in Abbildung 11.15 dargestellt sind.



(a) Metadaten-*Reader* (b) Metadaten-*Writer*

Abbildung 11.15: Klassenhierarchien der *Reader* und *Writer*

Der Zugriff auf alle Metadaten-*Reader* sowie -*Writer* erfolgt über die Klassen `MDReader` und `MDWriter`, die eine einheitliche Schnittstelle nach außen darstellen. Implementierungen spezieller Methoden sind auf dieser Ebene nicht zu erwarten, sodass es sich im Sinne von *Java* um echte *Interfaces* handelt und nicht um abstrakte Klassen. Auf der nächsten Ebene, den Klassen `XMLReader` und `DBReader` bzw. `XMLWriter` und `DBWriter` verhält es sich anders. Hier werden grundlegende Funktionen angeboten — meist technische Details verdeckend —, die in den Unterklassen genutzt werden. In den jeweiligen Klassen für *LOM* und *Dublin Core* auf der darunter liegenden Vererbungsebene steckt die Logik, die für den Zusammenbau der allgemein gültigen Metadatenstruktur bzw. die Abbildung in das entsprechende Zielformat benötigt wird. Am Beispiel von *Ariadne* wird gezeigt, wie der Entwurf durch *Application Profiles* erweiterbar ist. Durch Vererbung können existierende *Reader* und *Writer* um neue Metadateneinträge erweitert werden.

11.2.1 Datenstruktur

Im vorigen Abschnitt wurde bereits erwähnt, dass die interne Struktur hierarchisch als Baum aufgebaut ist. Diese Anordnung ist prädestiniert, weil sich alle gängigen Standards ohne Aufwand in diese Form bringen lassen. Sei es nun *Dublin Core* mit seinen 15 Elementen, die gleichberechtigt auf der obersten Ebene liegen, oder *LOM* mit der semantischen Aufteilung in 9 Kategorien und weiteren Unterkategorien. Letztendlich bietet ein allgemeiner Baum mit beliebiger Tiefe die Freiräume, um heutigen und künftigen Entwicklungen gerecht zu werden.

Mit der Bildung von Kategorien für alle Knoten im Baum bietet LOM einen äußerst flexiblen Mechanismus zur Strukturierung. Er wird daher in die allgemeine Datenstruktur mit aufgenommen, denn so lassen sich eigene *Application Profiles* durch Hinzufügen neuer Kategorien oder Metadatenfelder einfach entwickeln.

In der Entwurfsphase können bereits grob ein paar Kategorien benannt werden, die eine hohe Relevanz für den täglichen Einsatz haben. So sollte eine Kategorie für die didaktischen Eigenschaften enthalten sein, die Kontext, Zielgruppe, Benutzung, Schwierigkeitsgrad und andere subjektive Angaben umfasst. In LOM gibt es entsprechend die Kategorie *Educational* (siehe Abbildung 4.6(a)) und bei *Ariadne Pädagogik*. Bei *Dublin Core* müssen diese Angaben gezielt auf einzelne Elemente abgebildet werden, z.B. bietet sich *Subject* für die Klassifizierung und *Type* für die Form an. Letzteres gibt an, ob das Lernobjekt z.B. ein Text, ein Bild oder interaktiv ist.

Zu den relevanten objektiven Metadaten gehören Attribute wie beispielsweise „Titel“, „Autor“, „Typ“ und „Format“. LOM und *Ariadne* bieten hier eine große Auswahl vordefinierter Felder, die als obligatorisch oder optional festgelegt werden können. Im Interesse der einfachen Nutzbarkeit sollten Metadaten, wann immer möglich, als optional definiert sein. Lediglich ein Mindestsatz für Daten, die oft benötigt werden, wie z.B. bei Suchanfragen, muss obligatorisch sein. In der Regel sind Autoren/-innen gering motiviert, viele Metadaten einzugeben, sodass die Akzeptanz stark vom Komfort abhängt. Müssen bei der Erstellung von Lernobjekten erst zig Metadaten eingegeben werden, bevor die Speicherung durchgeführt wird, ist schnell die Toleranzschwelle überschritten. Da es sich aber um objektive Metadaten handelt, sollte das System in der Lage sein, wenigstens Vorschläge, wenn nicht sogar die richtigen Daten, einzutragen.

Eine besondere Bedeutung besitzen die technischen Rahmenbedingungen, die eigentlich auch zu den objektiven Metadaten gehören, aber wegen ihres Umfangs eine eigene Kategorie zugeteilt bekommen. Diese Kategorie gibt in erster Linie die Bedingungen vor, die für eine optimale Präsentation erfüllt sein müssen. Von der Bildschirmauflösung über den einzusetzenden Browser bis hin zu Speicher- und Prozessoranforderungen reicht das Spektrum möglicher Angaben. Abbildung 11.16 zeigt schematisch den Baum für die drei beschriebenen Kategorien.

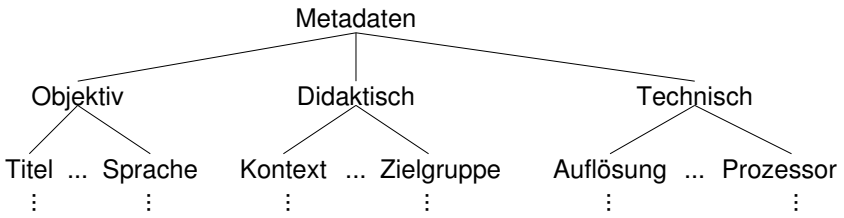


Abbildung 11.16: Metadatenkategorien

Neben der Anordnung der Metadaten sind auch die eingesetzten Datentypen der Attribute von Bedeutung. Hierunter werden verschiedene Wertebereiche verstanden, wie z.B. Text, Datum und Zahl. Sie müssen so allgemein gehalten sein, dass sie von allen Metadaten verarbeitenden Applikationen interpretiert werden können. Dem gegenüber steht der uneingeschränkte Einsatz eigener Metadaten, die durchaus exotischer sein dürfen. Hier muss ein geeigneter Kompromiss zwischen Verarbeitbarkeit und Flexibilität gefunden werden. Hinzu kommen noch komplexere Strukturen, die nicht vollständig durch die Basistypen abgedeckt werden. Im Folgenden werden wesentliche Merkmale der Datentypen diskutiert.

Manch ein Metadatum enthält ein oder mehrere Wörter, um eine bestimmte Eigenschaft eines Lernobjekts zu beschreiben. Soll beispielsweise der Schwierigkeitsgrad einer Aufgabe angegeben werden, bieten sich Wörter wie „schwierig“, „mittel“ und „einfach“ an. Eine ähnliche Bedeutung kann aber auch mit „anspruchsvoll“, „durchschnittlich“ und „leicht“ erreicht werden. Reicht die Auflösung dieser Bewertung nicht aus, können auch Gradpartikeln wie „sehr“, „ziemlich“ und „wenig“ hinzugefügt werden. Der Fantasie sind in natürlichen Sprachen kaum Grenzen gesetzt. Für die maschinelle Verarbeitung ergibt sich aus dieser Vielseitigkeit jedoch ein Problem, da eine Interpretation kaum möglich ist. Auf eine maschinennahe Kodierung, z.B. in Form eines Wertebereichs 1–6, sollte im Interesse der Anwender/-innen dennoch verzichtet werden. Die adäquate Lösung sind **Wörterbücher**, die einerseits das Verständnis natürlichsprachlicher Wörter beinhalten, andererseits wenig Komplexität in die Programmierung der Applikationen bringen.

Ein Wörterbuch ist eine vorgegebene Menge an Wörtern, die für ein Metadatum eingesetzt werden dürfen. Festgehalten in einem Standard, gewährleistet ein Wörterbuch für ein Metadatenfeld die semantische Interoperabilität beim Austausch zwischen unterschiedlichen Systemen. Obwohl die Standards explizit die Angabe nicht enthaltener Wörter erlauben, ist eine Einhaltung stark empfohlen, damit der Sinn eines Wörterbuchs nicht untergraben wird.

Die Standards *LOM* und *Dublin Core* definieren eine Reihe von Wörterbüchern, die in den jeweiligen Klassen der Metadaten-API unterstützt werden sollten. Für die Datenhaltung wird die Klasse `Dictionary` definiert, die eine schnelle Überprüfung ermöglicht, ähnlich einer Hash-Menge. Um nicht an Unterschieden in der Groß- und Kleinschreibung zu scheitern, wird für eine überschaubare Anzahl von Einträgen eine sortierte Liste für die Implementierung empfohlen.

Nicht alle Inhalte der Wörterbücher sind direkt in den Standards definiert, sondern verweisen auf externe Quellen, was bei der Realisierung der Klasse `Dictionary` zu ernsten Hindernissen führen kann. Lässt sich ein umfangreiches Wörterbuch mit den existierenden MIME-Types in Form einer Liste mit Aufwand umsetzen, sieht es beim „*Thesaurus of Geographic Names*“ (TGN) problematisch aus. Hierbei handelt es sich um eine strukturierte Sammlung von Vokabeln, die Millionen von Ortsnamen und andere Informationen wie Koordinaten enthält. Von Kontinent bis Provinz ist alles vertreten. Abgesehen von der Implementierung ist die eigenständige Pflege solcher Daten nur mit erheblichem Aufwand möglich. Daher müssen solche Daten extern eingekauft oder von entsprechenden Dienstleistern online überprüft werden. Wie die Realisierung der Klasse `Dictionary` letztendlich aussieht, hängt von den verwendeten Metadaten ab, sodass in der Entwurfsphase keine weiteren Details festgelegt werden können.

Ein anderes Problem beim Umgang mit Metadaten adressiert die **Internationalisierung** (I18N)[1]. Dieser Begriff wird überwiegend bei Programmen verwendet, die eine Anpassung an sprachliche und kulturelle Gegebenheiten erlauben. Hierzu gehören beispielsweise Texte in der Landessprache, Währungen sowie physikalische Einheiten wie Zeit, Längen und Gewichte. Im Bereich der Metadaten ist neben der Kodierung der unterschiedlichen Daten auch die Deklaration von Land und Sprache Gegenstand der Internationalisierung. Die passenden Regeln samt Standards wurden bereits bei der Vorstellung von LOM in Abschnitt 4.3 detailliert erläutert. Bei den Werten handelt es sich um Tupel, bestehend aus einem einfachen oder zusammengesetzten Code und einer Ressource. Hierbei ist es prinzipiell gleich, ob es Texte, Bilder oder andere Daten sind. Im Falle der Metadaten lässt sich diese Vielfalt auf Informationen in Form von Texten beschränken.

Der Entwurf einer Klasse für die Vorhaltung internationalisierter Daten gestaltet sich einfach. Neben der Speicherung von Schlüssel-Wert-Paare — wobei der Schlüssel beispielsweise eine Verbindung aus Länder- und Sprach-Code ist — muss die Gültigkeit der verwendeten Codes überprüft werden. Bei einer Programmiersprache wie *Java* liefert der Standardsatz an Libraries bereits solche Mechanismen, sodass die Implementierung schnell verrichtet ist. Weniger gute Unterstützung bieten die Programmiersprachen bei personenbezogenen Daten, die sich aus mehreren einfachen Daten, wie z.B. Name, Adresse und Organisation, zusammensetzen. Da der Austausch dieser Daten zwischen unterschiedlichen Systemen das Hauptanliegen ist, soll wie bisher verfahren werden und auf etablierte Vereinbarungen zurückgegriffen werden. Hierzu gehören die **VCards**, eine Art elektronische Visitenkarten, deren Inhalt und Form vom *International Mail Consortium* (IMC) vorgegeben wird. Dieser Standard wird überwiegend in Mails, HTML-Seiten, elektronischen Adressverwaltungen und im mobilen Sektor, wie z.B. dem Handy, eingesetzt. Eine übliche Kodierung, die alle Programme lesen können, ist ein Text mit folgender Struktur. Das Beispiel ist aus der Spezifikation der IMC entnommen.

```
BEGIN:VCARD
N:Wason;Thomas;D.;Dr.;Sr.
FN:Thomas D. Wason, Ph.D.
ORG:IMS Project;Meta Data Team
ADR:;IMS Project;1421 Park Drive;;North Carolina;27605-1727;USA
TEL:+1 919.839.8187
EMAIL;INTERNET:twason@imsproject.org
LABEL;QUOTED-PRINTABLE:IMS Project=0A= 1421 Park Drive
  =0A= Raleigh, NC 27605-1727=0A= USA
END:VCARD
```

[1] Die Abkürzung I18N stammt aus dem Englischen und steht für das Wort *Internationalization*. Mit der Zahl 18 werden die achtzehn Buchstaben zwischen „I“ und „N“ angegeben.'

Ohne auf die Details eingehen zu müssen, lässt sich für die Datenhaltung ein bekanntes Schema entdecken. Schlüssel werden mit Werten zu Paaren kombiniert. Auf diese Weise lassen sich auch einfache Parser realisieren, die dieses Muster erkennen. Ist eine genauere syntaktische Analyse gewünscht, sollte auf Produkte Dritter zurückgegriffen werden.

Abschließend soll die Kodierung von Zeit und Zeitspannen thematisiert werden. In Verbindung mit dem Standard LOM wurden die verwendeten Zeitformate kritisiert, weil sie Mehrdeutigkeiten zulassen. Doch wie sehen die Alternativen zu diesen Standards aus, besonders wenn sie explizit vorgeschlagen sind? Die Metadaten-API kommt also nicht umhin, diese Formate zu unterstützen. Ansonsten wäre die Kompatibilität zu anderen Anwendungen gefährdet, was wesentlich schlimmer ist, als die genannten Probleme.

11.2.2 Operationen

Mit einer Metadatenstruktur ist wenig anzufangen, wenn sie nicht verändert bzw. auf Richtigkeit überprüft werden kann. Aus diesem Grund bieten die Klassen der Metadaten-API verschiedene Operationen an. In dieser Arbeit sollen drei Formen unterschieden werden: **Produktion**, **Manipulation** und **Validierung**. Unter Produktion wird die eigentliche Erstellung neuer Metadaten verstanden. Veränderungen an existierenden Daten gehören zur Manipulation und die Überprüfung der syntaktischen Korrektheit ist Aufgabe der Validierung. Weil diese Aufteilung essentiell für die Gestaltung der Klassen ist, erfolgt nun eine detailliertere Ausführung.

Alle Methoden in den Klassen der Metadaten-API, mit denen die interne Metadatenstruktur aufgebaut wird, gehören zur Produktion. Es gibt verschiedene Kontexte für die Aufrufe, von denen die wichtigen in Abbildung 11.17 dargestellt sind.

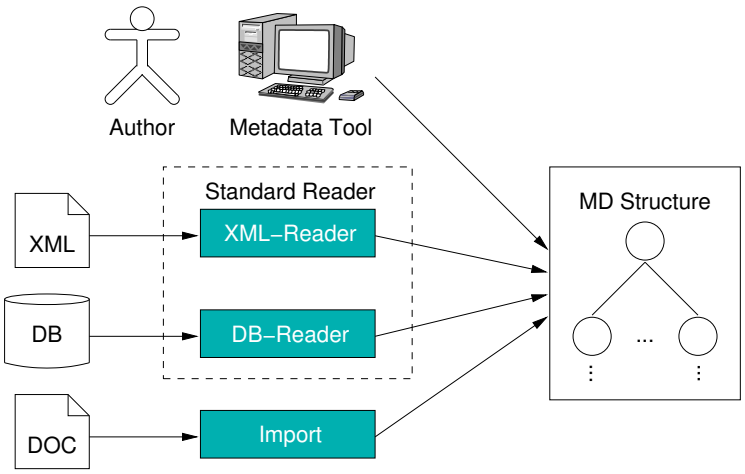


Abbildung 11.17: Produktion der internen Metadatenstruktur

In der Mitte befinden sich die *Reader*-Klassen, die in Abschnitt 11.2 vorgestellt wurden. Auch wenn es sich um die Konvertierung einer Repräsentation, in diesem Falle XML-Datei und Datenbank, in einen Objektbaum handelt, ist es aus Sicht der API ein neuer Aufbau von Metadaten. Ähnlich verhält es sich mit dem Import fremder Daten, ganz unten in der Abbildung. Angedeutet durch die Dateiendung `DOC` des Formats von *MS Word*, steht diese Art der Produktion für die Integration beliebiger Metadaten aus anderen Quellen. Hierbei ist festzuhalten, dass es niemals eine vollständige Abbildung geben kann. Im Falle des *Word*-Formats sind nur wenige Metadaten enthalten und die beschränken sich auf die Autoren/-innen, den Entwicklungsprozess und den Inhalt. Vielfältiger ist beispielsweise das Format *DocBook* ausgestattet, bei dem gezielt bestimmte Teile des Dokuments mit Metadaten versehen werden können. In der Praxis zeigt sich jedoch, dass eine 1:1 Abbildung auf die Standards wie *Dublin Core* oder LOM nicht möglich ist. Teilweise müssen Daten sogar verworfen werden, was die vollständige Automatisierung von Importprozessen schwieriger macht.

Freilich gibt es noch die Möglichkeit, Metadaten von Hand aufzubauen, was durch die Rolle *Author* mit ihrem Werkzeug, dem *Metadata Tool*, angedeutet ist. Besonders die subjektiven Daten zu einem Lernobjekt müssen vom Menschen festgelegt werden. In Anbetracht der Komplexität der Metadatenstandards stellt die adäquate Präsentation eine Herausforderung dar. Einerseits soll kein Metadatum vorenthalten werden, andererseits darf eine komplizierte Darstellung nicht verschrecken. Wie der Kompromiss aussieht, ist für die Metadaten-API von geringer Bedeutung. Sie kommt nicht umhin, alle Möglichkeiten anzubieten.

Doch wie gestaltet sich jetzt die Produktion von Metadaten? Zuerst muss ein Hauptknoten für die hierarchische Struktur erstellt werden, der nicht zwangsläufig Attribute beinhaltet. An ihn lassen sich die ersten Knoten für bestimmte Kategorien einhängen. Die Erzeugung dieser Objekte geschieht am besten über die Konstruktoren selbst, die bereits erste Attribute als Parameter annehmen. Bei weniger genutzten Attributen ist es sinnvoll, sie erst über spezielle Methoden zu setzen, um die Konstruktoren nicht unnötig umfangreich zu gestalten. Hiernach können weitere Unterkategorien eingehängt werden, die wiederum Unterkategorien mit jeweiligen Attributen akzeptieren. Auf diese Weise erfolgt die komplette Produktion der Metadatenstruktur.

Liegen die Metadaten nach der Produktion im Speicher vor, können sie über geeignete Methoden der API manipuliert werden. Grundsätzlich wird die Bearbeitung in Veränderungen der Struktur und der Inhalte unterschieden. Die Objekte der Klassen zur Kategorisierung lassen sich löschen, verschieben oder neu erzeugen, sodass eine neue Anordnung erstellt wird. Bei diesen Operationen muss besonders darauf geachtet werden, dass anschließend eine Abbildung auf standardkompatible Strukturen möglich ist. Weniger kritisch sieht es hingegen bei den Attributen aus. Als Bestandteil der Klassen zur Kategorisierung können sie entweder gesetzt oder gelöscht werden. Da bei allen gängigen Abbildungen die Attribute optional sind — eine sehr wichtige Eigenschaft für *Application Profiles* —, gibt es bis auf die Einhaltung des Wertebereichs wenig zu beachten.

Die Manipulation der Metadaten kann nur auf zwei Wegen geschehen, wie in Abbildung 11.18 dargestellt. Entweder nehmen die Anwender/-innen die Bearbeitung selbst vor oder ein Mechanismus eines Programms ändert die Metadaten, die bekannt sind oder sich ableiten lassen.

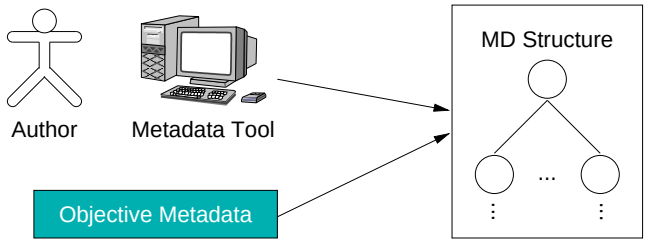


Abbildung 11.18: Manipulation der internen Metadatenstruktur

Alle Methoden zur semantischen und syntaktischen Überprüfung fallen unter die Kategorie der Validierung. Für diese Operationen ist besonders der Zeitpunkte der Konvertierung zwischen den Formaten kritisch, denn hierbei kann es im schlimmsten Fall zu unerwünschten Datenverlusten kommen, weil sich Metadaten nicht abbilden lassen. Da alle Standards Vorgaben zur Kodierung machen, können entsprechende Mechanismen meist problemlos realisiert werden.

Als Beispiel soll die Klasse `LOMReader` eine LOM-Datei im XML-Format einlesen und auf syntaktische Korrektheit überprüfen. Es wird eine vorhandene XML-Library genutzt, sodass im Entwurf die Entwicklung eines Parsers nicht berücksichtigt werden muss. Bei komplexeren Typen, wie einem Datum oder einer VCard, muss die Validierung nachträglich durchgeführt werden. Wird ein Fehler entdeckt, müssen die Anwender/-innen davon in Kenntnis gesetzt werden und gegebenenfalls bei der Behebung zu Rate gezogen werden. Sind die Fehler zu gravierend, sollte von einem Einlesen ganz abgesehen werden.

Bei dieser Richtung der Konvertierung kann die Metadaten-API wenig Einfluss nehmen. Anders sieht es bei Manipulationen des Objektbaums aus. Hier müssen die jeweiligen Klassen Methoden anbieten, mit denen sich Struktur und Werte der Attribute überprüfen lassen. Dies setzt freilich die Festlegung eines Formats voraus, auf das hin die Validierung erfolgt. Zwar sind besonders viele Standards erstrebenswert, aber die Relation zwischen Aufwand und Nutzen muss gewahrt sein. Aus diesem Grund wird wenigstens die Unterstützung von LOM empfohlen, weil es zu den wichtigsten Formaten zählt. Eine Überprüfung nach *Dublin Core* dürfte wegen der geringen Komplexität ebenfalls schnell implementiert sein, hat aber nicht die Bedeutung wie LOM. Letztendlich sind Umfang und Genauigkeit Sache der Implementation. Beim Entwurf muss nur berücksichtigt werden, dass eine Integration der Validierungsmechanismen möglich ist.

11.2.3 Kodierungen

Die interne Metadatenstruktur als Objektbaum ist der ideale Ausgangspunkt für die Erstellung und Bearbeitung von Metadaten, jedoch ist sie wenig für den direkten Austausch zwischen Applikationen geeignet. Es kann zwar auf Mechanismen der Programmiersprachen zurückgegriffen werden, wie z.B. die Serialisierung bei *Java*, aber dies führt zu einer zu engen Bindung an eine bestimmte Implementation. Auch Middleware-Lösungen, wie z.B. Corba, schränken die Nutzung zu sehr ein und haben hohe Anforderungen an die Technik. Der Weg über den Objektbaum im Speicher kann daher nie direkt begangen werden, sondern erfolgt immer über die *Reader*- und *Writer*-Klassen aus Abbildung 11.14. Über das Format XML soll in diesem Abschnitt nicht viele Worte verloren werden. Es ist obligatorisch und die Spezifikation der vorgestellten Metadatenstandards *LOM* und *Dublin Core* geben ausführlich Auskunft. Hingegen ist die Speicherung in Datenbanken ein wenig beachtetes Thema, obwohl sie gerade für Suchanfragen optimal geeignet ist.

Die direkte Repräsentation im Speicher ist für diese Aufgabe nämlich nur bedingt geeignet. Es ist wenig sinnvoll, die Metadaten aller Lernobjekte im Speicher vorzuhalten, um eine schnelle Suche durchführen zu können. Der Bedarf der hierfür nötigen Ressourcen steht in keiner Relation zum Nutzen. Auch die Kodierung in XML bringt keine nennenswerte Verbesserung, weil die resultierende Rechenzeit zu hoch ist. Zuerst müssten die Metadaten aller Lernobjekte geladen werden, um sie danach mit der Suchanfrage zu vergleichen. Abgesehen vom Zeitbedarf der Dateizugriffe sollte auch der Aufwand für die Berechnung bereits wenig komplizierterer Verknüpfungen nicht unterschätzt werden. Nicht ohne Grund gibt es hierfür Datenbanken, die genau für diese Art von Aufgaben optimiert sind. Es stellt sich nur die Frage, welche Datenbanktechnik gewählt wird, da diese Entscheidung Einfluss auf die entsprechenden *Reader* und *Writer* hat.

Neben den klassischen relationalen Datenbanksystemen gibt es auch XML-Datenbanken. Sie speichern ihre Daten nativ in XML und scheinen die ideale Lösung für Suchanfragen über Metadaten zu sein, weil sie eine direkte Abspeicherung der in XML vorliegenden Datensätze erlauben. Hierdurch entfallen lästige Konvertierungen in beide Richtungen, beim Import von Lernobjekten und beim Export. Leider ist die Technologie noch relativ jung und im Gegensatz zu etablierten Techniken nicht leistungsfähig genug. Besonders bei großen Datenmengen sind die Unterschiede gravierend, sodass den XML-Datenbanken erst ein prototypischer Stand attestiert werden kann und auf relationale Systeme zurückgegriffen werden muss. Sie profitieren von einem fundierten mathematischen Ansatz, der sich in der Praxis bewährt hat. Für die Modellierung bedeutet diese Entscheidung hingegen, dass Umstände in Kauf genommen werden. Zwar bieten einige Hersteller relationaler Datenbanken Erweiterungen an, die eine automatische Abbildung von XML-Daten auf ihr Produkt ermöglichen, aber trotzdem kann es bei unvorsichtiger Konfiguration zu Datenverlusten kommen. Details zu diesem Thema finden sich z.B. in [Schöning03].

Leider sind die Verfahren dermaßen unterschiedlich, dass es keinen einheitlichen Mechanismus für diese Aufgabe gibt. Als Konsequenz wird bei Verwendung dieser vorgefertigten

Lösungen Logik aus dem Programm in die Datenbank verlagert, was eine echte Schwäche des Entwurfs wäre und unter allen Umständen verhindert werden muss. Eine echte Unabhängigkeit kann daher nur mit einer eigenen Abbildung erreicht werden, die bereits im Entwurf Berücksichtigung findet.

Die Struktur und Daten eines XML-Dokuments müssen folglich „per Hand“ auf die Tabellen abgebildet werden. Hierfür soll das *object-relational Mapping* von [Bourret99] eingesetzt werden, bei dem eine hierarchische Struktur aus Objekten gebaut wird, die sich besonders einfach auf ein relationales Modell abbilden lassen. Es werden zwei XML-Elemente unterschieden: komplexe Elemente mit Unterelementen und einfache Elemente, die nur Text enthalten[2]. Die komplexen Elemente werden jeweils als Typ *Klasse* bezeichnet und die einfachen als Typ *Eigenschaft*. Eine Klasse hat immer andere Klassen und Eigenschaften als Attribute, wobei Eigenschaften lediglich aus einem einfachen oder zusammengesetzten Wert bestehen.

Vater-Kind-Beziehungen zwischen zwei Elementen eines XML-Dokuments werden als *inter-class*-Beziehung bezeichnet, wenn beide Elemente vom Typ Klasse sind. Ist das Kind-Element vom Typ Eigenschaft, so liegt eine *class-property*-Beziehung vor. Mit Hilfe dieser Unterscheidung kann das *object-relational Mapping* durchgeführt werden.

- Klassen auf Tabellen
- Eigenschaften auf Tabellenspalten
- *inter-class*-Beziehungen werden zu Primär-Fremdschlüssel-Paaren
- Eigenschaften mit einem einfachen Wert können auf eine Spalte einer Klassen-Tabelle oder als separate Tabelle abgebildet werden
- Eigenschaften mit zusammengesetztem Wert müssen auf separate Tabellen abgebildet werden
- Das Wurzelelement wird ignoriert, weil es nur eine syntaktische Funktion erfüllt
- Klassen, die nur Eigenschaften als Attribute besitzen, können aufgelöst werden, indem die Eigenschaften als Attribute der Basisklasse definiert werden

Anhand der Kategorie *General* des Standards LOM soll das Ergebnis dieses Verfahrens exemplarisch verdeutlicht werden. Die in Abbildung 4.6(b) angegebenen Unterkategorien ergeben das vereinfachte Resultat in Abbildung 11.19.

Die Unterkategorie *Identifier* ist reserviert und kann nicht berücksichtigt werden. Aus Gründen des Platzes und der Übersicht sind *Structure* und *Aggregation* ebenfalls nicht dargestellt, weil sie Vokabulare nutzen, die weitere Tabellen nach sich ziehen. Eine vollständige Umsetzung ist dann Aufgabe der Implementation.

Mit der Vollständigkeit der Klassen für den Umgang mit Metadaten kann nun wieder abschließend eine Komponente gebildet werden, wie sie in Abbildung 10.10 zu sehen ist. Aufgrund der Vielzahl von Klassen umfasst die Darstellung dieses Prozesses nur die besonders herausragenden. Andere Klassen, die zwar auch essentiell sind, werden in Abbildung 11.20 zu Gunsten der Übersicht nicht angezeigt.

11.3 Unterstützung von Multimedia

Bei der Erstellung modularer E-Learning-Inhalte kommt eine Autorenumgebung nicht umhin, multimediale Inhalte zu unterstützen. Die Anforderungsbeschreibung der Komponente `Multimedia Environment` in Unterabschnitt 10.2.2 beschrieb bereits grob den nötigen Funktionsumfang und soll nun verfeinert werden. Grundlegend wurde beschlossen, dass externe

[2] In XML-Notation tritt dieses Element als *Parsed Character Data* (PCDATA) oder *Character Date* (CDATA) auf.

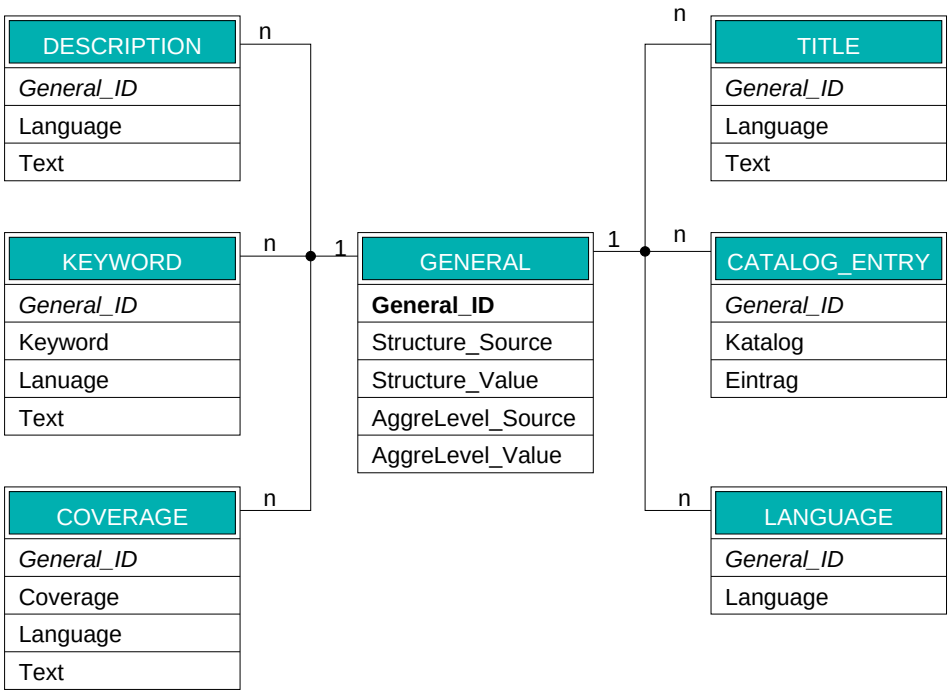


Abbildung 11.19: Datenbankschema für die Kategorie „General“ aus [Turan04]

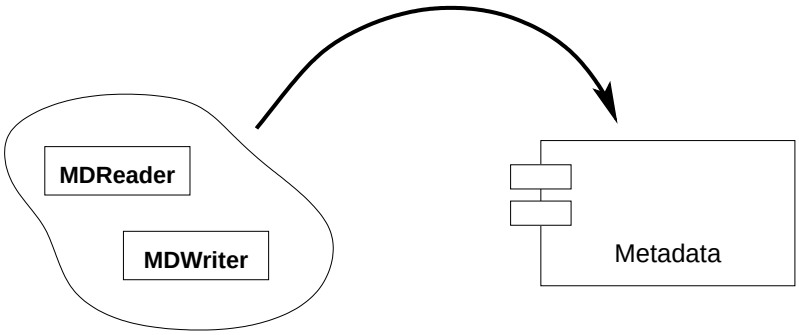


Abbildung 11.20: Bildung der Komponente `Metadata`

Programme für die Bearbeitung multimedialer Inhalte aufgerufen werden und eine Eigenentwicklung nicht in Frage kommt. In Anbetracht des Aufwands, den die adäquate Umsetzung eines Format wie z.B. Flash verursacht, ist diese Entscheidung vernünftig. Als Lösung wird nun eine Art Verknüpfung zwischen Anwendungen und Dateien vorgestellt.

Zunächst erhalten die Dateien eine Typisierung, um ihre Inhalte und deren Kodierungen unterscheiden zu können. Eine Möglichkeit sind die so genannten *Multipurpose Internet Mail Extensions* (MIME), deren Spezifikation sich über 5 RFCs erstreckt. Für diese Arbeit ist lediglich der zweite Teil, RFC 2046 [Freed96], interessant, weil er definiert, wie Haupt- und Nebentypen verschiedener Formate kodiert werden. Zur Zeit sind die sieben Haupttypen `text`, `application`, `image`, `audio`, `video`, `message` und `multipart` festgelegt. Ohne weitere Ausführungen lässt sich bereits erkennen, dass alle relevanten Typen multimedialer E-Learning-Inhalte unterstützt werden. Die Nebentypen geben genauere Auskunft darüber, was und vor allem wie es in einer Datei kodiert ist. Als Beispiel sei der Haupttyp `image` angeführt, der für Grafiken und Abbildungen jeglichen Formats steht. Einem Programm zur Anzeige oder Bearbeitung solcher Dateien reicht diese Information noch nicht aus, denn es gibt viele verschiedene Kodierungen, die ihre Vor- und Nachteile besitzen. Mit verlustbehafteten Komprimierungsverfahren wie z.B. JPEG lassen sich gut natürliche Bilder wie Fotografien komprimieren. Hingegen gehen bei synthetischen Grafiken mit vielen Linien, gleichfarbigen Flächen und Schriften wesentliche

Merkmale verloren. Gravierend wirkt sich die Unterabtastung auf die Qualität aus, wodurch das Bild unscharf erscheint. Durch verlustfreie Verfahren wie z.B. PNG werden bessere Ergebnisse erzielt. Für eine genaue Unterscheidung der Dateien gibt es folglich die Typen `jpeg` und `png`, die voll ausgeschrieben mit den Haupttypen angegeben werden, also `image/jpeg` bzw. `image/png`.

In der Praxis stellt sich jedoch die Erkennung eines Datentyps als Problem heraus, denn einer Datei ist von außen nicht ohne weiteres anzusehen, was sie enthält. Viele Dateiformate sind binär kodiert und zudem proprietär, sodass es schwer ist, ein allgemeines Verfahren zu benennen. Eine Möglichkeit ist die Erstellung so genannter *Fingerprints*. Mit Hilfe von *Byte Frequency Analysis*, *Byte Frequency Cross-Correlation Analysis* und *File Header/Trailer Analysis* werden Muster erkannt, die typisch sind für ein Dateiformat [McDaniel03]. Andere Verfahren erkennen „Magic Numbers“, also ganz bestimmte Byte-Folgen. Der große Nachteil gegenüber den *Fingerprints* ist die mangelnde Allgemeingültigkeit, denn für jedes Format muss mindestens eine eigene Regel hinterlegt sein.

Egal, welcher Erkennungsmenchanismus letztendlich eingesetzt wird, haben sie alle ein Problem gemein: Die zu untersuchende Datei muss mindestens ein Mal geöffnet und analysiert werden, was entsprechende Rechenzeit benötigt. In Hinblick auf das geplante Einsatzgebiet, bei dem in verschachtelten Lernobjekten mit möglicherweise hunderten von Dateien gearbeitet wird, kann es schnell zu Engpässen kommen. Bereits für die Anzeige der enthaltenen Dateien müssen die Dateitypen bekannt sein, um sie z.B. in einer grafischen Darstellung durch eigene *Icons* hervorzuheben. Die genannten Verfahren sind folglich ungeeignet und es muss nach einer anderen Lösung gesucht werden.

Das Betriebssystem *Windows* von *Microsoft* benutzt dreistellige Dateiendungen, um den Dateityp zu bestimmen. Aber nicht nur die auf den ersten Blick zu erkennende Fehldeutung, ausgelöst durch eine falsch eingegebene Dateiendung, offenbart Schwächen dieses Systems. Noch schlimmer sind Angriffe auf den Rechner mit Hilfe kompromittierter Dateien. Ein Mail-Filter, der sich auf diese Weise austricksen ließe, wäre nicht viel wert. Doch so schwerwiegend solche Argumente auch sein mögen, für das angestrebte Ziel sind sie wenig von Bedeutung. Schließlich sollen aus dem Autorensystem Programme aufgerufen werden, die bereits installiert sind. Der gewonnene Geschwindigkeitsvorteil gegenüber den analysierenden Verfahren rechtfertigt letztendlich die Verwendung von Dateiendungen.

Steht der Typ einer Datei erst fest, muss noch die gewünschte Operation bestimmt werden. Abhängig vom Typ lassen sich Dateien öffnen, bearbeiten, drucken, übersetzen und in vielen anderen unterschiedlichen Formen nutzen. Die Auswahl kann z.B. über ein Kontextmenü aufgerufen werden, das über eine bestimmte Taste oder Mausaktion geöffnet wird und alle möglichen Operationen anzeigt. Auf diese Weise ist es möglich, eine Datei mit verschiedenen Programmen zu öffnen, was die Flexibilität erhöht. Die Auswahl der gewünschten Operation erfolgt über ein **Verb**, das als Argument an das System übergeben wird. Weil das allgemeine Öffnen in der Praxis die am meisten genutzte Funktion ist, sollte sie in den Programmen mit dem Doppelklick der Maus verbunden sein. Dieses Verhalten hat sich auf vielen Systemen als Standard etabliert und sollte den Anwendern/-innen zuliebe beibehalten werden. Eine Sonderrolle spielt noch das Erstellen neuer Dateien eines Typs, denn dieser Prozess ist nicht abhängig vom Kontext irgendeiner Datei. Wo auch immer das aktuelle Arbeitsverzeichnis ist, sollte diese Aktion mit wenigen Handgriffen möglich sein.

Anhand der beschriebenen Funktionen lassen sich nun die Schnittstellen und Klassen bestimmen, die für die Unterstützung multimedialer Dateien benötigt werden. Ein wesentliches Merkmal ist die Vielzahl der möglichen Operation für einen Dateityp, die irgendwie angesteuert werden wollen. Aus diesem Grund werden sie in einer eigenständigen Klasse zusammengefasst, die sich über eine einheitliche Schnittstelle ansteuern lässt. Abbildung 11.3 zeigt, dass für den Aufruf lediglich zwei Methoden ausreichen. Der Signatur nach kann eine Datei plus ein optionales Verb für die Auswahl der Operation als Argumente übergeben werden. Ist kein Verb angegeben, soll eine Standardoperation, wie das bereits erwähnte Öffnen, ausgeführt wer-

den. Die Erstellung einer neuen Datei kommt sogar mit einer Methode aus, wie in Abbildung 11.21(b) zu sehen ist.

(a) Interface `ProcessFile` (b) Interface `NewFile`

Abbildung 11.21: Interfaces für den Zugriff und die Erstellung von Dateien

Es steht den Entwicklern/-innen frei, ob sie für die Bearbeitung und Erstellung jeweils unterschiedliche Klassen entwerfen oder sie in einer vereinen. Der Entwurf dieser Arbeit soll die wichtigsten Formate unterstützen, damit beim Einsatz nicht für jede „Standarddatei“ erst eigene Klassen erstellt werden müssen. Zu den unterstützten Formaten gehören Bausteine, Kurse und eine Beschreibungssprache für Inhalte in XML. Anstatt von Klassen zu reden, die zwei Schnittstellen implementieren, soll der Begriff **Handler** eingeführt werden. Zusammen mit einer Abkürzung des Dateiformats ergeben sich dann Klassen mit Namen, wie sie in Abbildung 11.22 verwendet werden[3].

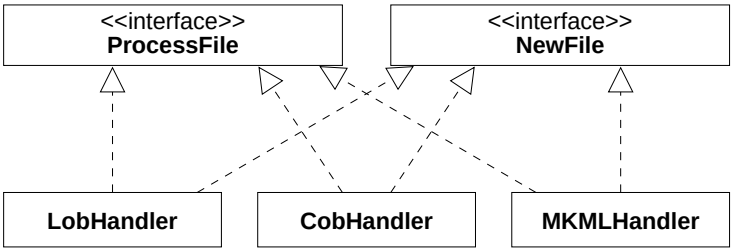


Abbildung 11.22: Drei *Handler*

Im folgenden Kapitel 12 wird auf die internen Details der Formate für Bausteine und Modelle eingegangen. An dieser Stelle sei festgehalten, dass die Klasse `LobHandler` für Bausteine zuständig ist und `CobHandler` für Kurse. Beide *Handler* erlauben es, standardkompatible E-Learning-Inhalte unter Berücksichtigung der Metaphern zu laden, bearbeiten und speichern. Die Abkürzung „MKML“ steht für die mαth-kit-Markup-Language, die in [Baudry03] genauer beschrieben steht. Diese Sprache ermöglicht die Trennung von Inhalt und Darstellung, wie sie in Abschnitt 3.7 für moderne Lernobjekte gefordert wird.

Mit den *Handlern* an sich lassen sich einzelne Dateitypen mit Verben erstellen, aber sie reichen nicht aus, um eine vollständige Dateiverarbeitung anzubieten. Wie soll beispielsweise mit verschiedenen Klassen für einen Dateityp umgegangen werden, denn bei aufwendigen Operationen kann es durchaus sinnvoll sein, diese auf mehrere Klassen aufzuteilen, was die Schnittstellen auch erlauben. Oder was soll passieren, wenn ein Verb aufgerufen wird, das von mehreren *Handlern* unterstützt wird? Diesen Fragen wird die Klasse `MimeTypeHandler` entgegengesetzt, deren Diagramm in Abbildung 11.23 dargestellt ist.

Diese Klasse verwaltet alle *Handler* eines MIME-Typs und delegiert von außen kommende Aufrufe an sie weiter. Über die Methode `setExtensions()` werden die Dateiendungen angegeben, auf die der `MimeTypeHandler` reagieren soll. Ein Aufruf der Methode `processFile` führt die gewählte Operation aus und sollte kein Verb übergeben worden sein, wird automa-

[3]Die Abkürzungen „Lob“ und „Cob“ sind historisch bedingt und haben sich während der Zeit ergeben, als die Metaphern sich noch nicht endgültig bis zur Ebene der Implementierung durchgesetzt hatten. Sie stehen für **L**earning **Ob**ject bzw. **C**ourse **Ob**ject und sind auch heute noch die verwendeten Dateiendungen. Weil sich alle Beteiligten des Projekts daran gewöhnt haben und diese Abkürzungen zu festen Begriffen verankert haben, soll von einer Änderung abgesehen werden.

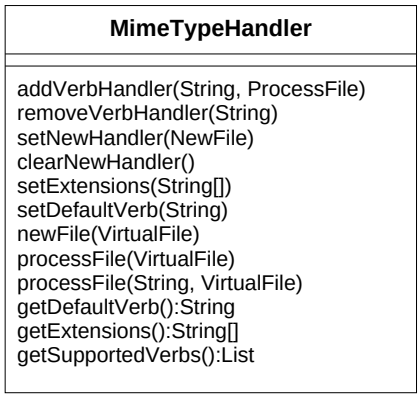


Abbildung 11.23: Klasse `MimeTypeHandler`

tisch das zuvor mit `setDefaultVerb()` gesetzte genommen. Neue Dateien des unterstützten MIME-Typs werden durch den Aufruf `newFile` erzeugt.

Mit dieser Klasse ist die Umsetzung des Aufrufmechanismus für einen MIME-Typ vollständig umgesetzt. Nun fehlt noch eine Verwaltung aller `MimeTypeHandler`-Objekte, die anhand der Dateiendung schnell einen passenden *Handler* auswählt und diesen zurück gibt. Für diese Aufgabe genügt eine Klasse, wie sie in Abbildung 11.24 zu sehen ist.

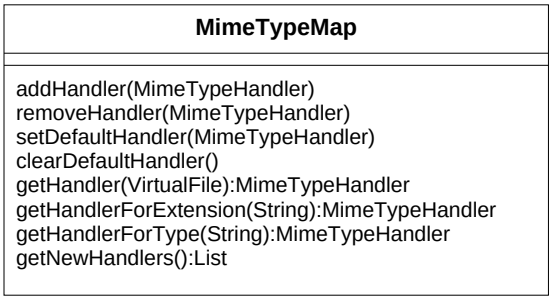


Abbildung 11.24: Klasse `MimeTypeMap`

Wie zu erwarten, bietet sie Methoden zum Hinzufügen, Löschen und Auffinden von Handlern an. Neben den geforderten Dateiendungen (`getHandlerForExtension()`) kann auch der MIME-Typ direkt für die Auswahl herangezogen werden (`getHandlerForType()`). Eine sehr interessante Methode ist `getDefaultHandler()`, mit dem Dateien eines Typs geöffnet werden, für die kein expliziter Handler definiert ist. Auf diese Weise kommt die Komponente für die Bearbeitung multimedialer Inhalte nie in die Verlegenheit, mit einer Datei nichts anfangen zu können. Für die Implementierung sind betriebssystemabhängige Standard-*Handler* vorgesehen, die bei Bedarf von dieser Methode zurückgegeben werden. Sie delegieren den Aufruf an das Betriebssystem weiter und können sogar weitere Informationen liefern, wie z.B. die unterstützten Verben. Der vorgestellte MIME-Typ-Mechanismus bietet somit mindestens die gleichen Möglichkeiten im Umgang mit Dateien an wie das Betriebssystem. Zuletzt sei noch die Methode `getNewHandlers()` erwähnt, die einen kontextfreien Zugriff auf alle *Handler* zur Erstellung einer Datei eines bestimmten Typs gestattet.

Ein kleines Beispiel soll anhand eines Objektdiagramms das Zusammenspiel der vorgestellten Klassen verdeutlichen. Der Einfachheit halber kommt der MIME-Typ-Mechanismus in Abbildung 11.25 mit zwei MIME-Typen aus, die von insgesamt sechs *Handlern* verarbeitet werden können.

Abschließend soll aus den erstellten Klassen eine Komponente für die Rolle *Developer* erstellt werden, wie sie in Abbildung 10.11 zu sehen ist. Der Herleitungsprozess in Abbildung 11.26 enthält wie immer der Übersicht halber nicht alle Klassen.

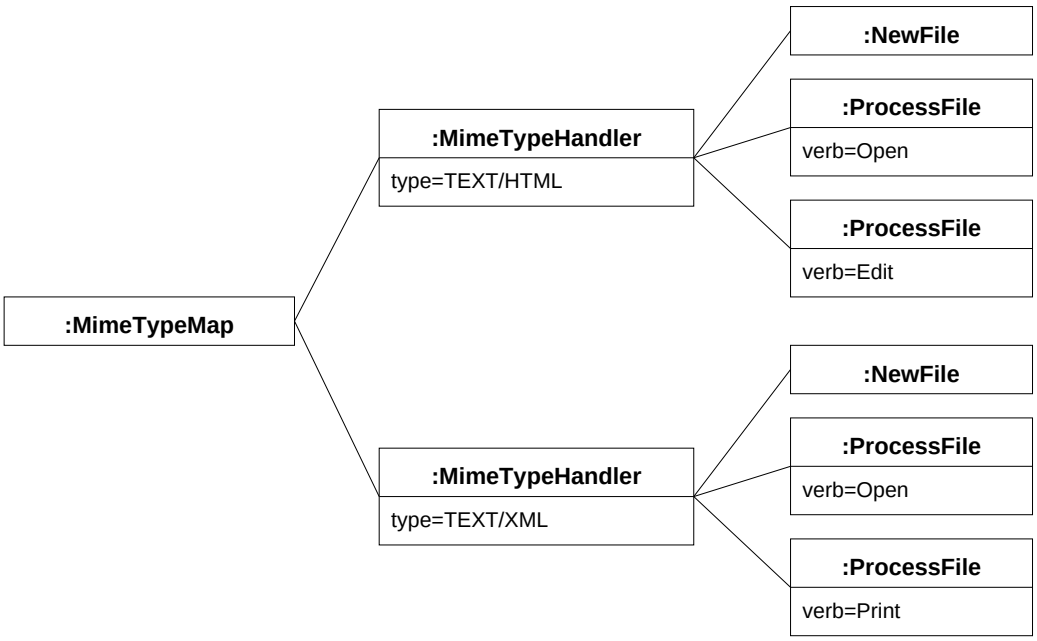


Abbildung 11.25: Objektdiagramm mit zwei unterstützen MIME-Types

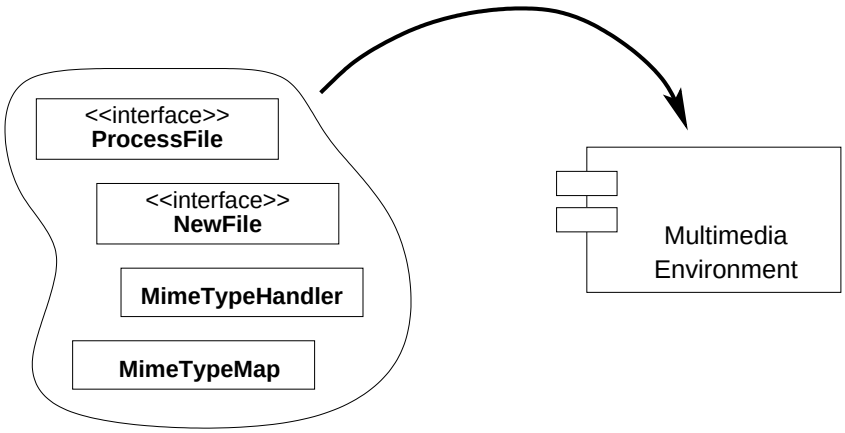


Abbildung 11.26: Bildung der Komponente `Multimedia Environment`

Kapitel 12

Baustein und Kurs

In diesem Kapitel werden alle Komponenten zur Erstellung, Bearbeitung und Verwaltung von modularen E-Learning-Inhalten entworfen. Hierbei unterscheidet sich die Vorgehensweise gegenüber dem in Kapitel 11 beschriebenen durch eine andere Klassenbildung. Gleichwohl auch bei diesen Komponenten das Ziel die Erstellung generischer Libraries ist, wird aufgrund der inhärenten Ähnlichkeit von Baustein sowie Kurs zum *Content Packages* eine abstraktere Abbildung entstehen. Nach außen präsentieren die Komponenten `Learning Object Development` und `Structure Development` unterschiedliche Schnittstellen, aber intern wird zu einem großen Teil auf die gleiche Klassen zurückgegriffen. Die grafische Ansteuerung ist sehr individuell und sollte nicht als Teil des Entwurfs betrachtet werden. Ein konkretes Beispiel für eine grafische Oberfläche, wie sie im Projekt mαth-kit eingesetzt ist, wird im Teil über die Implementierung gegeben.

12.1 Bindung an Standards

Ein erster Versuch der Modellierung wäre, das Problem bzw. den Prozess zu analysieren und daraus die passenden Klassen herzuleiten. Aus dieser „üblichen“ Perspektive geht allerdings schnell ein wichtiges Detail verloren: der Austausch von Inhalten mit anderen Systemen. Im Kapitel 3 über Lernobjekte wurde der Einsatz von Standards vorgeschlagen, um Inkompatibilitäten vorzubeugen. Die Gestaltung des Klassenmodells kommt somit nicht umhin, diesen bedeutenden Aspekt zu berücksichtigen. Hierbei treten die selben Widersprüche zwischen geringer Komplexität und Vielseitigkeit auf, wie bereits im vorherigen Kapitel 11 bei den Metadaten. Dem Wunsch, möglichst viele Standards unterstützen zu wollen, steht eine einheitliche und übersichtliche Schnittstelle der API entgegen. Wenn möglich, sollten keine Spezialfälle berücksichtigt werden, um nicht die Kohäsion der Klassen abzuschwächen.

Zu den verbreitetsten Standards für Lernobjekte sind das *IMS Content Packaging* und das *Sharable Content Object Reference Model* (siehe Abschnitt 3.5 und 3.6) zu zählen, die sich zum Glück sehr ähneln. Sie bieten sich an, den technischen Rahmen für den Entwurf von Baustein und Kurs zu stellen. Für die fachliche Beschreibung dienen die Definitionen der Metaphern aus Abschnitt 10.4, deren Eigenschaften das wesentliche Erscheinungsbild prägen.

Diese Herangehensweise, die Standards dermaßen einzubeziehen, ist nicht offensichtlich und bedarf einer Erklärung. Denn selbst den Spezifikationen ist eine derartige Nähe zu den Implementierungen nicht entnehmbar. Eigentlich sind die Standards für den Austausch zwischen den Systemen gedacht und nicht für die direkte Verwendung in Applikationen. Doch warum soll dieser ungewöhnliche Weg beschritten werden? Weil sich hieraus verschiedene Vorteile ergeben. Zwar ist es beim Entwurf freilich angenehmer, ein Klassenmodell zu erstellen, das keinen externen technischen Einschränkungen unterliegt, aber spätestens beim Austausch mit anderen Systemen muss die Standardkompatibilität bedacht werden. Mit mehr oder weniger Mühen muss dann das eigene Modell auf die vorgegebenen Datenstrukturen abgebildet werden, oft mit mäßigem Erfolg. Die beiden vorgeschlagenen Standards sind leider so flexibel,

dass *Content Packages* im schlimmsten Fall proprietäre Daten enthalten, die doch wieder nur mit speziellen Programmen genutzt werden können. Echte Austauschbarkeit bleibt so auf der Strecke und die resultierenden Konsequenzen zeigen sich in den vorgestellten Produkten aus Kapitel 5. Sie produzieren letztendlich inkompatible Dateien, die im Gewand der Standardkonformität daher kommen und Versprochenes nicht einlösen.

Durch die direkte Verwendung der Standards tritt dieses Problem erst gar nicht auf. Doch wie steht es mit den Anforderungen der Metaphern? Lassen sich die Standardformate so nutzen, dass sie sich wie Bausteine zu höheren Strukturen zusammensetzen lassen? Auch wenn diese Funktion abermals nicht explizit den Spezifikationen zu entnehmen ist, lässt sich solch ein rekursiver Ansatz verwirklichen. Über komplexer werdende Bausteine und Kurse wird sich der eigentlichen Lösung genähert. Abbildung 12.1 zeigt die einfachste Variante eines Lernobjekts als *Content Package*, angelehnt an die Bausteinmetapher.

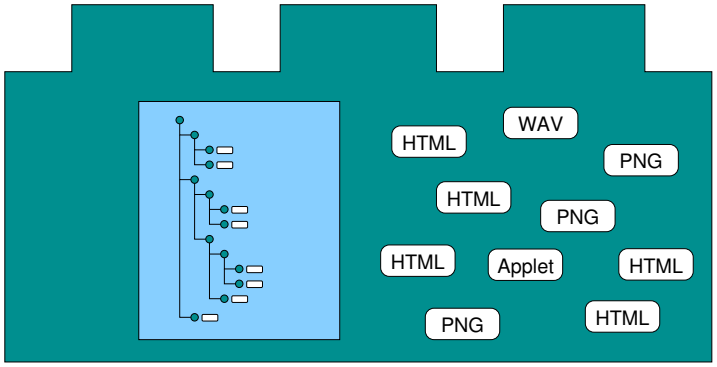


Abbildung 12.1: Ein einfacher Baustein aus [Bungenstock04a]

Auf der linken Seite ist das Manifest zu erkennen, das eine hierarchische Struktur enthält. Die Knoten haben als Attribute Referenzen auf die rechts stehenden physikalischen Dateien, die den Inhalt des Lernobjekts ausmachen. Es handelt sich um einen Baustein, wie er von der Rolle *Developer* erstellt wird. Bei dieser üblichen Form des *Content Packages* gibt es wenig technische Herausforderungen. Lediglich die Speicherung der Dateien und der Aufbau das Manifests müssen modelliert werden. Auch wenn diese Funktionen bereits einige Überlegungen benötigen, um zu einer geschickten Lösung zu gelangen, stellt der nächste Schritt, nämlich die Komposition verschiedener Bausteine zu einem Kurs, die eigentliche Herausforderung dar.

Es stehen zwei Möglichkeiten für die Umsetzung zur Auswahl. Die erste, bei der Bausteine auf Submanifeste abgebildet werden, ist von den Spezifikationen explizit vorgesehen. Bei der Komposition von Bausteinen werden die Dateien in ein *Content Package* kopiert und die Manifeste zu einem großen Manifest zusammengeführt. Jedes Manifest eines Bausteins wird auf diese Weise zu einem Submanifest, das über ein Item referenziert wird. Das Resultat ist in Abbildung 12.2 wieder als vereinfachte Bausteingrafik dargestellt.

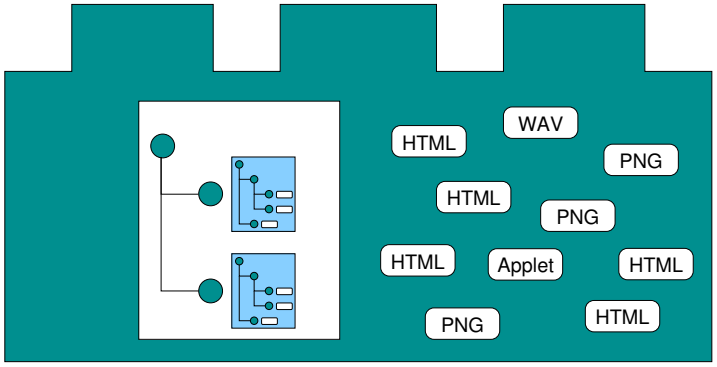


Abbildung 12.2: Baustein mit Submanifesten aus [Bungenstock04a]

Im Gegensatz zur vorherigen Abbildung setzt sich die hierarchische Struktur aus Submanifesten zusammen. Bei den Dateien hat sich gegenüber der ersten Variante nichts geändert, denn sie liegen zusammen auf einer physikalischen Ebene. Es gibt aber dennoch einen Unterschied, der den Prozess der Komposition betrifft. Wenn zwei Bausteine zusammengesetzt werden, kommt es zu einer Vermischung der Dateien auf Verzeichnisebene. So lange die Dateien unterschiedliche Namen haben, ist dieser Vorgang unkritisch. Doch sobald sie sich gleichen, müssen die Dateien mit unterschiedlichen Namen oder an verschiedenen Orten gespeichert werden. Letzteres kann über Verzeichnisse geregelt werden, die in *Content Packages* erlaubt sind. Diese Strategie führt zwar zu richtigen, aber möglicherweise nicht zu optimalen Ergebnissen. Es sei z.B. ein *Java Applet* angenommen, das über einen Konfigurationsmechanismus eine Vielzahl von Aufgaben und Tests ermöglicht. Aufgrund dieser Flexibilität ist es 2 MB groß und wird in fünf Bausteinen verwendet, die zu einer höheren Struktur kombiniert werden. Das Ergebnis ist ein *Content Package* mit einer Größe von mindestens 10 MB, obwohl eine Größe von ca. 2 MB möglich ist. Es kann sich folglich lohnen, die Dateien inhaltlich zu vergleichen, um solche Redundanzen zu vermeiden.

Diese Form der Komposition von Bausteinen ist einfach umzusetzen, weil sie dem Aufbau eines einzelnen Bausteins ähnelt. Lediglich Submanifeste und eine etwas umfangreichere Dateiverwaltung müssen integriert werden. Kritisch betrachtet, ähnelt dieser Ansatz aber nicht zusammengesetzten Bausteinen, denn für die Dekomposition von Kursen muss zuvor die interne Struktur analysiert werden. Aus einem Submanifest wird dann ein oberstes Manifest und ergibt zusammen mit allen Dateien einen neuen Baustein. Die ursprüngliche Form steht nicht mehr zur Verfügung, weil sie bei der Komposition verworfen wurde.

Um eine nähere Verbindung zu den Bausteinen und ihren Eigenschaften zu ermöglichen, wird nun die zweite Möglichkeit der Komposition vorgestellt. Anstatt die beteiligten *Content Packages* bei diesem Prozess aufzulösen, sollen sie lieber direkt als physikalische Ressourcen genutzt werden. Abbildung 12.3 zeigt die verschachtelten *Content Packages* in Form von Bausteinen und verdeutlicht den Unterschied zu der Lösung mit Submanifesten.

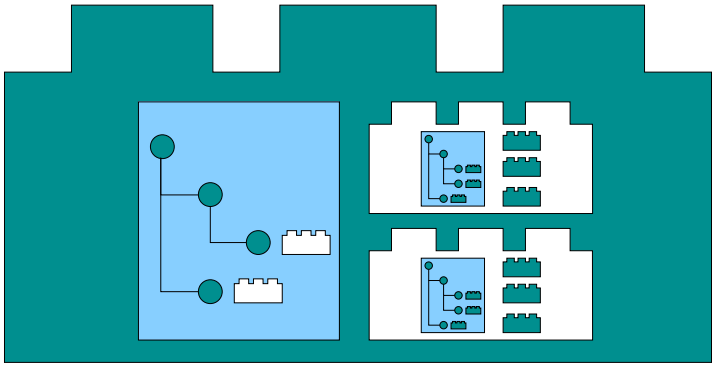


Abbildung 12.3: Verschachtelte Bausteine aus [Bungenstock04a]

Diese Kombination von Lernobjekten erscheint wesentlich intuitiver, zieht aber technische Konsequenzen nach sich, die nicht unterschätzt werden dürfen. Da es sich bei jedem *Content Package* um eine geschlossene physikalische Einheit handelt, ist der Aufwand für die Darstellung der Gesamtstruktur erhöht. Es ist nicht auf den ersten Blick ersichtlich, wie viele und welche Dateien in allen Bausteinen enthalten sind. Auch die Strukturbeschreibung liegt über mehrere Manifeste verteilt, die erst zusammengetragen werden müssen, bevor sie genutzt werden können. Im Sinne der Verständlichkeit und der Metaphern lohnt sich dieser Aufwand dennoch.

Doch wie steht es mit der Kompatibilität zu anderen Anwendungen? Sie ist ein wirkliches Problem, dass bereits im Entwurf angegangen werden muss. Andere Autorensysteme und Lernplattformen sind freilich nicht in der Lage, eine richtige Interpretation zu leisten. Nun ist aber genau der Austausch zwischen unterschiedlichen Systemen das Hauptargument für

den Einsatz der Standards gewesen und sollte nicht durch das Konzept verhindert werden. Diesem Manko kann durch Umwandlung begegnet werden, indem verschachtelte Bausteine zu einem einfachen *Content Package* konvertiert werden. Hierfür wird einfach der Prozess der Komposition nachgebildet, wie er bereits für Kurse mit Submanifesten beschrieben wurde. Werden die in den Abbildungen 3.8 und 3.9 dargestellten Regeln aus dem Abschnitt 3.5 auf die Submanifeste angewandt, können verschachtelte Bausteine sogar mit einem Hauptmanifest erzeugt werden. Dieses sehr einfache Format aus Abbildung 12.1 müssen alle Anwendungen interpretieren können, die sich als standardkompatibel ausgeben.

Eine Umsetzung der Bausteine und Kurse unter Einhaltung der Kompatibilität ist also konzeptionell möglich. Für die physikalische Speicherung der Daten innerhalb der *Content Packages* kann die Dateisystem-API aus dem Kapitel 11 herangezogen werden. Mit Hilfe der Vererbung entstehen neue Dateisysteme, die genau den Ansprüchen, insbesondere den Schwierigkeiten durch die mögliche Verschachtelung, gerecht werden. Letztendlich wird nur noch eine API für den Aufbau von Manifesten gebraucht, um die Umsetzung der Standards zu vervollständigen. Die nächsten beiden Abschnitte beschreiben die Herleitungen im Detail.

12.2 Physikalische Dateien

Wie in Abschnitt 3.5 beschrieben, unterscheidet die Spezifikation des *IMS Content Packaging* zwei Formen der physikalischen Datenhaltung: logische Verzeichnisse, *Package* genannt, und die Zusammenfassung in einer Datei, als *Package Interchange File* (PIF) bezeichnet. Wie gehabt wird der Begriff *Package* als Synonym für PIF genutzt, es sei denn, ein wesentlicher Unterschied soll herausgestellt werden. Da es sich bei dieser Unterscheidung eigentlich um ein technisches Detail handelt, soll der Zugriff über eine Schnittstelle, genauer gesagt eine Klasse, erfolgen. Für den Entwurf der Klassen ist es dennoch ungemein wichtig, diesen Unterschied zu berücksichtigen, um verschiedenen Implementationen und zukünftigen Entwicklungen gewachsen zu sein. Die Basisklassen für die folgenden Überlegungen sind die Klassen `VFSNode` (siehe Abbildung 11.7) und `VFS` (siehe Abbildung 11.9) aus Unterabschnitt 11.1.2.

Aus den Gemeinsamkeiten von *Package* und PIF kann eine Klasse entstehen. Doch welche Eigenschaften sind gleich, welche unterschiedlich? Besonders die PIFs können in ihrer Umsetzung stark variieren. Da die Spezifikation die unbedingte Unterstützung von RFC 1951 (ZIP) verlangt, soll sie als Referenzimplementation dienen. Eigentlich für die Archivierung und den Datenaustausch gedacht, weist dieses Format Schwächen auf, die sich auf die Umsetzung auswirken. So ist es beispielsweise nicht möglich, in eine bestehende Datei neue Dateien hinzuzufügen, sie zu entfernen oder in irgendeiner Form zu bearbeiten[1]. Als Konsequenz muss jedes PIF vor der Bearbeitung entpackt werden, z.B. in ein Verzeichnis eines anderen Dateisystems. Dieser Schritt entspricht einer Umwandlung von einem PIF zu einem *Package*, sodass, wenn er transparent erfolgt, lediglich ein Mechanismus für den Umgang mit *Packages* entwickelt werden muss. Eine abschließende Rückumwandlung bei PIFs, es wird der Vollständigkeit halber erwähnt, vollzieht sich genauso automatisch.

Die Idee für die Umsetzung dieser Funktionalität ist ein Dateisystem, das ein temporäres Verzeichnis erstellt, in dem das Dateisystem eines *Packages* vorübergehend gespeichert wird. Hierfür werden beim Öffnen alle Dateien umkopiert und nach Abschluss der Bearbeitung wieder zu einem Package zusammengesetzt. Eine interne Übersetzung der Pfade auf die temporären erfolgt über die Klasse `TempFSNode`, die in Abbildung 12.4 dargestellt ist.

Es wurden keine neuen Methoden hinzugefügt, sondern die abstrakten implementiert. Ein kurzes Beispiel soll ein besseres Verständnis der Funktionalität geben. Innerhalb eines *Packages* befinden sich zwei Dateien, ein Manifest in XML (`imsmanifest.xml`)und eine HTML-Seite (`index.html`). Das Dateisystem für temporäre Dateien, auf das gleich genauer eingegangen

[1] Ohne weiter auf Details eingehen zu wollen, erschließt sich diese Aussage aus der Natur der Kodierung mit einem Huffman-Code. Jegliche Änderung des Inhalts zieht eine Änderung am Alphabet nach sich und erfordert eine Umkodierung der anderen Inhalte.

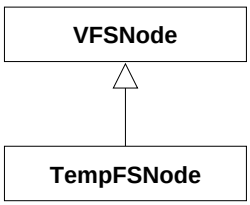


Abbildung 12.4: Klasse `TempFSNode`

wird, kopiert diese beiden Dateien in ein temporäres Verzeichnis mit dem Pfad `/var/tmp/pifs`, was den Anwendern/-innen nicht mitgeteilt wird. Sie benutzen in ihrer Anwendung die Pfade des *Packages*, also `/index.html` anstatt `/var/tmp/pifs/index.html`. Umgewandelt werden die Pfade in der Klasse `TempFSNode`.

Um nicht den Eindruck aufkommen zu lassen, dass dieser Mechanismus nur für PIFs benötigt wird, sei auf die Möglichkeit des Abbruchs hingewiesen. Alle ausgeführten Operationen müssen umkehrbar sein, wenn die Änderungen doch nicht gespeichert werden. Wurde die Bearbeitung auf dem Original ausgeführt, dann ist eine Herstellung des ursprünglichen Zustands schwierig. Deshalb ist es besser, auf einer Kopie zu arbeiten, die bei Bedarf zurückgespeichert wird. Doch bevor auf diese Operation näher eingegangen wird, soll noch das Dateisystem für temporäre Dateien behandelt werden. Abbildung 12.5 zeigt die Klasse `TempFS`, die zwei neue Methoden einführt.

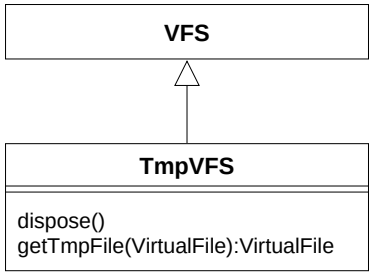


Abbildung 12.5: Klasse `TempFS`

Die Methode `dispose()` entfernt alle temporären Dateien und mit Hilfe von `getTmpfile()` lassen sich die lokalen Dateien auf die temporären physikalischen abbilden.

Weil die API allgemein gehalten wird und diese Klasse optimal für die Haltung temporärer Dateien ist, wird der Rückweg, die temporären Dateien in *Packages* zu speichern, in eine andere Klasse ausgelagert. Dies mag auf den ersten Blick nicht ersichtlich sein, aber durch die nächste Ebene, die lediglich eine allgemeine Schnittstelle für den Zugriff auf die Implementationen gestattet, wird der Entwurf klarer strukturiert. Bei der Speicherung müssen verschiedene Zustände berücksichtigt werden, die sich in den Methoden der Klasse `SavableFS` widerspiegeln. Weil sich keine strukturellen Veränderungen auf der Ebene der Dateien ergeben, muss für dieses Dateisystem keine eigenen Knoten-Klasse erstellt werden. Abbildung 12.6 zeigt das entsprechende Diagramm für das Dateisystem.

Entweder wird ein existierendes *Package* geöffnet, oder es wird neu erzeugt. Die Feststellung des initialen Ausgangspunkt erfolgt über die Methode `getFile()`. Wenn der Inhalt des Dateisystems die Kopie aus einer Datei ist, wird genau diese zurückgegeben. Andernfalls ist der Wert `null`. Als Vereinfachung für den Test in booleschen Ausdrücken wird das Prädikat `hasFile()` angeboten. Über die Methode `save()` wird, wenn das Prädikat den Wert „wahr“ liefert, der aktuelle Zustand des temporären Verzeichnisses in die Datei geschrieben. Hiernach kann mit dem Dateisystem normal weiter gearbeitet werden, bis die Methode `close()` aufgerufen wird, die endgültig alle belegten Ressourcen freigibt. Die eigentliche Speicherung erfolgt über die Methode `save(VirtualFile)`, der eine Datei als Ziel mitgegeben wird. Sie ist als abstrakt definiert und muss in den Unterklassen implementiert werden. Da bereits am Anfang

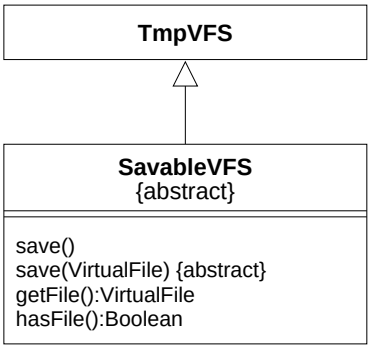


Abbildung 12.6: Klasse `SavableFS`

dieses Abschnittes festgestellt wurde, dass es nur zwei Formen der Speicherung gibt, sind in Abbildung 12.7 die Pendants abgebildet. Beide Klassen, `ZipFS` und `DirectoryFS`, bieten keine neue Methoden an, sondern implementieren die Speicherung.

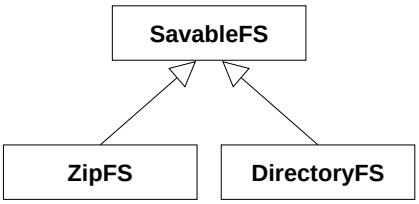


Abbildung 12.7: Die Unterklasse `ZipFS` und `DirectoryFS`

Es bleibt festzuhalten, dass über die Schnittstelle der Klasse `SavableFS` ein einheitlicher Zugriff auf verschiedene Typen von *Content Packages* möglich ist, der die Konstruktion von Baustein- und Kurs-Klassen sehr einfach hält. Auch die Handhabung verschachtelter Inhalte ist durch den orthogonalen Ansatz kein Problem mehr. Was noch fehlt, ist die Modellierung der Manifeste, die im folgenden Abschnitt behandelt wird.

12.3 Manifest

Das Manifest beschreibt die Struktur sowie Ressourcen der Bausteine und Kurse. Eine Modellierung dieses „Bauplans“ lässt sich leicht aus den Spezifikationen der Standards IMS CP und SCORM ableiten. Da die Manifeste für Bausteine eine echte Untermenge der Manifeste für Kurse sind, können beide über einen Entwurf dargestellt werden. Auch die marginalen Unterschiede zwischen den beiden Standards, die nur in zusätzlichen Attributen liegen, behindern dieses Vorhaben nicht. So wie es bereits *XML-Bindings* für Manifeste gibt, soll nun ein objektorientiertes Modell hinzukommen, das sich in dieser Arbeit **OO-Binding** nennt. Die Datenstruktur eines Manifests in Abbildung 3.7 ist ein guter Ausgangspunkt, um die folgenden Überlegungen besser nachvollziehen zu können.

In den *XML-Bindings* werden 10 Elemente mit ihren Attributen definiert. Für eine objektorientierte Darstellung ist es wichtig, Gemeinsamkeiten sowie Unterschiede deutlich herauszustellen und die Assoziationen untereinander zu benennen. Tabelle 12.1 listet alle Elemente sowie deren relevanten Eigenschaften auf.

Eltern-Elemente können andere Elemente als **Kind-Elemente** enthalten und **obligatorische Elemente** sind Elemente, die immer mit ihrem Eltern-Element bzw. als Wurzelelement im Manifest enthalten sein müssen. Nach UML wird diese Assoziation als Komposition bezeichnet, da die Existenzberechtigung obligatorischer Elemente vom Aggregat (dem Ganzen) abhängt. Ausnahmen sind die Elemente `manifest` und `item`, weil sie kontextabhängig sind. Als oberstes Element verkörpert `manifest` schlichtweg ein Manifest und ist obligatorisch ein-

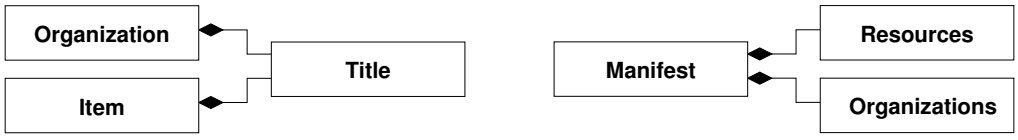
	Eltern-Element	Kind-Element	Obligatorisch	Bezeichner	Referenziert	XML:Base	Metadaten	Änderungen
`dependency`	○	●	○	○	●	○	○	●
`file`	○	●	○	○	○	○	●	●
`item`	●	●	◐	●	●	○	●	●
`manifest`	●	●	◐	●	○	●	●	●
`metadata`	●	●	○	○	○	○	○	●
`organization`	●	●	○	●	○	○	●	●
`organizations`	●	●	●	○	○	○	○	●
`resource`	●	●	○	●	○	●	●	●
`resources`	●	●	●	○	○	●	○	●
`title`	○	●	○	○	○	○	○	●

●: ja ○: nein ◐: beides

Tabelle 12.1: Gemeinsame Eigenschaften der Manifest-Elemente aus [Bungenstock04b]

zusetzen. Tritt es jedoch als Submanifest auf, ist es optional. Ähnlich verhält es sich mit dem `item`, denn in einem Element `organization` ist es obligatorisch und als Subitem optional. Manche Elemente sind mit **Bezeichnern** (ID) versehen, um sie von anderer Stelle aus **referenzieren** zu können. Diese IDs müssen innerhalb des gesamten Manifests eindeutig sein. Die Eigenschaft **XML:Base** ist für Kind-Elemente mit URLs als Attributwerten von Bedeutung, weil relative URLs immer gegen die nächste XML:Base aufgelöst werden. **Metadaten** sind zusätzliche Daten, wie sie in Kapitel 4 beschrieben sind. Mit der Eigenschaft **Änderungen** ist die Modifizierbarkeit der Elemente gemeint, die für alle zutrifft.

Bevor aus den Werten der Tabelle 12.1 die nötigen Klassen hergeleitet werden, können ein paar Elemente als relevante Klasse ausgeschlossen werden. Bei `title` handelt es sich mehr um ein Attribut als ein eigenständiges Element, sodass es durch eine einfache Zeichenkette dargestellt werden kann. Die Klasse `Title` in Abbildung 12.8(a) ist somit Bestandteil der Klassen `Organization` sowie `Item` und kann entfallen.



(a) Kompositionen mit `Title`

(b) Komposition mit `Resources` und `Organizations`

Abbildung 12.8: Strukturierte Adresse

Auch `resources` und `organizations` tragen keine komplexen Strukturen bei und können durch einfache Basistypen wie Array oder Liste realisiert werden. Abbildung 12.8(b) zeigt die mögliche Komposition, welche die Klassen `Resources` und `Organizations` überflüssig macht.

Aus Tabelle 12.1 ist ersichtlich, dass alle Elemente zwei Eigenschaften besitzen, die sich in einer gemeinsamen Basisklasse zusammenfassen lassen: Jedes Element kann als Kind-Element auftreten und ist modifizierbar. Um der hierarchischen Struktur gerecht zu werden, lautet ihr Name `HierarchicalElement`. Abbildung 12.9 zeigt die Basisklasse aller Elemente.

Die ersten drei Methoden sind für die Benachrichtigungen bei Veränderungen zuständig. Jede Klasse, die über Änderungen in einem Element in Kenntnis gesetzt werden möchte, muss das Interface `ModificationListener` implementieren und sich zur Laufzeit über die Methode `addListener()` als Objekt anmelden. Sobald ein Element eine Änderung erfährt, ruft

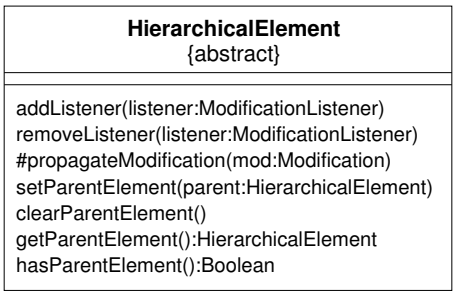


Abbildung 12.9: Klasse `HierarchicalElement`

es die Methode `propagateModification()` auf, die alle angemeldeten Objekte benachrichtigt. Abbildung 12.10 zeigt das zugehörige Sequenzdiagramm.

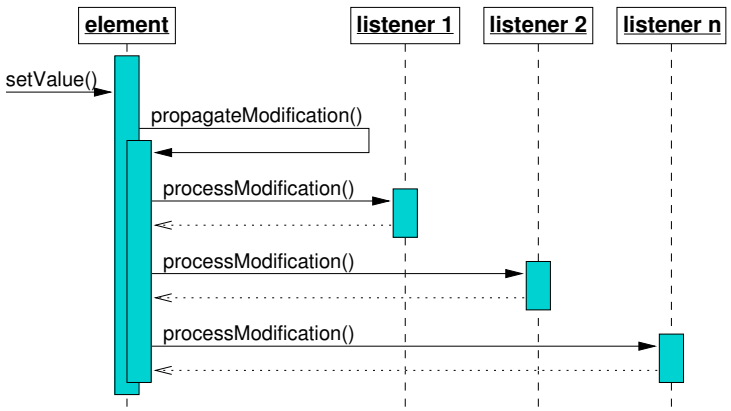


Abbildung 12.10: Sequenzdiagramm für den Benachrichtigungsmechanismus

Soll ein Objekt nicht weiter über Veränderungen benachrichtigt werden, kann es dies über die Methode `removeListener` bekannt geben. Die restlichen Methoden sind für das Elternelement zuständig, mit denen es gesetzt, gelöscht und abgefragt werden kann.

Die Metadaten werden selbstverständlich über die Klassen aus Abschnitt 11.2 verwaltet. Für den Zugriff über das Manifest wird die Klasse `MDElement` eingeführt, die eine Spezialisierung von `HierarchicalElement` ist. Über ihre Methoden können Metadatenstrukturen mit der Schnittstelle `MetaData` gesetzt und abgefragt werden, sodass keine direkte Verbindung zu einem bestimmten Standard besteht. Abbildung 12.11 zeigt das entsprechende Diagramm.

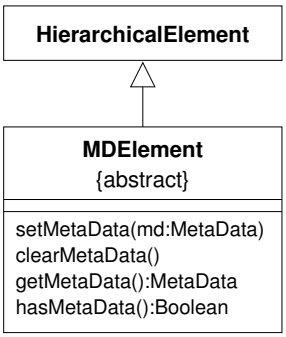


Abbildung 12.11: Klasse `MDElement`

Vier Elemente haben Bezeichner, über die sie eindeutig identifiziert werden können. Hierfür muss gewährleistet sein, dass jeder Bezeichner eindeutig ist und im Falle einer Referenzierung auch tatsächlich existiert. Da alle Elemente mit einem Bezeichner auch Metadaten haben

können, kann die Klasse `IDElement` direkt von `MDElement` erben. Abbildung 12.12 zeigt das zugehörige Klassendiagramm.

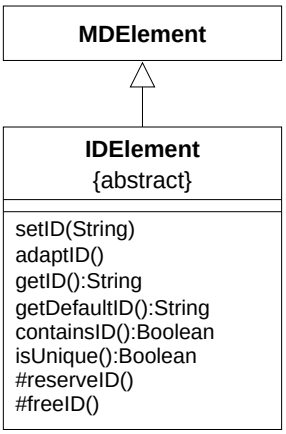


Abbildung 12.12: Klasse `IDElement`

Wenn ein Objekt mit den Eigenschaften der Klasse `IDElement` erzeugt wird, gibt es verschiedene Möglichkeiten, den Bezeichner zu bestimmen. Initial wird die Methode `getDefaultId()` im Konstruktor aufgerufen und mit `setID()` gesetzt. Letztere Methode kann auch nachträglich aufgerufen werden, um einen eigenen Bezeichner zu vergeben. Hierbei wird allerdings keine Konsistenzprüfung durchgeführt, sodass entweder durch das Prädikat `isUnique()` die Eindeutigkeit bestätigt wird oder gleich mit `adaptID()` eine automatische Anpassung erfolgt.

Die Überprüfung selbst ist keine triviale Angelegenheit, weil sie in einer hierarchischen Struktur erfolgt. Aus diesem Grund gibt es zwei Methoden, die bei der Reservierung und Freigabe von Bezeichnern unterstützend wirken. Mit `reserveID()` wird im gesamten Baum bekannt gegeben, dass ein Element einen bestimmten Bezeichner für sich beansprucht. Ob dies überhaupt möglich ist, kann mit `containsID()` vorab überprüft werden. Bei der Entfernung eines Elements wird über `freeID()` der Bezeichner freigegeben, damit er bei Bedarf neuen Elementen zur Verfügung steht.

Eine weitere Aufteilung der gemeinsamen Eigenschaften auf allgemeine Klassen lässt sich nicht sinnvoll weiterführen. Daher sollen nun die Klassen für die konkreten Manifest-Elemente hergeleitet werden. In umgekehrter Reihenfolge der entwickelten Basisklassen, also von `IDElement` zu `HierarchicalElement`, werden sie vorgestellt. Aus Tabelle 12.1 wird entnommen, dass es genau vier Elemente mit Bezeichnern gibt, nämlich `item`, `manifest`, `organization` und `resource`. Abbildung 12.13 zeigt das entsprechende Klassendiagramm.

Die einzelnen Klassen zeigen bei Weitem nicht alle Methoden, denn zu spezielle Funktionen und Attribute wurden übersichtshalber ausgelassen. Aus dieser Vereinfachung darf freilich nicht eine Irrelevanz für die Implementierung gefolgert werden. Im Wesentlichen verwalten die vier Klassen ihre Unterelemente, deren genauen Beziehungen sich gut in Abbildung 3.7 nachvollziehen lassen. Einige der Methoden bieten Zusatzfunktionen zur Umsetzung der Metaphern an. Beispielsweise integriert `flatten()` alle Submanifeste in das oberste Manifest, sodass die Umwandlung von verschachtelten Bausteinen zu flachen *Content Packages* vereinfacht wird.

Die Klasse `File` erbt direkt von `MDElement`, weil sie zwar Metadaten zu der jeweiligen Datei anbietet, aber keinen Bezeichner erhält. Abbildung 12.14 zeigt das Klassendiagramm.

Intern nutzt diese Klasse eine URL zum Adressieren einer physikalische Datei. Neben den Methoden zur Manipulation dieser Referenz bietet `File` das Prädikat `isLocal()` an, mit dem überprüft wird, ob die Datei Bestandteil des *Content Packages* ist. Diese Funktion ist besonders für die Datenhaltung von E-Learning-Inhalten wichtig, weil sie das Auffinden von Abhängigkeiten vereinfacht.

Das letzte Element ist `dependency` und dessen Klasse erbt direkt von der `Hierarchical-`

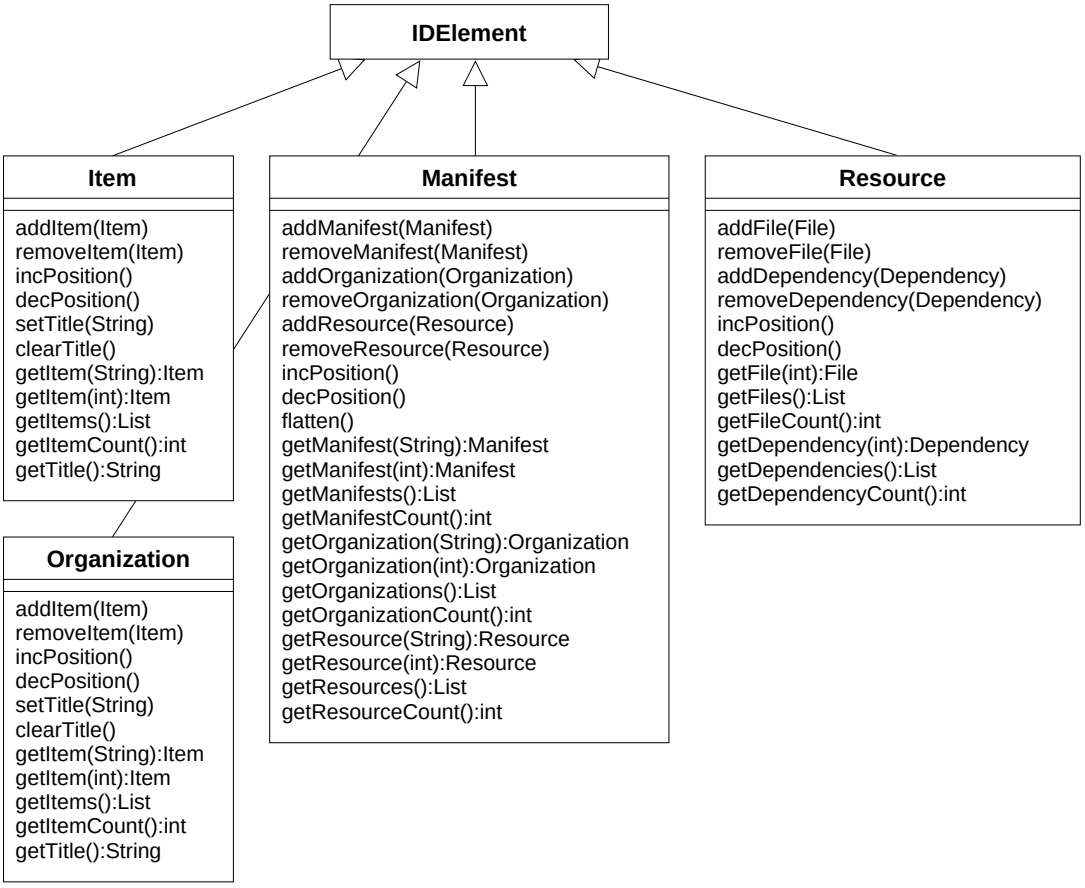


Abbildung 12.13: Die Klassen `Item`, `Manifest`, `Resource` und `Organization`

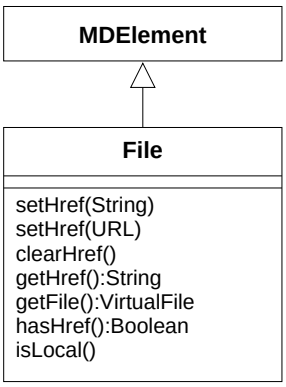


Abbildung 12.14: Klasse `File`

`Element`. Sie enthält lediglich eine Referenz auf ein anderes `Resource`-Objekt, das für eine ordentliche Ausführung benötigt wird. Abbildung 12.15 zeigt das Klassendiagramm.

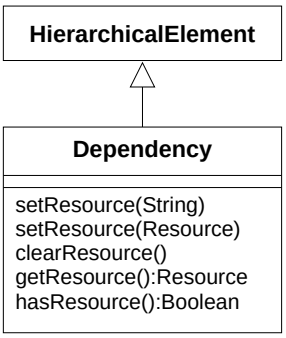


Abbildung 12.15: Klasse `Dependency`

Nun ist das *OO-Binding* vollständig und kann zur Erstellung und Bearbeitung von Manifesten genutzt werden. Die Klassenhierarchie in Abbildung 12.16 fasst die Ergebnisse dieses Abschnitts zusammen.

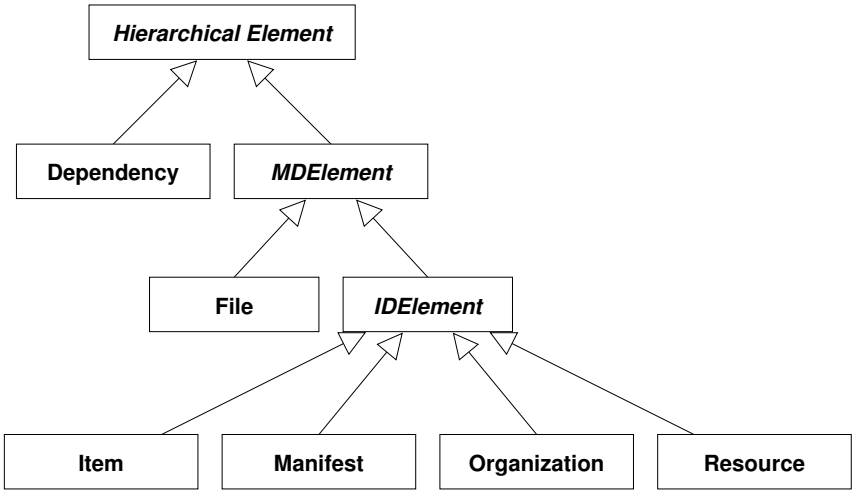


Abbildung 12.16: Klassenhierarchie der Manifest-Elemente

12.4 Content Package

Mit den Klassen für die Speicherung der physikalischen Dateien und des Manifests sind die Grundlagen für die Modellierung des *Content Packages* gelegt. Weil bereits auf dieser Ebene die Standardkompatibilität gewährleistet ist, fällt die folgende Klassenstruktur übersichtlich und kompakt aus. Im Grunde genommen sind die Klassen `SavableFS` und `Manifest` Aggregate in einer kapselnden Klasse, die hier den Namen `ContentPackage` trägt. Abbildung 12.17 zeigt die Komposition und die relevanten Methoden.

Die Schnittstelle setzt sich im Wesentlichen aus Methoden zusammen, die Aufrufe an Dateisystem und Manifest weiterleiten. Lediglich die Unterscheidung, ob es sich bei einem *Content Package* um ein logisches Verzeichnis oder eine gepackte Datei handelt, wird in dieser Klasse verwaltet. Hierzu wird im Konstruktor der übergebene Pfad überprüft und das jeweilige Objekt erzeugt. Ist dies bei einem Verzeichnis noch recht einfach, die Klasse `VirtualFile` bietet entsprechende Methoden an, kann die Unterscheidung zwischen ZIP-, CAB- und anderen Dateien aufwendiger sein. Das folgende Verfahren ist recht simpel und kann in der Implementierung durch ein effizienteres ersetzt werden. In der Regel erkennen die Klassen für das

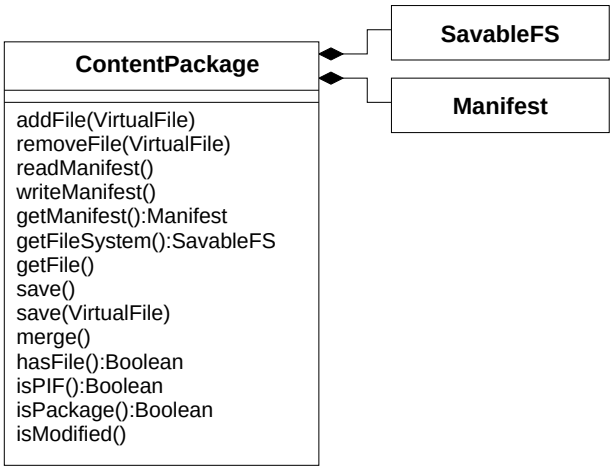


Abbildung 12.17: Klasse `ContentPackage`

Einlesen eines bestimmten Dateityps, ob sie die Datei ordnungsgemäß verarbeiten können. Ist dies nicht der Fall, werfen sie eine Ausnahme (*Exception*), die abgefangen werden muss. Der Reihe nach werden alle verfügbaren Klassen zum Lesen eines *Content Packages* aufgerufen, bis die Operation erfolgreich durchgeführt wurde. Schlugen alle Aufrufe fehl, dann wird das Format nicht unterstützt oder die Datei ist fehlerhaft. Dieser Mechanismus kann noch ein wenig verfeinert werden, indem z.B. die Dateiendungen überprüft werden.

Eine sehr wichtige Funktion für die Umsetzung der Metaphern stellt die Methode `merge()` bereit, die eine Konvertierung von verschachtelten *Content Packages* zu einem mit mehreren Submanifesten durchführt. Dies entspricht der Umwandlung eines Kurses aus Abbildung 12.3 zu einem Kurs aus Abbildung 12.2. Mit der Methode `flatten()` aus dem vorherigen Abschnitt (siehe Abbildung 12.13) können die Submanifeste noch zu einem Manifest zusammengefasst werden, sodass Kurse wie aus Abbildung 12.1 entstehen, die auch einfache Systeme unterstützen.

Mit der Klasse *ContentPackage* können modulare E-Learning-Inhalte in standardkompatiblen Formaten erstellt und bearbeitet werden. Jetzt gilt es, die Metapher Baustein explizit anzuwenden, um die gewünschte Modellierung zu erhalten. Abbildung 12.18 illustriert, wie die Umsetzung eines Bausteins über Vererbung realisiert ist.

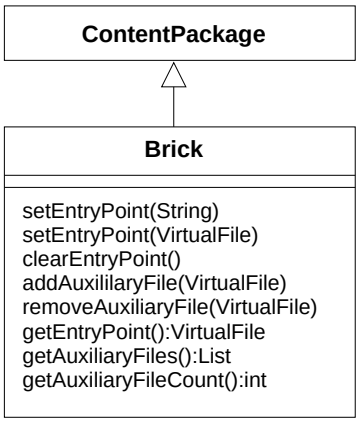


Abbildung 12.18: Klasse `Brick`

Ein Baustein ist im Kontext dieser Arbeit die kleinste technische Einheit, die von der Infrastruktur für modulare E-Learning-Inhalte verarbeitet wird. Er enthält eine Datei als Einstiegspunkt und mehrere zugehörige Dateien, die zur Darstellung notwendig sind. Die Schnittstelle der Klasse `Brick` bietet für die Erstellung und Bearbeitung verschiedene Methoden an,

die intern auf die Methoden der Klasse `ContentPackage` abgebildet werden. Es wird somit keine echte Funktionalität hinzugefügt, sondern der Blick aufs Wesentliche konzentriert. Anstatt beliebige Dateien in das *Content Package* zu kopieren, wird zwischen Einstiegspunkt und Hilfsdateien unterschieden. Bei der Implementierung kann auch darauf geachtet werden, dass alle anderen Elemente, wie z.B. Organisationen, ausgeschaltet sind. Letztendlich besitzt die Klasse `Brick` eine überschaubare Schnittstelle und die technischen Details sind in den Basisklassen gekapselt.

Ein Kurs ist rekursiv definiert und beinhaltet andere Kurse und Bausteine. Diese flexible Definition erlaubt den Aufbau beliebiger Strukturen, führt aber bei der Schnittstelle zu einem komplexeren Aufbau. Nicht alle Funktionen lassen sich direkt über eine Klasse steuern, weil auch hierarchische Datenstrukturen verwaltet werden. Dies ändert freilich nichts an der Vererbungshierarchie, wie Abbildung 12.19 verdeutlicht.

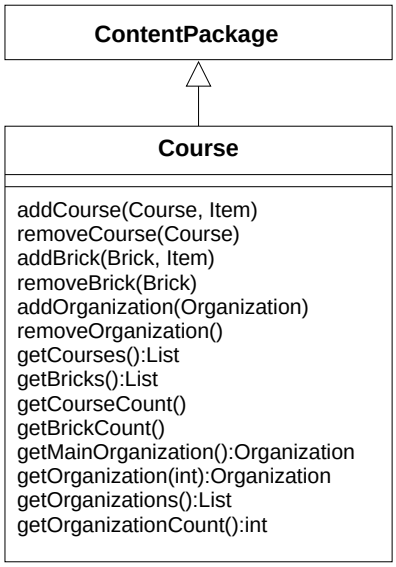


Abbildung 12.19: Klasse `Course`

Die Methoden der Klasse `Course` unterstützen das Hinzufügen und Entfernen von Bausteinen und Kursen. Wie schon bei der Klasse `Brick` ist die Schnittstelle eine spezialisierte Sicht auf ein *Content Package* mit der Einschränkung, dass es nur Bausteine und Kurse anstatt physikalischer Dateien gibt. Abschließend zeigt Abbildung 12.20 die gesamte Klassenhierarchie für *Content Packages*.

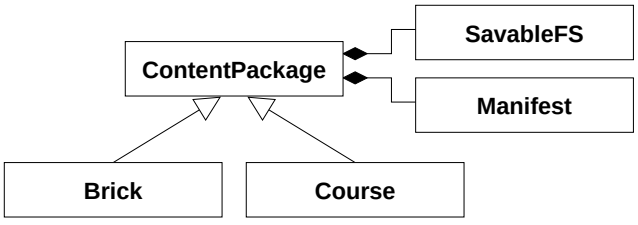
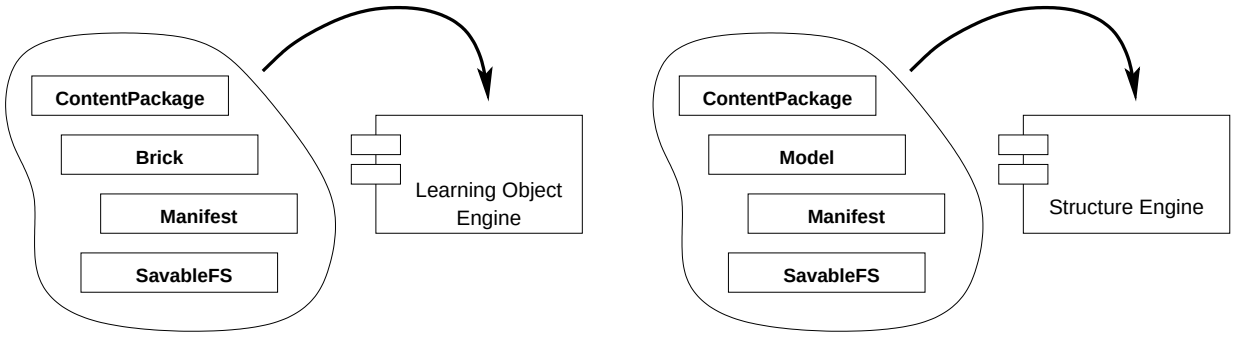


Abbildung 12.20: Klassenhierarchie der *Content Packages*

Mit den Klassen `Brick` und `Course` sind die letzten Klassen beschrieben, die für eine Komponentenbildung notwendig sind. Sie sind es, deren Funktionen über die Schnittstellen der Komponenten `Learning Object Engine` (siehe Abbildung 10.11) und `Structure Engine` (siehe Abbildung 10.12) angesprochen werden. In den Abbildungen 12.21(a) und 12.21(b) sind die jeweiligen Prozess zu sehen, die wie gehabt nur ausgewählte Klassen anzeigen.

Der wesentliche Unterschied zu den vorhergegangen Komponentenbildungen ist die Wiederverwendung einer ganzen Reihe von Klassen in zwei Komponenten. Dieser Sachverhalt



(a) Bildung der Komponente `Learning Object Engine`

(b) Bildung der Komponente `Structure Engine`

Abbildung 12.21: Komponentenbildung

verdeutlicht auf praktische Weise, warum es wenig ratsam ist, Komponenten zur physikalischen Gruppierung einzusetzen. In diesem Fall träte eine unerwünschte Redundanz auf, die fachlich motiviert wäre, aber technisch nicht nötig ist.

Kapitel 13

Rahmenwerk

Im zweiten Teil der Arbeit wurde am Anfang ein fachliches Modell für den Umgang mit modularen E-Learning-Inhalten aufgestellt, das am Ende in ein weit technischeres überführt wurde. Dieser Prozess ist noch nicht vollends abgeschlossen, denn es fehlen noch die Kompositionen der entwickelten Basiskomponenten zu vollständigen Werkzeugen der einzelnen Rollen. Zudem soll noch eine Zusammenfassung zu Libraries bzw. Paketen erfolgen, wie es in Kapitel 11 einleitend angeführt wurde. Erst in dieser Konstellation mit einer geeigneten Beschreibung der API lässt sich das entwickelte Gesamtmodell als Rahmenwerk nutzen.

Ohne eine genaue Definition eines Rahmenwerks zu geben und alle Aspekte dieses Themas auszuloten, soll kurz eine Unterscheidung zwischen Rahmenwerk und Library gegeben werden. Die wesentlichen Unterschiede lassen sich in der Nutzung ausmachen. Geht es bei Libraries in erster Linie um die Zusammenfassung kohärenter Klassen, die nach außen hin eine einheitliche Schnittstelle bilden, steht beim Rahmenwerk die Erweiterung um eigene Funktionen im Vordergrund. Nach dem Hollywood-Prinzip „Don't call us, we call you" [Sweet85] werden die hinzugefügten Unterklassen über Nachrichten des Rahmenwerks aufgerufen. Dieses Prinzip wird in allen Klassen der erstellten Komponenten eingehalten. So lassen sich z.B. eigene Dateisysteme, Metadaten und multimediale Ergänzungen auf einfache Art hinzufügen.

Obwohl die Begriffe Komponente, Library und Rahmenwerk im ersten Anschein widersprüchlich erscheinen, sind alle drei Betrachtungsweisen mit dem vorgestellten Entwurf realisierbar. Dies ist nur möglich, weil nicht dogmatisch Definitionen befolgt werden, die in der Praxis wenig Bedeutung haben und kontraproduktiv sind. Im nächsten Teil dieser Arbeit wird eine Implementation vorgestellt, die auf die verschiedenen Weisen eingesetzt werden kann. Die fertigen Komponenten lassen sich mit wenig Aufwand zu vollständigen Anwendungen zusammensetzen oder ergänzen Eigenentwicklungen um spezielle Funktionalitäten. Wird mehr Kontrolle gewünscht, lassen sich Teile der API wie eine Library ansprechen, mit denen sich umfangreiche Eigenentwicklungen hochziehen lassen. Erst wenn spezielle Erweiterungen benötigt werden, die entweder zu exotisch waren, um sie in die Implementierung aufzunehmen, oder sich zu einem späteren Zeitpunkt etablierten, kommt der Charakter eines Rahmenwerks zum Vorschein. Es sei an dieser Stelle abermals darauf hingewiesen, dass der bisherige Entwurf unabhängig von jeglicher Programmiersprache ist.

Im nächsten Abschnitt werden nun die letzten drei Komponenten der Entwurfsphase erstellt. Dieser Prozess entspricht dem Muster der Erstellung einer **Fassade**, wie in Abbildung 13.1 illustriert ist.

Diese Architektur verdeckt die Kommunikation und Abhängigkeiten unter den einzelnen Teilkomponenten und bietet eine vereinfachte, einheitliche Schnittstelle nach außen. Die nun entstehenden Komponenten bieten somit keine neue Funktionalität, wodurch der Erstellungsprozess sehr einfach gehalten ist.

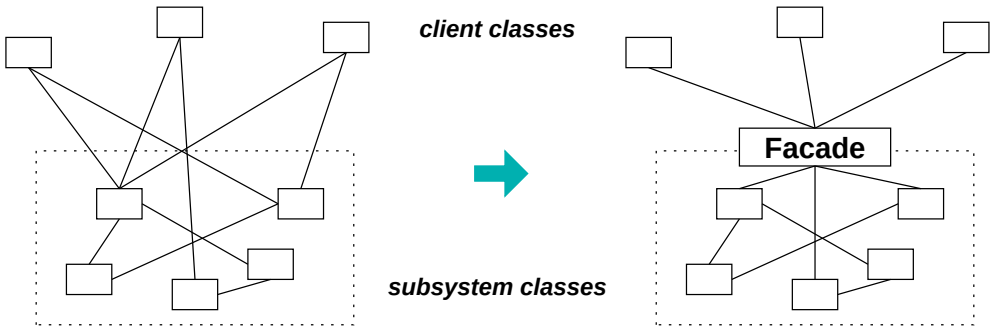


Abbildung 13.1: Das Muster *Fassade* [Gamma95, S. 185]

13.1 Zusammengesetzte Komponenten

Nach Tabelle 10.1 fehlen im Bereich des Entwurfs noch die zwei zusammengesetzten Komponenten `Learning Object Development` (siehe Abbildung 10.11) und `Structure Development` (siehe Abbildung 10.12). Die Komponente `Publishing Environment` (siehe Abbildung 10.13) wird, wie in der Tabelle angegeben, in anderen Arbeiten ausführlich beschrieben und soll an dieser Stelle als gegeben betrachtet sein.

Die Rolle *Developer* benutzt die Komponenten `Multimedia Environment`, `Import Engine` und `Learning Object Engine` zur Erstellung sowie Bearbeitung modularer E-Learning-Inhalte. Bis auf die `Import Engine` wurde in den letzten Kapiteln die Herleitung und Funktionalität dieser Komponenten beschrieben, sodass sie keiner weiteren Erklärung bedürfen. Auch ihr Zusammenspiel ist nicht sonderlich kompliziert. Grundlage für den Umgang mit standardkompatiblen Lernobjekten ist die `Learning Object Engine`, sei es nun beim Import fremder Inhalte oder der Bearbeitung enthaltender Multimedia-Dateien. Letztendlich müssen für das Zusammenspiel ein paar Initialisierungen und Abhängigkeiten bei den gegenseitigen Aufrufen beachtet werden, die von einer neuen Klasse übernommen werden. Ihre Schnittstelle bietet weitestgehend die gleiche Funktionalität wie die Einzelkomponenten an, weshalb auf eine ausführliche Beschreibung verzichtet wird. Die wesentliche Struktur ist Abbildung 13.2 zu entnehmen.

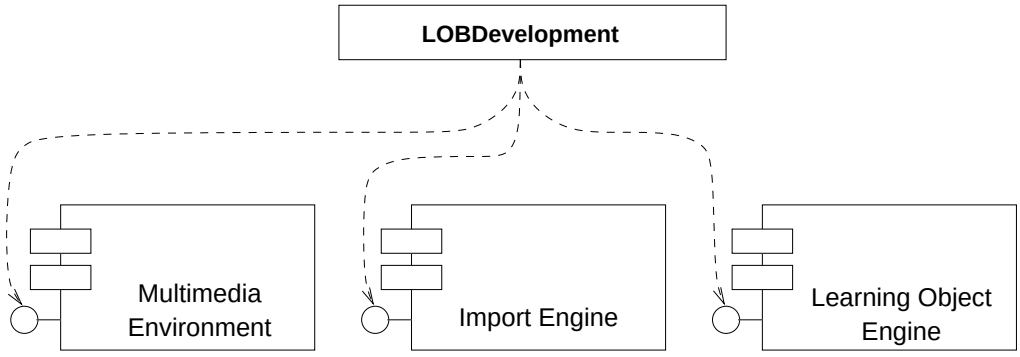


Abbildung 13.2: Bildung der Komponente `LOBDevelopment`

Mit der Klasse `LOBDevelopment` bekommen die Entwickler/-innen eine einheitliche Schnittstelle an die Hand, mit der sich auf einfache Weise eigene Erweiterungen, wie z.B. eine grafische Repräsentation oder ein Kommandozeilenprogramm, erstellen lassen. Die bisher eingesetzte Komponentenbildung, wie z.B. in Abbildung 12.21 dargestellt, wird aufgrund der Trivialität des Prozesses — schließlich wird nur die Klasse `LOBDevelopment` hinzugefügt — an dieser Stelle nicht angeführt. Das Endresultat ist aus Abbildung 10.11 bekannt.

Auch bei den Tätigkeiten der Rolle *Composer* werden hauptsächlich drei Komponenten eingesetzt, namentlich als `Import Engine`, `Search Engine` und `Structure Engine` vorgestellt.

Letztere ist die zentrale Komponente, beschrieben im vorherigen Kapitel. Egal ob nun fremde Inhalte importiert werden, oder eine Aufarbeitung bestehender Lerninhalte für die Suche erfolgt. Es wird stets der Zugriff auf die internen Strukturen und die inhaltlichen Zusammenhänge benötigt. Die Komponente *CBK-Management-Application* aus der Diplomarbeit [Vollmann04] wird als `Search Engine` verwendet, die nach dem Verfahren *Case-Based Reasoning* (CBR) arbeitet [Kolodner93; Lenz98]. Abbildung 13.3 zeigt den inneren Aufbau. Wer Interesse an der umfangreichen Schnittstelle hat, findet in der Diplomarbeit eine ausführliche Beschreibung.

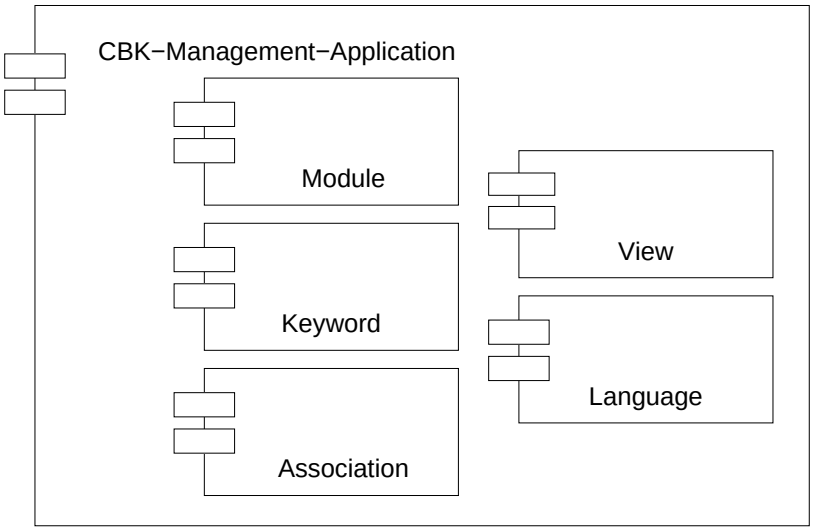


Abbildung 13.3: Aufbau der Komponente *CBK-Management-Application* nach [Vollmann04, S. 128]

Die Initialisierungen und Abhängigkeiten unter den Komponenten werden wieder über das Entwurfsmuster *Fassade* gekapselt, um einen einfachen Zugriff zu ermöglichen. Abbildung 13.4 zeigt die hierfür eingeführte Klasse. `StructureDevelopment`.

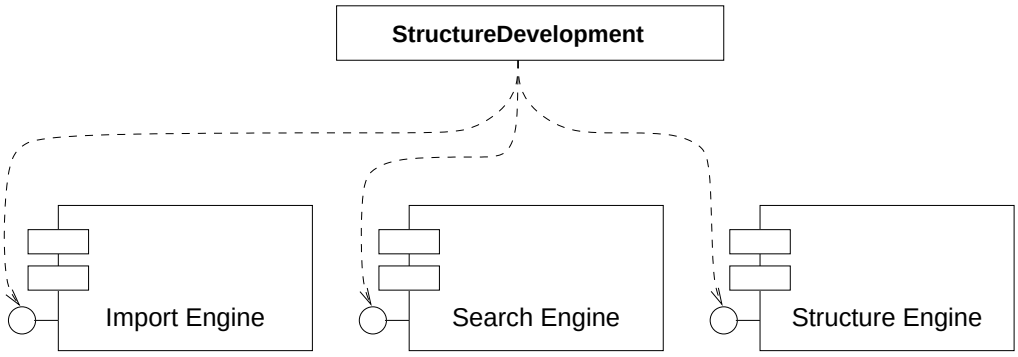


Abbildung 13.4: Bildung der Komponente `StructureDevelopment`

Auf die Darstellung einer expliziten Komponentenbildung soll wiederum verzichtet werden, denn auch hier ist das Endresultat aus Abbildung 10.12 bekannt.

Nun sind alle Komponenten erstellt, mit denen größere oder kleinere Aufgaben im Bereich modularer E-Learning-Inhalte übernommen werden können. Es lassen sich Lernobjekte als Bausteine erstellen sowie bearbeiten, mit anderen Lernobjekten zu Kursen zusammensetzen und in jedes beliebige Ausgabeformat überführen. Alle beteiligten Funktionen stehen in den drei Komponenten `Learning Object Development`, `Structure Development` und `Publishing Environment` zur Verfügung, doch der Zusammenhang zwischen ihnen fehlt. Mit Hilfe einer Komponente soll quasi die Funktionalität einer vollständigen Applikation angeboten wer-

den. Es wird wieder nach dem Muster *Fassade* verfahren, dessen Resultat in Abbildung 13.5 zu sehen ist.

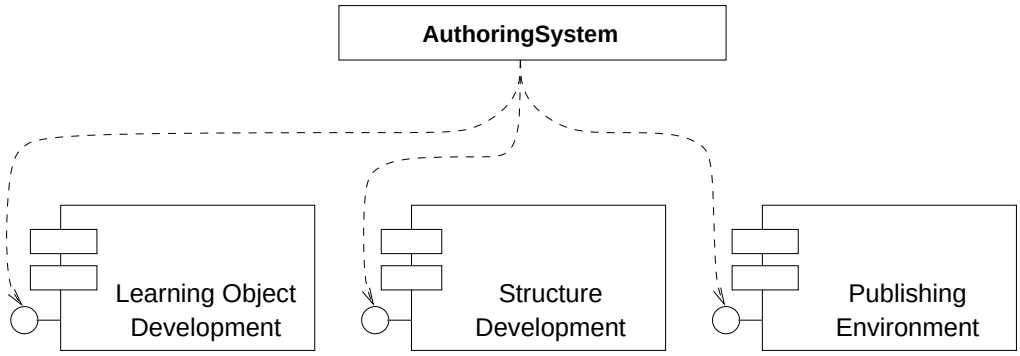


Abbildung 13.5: Bildung der Komponente `AuthoringSystem`

Die Klasse `AuthoringSystem` kapselt die Initialisierungen und kümmert sich um Abhängigkeiten. Zusätzlich erlaubt sie die Steuerung der Konfiguration über Umgebungsvariablen und spezielle Dateien. Auf diese Weise lässt sich das Verhalten der Anwendung ohne Programmierung steuern, sodass im optimalen Fall für den Einsatz dieser Komponente keine Entwicklungsarbeit notwendig ist. Freilich kann es sich hierbei nur um rudimentäre Tätigkeiten ohne Interaktion mit den Anwendern/-innen handeln.

Der Entwurf ist jetzt so weit, dass die nächsten Entwicklungsschritte nur noch in Abhängigkeit von Programmiersprachen, Libraries und Betriebssystemen sinnvoll vorangetrieben werden können. Hier endet nun die kreative Arbeit und geht über in Tätigkeiten, die zunehmend von konkreten Erfahrungen geprägt sind. So wird beispielsweise Wissen benötigt, wie sich grafische Oberflächen am elegantesten zusammenstellen lassen, wie Ressourcen des Rechners möglichst effizient genutzt werden und wie sich Programmieraufwand reduzieren lässt. Diese Überlegungen passen nicht in einen unabhängigen Entwurf, weshalb nun zur Implementierung übergegangen wird.

Teil III

Implementierung

Kapitel 14

Baukasten

Im vorherigen Teil dieser Arbeit wurden die fachlichen Klassen und Komponenten entwickelt, die für die technische Kodierung und Bearbeitung modularer E-Learning-Inhalte benötigt werden. Sie sind weitestgehend unabhängig von jeglichen Programmiersprachen und sonstigen Implementierungsdetails. Aus Sicht der Entwickler/-innen ist eine API entstanden, die eine gute Basis für die Erstellung eigener Anwendungen darstellt. In Anbetracht der Zielsetzung dieser Arbeit muss aber noch ein Schritt weitergegangen werden, denn es soll eine komplette Infrastruktur geschaffen werden, nicht nur ein konzeptioneller Entwurf. Was also fehlt, ist die grafische Darstellung (*View*), um das MVC-Muster zu vervollständigen. Sie ist stark von der Implementierung abhängig und deshalb in diesem Teil der Arbeit untergebracht, obwohl auch hier überwiegend konzeptionelle Überlegungen angeführt werden.

Da bis jetzt lediglich die Daten modelliert sind, nämlich Bausteine, Kurse und ihre Metadaten, sollen nun die passenden Werkzeuge entwickelt werden. Hierbei handelt es sich um die Komponenten aus Abschnitt 10.2, die nun zu einer prototypischen Anwendung zusammengesetzt werden.

Für die Programmiersprache *Java* gibt es verschiedene Libraries, die bei der Erstellung grafischer Oberflächen helfen. Implementierungen der *Virtual Machine* von *Java* unterstützen in der Regel die Standard-Libraries *Abstract Window Toolkit* (AWT) und *Swing*. Sie müssen im Gegensatz zu den Alternativen, wie z.B. das *Standard Widget Toolkit* (SWT) des Projekts *Eclipse*[1], nicht zusätzlich installiert werden. Das AWT ist die älteste Library und bietet wenig Komfort, da lediglich wenige Standardkomponenten direkt angeboten werden. Bäume, Tabellen und weitere komplexere *Widgets* stehen erst in *Swing* zur Verfügung, das intern auf dem AWT aufbaut. Im Gegensatz zu AWT und SWT sind die *Widgets* unabhängig vom Betriebssystem, da sie als *Lightweight Components* realisiert sind, sich also selbst in eine vorgegebene Fläche zeichnen. *Heavyweight Components* basieren hingegen auf den *Widgets* nativer Libraries oder dem Betriebssystem und benötigen unter Umständen mehr Ressourcen zur Ausführung.

Die Entscheidung in dieser Arbeit fällt zu Gunsten von *Swing* aus, weil es einen Kompromiss zwischen der Verfügbarkeit bei Standardinstallationen und der Anzahl von *Widgets* darstellt. Das AWT steht wegen seines Alters und des wenig durchdachten Entwurfs — die Entwickler beteuern selbst, dass es am Zeitdruck bis zur ersten Veröffentlichung von *Java* lag — außen vor. Bei dem SWT sieht es mit der Verfügbarkeit und der Stabilität heute wesentlich besser aus, als zu Beginn des Projekts mαth-kit, sodass Der Vorzug von *Swing* historisch bedingt ist.

In den folgenden Abschnitten werden die grafischen Komponenten hergeleitet, mit denen Bausteine sowie Kurse erstellt, bearbeitet und konvertiert werden. Wie in dem Teil Entwurf, werden erst die Basisfunktionen Dateizugriff und Metadaten angegangen, um anschließend die Komponenten für die Rollen *Developer*, *Composer* und *Publisher* fertig zu stellen.

[1] *Eclipse* ist eine offene flexible Plattform für die Integration von Entwicklungswerkzeugen. Die grafische Oberfläche basiert auf dem SWT. Mehr Details finden sich unter `http://www.eclipse.org` (29.10.05)

14.1 Script-Steuerung

Für eine direkte Ansteuerung ohne aufwändige Programmierung soll eine Komponente zur Script-Ansteuerung angeboten werden. Mit ihr lassen sich kleinere Routinetätigkeiten elegant erledigen, ohne jedes Mal aufwändig programmieren zu müssen. Für die Anwender/-innen bieten Script-Sprachen den idealen Kompromiss zwischen direkter Kontrolle und komplizierten Aufrufen von Komponenten. Die Position der Komponente `Scripting Environment` im Autorensystem kann in Abbildung 10.17 nachvollzogen werden.

Java ist eine stark typisierte Programmiersprache, was nichts anderes bedeutet, als dass der Quellcode übersetzt wird und während dieses Prozesses einer Prüfung der Typkonsistenz erfolgt. Passen irgendwelche Datentypen nicht zusammen, wird die Übersetzung mit einer entsprechenden Fehlermeldung abgebrochen. Dieses Konzept ist freilich schwer mit Script-Sprachen zu kombinieren, da sie in der Regel erst zur Laufzeit interpretiert werden. Es ist z.B. durchaus üblich, dass eine Variable im gleichen Geltungsbereich verschiedene Typen annimmt. In *Java* ist dies nicht möglich, doch wie lassen sich diese beiden Welten vereinen?

Klassische Script-Sprachen wie TCL[2], Perl[3] oder gar PHP[4] sind imperativ und kommen für diese Aufgaben freilich nicht in Frage, denn sie lassen sich nur sehr schwer oder gar nicht mit *Java* verbinden. Eine objektorientierte Script-Sprache wie Python[5] ist da wesentlich besser geeignet und es gibt interessante Implementierungen auf dem Markt. Herausragend ist z.B. der Interpreter *Jython*[6], der vollständig in *Java* umgesetzt wurde und eine nahtlose Verbindung zwischen Python und *Java* schafft. Aus technischer Sicht scheint dieser Ansatz eine gangbare Lösung zu sein.

Bei genauer Betrachtung von Python zeigt sich, dass es sich um eine vollständige Programmiersprache handelt, deren Merkmale z.B. Module, Klassen, *Exceptions* und höhere Datentypen sind. Nun unterscheidet sich die Syntax zu *Java* teilweise wesentlich und wer Python nicht kennt, hat einen gewissen Lernaufwand zu leisten. Bei den vielen Projekttreffen von mαth-kit hat sich deutlich herausgestellt, dass der größte Teil der Partner/-innen im Umgang mit *Java* vertraut ist bzw. sich in diese Richtung weiterbildet. Die Bereitschaft, eine weitere Programmiersprache zu erlernen, war verständlicher Weise gering. Aus diesem Grund ist die Entscheidung auf die *BeanShell*[7] gefallen, einen Interpreter speziell für *Java*. Die Syntax dieser Script-Sprache ist stark an *Java* angelehnt, wie das Beispiel in Abbildung 14.1 zeigt.

Es wird ein neuer Baustein erzeugt und der Bezeichner der Ressource ausgegeben. Weil die *BeanShell* kompakt ist und bereits mit einer eigenen grafischen Oberfläche ausgestattet ist, müssen keine weiteren Entwicklungsschritte vorgenommen werden, um die Komponente *Scripting Environment* aus Abbildung 10.17 zu erstellen. Lediglich bei der Initialisierung der *BeanShell* werden ein paar nützliche Objekte in den *Scope*[8] geladen, wie z.B. das spezielle Dateisystem. Dies reduziert den Aufwand für neu entwickelte Scripte, weil alle wichtigen Komponenten im direkten Zugriff vorliegen.

14.2 Grafische Basiskomponenten

In Kapitel 11 wurden die funktionalen Klassen für den Zugriff auf Dateien und Metadaten definiert. Nun werden sie um eine grafische Repräsentation ergänzt, um sie als Komponenten leichter wieder verwenden zu können. Bei der Darstellung eines Dateisystems gibt es zwei wesentliche Ansätze mit kleinen Varianten. Entweder erfolgt die Anzeige als Baum oder Liste, was verschiedene Vor- und Nachteile mit sich bringt. Beim Baum profitieren die Anwender/-

[2] `http://www.tcl.tk` (29.10.05)
[3] `http://www.perl.org` (29.10.05)
[4] `http://www.php.net` (29.10.05)
[5] `http://www.python.org` (29.10.05)
[6] `http://www.jython.org` (29.10.05)
[7] `http://www.beanshell.org` (29.10.05)
[8] Mit *Scope* wird der Bereich bezeichnet, in dem eine Variable definiert ist.

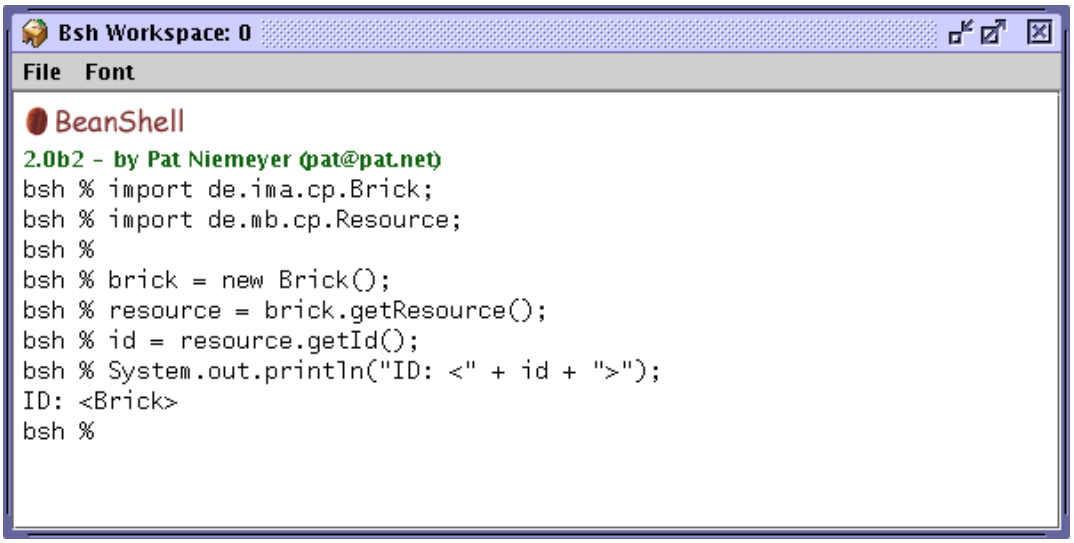


Abbildung 14.1: Screenshot der *BeanShell*

innen von einer guten Gesamtübersicht und sie erreichen schnell die gewünschten Dateien, auch wenn mehrere Verzeichnisse gleichzeitig geöffnet sind. Diese Datenfülle gereicht der Geschwindigkeit dieser Darstellung aber zum Nachteil, denn es müssen einige Informationen aus dem Dateisystem gesammelt werden. Angefangen bei der Unterscheidung von Dateien und Verzeichnissen, über Dateiattribute bis hin zu den grafischen Icons erhöhen viele Operationen die Belastung. Hinzu kommt ein Überwachungsmechanismus zur Aktualisierung der Darstellung, wenn von Dritten Änderungen im Dateisystem erfolgen. Auf dem lokalen Dateisystem des Betriebssystems mag eine adäquate Geschwindigkeit noch gegeben sein, aber spätestens bei einem Dateisystem im Netzwerk oder tief verschachtelten *Content Packages* sind massive Verzögerungen möglich.

Die Darstellung als Liste ist von diesen Probleme weniger betroffen, weil lediglich der Inhalt eines Verzeichnisses angezeigt wird. Die Anzahl der Dateien und Verzeichnisse hält sich somit in Grenzen. Bei der gleichzeitigen Arbeit mit vielen Verzeichnissen wirkt sich die Liste nachteilig aus, denn entweder wird oft die Ansicht gewechselt oder es werden mehrere Listen nebeneinander geöffnet. Es sei an dieser Stelle nicht verschwiegen, dass bei sehr vielen Dateien in einem Verzeichnis auch die Liste vor Verzögerungen nicht gefeit ist.

Die Darstellung von Bäumen und Listen lässt sich beschleunigen, indem sie in mehrere Schritte aufgeteilt wird. Zunächst werden alle Namen der Dateien und Verzeichnisse abgefragt, um sie danach sofort anzuzeigen. Danach werden die Einträge von einem Prozess oder Thread im Hintergrund um weitere Attribute ergänzt. Der Vorteil dieser „dynamischen“ Darstellung liegt auf der Hand: Die Anwender/-innen bekommen den Inhalt des Dateisystems schneller angezeigt und haben sofort Zugriff auf die Dateien. Wer sich an kleinen Icons und anderen Eigenschaften orientieren möchte, muss die benötigte Wartezeit in Kauf nehmen.

In Anbetracht der Aufgabenstellung ist die Liste die geeignetere Darstellungsform für Dateisysteme, denn es sollen in erster Linie die Inhalte von Bausteinen und Kursen angezeigt werden, die meist eine flache Struktur haben. Deswegen wird in dieser Arbeit die grafische Komponente verwendet, wie sie in Abbildung 14.2 zu sehen ist.

In diesem Screenshot wird der Inhalt des Verzeichnisses `Eigene Dateien` angezeigt, der aus drei Verzeichnissen besteht. Die Anzeige des aktuellen Verzeichnisses ist eine *Combobox*, die alle höher liegenden Verzeichnisse anzeigt und einen Aufstieg im Verzeichnisbaum ermöglicht. Über einen Doppelklick auf einen Verzeichnisnamen wird dessen Inhalt angezeigt, was einem Abstieg im Verzeichnisbaum entspricht. Mit den ersten beiden Knöpfen in der rechten oberen Ecke kann direkt zu oft genutzten Verzeichnissen gesprungen werden und mit dem letzten wird ein neues Verzeichnis angelegt.

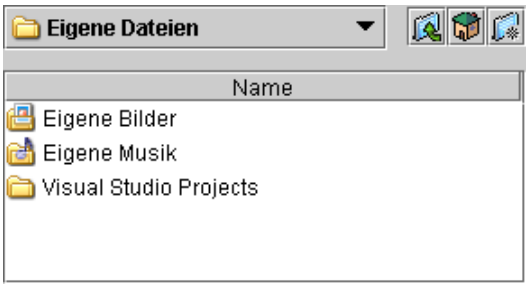


Abbildung 14.2: Visualisierung physikalischer Dateien in *Content Packages*

Die Entwicklung einer grafischen Komponente für Metadaten ist aufgrund ihrer Komplexität umfangreich und wurde im Projekt mαth-kit als Diplomarbeit [Turan04] vergeben. Ausgangspunkt waren die Klassen für Metadaten aus Abschnitt 11.2, deren Elemente und Attribute auf geeignete Weise visualisiert werden mussten. Das Ergebnis ist in Abbildung 14.3 zu sehen.

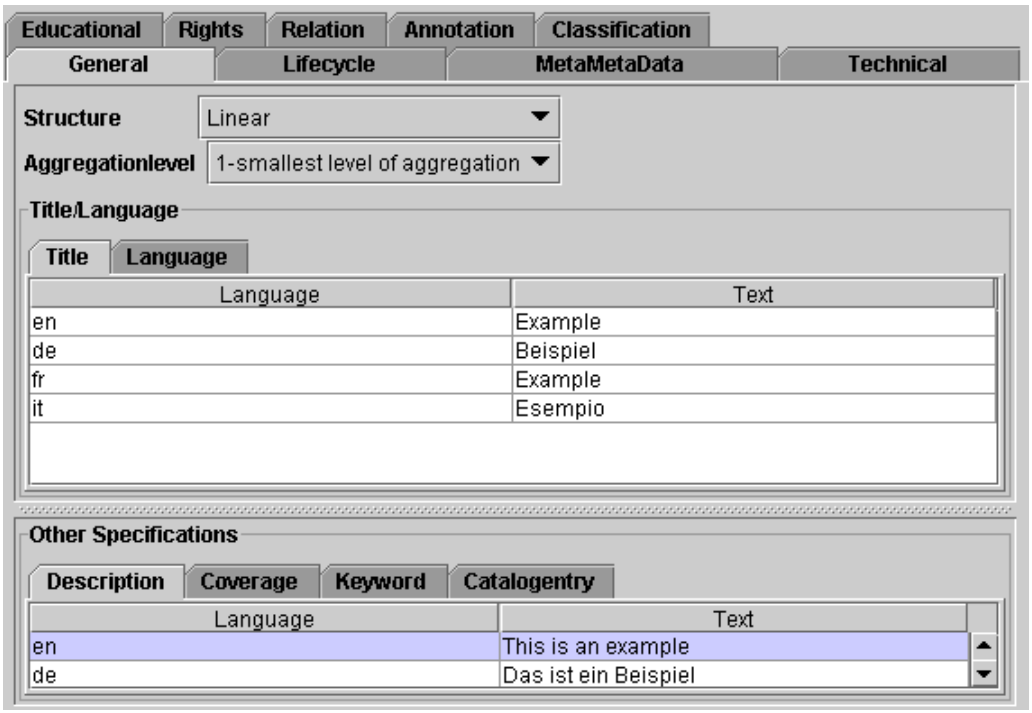


Abbildung 14.3: Komponente für Metadaten

Für die Unterteilung der einzelnen Kategorien wurden Reiter gewählt, deren Seiten den gesamten Inhalt anzeigen. Ohne zu sehr in die Details gehen zu wollen, lässt sich aus Abbildung 14.3 gut die Vorgehensweise bei der Umsetzung nachvollziehen. Zuerst wurden für alle Datentypen der Metadatenbeschreibung die passenden *Widgets* festgelegt und bei Bedarf angepasst. Wörterbücher lassen sich beispielsweise gut als *Comboboxen* umsetzen, wie bei den Elementen `Structure` und `Aggregationlevel` zu sehen ist. Mehrsprachige Daten, die sich aus einem Sprachkürzel und dem Text zusammensetzen, sind in Tabellen gut aufgehoben. Um doppelte Einträge von vornherein zu vermeiden, lassen sich die Sprachkürzel in der Spalte `Language` nur über eine *Combobox* auswählen, die alle unterstützten Sprachen enthält. Diese Tabelle lässt sich für viele Elemente anwenden, wie die sichtbaren Elemente `Title` und `Description` beweisen. Es gibt auch Datentypen, die sich nicht in vorhandenen *Widgets* darstellen lassen, wie z.B. die Datumsauswahl oder die *VCards*, und eigene Implementierungen benötigen. Genaue Ausführungen hierüber finden sich in der genannten Diplomarbeit.

14.3 Rahmenwerk für Werkzeuge

Bevor mit den eigentlichen Komponenten für Bausteine und Kurse begonnen wird, soll ein kleines Rahmenwerk geschaffen werden, dass ihren Aufbau und Einsatz vereinfacht. In einer Anwendung für modulare E-Learning-Inhalte ist es praktisch, mehrere Bausteine und Kurse zur gleichen Zeit geöffnet zu haben. Dies bedeutet jedoch für alle Implementierungen einen zusätzlichen Aufwand, der ein tieferes Verständnis der gesamten Klassenstruktur voraussetzt. Um potentiellen Entwicklern/-innen den Einstieg zu vereinfachen, sollen daher für typische Konstruktionen vorgefertigte Klassen angeboten werden, die eine Integration in die eigene Anwendung vereinfachen.

Besonders die Verschachtelung von Kursen stellt eine Herausforderung bei der Visualisierung dar. Ein einfacher Baum, der sonst bei hierarchischen Strukturen zum Einsatz kommt, ist nicht geeignet, da sich die Inhalte der Bausteine und Kurse nur schwer als Blätter umsetzen lassen. Hinzu kommen die vielen Möglichkeiten zur Platzierung von Metadaten und wie Abbildung 14.3 deutlich zeigt, ist die Komponente in ihren Ausmaßen nicht klein.

In einem ersten Prototyp für das Projekt mαth-kit wurde jeder Baustein und Kurs in einem eigenen Fenster angezeigt, auch wenn es sich um verschachtelte Inhalte handelte. Die Bezugslosigkeit der Fenster untereinander ist an dieser Form besonders gravierend, denn auf den ersten Blick ist nicht mehr ersichtlich, welche von ihnen zusammen gehören. Daher muss eine bessere Darstellung gefunden werden, die einen näheren Bezug zur Verschachtelung besitzt.

Zunächst lässt sich feststellen, dass Bausteine, Kurse und Metadaten nach der Erstellung bzw. Bearbeitung entweder verworfen oder gespeichert werden können. Hierfür erbt die Klasse `JSavablePanel` von der *Swing*-Klasse `JPanel` und stellt verschiedene **Actions** zur Verfügung, wie in Abbildung 14.4 zu sehen ist.

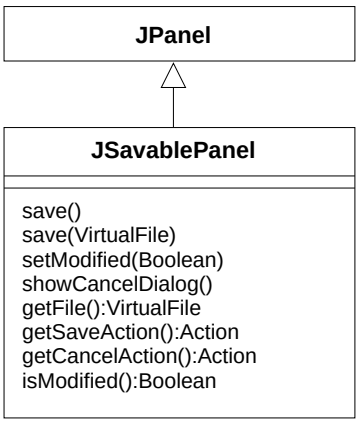


Abbildung 14.4: Klasse *JSavablePanel*

Bei einer *Action* handelt es sich um einen Steuerungsbefehl, der von Knöpfen oder Menüeinträgen abgeschickt wird, sodass eine Ansteuerung des `JSavablePanel` von außen möglich ist. Zusätzlich können Objekte dieser Klasse mit `isModified()` überprüfen, ob sich ihr Inhalt verändert hat. Soll das Objekt geschlossen werden und es wurde modifiziert, dann geht automatisch ein Dialog auf, der auf den möglichen Datenverlust hinweist und explizit eine Bestätigung verlangt. Auf diese Weise gehen nicht ungewollt Daten verloren.

Für die Verwaltung der Verschachtelungen soll eine eigene Klasse zuständig sein, die mehrere Objekte von `JSavablePanel` aufnehmen kann. Abbildung 14.5 gibt eine schematische Ansicht.

Es sind vier verschiedene Ebenen mit verschachtelten Inhalten, die übersichtshalber versetzt dargestellt sind. Beispielsweise ist `Panel 4` in `Panel 3` enthalten und `Panel 3` in `Panel 2`. Im Moment kann nur auf `Panel 4` gearbeitet werden, bis eine andere Ebene ausgewählt wird. Für die Navigation ist auf dem `Panel 1` eine *Combobox* angebracht, mit der auf höher liegende Ebenen zugegriffen wird. Bei der Auswahl, z.B. von `Panel 2`, werden `Panel 4` und

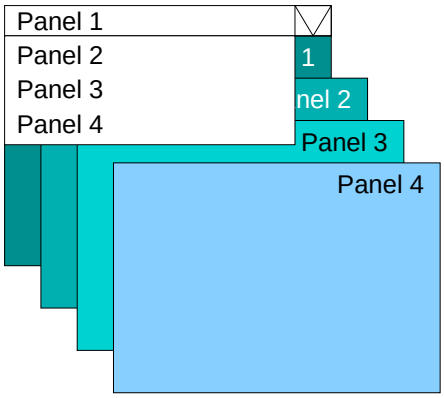


Abbildung 14.5: Verschachtelte Inhalte

`Panel 3` geschlossen. Bevor dies geschieht, werden diese selbstverständlich auf Änderungen geprüft, um eine Speicherung zu ermöglichen.

Um das Beispiel weniger abstrakt zu halten, kann auch der Inhalt von `Panel 4` als Baustein angenommen werden, der in einem anderen Kurs, dargestellt auf `Panel 3`, enthalten ist. Die Ebenen `Panel 2` und `Panel 1` müssen selbstverständlich ebenfalls für Kurse stehen.

Bei der Klasse `JNestedPanel` handelt es sich um eine Spezialisierung der Klasse `JSavable`, wie in Abbildung 14.6 dargestellt.

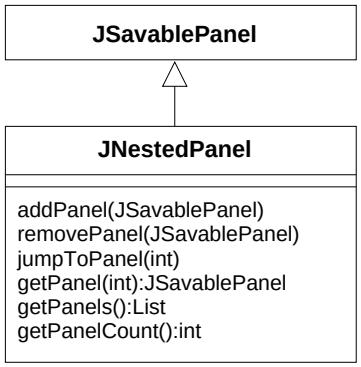


Abbildung 14.6: Klasse `JNestedPanel`

Hauptaufgabe der Klasse ist die Verwaltung der Ebenen. Mit den Methoden `addPanel()` und `removePanel()` werden Objekte der Klasse `JSavablePanel` hinzugefügt bzw. entfernt. Bei der Auswahl einer anderen Ebene in der *Combobox* wird *jumpToPanel()* aufgerufen, woraufhin alle Ebenen zwischen der aktuellen und der ausgewählten geschlossen werden. Ansonsten wäre die Navigation in der *Combobox* inkonsistent.

Metadaten können ebenfalls auf verschiedenen Ebenen auftreten, sodass es sinnvoll ist, diese Komponente aus Abbildung 14.3 in den Mechanismus zu integrieren. Als Spezialisierung der Klasse `JSavablePanel` bettet sich die Darstellung und Speicherung der Metadaten nahtlos ein. Da keine neuen Funktionen hinzugefügt werden, soll an dieser Stelle auf ein Klassendiagramm verzichtet werden.

Bei vielen gleichzeitig geöffneten Bausteinen und Kursen bietet sich die Darstellung mehrerer Fenster in einem Hauptfenster an, die auch als *Multiple Document Interface* (MDI) bezeichnet wird. *Swing* sieht hierfür die zwei Klassen `JDesktopPane` und `JInternalFrame` vor, mit denen solche Anwendungen schnell erstellt sind. Um das Rahmenwerk auch für diese Form auszurichten, wird es um die Klassen `JSavableInternalFrame` und `JNestedInternalFrame` ergänzt. Sie nehmen die *Panels* des Rahmenwerks auf und steuern die Kommunikation. Abbildung 14.7 zeigt die vollständige Vererbungshierarchie, um die beschriebenen Zusammenhänge zu verdeutlichen.

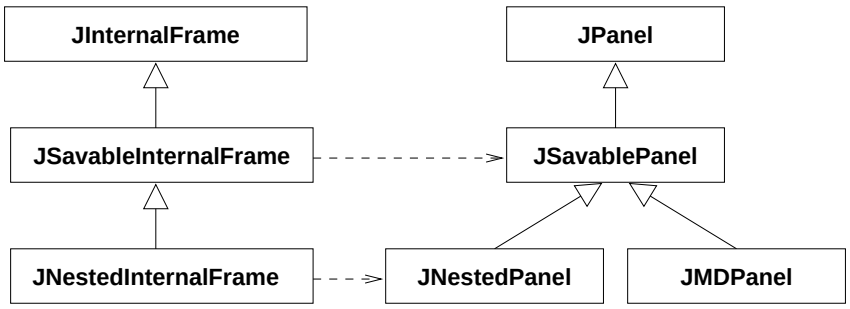


Abbildung 14.7: Klassenhierarchie der grafischen Basisklassen

14.4 Visualisierung der Bausteine und Kurse

In diesem Abschnitt werden grafische Komponenten für die Erstellung und Bearbeitung von Bausteinen und Kursen hergeleitet. Sie visualisieren die Funktionen der Komponenten *Learning Object Engine* aus Unterabschnitt 10.2.2 und *Structure Engine* aus Unterabschnitt 10.2.3. Zuerst wird eine allgemeine Klasse `JCPPanel` für *Content Packages* definiert, die interne Verwaltungsaufgaben übernimmt und noch keine Visualisierung der Inhalte durchführt, denn diese ist in spezialisierte Klassen ausgelagert. Abbildung 14.8 zeigt das Klassendiagramm.

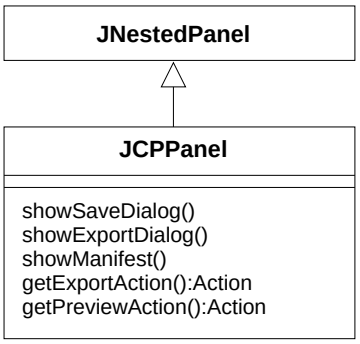


Abbildung 14.8: Klasse `JCPPanel`

`JCPPanel` erlaubt über die Methode `showSaveDialog()` die Auswahl einer bestimmten Datei, in die der Baustein oder Kurs als *Content Package* gespeichert wird. Nach erfolgreicher Bestimmung einer Datei wird die Methode `save()` der Klasse `JSavablePanel` aufgerufen. Hinter den Methoden `showExportDialog`, `getExportAction()` und `getPreviewAction` verbergen sich Aufrufe für Werkzeuge der Rolle *Publisher*, deren Funktionalität im nächsten Abschnitt erläutert wird. An dieser Stelle soll genügen, dass mit dem Export-Dialog Bausteine und Kurse in andere Formate übersetzt werden. In der Regel handelt es sich beim Quellformat um XML und bei dem Zielformat um HTML oder PDF. Die Methode `showManifest()` öffnet ein neues Fenster, in dem das in XML kodierte und formatierte Manifest zu sehen ist. Abbildung 14.9 zeigt eine typische Darstellung, in der die Organisation mit ihren Einträgen und Verweisen auf die Referenzen gut zu sehen ist.

Jetzt sind die Voraussetzungen für die Visualisierung von Bausteinen und Kursen gelegt. Die resultierenden Klassen bieten keine neue Funktionalität an, da sie praktisch eine eingeschränkte, auf das Wesentliche reduzierte Darstellung allgemeiner *Content Packages* sind. Aus diesem Grund wird auf entsprechende Klassendiagramme verzichtet und die interessantere Gestaltung der Komponenten diskutiert. Abbildung 14.10 zeigt zunächst die Bausteinkomponente mit einem geöffneten Beispiel.

Im unteren Teil der Komponente ist die Dateisystemkomponente eingebettet, in der die physikalischen Dateien des *Content Packages* zu sehen sind. Neben den vier Dateien ist auch ein Verzeichnis mit dem Namen `data` enthalten. Welche Dateien darin liegen, soll an dieser

Abbildung 14.9: Manifest mit farblicher Syntax-Hervorhebung

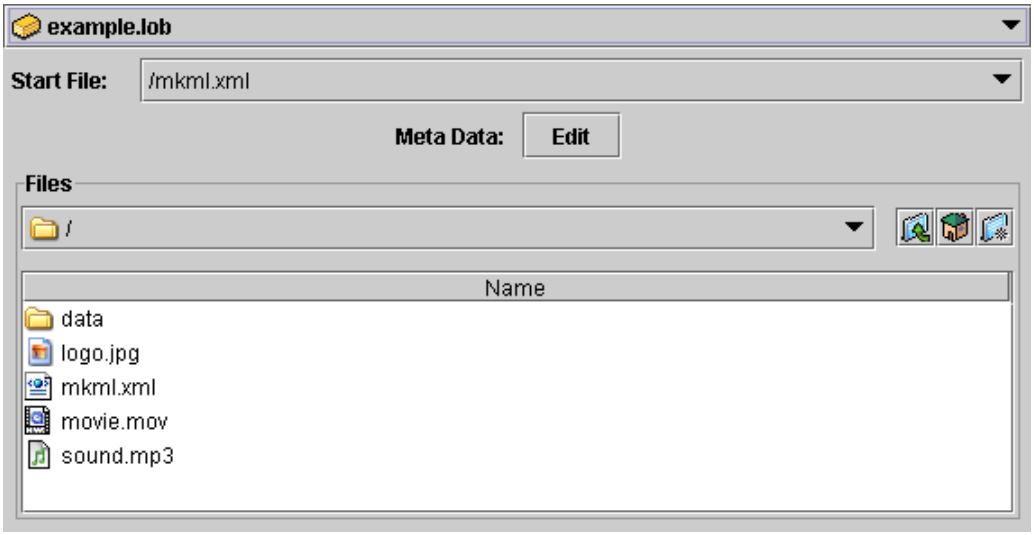


Abbildung 14.10: Komponente für Bausteine

Stelle nicht interessieren, vielmehr soll die mögliche Strukturierung der Dateien innerhalb eines Bausteins verdeutlicht werden. Obwohl mit Verzeichnissen in Bausteinen sparsam umgegangen werden soll, ist ihre Verwendung bei sehr vielen Dateien sinnvoll. In der oberen *Combobox* mit der Beschriftung „Start File“ ist die Datei `mkml.xml` als Einstiegspunkt für diesen Baustein festgelegt. Alle anderen Dateien des Bausteins werden von ihr referenziert und nie direkt aufgerufen. Die Metadaten für diesen Baustein werden über die Metadatenkomponente eingegeben, die über einen Klick auf den Knopf „Meta Data“ als neue Ebene eingeblendet wird.

Durch die Dateisystemkomponente ist die Arbeit mit den enthaltenen Dateien sehr einfach. Neue Dateien werden über ein Kontextmenü neu erzeugt oder per *Drag'n'Drop* von außen hinzugefügt. Für die Bearbeitung stehen je nach Datentyp unterschiedliche Operationen bereit, die externe Programme aufrufen. Zur besseren Strukturierung der physikalischen Dateien können zusätzlich Unterverzeichnisse angelegt werden.

Die Visualisierung der Kurse ist im Gegensatz zu den einfachen Bausteinen etwas umfangreicher, denn neben den physikalischen Dateien müssen auch Ressourcen und die internen Strukturen abgebildet werden. Abbildung 14.11 zeigt einen Screenshot mit einem geöffneten Kurs.

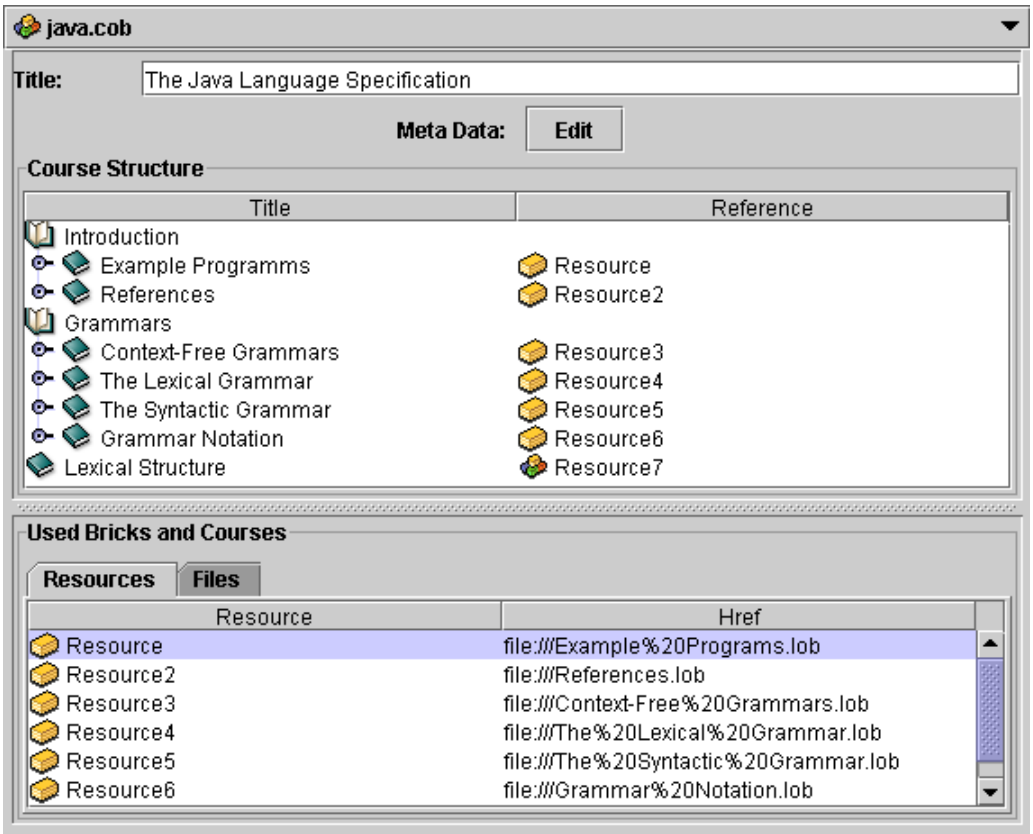


Abbildung 14.11: Komponente für Kurse

Der Titel des Kurses, *The Java Language Specification*, ist der obersten Zeile zu entnehmen und kann bearbeitet werden. Allgemeine Metadaten, die den gesamten Kurs betreffen, können wieder über den entsprechenden Knopf aufgerufen werden. Die Herausforderung bei der Visualisierung ist es, die Übersicht beizubehalten. Über eine Zweiteilung des *Panels*, gekennzeichnet mit den Beschriftungen *Course Structure* und *Used Bricks and Courses*, soll eine klare Trennung zwischen Struktur und Ressourcen angezeigt werden. Der obere Baum entspricht der *Item*-Struktur einer *Organization*, wobei als Attribute lediglich Titel und Referenz angezeigt werden. Für eine genauere Steuerung der Lernpfade werden allerdings wesentlich mehr Attribute benötigt, um z.B. Vorbedingungen, Zeiten, Punkte, etc. anzugeben. Jedoch ist die Baumdarstellung nicht für diese Anzahl von darzustellenden Werten ausgelegt, weshalb

über ein Kontextmenü ein separates Fenster geöffnet werden kann. Abbildung 14.12 zeigt alle unterstützten Attribute, wobei die Felder für SCORM-Werte extra gekennzeichnet sind. Sollte ein Baustein oder Kurs im IMS-Format gespeichert werden, können diese Attribute nicht übernommen werden.

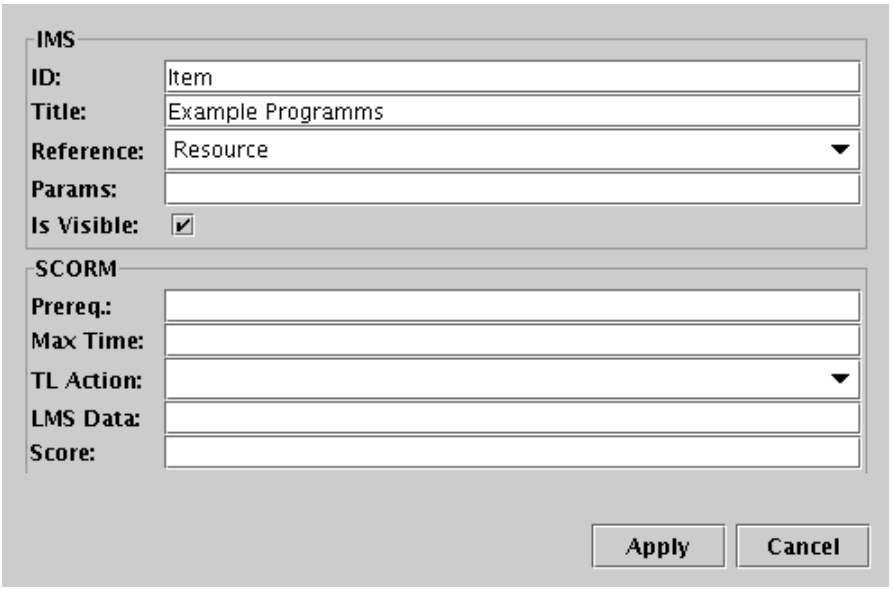


Abbildung 14.12: Ansicht der *Item*-Properties

Die Darstellung der Ressourcen in Abbildung 14.11 ist dank der eingeschränkten Definition aus Abschnitt 12.4 sehr einfach gehalten: Kurse dürfen nämlich nur Bausteine und andere Kurse als Ressourcen enthalten. In der zweiten Spalte mit dem Titel „Href“ steht die jeweilige URL[9], mit der die Ressource verbunden ist. Alternativ können die Bausteine und Kurse auch als physikalische Dateien angezeigt werden, indem diese Darstellung über den Reiter „Files“ ausgewählt wird.

14.5 Steuerung des Exports

Obwohl in dieser Arbeit die Komponenten für die Rolle *Publisher* lediglich hergeleitet wurden (siehe Abschnitt 10.2.4), aber eine detaillierte Beschreibung des inneren Aufbaus ausblieb, soll wenigstens kurz auf die grafische Repräsentation eingegangen werden. Sie reicht vollkommen aus, um eine Idee der Tätigkeit zu vermitteln. Im Wesentlichen geht es um die Umwandlung von Bausteinen und Kursen, die intern XML zur Kodierung verwenden (siehe Abschnitt 3.7). Die gängigsten Zielformate sind HTML und PDF und können mit XSLT sowie XSL-FO generiert werden. Zusammen mit einigen Hilfsklassen, auf die an dieser Stelle nicht weiter eingegangen wird, bilden diese Umwandlungssprachen die **Transformation Packages** (TP), die wesentlich Ausgabeformat und Erscheinungsbild prägen. Zusätzlich können Module aktiviert werden, mit denen z.B. Konvertierungen von Formeln oder Bildern durchgeführt werden. Sie sind unabhängig von den TPs und können als Präprozessoren verstanden werden. Abbildung 14.13 zeigt die genannten Funktionen in einem Dialog-Fenster an.

In der *Combobox* ist das *GET Lab HTML Package* ausgewählt, das HTML-Dateien in der *Corporate Identity* des *GET Labs*[10] erzeugt. Von den optionalen Modulen ist ein Formelkonverter für LaTeX eingeschaltet, was am Häkchen in der Spalte *Active* zu erkennen ist. Da in HTML keine LaTeX-Formeln dargestellt werden können und die Unterstützung von *MathML* in den gängigen Browsern nicht vorausgesetzt werden darf, müssen die Formeln mit Hilfe dieses

[9] Das Zeichen `%20` steht für das Leerzeichen.
[10] `http://getwww.upb.de` (29.10.05)

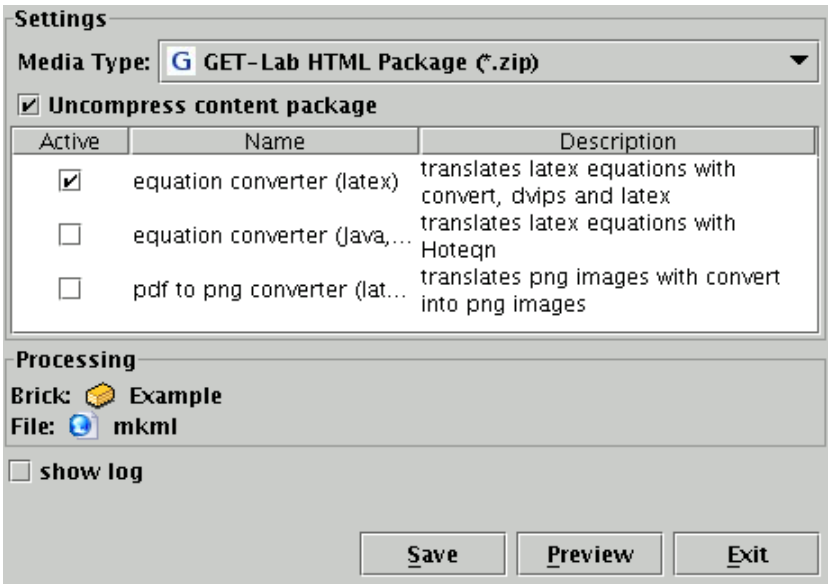


Abbildung 14.13: Dialog für Export-Einstellungen

Moduls als Grafiken eingebunden werden. Die *Checkbox Uncompress content package* legt fest, ob ein *Content Package* erzeugt wird oder lose Dateien. Im Feld *Processing* werden der aktuelle Baustein und die in Übersetzung befindliche Datei angezeigt. Sind mehr Informationen über den aktuellen Durchlauf erwünscht, kann über die *Checkbox show log* eine detailliertere Ausgabe hinzugeschaltet werden. Soll das Ergebnis einer vorherigen Prüfung unterzogen werden, kann über den Knopf *Preview* eine Voransicht erzeugt werden. Ein Klick auf den Knopf *Save* schreibt das Resultat direkt in eine auszuwählende Datei.

14.6 Lyssa

Mit den erstellten Komponenten aus den letzten Kapiteln kann nun das Autorenwerkzeug entsprechend den Abbildungen 10.20 und 10.23 zusammengesetzt werden. In Abschnitt 10.3 wurde festgelegt, dass eine einzelne Anwendung den Belangen der Rollen *Developer*, *Composer* und *Publisher* gerecht werden muss. Durch diese enge Verzahnung der Komponenten ist ein rasches Wechseln zwischen den Rollen möglich, was den Arbeitsablauf verbessert. Weil bisher alle grafischen Komponenten *Java Swing* verwenden, soll auch das Autorenwerkzeug diese Library nutzen.

Im Projekt mαth-kit hat sich **Lyssa**[11] als Name für das Autorenwerkzeug durchgesetzt und wird stellvertretend als Begriff für das Programm verwendet. Ein wesentliches Merkmal von Lyssa ist die übersichtliche und einfache Bedienung. Hierzu gehört auch der Aufbau der grafischen Oberfläche, die aus drei Teilen besteht und nur so viele Daten anzeigt wie nötig. Abbildung 14.14 zeigt einen typischen Screenshot während der Arbeit.

Ganz oben befindet sich die *Toolbar*, die eine Ansteuerung der Basisfunktionen erlaubt und sich dem jeweils ausgewählten Baustein oder Kurs im *Arbeitsbereich* anpasst. Abbildung 14.15 zeigt die *Toolbar* im initialen Zustand, wenn kein Baustein oder Kurs geöffnet ist. Eigentlich wären die Knöpfe `Speichern` und `Speichern als` ausgegraut, denn ihre Funktion steht nur mit geöffneten Dateien zur Verfügung. Zur besseren Identifikation sind sie aber aktiviert dargestellt.

[11] Dieser merkwürdig anmutende Name hat sich aus der Entwicklungsgeschichte ergeben. Jedes Teilprojekt wurde intern nach einem Gott oder einer Göttin aus der griechischen Mythologie benannt. Nach der Fertigstellung der ersten Version des Autorensystems wurde der Name beibehalten, obwohl freilich inhaltlich keine Verbindung zu ihm besteht. Lyssa ist nämlich die Göttin der Rage und hat Herakles mit einer vorübergehenden Verstandstrübung dazu gebracht, seine Frau und Kinder umzubringen.

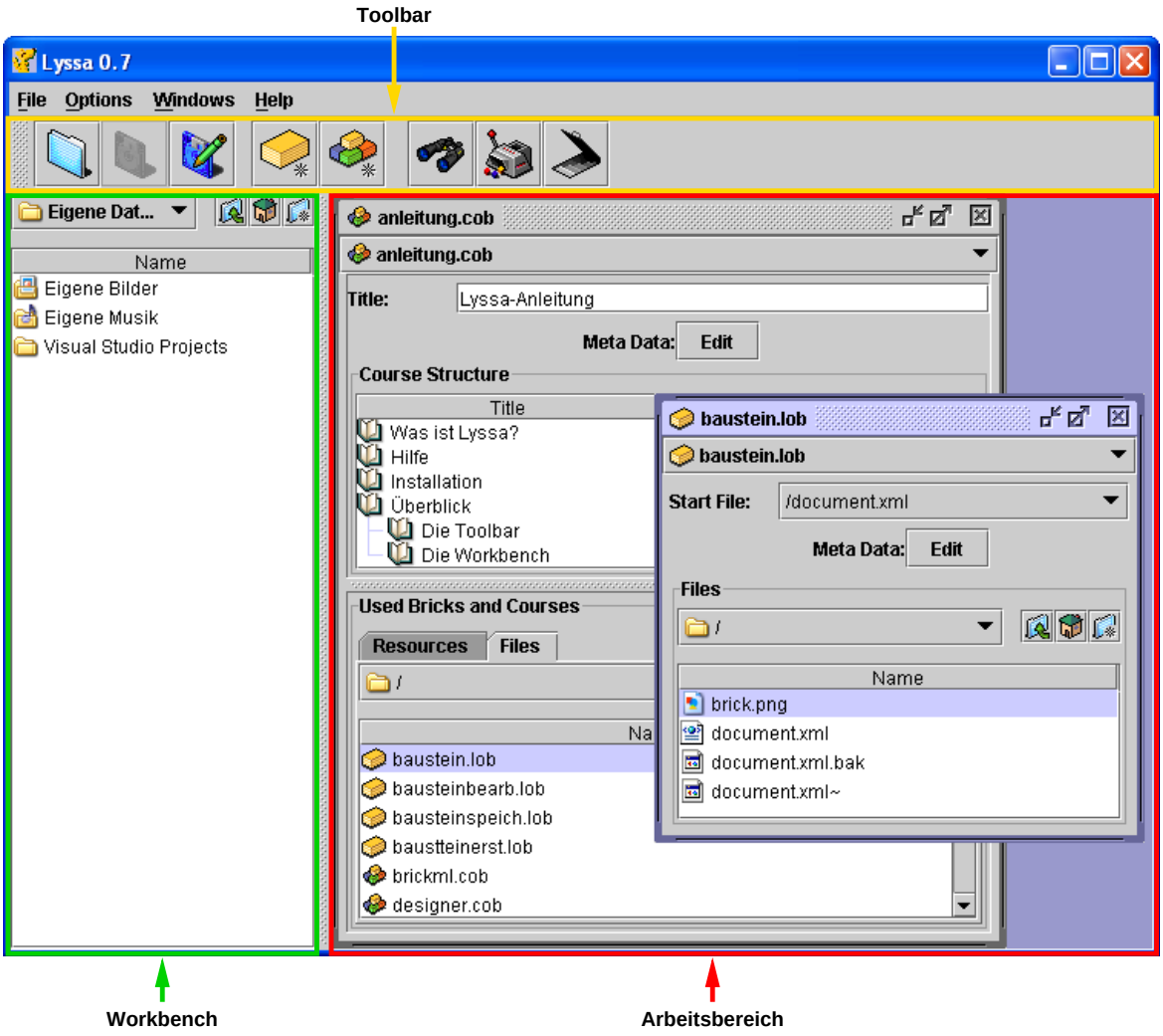


Abbildung 14.14: Screenshot von Lyssa

Über den Knopf `Laden` wird ein Dateidialog geöffnet, der die Auswahl eines Bausteins oder Kurses anbietet. Mit den Knöpfen `Neuer Baustein` und `Neuer Kurs` werden entsprechende Objekte neu angelegt und auf der Arbeitsfläche präsentiert. Sobald ein oder mehrere Inhalte angezeigt werden, erweitert sich die *Toolbar* um abhängige Funktionen. Ist z.B. ein Kurs ausgewählt, dann sieht die *Toolbar* wie in Abbildung 14.16 aus.

Bei den Erweiterungen der *Toolbar* handelt es sich um ausgelagerte Ansteuerungen der Komponenten aus den Abbildungen 14.10 und 14.11. Die Motivation für dieses Vorgehen liegt in der besseren Übersicht begründet. Anstatt die gleichen Funktionen in die Komponenten zu stecken und somit über die Arbeitsfläche zu verteilen, befinden sich die Knöpfe immer an der selben Stelle. Ein Klick auf den Knopf `Preview` übersetzt den ausgewählten Inhalt mit einem voreingestellten *Transformation Package* und öffnet das Resultat in einem Anzeigeprogramm. Ist mehr Kontrolle gewünscht, kann über `Export` der bekannte Dialog aus Abbildung 14.13 geöffnet werde, mit dem sich die Übersetzungsparameter einstellen lassen. Der letzte Knopf `Manifest` öffnet die Manifestansicht aus Abbildung 14.9.

Die bereits erwähnte Arbeitsfläche arrangiert alle geöffneten Bausteine und Kurse in eigenen Fenstern. Wie „normale“ Fenster auch, lassen sie sich verkleinern, vergrößern und verschiedenartig ausrichten. Über Befehle in der Menüleiste können einzelne Fenster gezielt ausgewählt werden.

Als letzter Bestandteil von Lyssa bleibt die in Abbildung 14.14 auf der linken Seite liegende *Workbench* übrig, die für den Datenaustausch mit dem Betriebssystem und anderen Rechnern zuständig ist. Im Prinzip ist es die um ein spezielles Dateisystem erweiterte Komponente zur

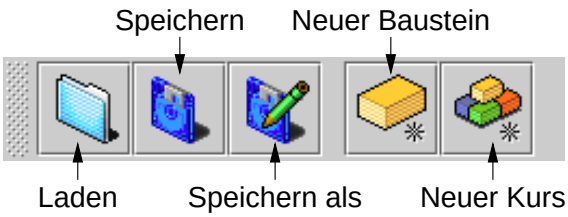


Abbildung 14.15: Screenshot der *Toolbar*

Abbildung 14.16: Screenshot der erweiterten *Toolbar*

Dateiansicht aus Abbildung 14.2. Im Gegensatz zu anderen Programmen bietet Lyssa eine abstraktere Sichtweise auf Dateien und ihre Verteilung. Die Struktur des Hauptverzeichnisses ist in Abbildung 14.17 dargestellt.

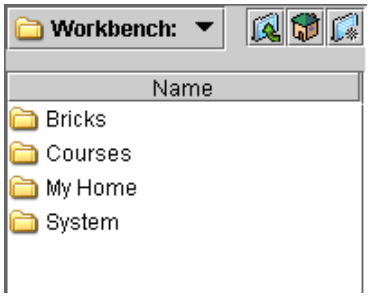


Abbildung 14.17: Screenshot der *Workbench*

Für eigene erstellte Bausteine und Kurse sind die Verzeichnisse `Brick` und `Courses` vorgesehen. Wo die Dateien physikalisch gespeichert werden, ist nicht festgelegt und kann von der voreingestellten Festplatte z.B. auf ein Netzlaufwerk umgestellt werden. Das Verzeichnis `My Home` ist mit dem Home-Verzeichnis des eingeloggten *User Accounts* verbunden und `System` mit dem Wurzelverzeichnis des Betriebssystems. Über das Kontextmenü der *Workbench* können noch beliebig weitere Verzeichnisse mit unterschiedlichen Dateisystemen hinzugefügt werden. Im nächsten Kapitel wird z.B. ein Server zur zentralen Datenhaltung vorgestellt, der seine Inhalte über WebDAV anbietet. So ein WebDAV-Laufwerk lässt sich ohne weiteres in die *Workbench* integrieren.

Kapitel 15

Repository

Für die Vervollständigung der angestrebten Architektur aus Abbildung 10.23 muss noch das *Repository* für die zentrale Datenhaltung implementiert werden. Die hierfür notwendigen Klassen und Komponenten stehen größtenteils aus den vorherigen Kapiteln zur Verfügung, wie Abbildung 10.18 mit der Komponente *Learning Content Repository* zu entnehmen ist. Alle eingezeichneten Subkomponenten sind bereits beschrieben worden. Was noch fehlt, ist die Umsetzung der Infrastruktur für den parallelen Zugriff mehrerer Autorensysteme über das Netzwerk. Sie wird durch die Komponente `Web Server` aus Abbildung 10.19 repräsentiert, die eine Web-gestützte Oberfläche und *Remote Procedure Calls* (RPC) anbietet.

In Kapitel 7 wurden bereits die Web-Technologien vorgestellt, die für eine Umsetzung des Repositories geeignet scheinen. Als Resümee dieser Betrachtung ergibt sich, dass mit Hilfe vorhandener Server, Module und Rahmenwerke anspruchsvolle *Web Applications* bei angemessenem Aufwand realisiert werden können. Teilweise wurden mehrere Produkte für eine Lösung vorgestellt, weshalb nun eine konkrete Entscheidung für die Implementierung getroffen wird. Als zentraler Web-Server wird *Tomcat* der *Apache Group* eingesetzt, weil die meisten Beteiligten des Projekts mαth-kit mit diesem Programm vertraut sind. Für die strukturierte Implementierung der *Web Application* wurden kurz die Rahmenwerke *Struts* und *Spring* vorgestellt, die ganz unterschiedliche Qualitäten besitzen. *Struts* stellt eine einfache, robuste Umsetzung des Entwurfsmusters MVC dar, die sich in vielen Projekten bewährt hat. Sie deckt genau die Ansprüche ab, die sich aus der Umsetzung eines Repositories für modulare E-Learning-Inhalte ergeben. *Spring* hingegen macht einen moderneren, ganzheitlichen Eindruck für die gesamte Umsetzung von *Web Applications*. In Anbetracht der bisher entwickelten Komponenten ist der Aufwand jedoch recht hoch, diese mit *Spring* zu verbinden. Dies liegt unter anderem am Konzept der *Beans* für die Datenhaltung, die über eine eindeutige Identität verfügen und somit für Datenbanken prädestiniert sind. Bausteine und Kurse bestehen aber aus vielen Dateien, teilweise in binären Formaten und lassen sich nur schwer mit Beans modellieren. Aus dieser Erwägung heraus fällt die Entscheidung für die Verwendung von *Struts*.

Mit *Axis* steht eine freie Implementierung des SOAP-Protokolls für *Web Services* zur Verfügung, die sehr gut mit dem Web-Server *Tomcat* zusammen arbeitet. Über die Web-Service-Schnittstelle soll die Suchfunktionalität der Komponente *Search Engine* durch andere Applikationen aufgerufen werden. Somit können Autorensysteme wie Lyssa Suchanfragen für Bausteine und Kurse direkt an das Repository schicken, ohne zwingend eine Web-Oberfläche nutzen zu müssen. Der Datenaustausch selbst soll über die Übertragungstechnik *WebDAV* erfolgen. Weil der Web-Server *Tomcat* mit *WebDAV*-Unterstützung ausgeliefert wird, ist lediglich ein wenig Konfigurationsarbeit zu leisten.

Ein Vorteil der freien Software soll noch hervorgehoben werden, der sich in der Praxis als äußerst nützlich erwiesen hat. Alle Programme und Pakete liegen im Quellcode vor, sodass die internen Vorgänge wesentlich einfacher nachzuvollziehen sind. Im Fall der *WebDAV*-Library konnte sogar ein Fehler selbst behoben werden, der während der Tests aufgetreten war.

15.1 Construction Kit Server

Auch für das *Repository* wurde im Projekt mαth-kit ein eigener Produktname vergeben. Um den modularen Ansatz hervorzuheben, der auf der Baustein-Metapher beruht, lautet er **Construction Kit Server** (CKS). Durch den Einsatz des Rahmenwerks *Struts* ist die Architektur des CKS bereits grob vorgegeben. Der größte Teil der Programmlogik zum Aufrufen der Komponente *Learning Content Repository* liegt über mehrere *Controller* verteilt, die über eine definierte Schnittstelle von *Struts* aufgerufen werden. Überwiegend handelt es sich um die Steuerung von Standardabläufen, wie z.B. die Eingabe einer Suchanfrage, die auf Richtigkeit überprüft werden muss und gegebenenfalls mehrere Berichtigungszyklen durchläuft. Bei einer erfolgreichen Anfrage kann das Suchergebnis so viele Einträge enthalten, dass diese nicht übersichtlich auf einer Seite dargestellt werden können. Dank mitgelieferter Komponenten, die sich über Vererbung erweitern lassen, werden die lästigen Routineaufgaben übernommen.

Es ist daher im Gegensatz zur Herleitung des Autorensystems Lyssa nicht sinnvoll, an dieser Stelle die involvierten Klassen aufzuzählen, denn die Schnittstellen sind überwiegend gleich. Auch der Einsatz von Sequenzdiagrammen ist nicht angebracht, weil überwiegend Methoden des Rahmenwerks *Struts* involviert sind. Von daher reicht es aus, den Aufbau des *Repositories* schematisch darzustellen. Abbildung 15.1 zeigt, wie Web-Server, Module, Rahmenwerke und die selbst entwickelte Komponente zusammenwirken.

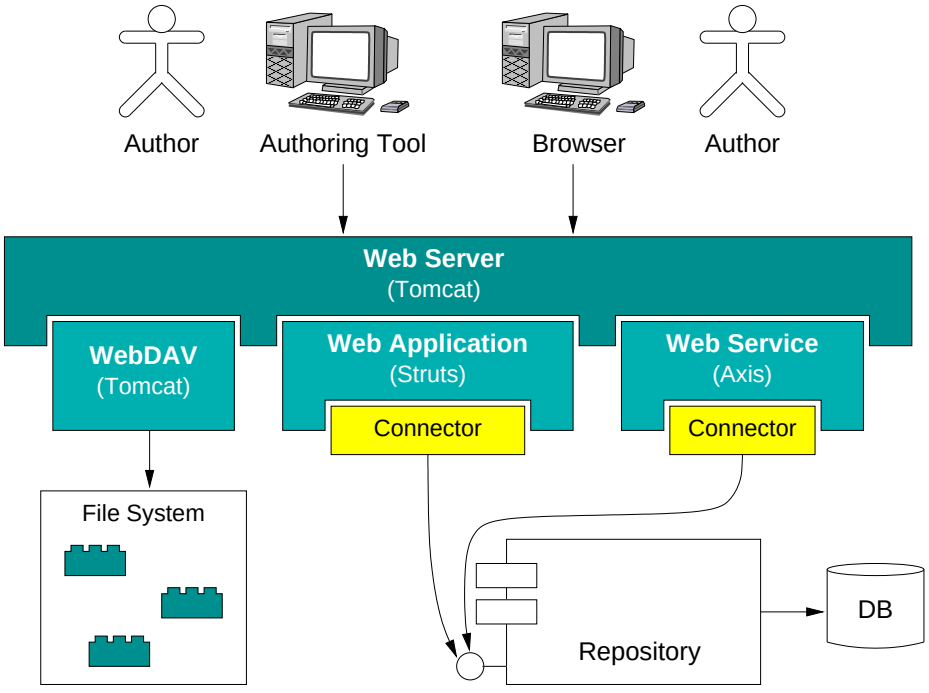


Abbildung 15.1: Aufbau des *Construction Kit Servers*

Als Web-Server nimmt der *Tomcat* alle Anfragen der Clients entgegen, entweder vom Autorenwerkzeug oder einem Web-Browser. Hiernach wird die Eingabe verarbeitet und an das entsprechende Modul, entweder *WebDAV*, *Web Application* oder *Web Service*, delegiert. Die Boxen mit der Beschriftung „Connector“ kennzeichnen die zusätzlich entwickelten Klassen, die das Rahmenwerk *Struts* und die Komponente *Learning Content Repository* verbinden. Die eingezeichnete Datenbank sowie das Dateisystem dienen zur Speicherung der Kurse, Bausteine und Metadaten.

15.2 Web-Oberfläche

Der CKS verfügt auch über eine eigene Web-Oberfläche, die den Zugriff auf die wichtigsten Funktionen zur Baustein- und Kursrecherche ermöglicht. Eine eigene Authentifizierung und Autorisierung auf Dateiebene schützt die Daten gegenüber unbefugten Zugriffen. Daher müssen sich die Benutzer/-innen zunächst über die CKS-Anmeldemaske aus Abbildung 15.2 anmelden.

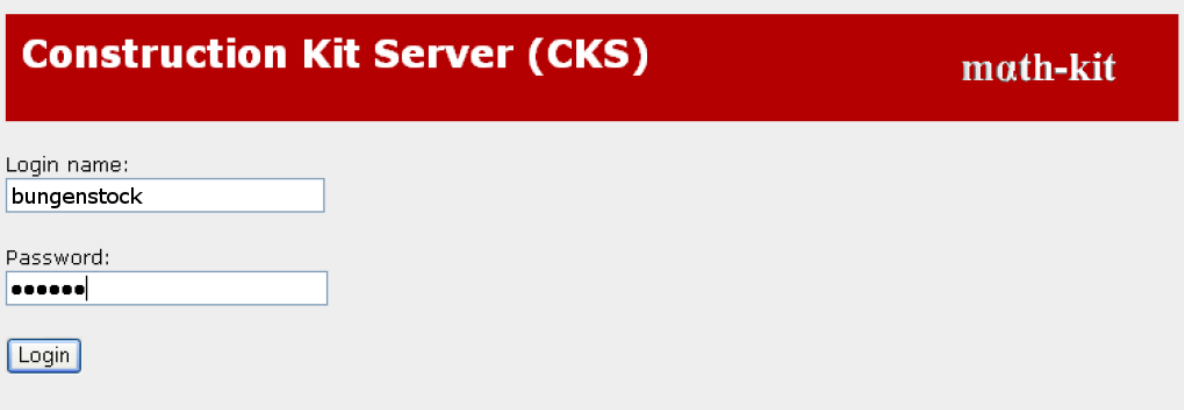


Abbildung 15.2: Screenshot der CKS-Anmeldemaske

Nach erfolgter Anmeldung kann über die gespeicherten Bausteine und Kurse navigiert werden. Abbildung 15.3 zeigt einen kleinen Datenbestand, mit allgemeinen Zusatzdaten.

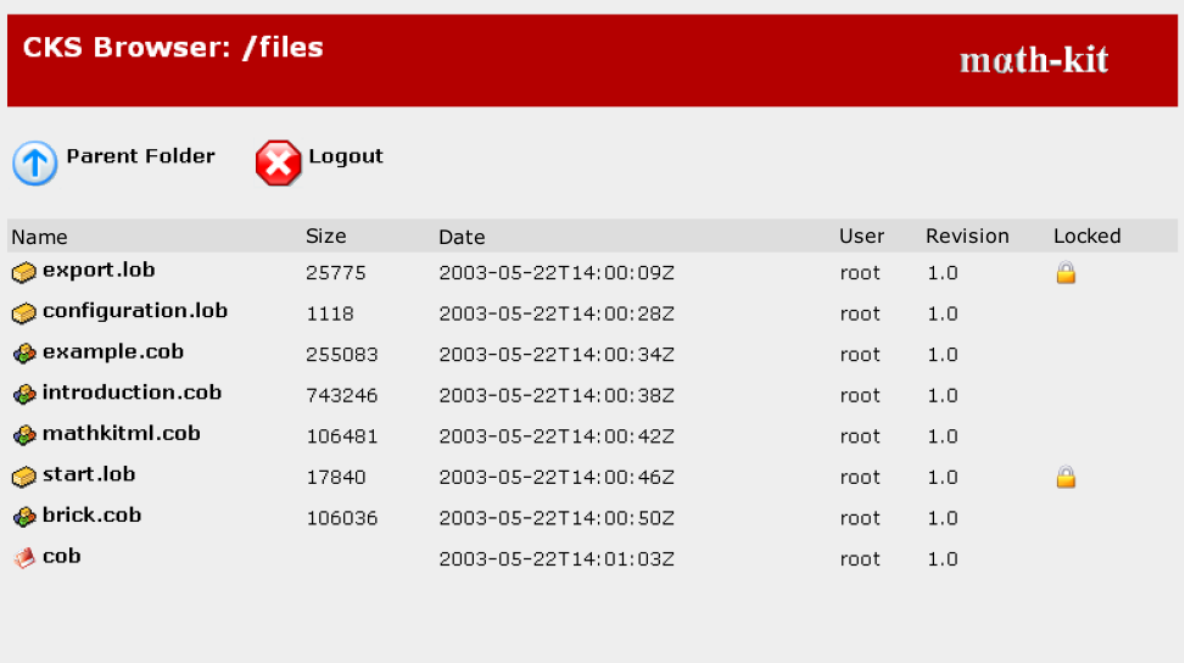


Abbildung 15.3: Screenshot der CKS-Dateiansicht

Neben Größe, Datum, Besitzer/-in sind besonders die Versionsnummern (*Revision*) und der Freigabestatus (*Locked*) interessant. Wird ein Baustein oder Kurs verändert, dann erhöht sich die Versionsnummer, sodass jede Speicherung nachvollzogen und bei Bedarf wieder rückgängig gemacht werden kann. Ist ein exklusiver Zugriff auf eine Datei erwünscht, lässt sie sich für andere Autoren/-innen sperren (angezeigt durch das Vorhängeschloss). Erst nach einer expliziten Freigabe steht die Datei wieder für alle Berechtigten zur Verfügung.

Teil IV

Analyse

Kapitel 16

Ausgewählte Beispiele

In den vorangegangenen Teilen „Entwurf“ und „Implementierung“ wurde ein ganzheitliches Konzept für modulare E-Learning-Inhalte erstellt und umgesetzt. Aufgrund der Detailvielfalt kann der Blick für das Wesentliche verloren gehen, sodass in diesem Kapitel eine Reihe ausgewählter Beispiele die Funktionalität verdeutlichen soll. Von der Erstellung von Bausteinen, über die Aggregation zu Kursen bis hin zum Datenaustausch mit anderen Systemen wird Schritt für Schritt die Arbeit in der Praxis gezeigt. Freilich können nicht alle Aspekte und Möglichkeiten abgedeckt werden, aber der größte Teil des Systems wird durch die Beispiele veranschaulicht.

16.1 Erstellung neuer Bausteine

Zur Erstellung von Bausteinen nehmen die Benutzer/-innen die Rolle *Developer* ein, in der ihnen alle Funktionen des Autorensystems zur Verfügung stehen, die sie für die Erfüllung ihrer Aufgabe benötigen. Es gibt verschiedene Szenarien bei der Tätigkeit, die zu leicht abgewandelten Arbeitsschritten führen, aber letztendlich sind die Unterschiede nicht so gravierend, weshalb die Darstellung eines Beispiels in mehreren Schritten ausreichend ist. An dieser Stelle wird davon ausgegangen, dass die Texte, Bilder und ein *Java Applet* bereits vorliegen. Die Erstellung solcher Materialien ist stark von den eingebundenen Programmen abhängig, die über die Komponente `Multimedia Environment` aus Abbildung 10.11 angesteuert werden. Aus diesem Grund sollen die atomaren Dateien als gegeben angesehen werden, damit nicht zu viel Zeit auf andere Programme verwendet wird.

Als Beispiel soll nun ein Baustein erstellt werden, wie er sich tatsächlich in der Praxis des Projekts mαth-kit ergeben hat. Ein Thema unter vielen sind die komplexen Zahlen, die sich sehr gut als repräsentatives Beispiel anbieten. Für die komplexen Zahlen werden zum einen sehr allgemeine Bausteine benötigt, die sich sehr flexibel in verschiedene Kontexte einbetten müssen, und zum anderen aber auch sehr spezielle Bausteine, die nur für ein spezielles Thema geeignet sind. Da die Gruppen des Projekts mαth-kit in sehr unterschiedlichen Bereichen tätig sind — hier sind Technische Informatik, Mathematik und Ingenieurwissenschaften zu nennen — und mit komplexen Zahlen in Berührung kommen, lässt sich an ihnen der interdisziplinäre Einsatz und die Wiederverwendung gut demonstrieren.

Zunächst muss die Idee für einen Baustein reifen und es schadet nicht, Überlegungen zu den späteren Einsatzgebieten einfließen zu lassen. An dieser Stelle soll ein einführender Baustein entwickelt werden, der unterschiedliche Darstellungen der komplexen Zahlen verdeutlicht. Weil das Leitbild von mαth-kit der multimediale Baukasten ist, soll das Beispiel auch um eine multimediale Komponente bereichert werden. Nach reiflicher Überlegung ist der Entschluss gefallen, dieses Vorhaben mit einem *Java Applet* zu realisieren, denn die geplanten Interaktionen setzen ein hohes Maß an Steuerbarkeit voraus. Mit der Unterstützung mehrerer Entwickler/-innen wurde eine Lösung erstellt, die der *Screenshot* in Abbildung 16.1 nur ansatzweise übermitteln kann.

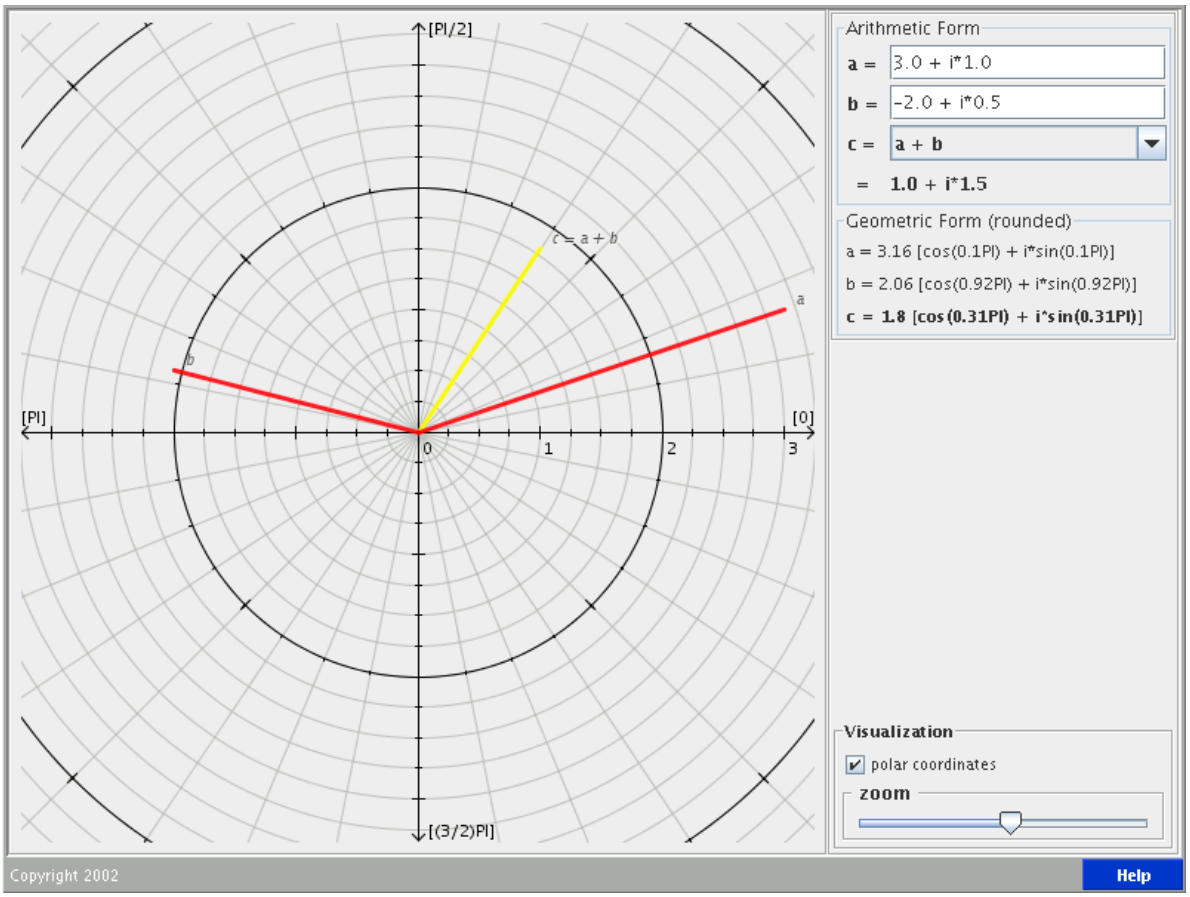


Abbildung 16.1: *Screenshot* des Applets für komplexe Zahlen

Nicht nur die reine Umsetzung der Funktionalität verdient eine explizite Erwähnung, die mit Umschaltung von Koordinatensystemen, Zoom-Funktion und unterschiedlichen Rechenoperationen die verschiedenen Facetten des Themas abdeckt, sondern auch die Anpassungsfähigkeit an den einbettenden Kontext. Rahmen, Hilfe, Farben, Zeichensätze, Schriftgrößen und viele weitere Parameter lassen sich nämlich von außen konfigurieren, sodass dasselbe *Applet* in unterschiedlichen Ausgaben nicht durch ein einmal festgelegtes Äußeres als Fremdkörper hervorsticht. Die störenden Auswirkungen bei fehlender Adaptierbarkeit dürfen nicht unterschätzt werden.

Nach der Entwicklung des *Applets* wird der Text geschrieben. Das Autorensystem selbst enthält keinen eigenen Editor zum Schreiben von Texten oder XML-Dokumenten. Hier gibt es aber viele freie sowie kommerzielle Programme auf dem Markt, die aus der alltäglichen Arbeit bereits bekannt sind und sich für diese Aufgabe anbieten. Der Vorteil dieser Vorgehensweise ist die Freiheit für die Anwender/-innen, denn sie nutzen das Werkzeug ihrer Wahl und haben im idealen Fall keine Einarbeitungszeit. Folgender Ausschnitt des XML-Codes soll eine Idee des Textes vermitteln:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
2
<tco title="Polar Coordinates, Geometrical Interpretation of Complex Multiplication"
4 xml:lang = "en">
<p>
6   Let
  <formula text="true">
8     z = a + ib \not= 0
  </formula>
10  be an arbitrary point in the complex plane. We draw a line from 0 to
  <formula text="true">
```

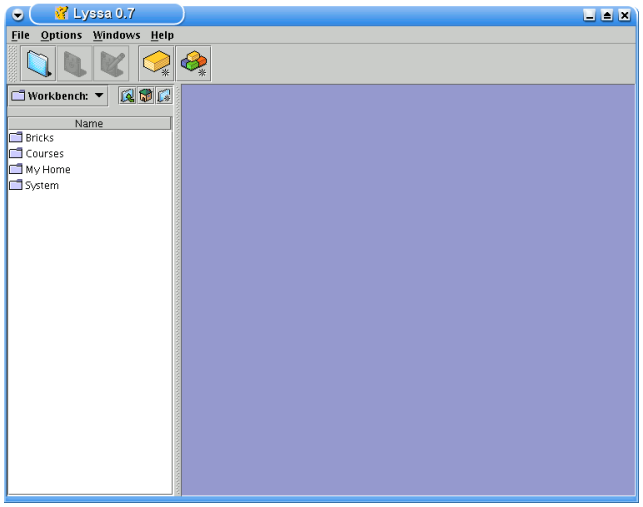
```
12      z \,
    </formula>
14  </p>

16    ...

18  <p>
      Hence, the multiplication of two complex numbers means the
20    multiplication of the two absolute values and the addition of
      the arguments. (The following applet visualises this calculation.)
22  </p>

24  <p>
      <mmo type="applet"
26     code="ComplexApplet.class"
       archive="complex.jar"
28     width="800"
       height="600">
30       <param name="showPolar" value="true"></param>
         <param name="language" value="en"></param>
32       <param name="copyright" value="Copyright 2002"></param>
         <param name="help" value="help.html"></param>
34     </mmo>
    </p>
36  </tco>
```

Nach der Erstellung einer Abbildung — wieder mit einem Programm der Wahl — ergeben das Applet, der XML-Text und die Bilddatei zusammen einen Baustein. Sie werden in ein *Content Package* kopiert und anschließend bestimmt die Auswahl einer Datei den Einstiegspunkt des Bausteins. Eine Angabe von Metadaten vervollständigt den Baustein, sodass er auch in größeren Datenbeständen, wie z.B. dem Repository, leicht aufzufinden ist. Die Bildfolge in Abbildung 16.2 illustriert die einzelnen Arbeitsschritte.

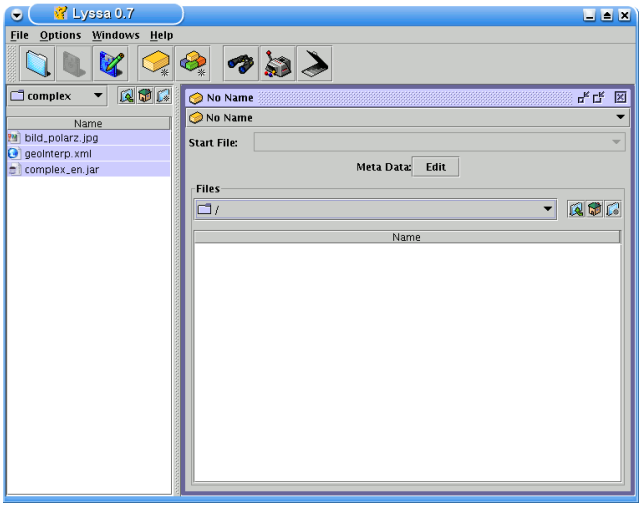
Nach dem Start des Autorensystems öffnet sich ein Fenster, das die *Workbench* auf oberster Ebene und eine leere Arbeitsfläche zeigt (Abbildung 16.2(a)). Durch Drücken des Knopfs „Neuer Baustein“ in der Werkzeugleiste geht ein Baustein ohne Namen auf. Weil die benötigten Dateien bereits vorliegen, muss lediglich das entsprechende Verzeichnis in der Workbench geöffnet werden (Abbildung 16.2(b)). Nachdem alle Dateien mit der Maus markiert wurden, lassen sie sich per Drag'n'Drop in den Dateibereich des Bausteins ziehen. Obwohl die XML-Datei automatisch als Einstiegspunkt angegeben wird, kann es zu Dateikonstellationen kommen, bei denen diese Entscheidung nicht einwandfrei getroffen werden kann. Ist die zugewiesene Datei nicht korrekt, lässt sie sich durch eine *Combobox* manuell auswählen, die alle Dateien des Bausteins anzeigt (Abbildung 16.2(c)). Nun ist der Baustein grundlegend fertig gestellt, sollte aber durch die Vergabe von Metadaten erklärt werden (Abbildung 16.2(d)). Wenigstens die Daten der Kategorie *General* sind einzugeben, sodass die Suchmaschine des Repositories den Baustein über die Schlüsselwörter identifizieren kann. Abschließend wird der Baustein an einem beliebigen Ort gespeichert, entweder lokal, wenn er z.B. noch nachbereitet werden soll, oder im Repository, um ihn zentral zur Verfügung zu stellen.

16.2 Erstellung neuer Kurse

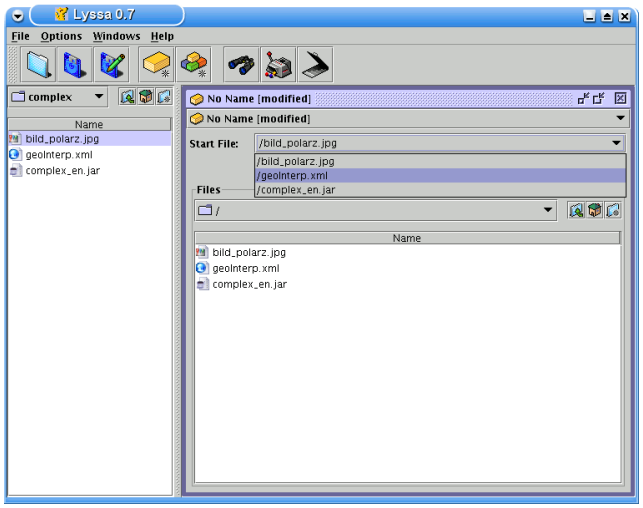
In der Rolle *Composer* erstellen Benutzer/-innen Kurse aus Bausteinen und anderen Kursen. Hierbei ist weniger technisches Wissen gefragt, sondern der Blick für das Gesamte. Lediglich das Verständnis für die Rahmenbedingungen, die einen reibungslosen Einsatz gestatten, darf vorausgesetzt werden. Um das Beispiel nicht zu überfrachten, wird in diesem Abschnitt nicht weiter auf die Suche in Repositories eingegangen. Im vorherigen Beispiel 16.1 wurde bereits



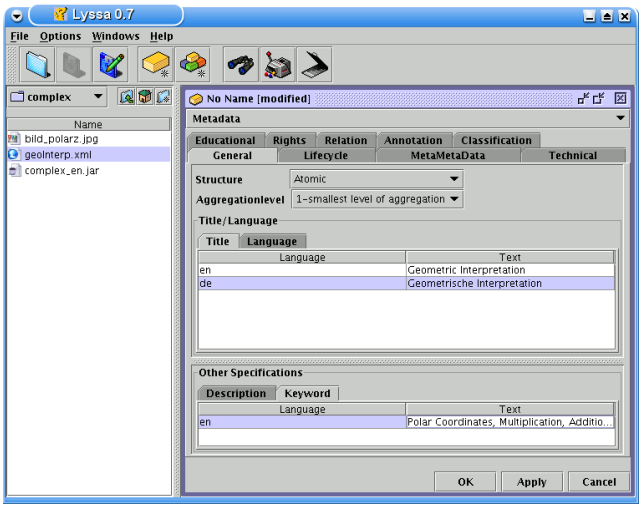
(a) Initialer Zustand



(b) Leerer Baustein



(c) Auswahl der Einstiegsdatei



(d) Eingabe der Metadaten

Abbildung 16.2: Erstellung eines Bausteins in vier Momentaufnahmen

ein Baustein für komplexe Zahlen erstellt, der nun mit weiteren Bausteinen zu einem Kurs zusammengesetzt wird.

Es gibt zwei wesentliche Varianten, einen Kurs zusammenzusetzen, die sich auch kombinieren lassen. Bei der ersten werden alle Bausteine auf ein Mal in den Kurs gezogen und anschließend zu einer Struktur verknüpft. Dieser Aufbau wird bei der zweiten Variante direkt gesteuert, indem jeder Baustein einzeln in die Struktur und nicht in den Ressourcenbereich gezogen wird. Besonders bei vielen Bausteinen ist letzteres Vorgehen übersichtlicher. Abbildung 16.3 verdeutlicht die einzelnen Schritte, wobei die erste Reihe die Variante mit der nachträglichen Verknüpfung zeigt und die zweite Reihe die direkte.

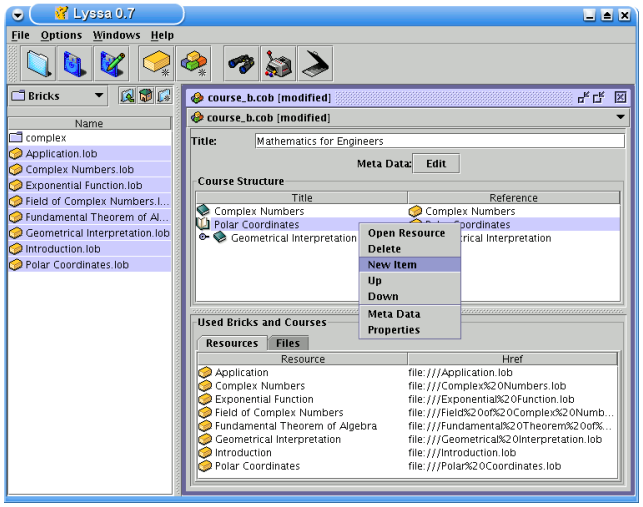
In dieser Darstellung wurden die ersten Schritte ausgelassen, weil sie denen aus Abbildung 16.2(a) gleichen. Nach dem Start befindet sich das Autorensystem im gewohnten Anfangszustand und eine Betätigung des Knopfs „Neuer Kurs“ öffnet einen leeren Kurs. Neben der Auswahl des Verzeichnisses, das die gewünschten Bausteine enthält, wird noch der Titel des Kurses eingegeben, der in diesem Fall „Mathematics for Engineers“ lautet. Anschließend werden alle Dateien ausgewählt und per Drag'n'Drop in den Ressourcenbereich gezogen. Nun kann die Struktur aufgebaut werden, indem über das Kontextmenü des Strukturbereichs der Befehl `New Item` aufgerufen wird. Befindet sich der Mauszeiger beim Öffnen des Menüs auf einem bereits existierenden Knoten, dann wird ein neuer Unterknoten eingehängt. Andernfalls erscheint der Knoten auf oberster Ebene. Abbildung 16.3(a) zeigt den Zustand des Kurses, nachdem bereits zwei Knoten auf oberster Ebene und ein Unterknoten erstellt wurden. Da das Kontextmenü auf Höhe des Knotens „Polar Coordinates“ geöffnet ist, wird der neue Knoten unter diesem erscheinen, wie in Abbildung 16.3(b) zu sehen ist. Die Auswahl der Ressource erfolgt über eine spezielle *Combo Box*, die eine vollständige Liste aller Ressourcen anbietet. Sobald eine ausgewählt ist, wird auch der Knoten automatisch benannt. Eine Eingabe von Metadaten ist selbstverständlich auch für Kurse angeraten, wird aber aus Platzgründen nicht in einer Abbildung dargestellt. Wenn auch dieser Schritt abgeschlossen ist, kann der Kurs an beliebiger Position gespeichert werden.

Abbildung 16.3(c) zeigt die Variante mit dem separaten Einfügen jedes Bausteins in die Struktur. Es ist gerade der Baustein `Application.lob` ausgewählt, der mit der Maus auf den Knoten „Polar Coordinates“ gezogen wird. In Abbildung 16.3(d) ist das Resultat zu sehen. Neben dem neuen Unterknoten wurden gleichzeitig Titel und Referenz auf die Ressource angelegt, sodass mit einer Mausbewegung ein vollständiger Knoten entsteht. Da ein Knoten mehr Eigenschaften besitzt als gleichzeitig in der Baumdarstellung präsentiert werden können, wird ein zusätzliches Eigenschaftsfenster eingeblendet, in dem sich alle Werte einstellen lassen. Wenn der Kurs fertig gestellt ist, folgt nach der Metadateneingabe das Speichern.

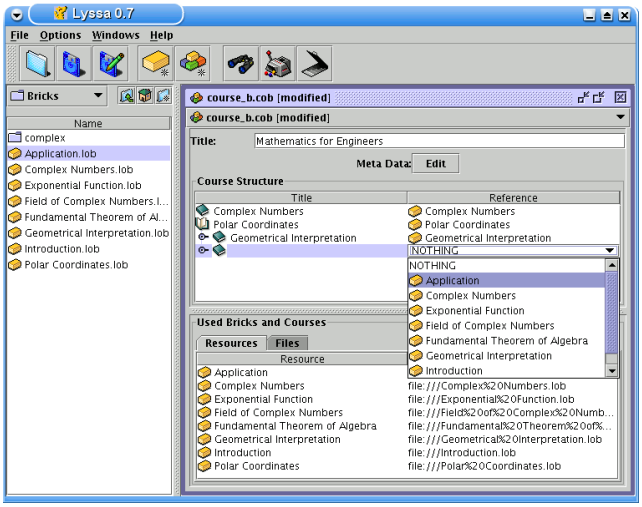
16.3 Inhalte publizieren

Die Rolle *Publisher* ist für die Übersetzung von Bausteinen und Kursen in andere Formate, wie z.B. HTML oder PDF, zuständig. Für diese Tätigkeit stehen verschiedene Werkzeuge zur Verfügung, von denen in diesem Beispiel lediglich die Übersetzungsprozesssteuerung vorgestellt wird. Alle wesentlichen Funktionen wurden bereits in Abschnitt 14.5 erläutert, weshalb sich folgender Text mehr auf die Resultate konzentriert. Zu den anderen Werkzeugen sei noch gesagt, dass ihre Bedienung relativ komplex ist und sie in der Regel nicht oft eingesetzt werden. Ihr wesentlicher Zweck ist die Erstellung der *Transformation Packages* (TP), in denen Java-Klassen, Übersetzungsregeln und sonstige Ressourcen enthalten sind. Einmal erstellt, was durchaus viel Arbeit bereiten kann, sollte das TP nach einer kurzen Anpassungszeit nur noch wenige Änderungen benötigen.

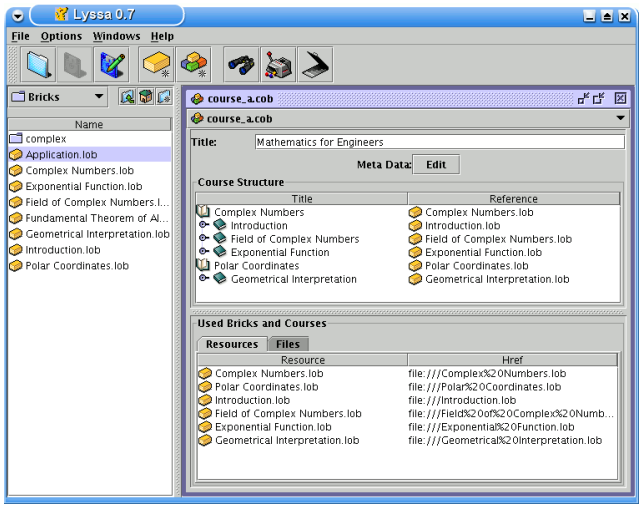
Welche Auswirkungen ein TP auf das Resultat hat, wird nun anhand drei verschiedener Beispiele für den gleichen Inhalt demonstriert. Bei dem Inhalt handelt es sich um ein Skript zur Technischen Informatik von Prof. Dr.-Ing. Bärbel Mertsching, das ursprünglich in LaTeX gesetzt und mit Hilfe des Importmechanismus konvertiert wurde. Das erste TP ist im Stil des



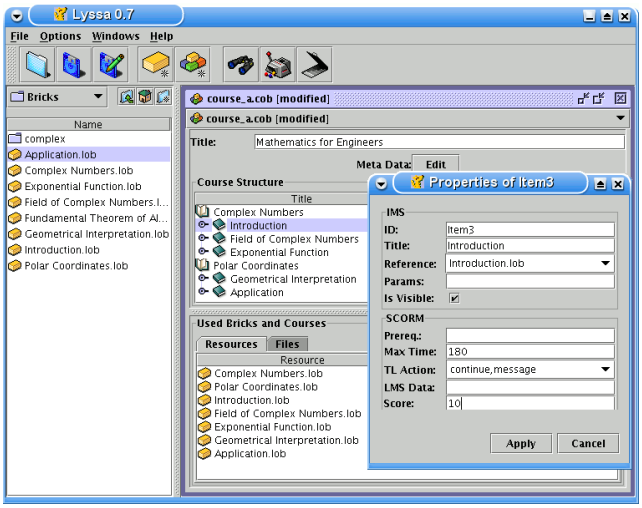
(a) Hinzufügen eines neuen Knotens



(b) Auswählen der Referenz



(c) Separates Einfügen von Bausteinen



(d) Bearbeitung der Eigenschaften eines Knotens

Abbildung 16.3: Erstellen eines Kurses in zwei Varianten

GET Lab-Web-Auftritts gehalten, wodurch sich erstellte Kurse nahtlos in die *Site* integrieren. Abbildung 16.4 zeigt einen *Screenshot* des Ergebnisses mit einer Seite über „Codes“. Es handelt sich um HTML-Seiten für einen gängigen Web-Server, weshalb auch eine Navigation erzeugt wurde. Wäre der Inhalt für ein LMS bestimmt, ließe sich dieser zusätzliche Übersetzungsschritt selbstverständlich auslassen.

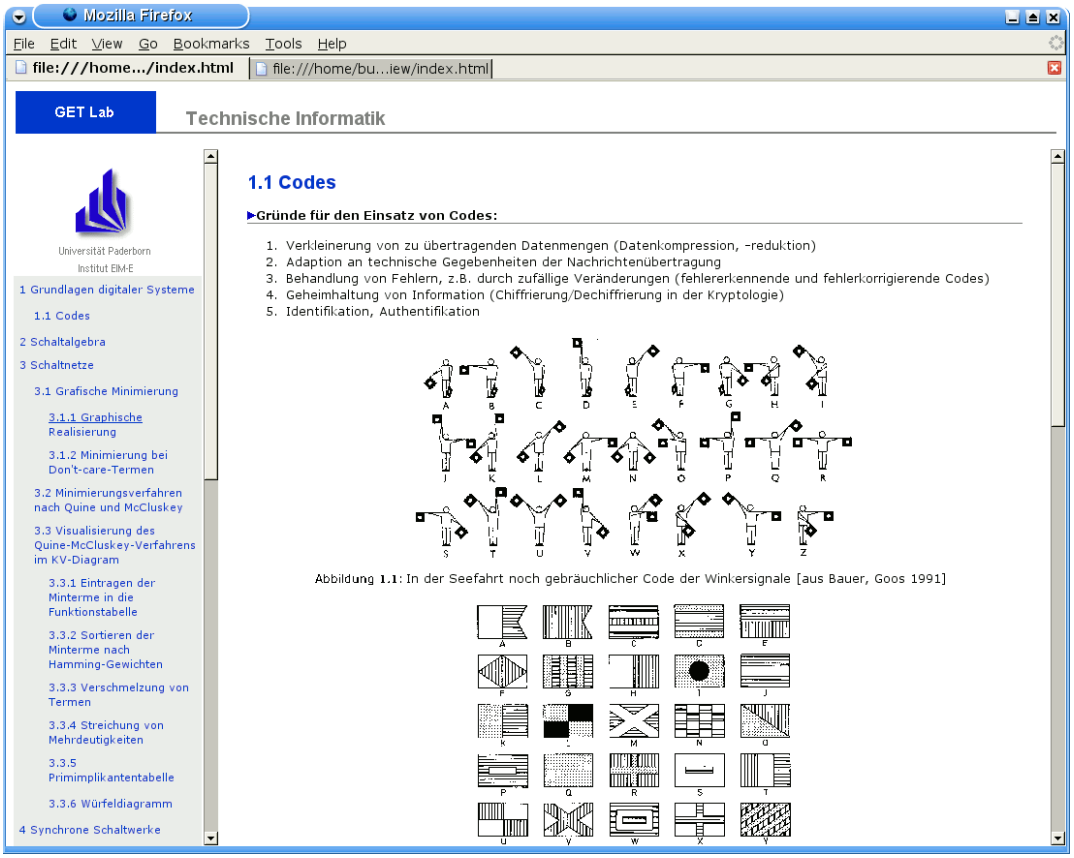


Abbildung 16.4: Übersetzungsergebnis im Layout des *GET Labs* (HTML)

Das zweite Beispiel in Abbildung 16.5 zeigt den gleichen Inhalt, übersetzt mit dem TP des Projekts mαth-kit. Abgesehen von den Farben und den Logos ähnelt die Darstellung dem ersten Beispiel, wo hingegen die Darstellung in einem anderen Format auch zu einem unterschiedlichen Ergebnis führt, wie Abbildung 16.5 verdeutlicht. In diesem *Screenshot* ist der Inhalt als PDF zu sehen, das mit einem weiteren TP des Projekts mαth-kit übersetzt wurde. Die sichtbare Navigation wurde im Gegensatz zur HTML-Variante nicht generiert, weil PDF diese Funktionalität automatisch anbietet. Ein wesentlicher Unterschied macht die Platzierung der Bausteine auf dem Bildschirm aus. Aufgrund der kleineren Darstellung werden bei PDF mehrere Bausteine auf eine Seite gesetzt. Weil HTML hauptsächlich für die Bildschirmdarstellung genutzt wird, darf mit dem Platz großzügiger umgegangen werden. Jeder Baustein wird zu einer HTML-Seite umgewandelt, die sich bei Bedarf vertikal scrollen lässt.

16.4 Explorationsumgebung

Die in dieser Arbeit vorgestellte Infrastruktur für modulare E-Learning-Inhalte unterstützt eine Vielzahl verschiedener Lernparadigmen. In diesem Beispiel soll nun aus Sicht der Rolle *Student* die Arbeit mit einer Explorationsumgebung verdeutlicht werden, einem wesentlichen Konzept im Projekt mαth-kit. Eine Explorationsumgebung dient in erster Linie zur praktischen Vertiefung von bereits angeeignetem Wissen, also z.B. als Ergänzung zu einer Vorlesung.

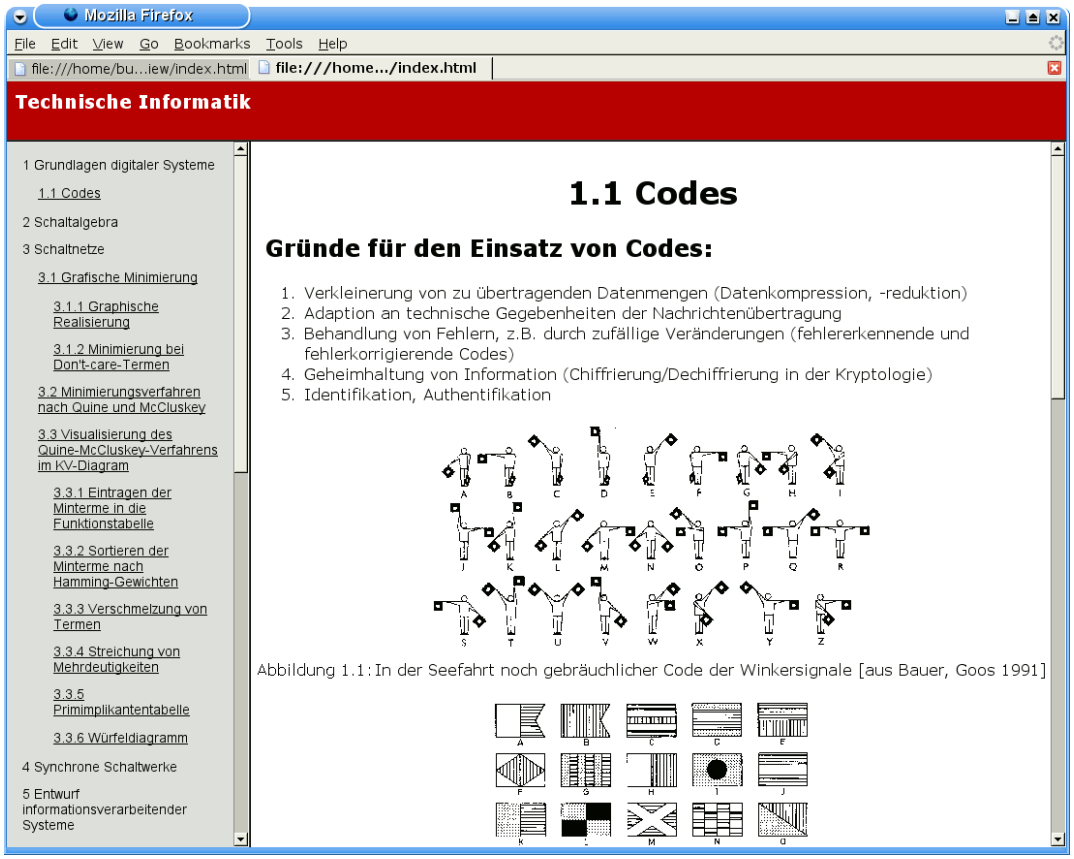


Abbildung 16.5: Übersetzungsergebnis im Layout von mαth-kit (HTML)

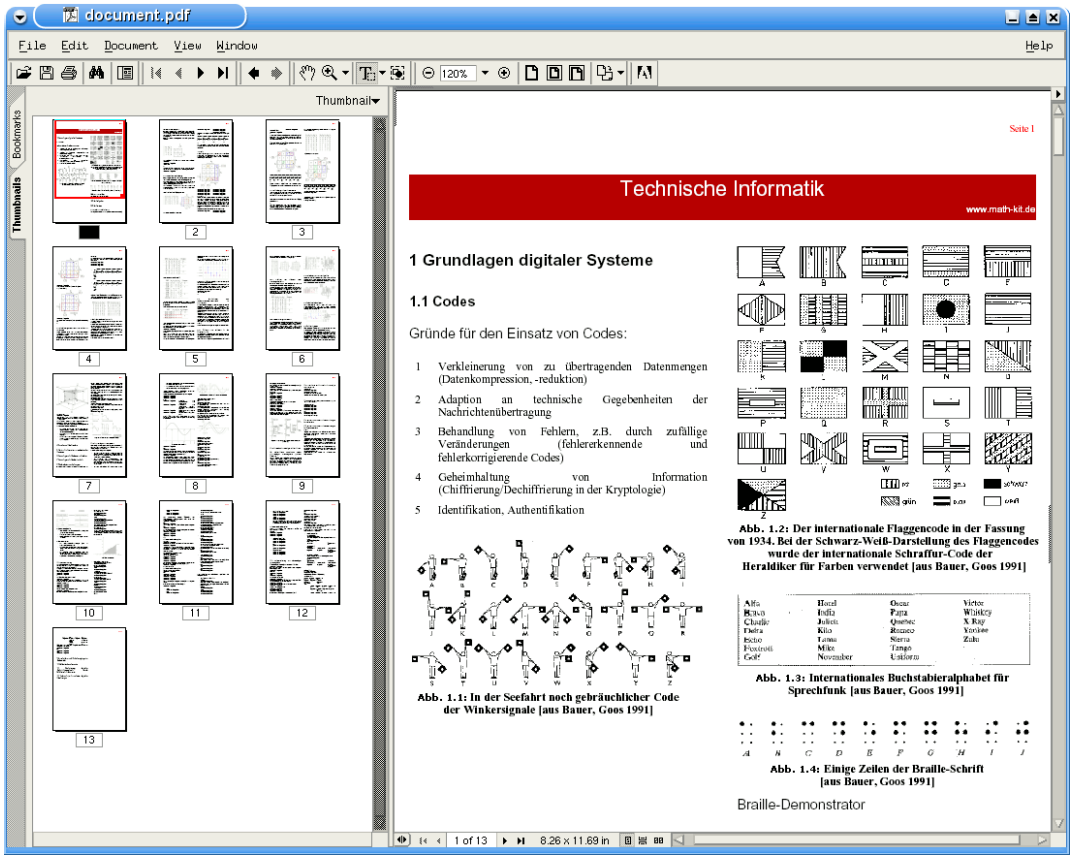


Abbildung 16.6: Übersetzungsergebnis im Layout von mαth-kit (PDF)

Benutzer/-innen in der Rolle *Student* können auf diese Weise im Selbststudium ihren aktuellen Wissensstand überprüfen und erweitern. Die zugrundeliegende Lerntheorie für Explorationsumgebungen ist der Konstruktivismus (siehe Abschnitt 2.2.3).

In diesem Beispiel sollen die bereits verwendeten komplexen Zahlen als Anschauungsobjekt dienen. Eine Explorationsumgebung beginnt zunächst mit einem theoretischen Teil, der das nötige Grundwissen vermittelt. Abbildung 16.7 zeigt den Unterabschnitt „Geometrische Interpretation der komplexen Zahlen“ des Abschnitts „Gaußsche Zahlenebene“. Sollten die theoretischen Grundlagen bereits vorhanden sein, kann auch direkt zum Explorationsteil übergegangen werden. Abbildung 16.8 zeigt ein Applet zum Rechnen mit komplexen Zahlen im kartesischen Koordinatensystem. Nach der konstruktivistischen Sichtweise gibt der Explorationsteil keine starre Aufgabe oder Ausführungsfolge vor, sondern erlaubt der Rolle *Student* eine individuelle Erfahrung. In diesem Beispiel erlaubt das Applet, verschiedene Operationen mit komplexen Zahlen interaktiv durchzuführen. Um den Transfer des neu erlangten Wissens in die Praxis ein wenig zu vereinfachen, folgt ein Anwendungsteil mit spezifischen Beispielen.

Viel wichtiger als dieser Praxisbezug ist jedoch der Übungsteil, der den aktuellen Lernfortschritt wiedergibt. Ohne diese Überprüfung kann die eigene Leistung der Rolle *Student* nur schwer eingeschätzt werden. Noch gravierender wirken sich falsche Rückschlüsse oder Missverständnisse aus, die sich vielleicht aus dem Explorationsteil ergeben. Um ihnen entgegen zu wirken, sollten die Übungen das gesamte Spektrum einer Explorationsumgebung abdecken. In welcher Form dies genau geschieht, ist dabei von geringer Wichtigkeit. Mit dem Autorensystem Lyssa lassen sich z.B. Quiz, Puzzles und Multiple-Choice-Aufgaben ohne Aufwand integrieren. In Abbildung 16.9 ist z.B. eine Übung zur Umwandlung der Darstellungsarten zu sehen, die mit der eigenen Auszeichnungssprache des Projekts mαth-kit erstellt wurde [Baudry03; Baudry04b].

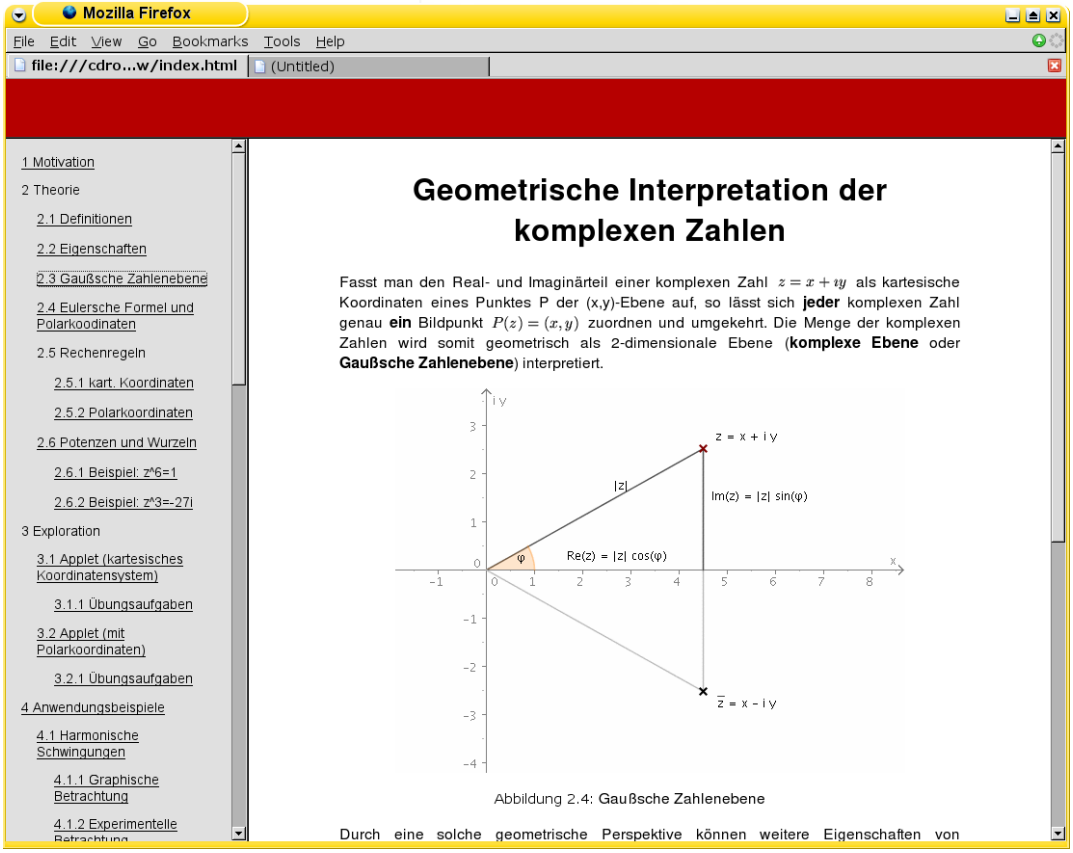


Abbildung 16.7: Theorieteil

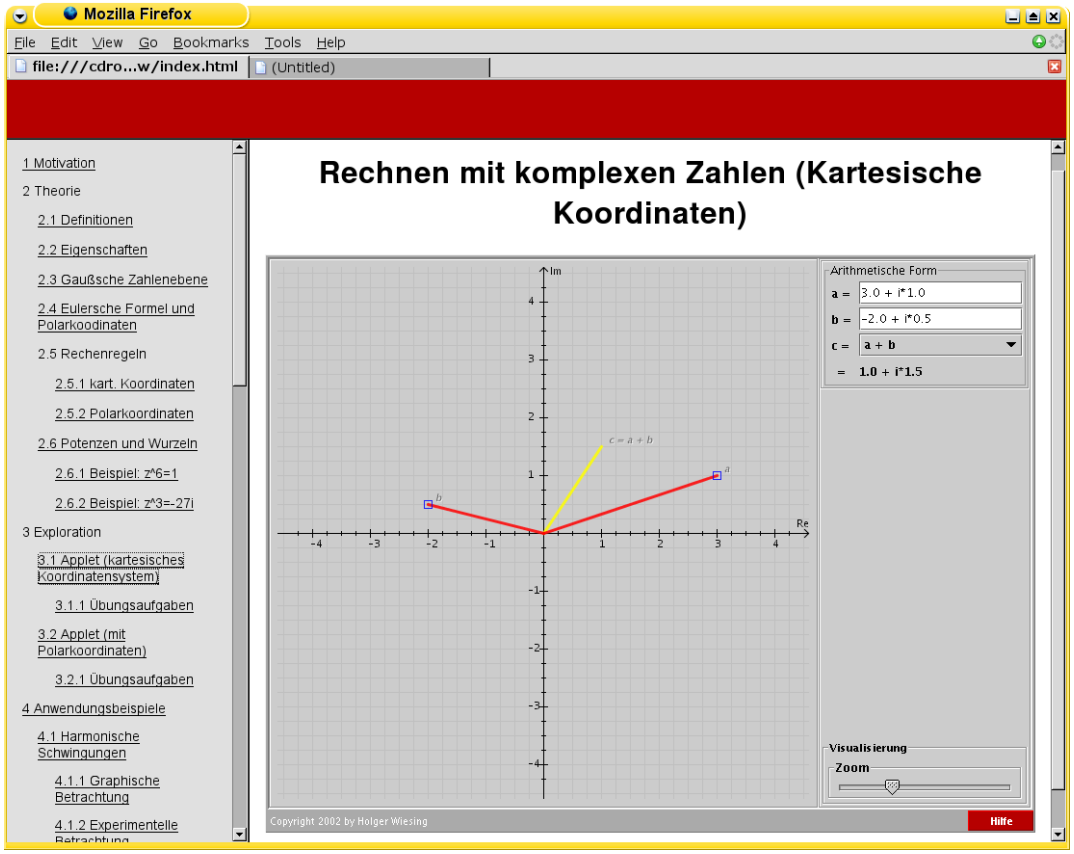


Abbildung 16.8: Explorationsteil

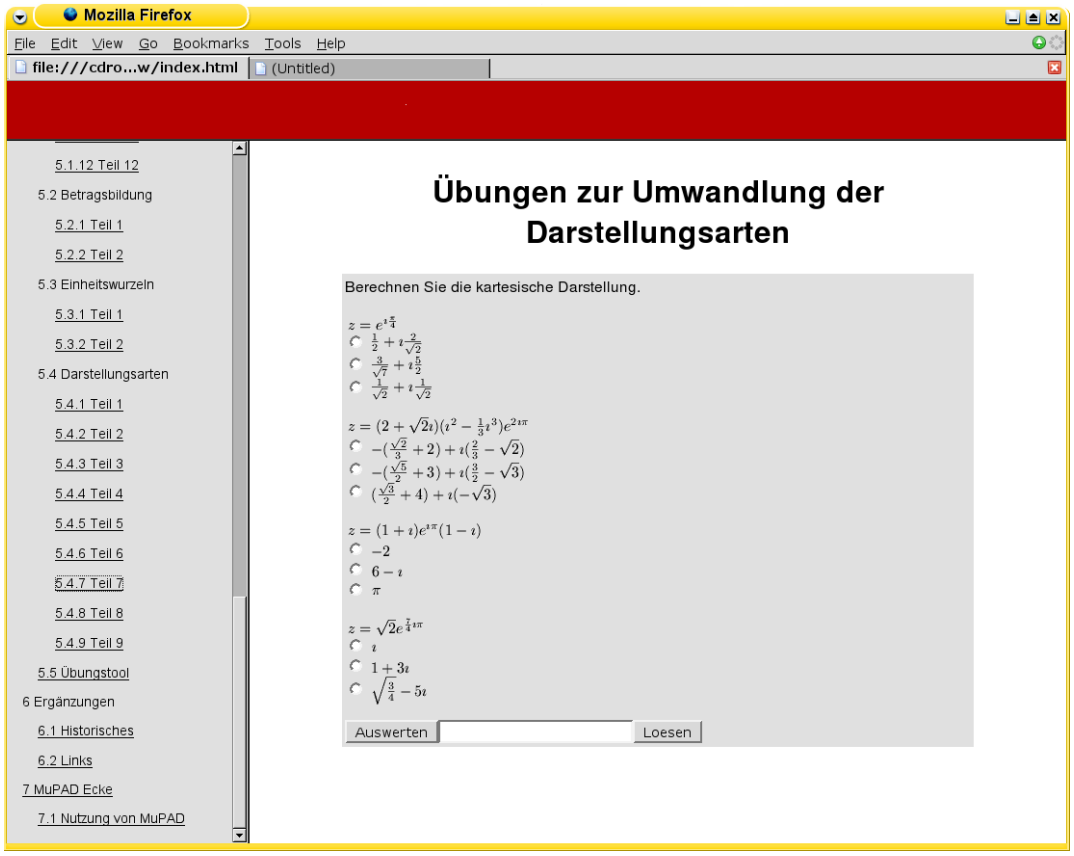


Abbildung 16.9: Übungsteil

Kapitel 17

Zusammenfassung und Bewertung

Das Ziel dieser Arbeit ist, Entwurf, Implementation und Einsatz eines ganzheitlichen Konzepts für modulare E-Learning-Inhalte zu realisieren. Der vorgegebenen Systematik folgend wurden im ersten Teil der Stand der Wissenschaft wiedergegeben und die aktuellen Grenzen aufgezeigt. Als besondere Herausforderung stellte sich die vorherrschende Betrachtung von Einzelproblemen in der wissenschaftlichen Gemeinschaft heraus. Keine der Arbeiten deckt das gesamte Spektrum dieses Gebietes ab, sondern betrachtet Details, die für sich genommen wichtig sind, sich als Teil eines Ganzen jedoch in der Praxis anders verhalten. So galt es, die einzelnen Theorien, Definitionen sowie Umsetzungen zu kombinieren und um neue Ideen bzw. eine erweiterte Sicht anzureichern. Dass dieses Unterfangen nicht einfach wird, zeichnete sich bereits in Kapitel 3 über die Lernobjekte ab. Viele Definitionen, die teilweise ähnlich und doch wieder sehr verschieden sind, zeigten, wie unterschiedlich die Ansichten und Bedürfnisse sind. Auch in dieser Arbeit wird die Problemstellung aus einem eigenen Blickwinkel betrachtet. Herausgekommen ist deshalb eine Lösung, die stark technisch angelegt ist, aber durch ihre generische Struktur keine Barrieren für unterschiedliche didaktische Konzepte aufstellt.

Mit den erworbenen Erkenntnissen wurde in Kapitel 10 eine Vision für das angestrebte System entwickelt. Ein besonderes Augenmerk lag auf der genauen Feststellung des Bedarfs der späteren Benutzer/-innen, indem über Rollen die einzelnen Tätigkeiten gruppiert wurden. Einerseits lassen sich durch diese Abgrenzungen die einzelnen Anwendungsfälle leichter finden und vervollständigen, andererseits wird ein modularer Aufbau des späteren Systems gefördert. Jede Tätigkeit lässt sich nämlich wieder einer bestimmten Anzahl von Komponenten zuordnen, die für sich einen abgeschlossenen Funktionsumfang besitzen. Hierdurch lassen sich einzelne Teile des Systems auch in anderen Kontexten einsetzen, was die Wiederverwendbarkeit des Systems erhöht. Aus diesem Grund muss nicht die gesamte Anwendung installiert und konfiguriert werden, nur weil eine bestimmte Funktionalität benötigt wird. Die Abhängigkeiten der Komponenten untereinander sind durch den umfangreichen Entwurf auf ein Minimum reduziert.

Für ein besseres Verständnis wurde zusätzlich die Metapher „Baukasten“ eingeführt, die weitere Begriffe wie z.B. den Baustein implizierte. Diese zusätzlich im Entwurf eingezogene Ebene gibt Entwicklern/-innen wie Benutzern/-innen eine vereinfachte Ansicht auf das komplexe System, indem sie Assoziationen und eine gewisse Vertrautheit hervorruft. Anstatt technische Details zu betonen, werden die wesentlichen Eigenschaften des Systems in den Vordergrund gestellt: Bausteine lassen sich vielfältig und „kinderleicht“ kombinieren. Die Metapher wirkte sich nicht nur auf das *User Interface* (UI) aus. Jede einzelne Komponente, sei sie noch so klein oder allgemein, wurde speziell auf dieses Prinzip ausgelegt, sodass sich ein konsistenter Entwurf ergab, der ohne Hilfskonstruktion auskommt.

Der Entwurf der einzelnen Komponenten gestaltet sich orthogonal, indem neue Komponenten die bestehenden für ihre Aufgaben nutzen. Auf unterster Ebene befinden sich die Basiskomponenten aus Kapitel 11, die auf den ersten Blick wenig mit E-Learning zu tun haben. Sie

bereiteten jedoch den Weg für die eigentlich genutzten Komponenten, indem sie den technischen Zugriff auf Dateien und externe Anwendungen abstrahieren. Erst durch die vereinfachte Sicht auf die Inhalte der Bausteine und Kurse war es möglich, die komplexen Konvertierungs- und Integrationsfunktionen in der Form anzubieten. In Kapitel 12 wurde gezeigt, wie sich die Basiskomponenten mit vorherrschenden Standards sowie Metaphern zu einer vielseitigen Komponente für modulare E-Learning-Inhalte vereinen lassen. Diese Komponente für Bausteine und Kurse bietet sich an, auch außerhalb dieser Arbeit in anderen Projekten verwendet zu werden. Unabhängig von den eingesetzten Metaphern — es handelt sich um eine austauschbare Schicht —, erlaubt sie die Erstellung, Wartung und Nutzung standardkompatibler Lernobjekte. Sollte in einem Projekt die Auffassung vorherrschen, z.B. lieber Wileys Atom-Metapher aus Abschnitt 3.2.3 einzusetzen, ist der Aufwand für die Anpassung der Schnittstelle gering.

Zusammen mit den anderen Komponenten für den Import von Inhalten, der Suche und der Übersetzung in andere Formate ergab sich ein vollständiges Rahmenwerk in Kapitel 13. Es wurden die Bereiche Erstellung von Inhalten, ihre Komposition und die Konvertierung abgedeckt, auf denen Komponenten zur Darstellung sowie Steuerung aufsetzen. Hierdurch war der funktionale Grundstein gelegt, um die angestrebte Umgebung für modulare E-Learning-Inhalte im vollen Umfang zu realisieren. Die mit den Benutzern/-innen kommunizierenden Komponenten können beliebig nach eigenen didaktischen Gesichtspunkten gestaltet werden, weil das Rahmenwerk hierüber keine Vorgaben macht. Auch technische Entscheidungen, ob das System z.B. verteilt mit vielen Autoren/-innen arbeiten soll oder doch als Einzelplatzanwendung realisiert wird, sind noch offen. Dank der Skalierbarkeit des Rahmenwerks, können auch wesentlich kleinere oder spezialisierte Lösungen umgesetzt werden, als sie hier angedacht sind. Es stehen alle Möglichkeiten offen.

Für das gesetzte Ziel musste auch eine konkrete Implementierung umgesetzt werden, die sich in der Praxis bewährt. Die Systemvision in Abbildung 10.23 gibt genau vor, dass die Entwicklungsumgebung für modulare E-Learning-Inhalte aus einem zentralen Repository besteht, auf das die Autoren/-innen mit einem Werkzeug zugreifen. Angelehnt an die gewählte Metapher Baukasten, wird in Kapitel 14 das Autorenwerkzeug umgesetzt. Der Fokus lag hierbei auf einer einfachen wie flexiblen Anwendung, bei der mit wenigen Arbeitsschritten die gewünschten Ergebnisse erzielt werden. Wenn immer möglich, wurden grafische Komponenten herausgearbeitet, wie z.B. die Dateiansicht, die in vielen Kontexten ihren Einsatz finden. Hierdurch wird die ohnehin kurze Einarbeitungszeit reduziert. Auch beim Autorenwerkzeug soll die Skalierbarkeit betont werden, denn durch den Zugriff über die *Workbench* kann das Programm als Einzelplatzlösung oder als Teil einer verteilten Anwendung betrieben werden.

Das Repository in Kapitel 15 setzt sich größtenteils aus fertigen Servern und Libraries zusammen, die aber entsprechend konfiguriert und angepasst wurden. Auch bei dieser Aufgabe kam der Komponentenentwurf entgegen, weil sich die benötigte Funktionalität einfach integrieren ließ. Der Entwicklungsaufwand war relativ gering und das Ergebnis überzeugend. Eine Reihe verschiedener Protokolle sowie Mechanismen ermöglichen nun den Zugriff auch von anderen Programmen, die nicht in dieser Arbeit entwickelt wurden. Wenn nun z.B. ein anderes Autorenwerkzeug genutzt wird, das standardkompatibel ist und WebDAV unterstützt, kann es auf das Repository zugreifen.

In Anbetracht der Zielsetzung aus Abschnitt 1.2 und ihrer Verfeinerung aus Abschnitt 9.1 wurde in dieser Arbeit eine adäquate Lösung herausgearbeitet, die teilweise sogar über die Ansprüche hinausgeht. Dank der Skalierbarkeit des gesamten Systems können nicht nur große Infrastrukturen für modulare E-Learning-Inhalte aufgebaut werden. Je nach Belieben können einzelne Komponenten entnommen werden, die sich aufgrund ihrer Standardkompatibilität mit Systemen gleicher Ausrichtung wirkungsvoll integrieren lassen. Mit dieser Arbeit ist es gelungen, das differente Feld der Lernobjekte von der Theorie bis zur Implementierung zusammenzufassen und ein ganzheitliches Konzept zu liefern.

Kapitel 18

Ausblick

In dieser Arbeit haben sich neben den behandelten Fragestellungen viele weitere interessante Themen aufgetan, die nicht weiter behandelt werden konnten. Gründe hierfür sind zum einen der zu große Umfang, den eine gerecht werdende Diskussion einnehmen würde, oder der späte Zeitpunkt des Auftretens, der eine direkte Berücksichtigung unmöglich machte. In diesem Kapitel nun sollen diese Themen wenigstens angesprochen werden, um ihre wissenschaftliche Behandlung in anderen Arbeiten anzuregen.

Viele in der Arbeit vorgestellten Standards unterliegen einer stetigen Weiterentwicklung. Ein besonders wichtiger im Zusammenhang mit modularen E-Learning-Inhalten ist SCORM (siehe Abschnitt 3.6), der mittlerweile *SCORM 2004* [Dodd04a; Dodd04b; Dodd04c] heißt und neben einigen Verbesserungen auch neue Aspekte behandelt. So wird die Sequenzierung von Inhalten (siehe Abschnitt 3.4) nun ausführlich in einem eigenen Dokument behandelt [Dodd04c]. All diese Neuerungen sollten in einer nächsten Version des entwickelten Rahmenwerks einfließen, um die Standardkompatibilität auch für neue SCORM-Lernobjekte zu garantieren.

Einige Standards haben keinen Einfluss auf diese Arbeit gefunden, weil sie zum Zeitpunkt der Entwicklung noch nicht die nötige Reife erlangt hatten. Hierzu gehört das *IMS Digital Repositories Interoperability* [IMS03a], das andere Standards wie z.B. *IMS Meta-Data* und *IMS Content Packaging* nutzt, um ein digitales Repository für Lernobjekte aufzubauen. Im Grunde genommen werden in diesem Standard die formalen Rahmenbedingungen festgelegt, wie sie im Entwurf des Repositories dieser Arbeit (siehe Kapitel 15) befolgt werden. Da auch hier die Einhaltung der Standards einen wesentlichen Punkt ausmachte, sollte es ein leichtes sein, durch leichte Modifikationen ein für diesen Standard kompatibles Repository zu erzeugen.

Ein anderes Themengebiet, das in dieser Arbeit nicht wesentlich behandelt wurde, sind die Möglichkeiten bei der inhaltlichen Gestaltung von Lernobjekten. Neben den Formaten zur Kodierung der Inhalte (siehe Abschnitt 3.7) und den Metadaten (siehe Kapitel 4), gibt es Standards für das Verhalten von Lernobjekten zur Laufzeit. Da wären zum einen Quiz, mit denen die Studierenden z.B. ihren Wissensstand selbst überprüfen oder Leistungsnachweise erbringen könnten. Grundlage für den Einsatz in modularen E-Learning-Inhalten wären freilich Standards, wie z.B. *IMS Question and Test Interoperability* [IMS04b], die vom Autorensystem und von der Lernplattform korrekt verarbeitet werden. Zu den ergänzenden Verfahren gehört auch das Sammeln von Daten über die Studierenden, die z.B. nach einem Standard wie dem *IMS Learner Information Package* [IMS05] in der Lernplattform oder dem Repository gespeichert werden.

Den Urhebern/-innen digitaler Inhalte gereichen die eigentlich positiven Eigenschaften des Internets oft zum Nachteil. Über das Internet können Lernobjekte ubiquitär verbreitet werden und eine Kopie ist so gut wie das Original. Diese Verteilung lässt sich ohne geeignete Mittel nicht kontrollieren, sodass gewollt oder ungewollt die Urheberrechte schnell verletzt sind. Denn sobald Lizenzgebühren mit dem geistigen Eigentum verbunden sind, reichen einfache Copyright-Vermerke in den Metadaten nicht aus. Das zeigt sich eindeutig an der Be-

liebtheit freier P2P-Netzwerke (*Peer-to-Peer*), in denen vorwiegend Raubkopien gehandelt werden. Es müssen technische Vorkehrungen getroffen werden, mit denen die Verbreitung und Nutzung kontrolliert wird. Der dahinter stehende Mechanismus nennt sich *Digital Rights Management* (DRM) [Downes03; Iannella01] und wird in anderen Medienbereichen, wie z.B. Handy-Klingeltöne, Musikstücke oder Videos bereits kommerziell genutzt. Inhalte werden verschlüsselt übertragen und können nur mit spezieller Software oder Hardware in das ursprüngliche Format übertragen werden. Voraussetzung hierfür ist eine gültige Lizenz.

Auf dem Gebiet der Lernobjekte hat diese Technik in der Praxis noch keinen Einzug gefunden und auch die Standardisierung ist nicht weit fortgeschritten. Hier gibt es noch Forschungsgebiete, die wissenschaftlich erschlossen werden müssen, um sie in das Rahmenwerk dieser Arbeit zu integrieren. Auf jeden Fall ist DRM für die kommerzielle Nutzung von modularen E-Learning-Inhalten ein wichtiges Thema, das zukünftige Systeme anbieten sollten. Ein Beispiel für ein einfaches DRM-Rahmenwerk für Lernobjekte ist z.B. in [Santos04] zu finden.

Ein anderes interessantes Thema im Zusammenhang mit Lernobjekten ist die Einführung semantischer Ontologien, um Dokumente untereinander zu verknüpfen [Krieg-Brückner04]. Dies könnte ein Schlüssel in Richtung personalisiertes Lernen sein, bei dem Lernende ihr eigenes Wissen mit genau auf ihre Bedürfnisse angepassten Lernmaterialien aufbauen. Jeder Mensch bringt beim Lernen andere Voraussetzungen mit, denen mit individuellen, angepassten Lernpfaden [Farrell04; Atif03] begegnet werden soll. Hierfür muss das System Annahmen treffen, die auf Daten über die Lernenden, z.B. durch automatische Tests oder Bewertungen durch Lehrende, und den Metadaten der Lernobjekte beruhen. Auch auf diesem Gebiet gibt es zur Zeit mehr Fragen als Antworten, die von der Wissenschaft erst noch beantwortet werden müssen. Erste Ansätze gibt es z.B. in [Dolog04; Brusilovsky04]. Andere Systeme versuchen einen einfacheren Weg einzuschlagen, indem sie Empfehlungen für einzelne Lernobjekte geben [Rashid02]. Das Rahmenwerk dieser Arbeit sollte um entsprechende Algorithmen für personalisiertes Lernen erweitert werden, sobald die Forschung einen akzeptablen Stand erreicht hat.

Für das Problem des Auffindens geeigneter Lernobjekte gibt es auch andere Vorschläge, die nicht auf direkte Automatisierung setzen. Menschen können besser abschätzen, was sich hinter einem bestimmten Lernobjekt verbirgt, welche Voraussetzungen für einen erfolgreichen Einsatz notwendig sind und wie sie sich mit anderen Lernobjekten kombinieren lassen. Die Daten hierüber werden an einer zentralen Stelle gespeichert und anderen Interessierten zur Verfügung gestellt. Solch ein Bewertungssystem für Lernobjekte bietet z.B. das *Learning Object Review Instrument*[1] (LORI) an [Leacock04; Nesbit04]. Eine Überlegung für die zukünftige Entwicklung des Rahmenwerks ist die Integration eines Bewertungssystems in das Rahmenwerk, um es an verschiedenen Stellen, z.B. direkt im Autorensystem oder im Repository, zur Verfügung zu stellen.

[1] `http://www.eLera.net` (29.10.05)

Literaturverzeichnis

[Ahronheim98] Judith R. Ahronheim: Descriptive metadata: Emerging standards. In: *Journal of Academic Librarianship*, 24(5), 1998, pp. 395–403.

[Aliprand03] Joan Aliprand, Julie Allen and Ken Whistler (eds.): *The Unicode Standard Version 4.0.* Addison Wesley, 2003.

[Ammelburger03] Dirk Ammelburger: *XML.* Hanser Fachbuchverlag, 2003.

[Aristoteles82] Aristoteles: *Poetik, griechisch-deutsch.* Übersetzt von Manfred Fuhrmann, Reclam, 1982.

[Atif03] Yacine Atif, Rachid Benlamri and Jawad Berri: Learning objects based framework for self-adaptive learning. In: *Education and Information Technologies*, 8(4), 2003.

[Atkinson69] Richard C. Atkinson and H. A. Wilson (eds.): *Computer-assisted instruction: a book of readings.* Academic Press, 1969.

[Baker00] Thomas Baker: A grammar of dublin core. In: *D-Lib Magazine*, 6(10), 2000.

[Baker01] Thomas Baker, Makx Dekkers, Rachel Heery, Manjula Patel and Gauri Salokhe: What terms does your metadata use? application profiles as machine-understandable narratives. In: *Journal of Digital Information*, 2(2), 2001.

[Baker03] Thomas Baker, Thomas Baker, Dan Brickley, Erik Duval, Erik Duval, Pete Johnston, Pete Johnston, Heike Neuroth and Heike Neuroth: Principles of metadata registries. Technical report, DELOS Working Group on Registries, 2003.

[Balbieris02] Giedrius Balbieris and Vytautas Reklaitis: Reshaping e-learning content to meet the standards. In: *Informatics in Education*, 1(1), 2002, pp. 5–16.

[Balzert00] Helmut Balzert: *Lehrbuch der Software-Technik.* Spektrum Akademischer Verlag, 2000.

[Bauch03] Manfred Bauch and Luise Unger: Interactive mathematics with math-kit — distance learning versus face-to-face learning. In: *Proc. 21st ICDE World Conference on Open Learning & Distance Education*, 2003.

[Baudry02a] A. Baudry, M. Bungenstock and B. Mertsching: Architecture of an e-learning system with embedded authoring support. In: Margaret Driscoli and Thomas C. Reeves (eds.): *E-LEARN 2002–World Conference on E-Learning in Corp., Govt., Health., & Higher Ed.*, 2002, pp. 110–116.

[Baudry02b] A. Baudry, M. Bungenstock and B. Mertsching: Ein multimediales Rahmenwerk für die Mathematiklehre nach der Baukastenmetapher. In: Klaus P. Jantke, Wolfgang S. Wittig and Jörg Herrmann (eds.): *Von e-Learning bis e-Payment. Das Internet als sicherer Marktplatz (LIT '02).* Infix, 2002, pp. 300–307.

[Baudry03] A. Baudry, M. Bungenstock and B. Mertsching: Nyx - a tool for generating standard compatible e-learning courses with consistent and adaptable presentation. In: *The IASTED International Conference on Computers and Advanced Technology in Education (CATE 2003)*, 2003, pp. 265–269.

[Baudry04a] A. Baudry, M. Bungenstock and B. Mertsching: Administration and development of modular learning units with the construction kit server. In: *Proc. ED-MEDIA 2004–World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2004.

[Baudry04b] A. Baudry, M. Bungenstock and B. Mertsching: Reusing document formats for modular course development. In: *IASTED International Conference on WEB-BASED EDUCATION (WBE 2004)*, 2004, pp. 535–537.

[Baumgartner97] Peter Baumgartner and Sabine Payr: *Konstruktivismus und Kognitionswissenschaft. Kulturelle Wurzeln und Ergebnisse*, chapter Erfinden lernen, pp. S. 89–106. Springer, 1997.

[Baumgartner99] Peter Baumgartner and Sabine Payr: *Lernen mit Software.* Studienverlag Ges. mbH, 1999. ISBN:3706514443.

[Baumgartner02a] Peter Baumgartner, Hartmut Häfele and Kornelia Maier-Häfele: *E-Learning Praxishandbuch, Auswahl von Lernplattformen.* Studien Verlag, 2002.

[Baumgartner02b] Peter Baumgartner, Kornelia Häfele and Hartmut Häfele: E-Learning: Didaktische und technische Grundlagen. In: *CD Austria*, 5, pp. 4–32.

[Bearman99] David Bearman, Godfrey Rust, Stuart Weibel, Eric Miller and Jennifer Trant: A common model to support interoperable metadata. In: *D-Lib Magazine*, 5(1), 1999.

[Beckett03] Dave Beckett. Rdf/xml syntax specification (revised), December 2003.

[Belqamsi02] Youssef Belqamsi: Using XML in elearning courses. In: *E-LEARN 2002–World Conference on E-Learning in Corp., Govt., Health., & Higher Ed.*, 2002, pp. 1183–1186.

[Bentley95] Richard Bentley, Thilo Horstmann, Klaas Sikkel and Jonathan Trevor: Supporting collaborative information sharing with the world wide web: The bscw shared workspace system. In: *The World Wide Web Journal: Proceedings of the 4th International WWW Conference*, pp. 63–74.

[Berners-Lee94] T. Berners-Lee, L. Masinter and M. McCahill: Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force, December 1994.

[Berners-Lee98] T. Berners-Lee, R. Fielding, U.C. Irvine and L. Masinter: Uniform Resource Identifiers (URI). RFC 2396, Internet Engineering Task Force, August 1998.

[Binstock02] Cliff Binstock, David Peterson and Mitchell Smith: *The XML Schema Complete Reference.* Addison-Wesley, 2002.

[Biron01] Paul V. Biron and Ashok Malhotra. Xml schema part 2: Datatypes, May 2001.

[Black62] Max Black: *Models and Metaphorsr. Studies in language and Philosophy.* Cornell University Press, 1962.

[Blanchi01] Christophe Blanchi and Jason Petrone: Distributed interoperable metadata registry. In: *D-Lib Magazine*, 7(12), 2001.

[Bodendorf90] Freimut Bodendorf: *Computer in der fachlichen und universitären Ausbildung.* Oldenbourg, 1990.

[Bodoff04] Stephanie Bodoff, Eric Armstrong, Jennifer Ball and Debbie Bode Carson: *The J2EE Tutorial, Second Edition.* Addison Wesley, 2004.

[Bourret99] Ronald Bourret, Christof Bornhövd and Alejandro P. Buchmann: A generic load/extract utility for data transfer between xml documents and relational databases. Technical Report DVS99-1, Department of Computer Science - Darmstadt University of Technology, December 1999.

[Britain00] Sandy Britain and Oleg Liber: A framework for the pedagogical evaluation of virtual learning environments. In: *Proc. of: ALT-C 2000*, 2000.

[Bro91] Brockhaus-Enzyklopädie, Bd. 14: mag.–mod., 1991.

[Brusilovsky04] Peter Brusilovsky: Knowledgetree: A distributed architecture for adaptive e-learning. In: *Proceedings of the 13th international World Wide Web conference on Alternate Track Papers & Posters*, 2004, pp. 104–113.

[Bungenstock02] M. Bungenstock, A. Baudry and B. Mertsching: The construction kit metaphor for a software engineering design of an e-learning system. In: Philip Barker and Samuel Rebelsky (eds.): *Proc. ED-MEDIA 2002–World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2002, pp. 216–217.

[Bungenstock03a] M. Bungenstock, A. Baudry and B. Mertsching: Data exchange between lyssa and learning management systems. In: *E-Learn 2003–World Conference on E-Learning in Corporate, Gov ernment, Healthcare, & Higher Education*, 2003, pp. 31–34.

[Bungenstock03b] M. Bungenstock, A. Baudry and B. Mertsching: Datenaustausch zwischen Lyssa und Learning Management Systemen. In: *Von e-Learning bis e-Payment. Das Internet als sicherer Marktplatz (LIT '03)*, 2003, pp. 126–132.

[Bungenstock04a] M. Bungenstock, A. Baudry and B. Mertsching: Design of a common api for learning objects. In: *The IASTED International Conference on Computers and Advanced Technology in Education (CATE 2003)*, 2004, pp. 345–350.

[Bungenstock04b] M. Bungenstock, A. Baudry and B. Mertsching: Entwicklung eines technischen Rahmenwerks für standardkompatible Lernobjekte. In: *DELFI 2004 — Tagungsband der 2. e-Learning Fachtagung Informatik*, 2004, pp. 151–162.

[Busch98] Carsten Busch: *Metaphern in der Informatik: Modellbildung, Formalisierung, Anwendung.* Deutscher Universitäts-Verlag GmbH, 1998.

[Büchmann94] Georg Büchmann and Eberhard Urban: *Der neue Büchmann. Geflügelte Worte.* Bassermann, 1994.

[Carnell03] John Carnell, J. Linwood and M. Zawadzki: *Professional Struts Applications*. Apress, 2003.

[Case78] R. Case: A developmentally based theory and technology of instruction. In: *Review of Educational Research*, 48, pp. 439–463.

[Case85] R. Case: *Thinking and Learning Skills — Research and Open Questions*, vol. 2, chapter A Developmentally based Approach to the Problem of Instructional Design, pp. 537–545. Lawrence Erlbaum Assoc, 1985.

[Cavaness04] Chuck Cavaness: *Programming Jakarta Struts*. O'Reilly, 2004.

[CEN03] European Committee For Standardization: *CEN Workshop Agreement CWA 14855: Dublin Core Application Profile Guidelines*, November 2003.

[Chapman03] Bryan Chapman: *LCMS Report: Comparative Analysis of Enterprise Learning Content Management Systems*. brandon-hall.com, 2003.

[Cis99] Cisco Systems: *Reusable Information Object Strategy*, June 1999. Version 3.0.

[Clemm99] G. Clemm, J. Amsden, T. Ellison, C. Kaler and J. Whitehead: Versioning extensions to webdav. RFC 3253, Network Working Group, Februar 1999.

[CTGV90] CTGV: Anchored instruction and its relationship to situated cognition. In: *Educational Researcher*, 19(6), 1990, pp. 2–10.

[CTGV93] CTGV: Anchored instruction and situated cognition revisted. In: *Educational Technology*, 33(3), 1993, pp. 52–70.

[Dahn01] Ingo Dahn: Slicing book technology — providing online support for textbooks. In: *ICDE 2001, International Conference on Distant Education*, 2001.

[Dahn02] Ingo Dahn, Michael Armbruster, Ulrich Furbach and Gerhard Schwabe: *Writing hypertext and learning: Conceptual and empirical approaches*, chapter Slicing Books The Authors' Perspective. Pergamon, 2002.

[Dawson98] F. Dawson and T. Howes: vcard mime directory profile. RFC 2426, Network Working Group, September 1998.

[Dekkers01] Makx Dekkers: Application profiles, or how to mix and match metadata schemas. In: *Cultivate Interactive*, 3.

[Depke99] Ralph Depke, G. Engels, K. Mehner, S. Sauer and A. Wagner: Ein Vorgehensmodell für die Multimedia-Entwicklung mit Autorensystemen. In: *Informatik Forschung und Entwicklung*, 14, pp. 83–94.

[Deutsch96] Peter Deutsch: DEFLATE compressed data format specification version 1.3. RFC 1951, Aladdin Enterprises, May 1996.

[Dittler03] Ullrich Dittler (ed.): *E-Learning : Einsatzkonzepte und Erfolgsfaktoren des Lernens mit interaktiven Medien*. Oldenbourg, 2003.

[Dodd04a] Philip Dodd and Schawn E. Thropp: Sharable content object reference model (scorm) — content aggregation model (cam) version 1.3.1. Technical report, Advanced Distributed Learning, 2004.

[Dodd04b] Philip Dodd and Schawn E. Thropp: Sharable content object reference model (scorm) — run-time environment (rte) version 1.3.1. Technical report, Advanced Distributed Learning, 2004.

[Dodd04c] Philip Dodd and Schawn E. Thropp: Sharable content object reference model (scorm) 2004 — overview. Technical report, Advanced Distributed Learning, 2004.

[Dolog04] Peter Dolog, Nicola Henze, Wolfgang Nejdl and Michael Sintek: Personalization in distributed e-learning environments. In: *Proceedings of the 13th international World Wide Web conference on Alternate Track Papers & Posters*, 2004, pp. 170–179.

[Downes00a] Stephen Downes. Learning objects. Presented at Leaders in Learning 2000, May 2000.

[Downes00b] Stephen Downes: Nine rules for good technology. In: *The Technology Source.*

[Downes02] Stephen Downes. The learning object economy. Published by Contact North, October 2002.

[Downes03] Stephen Downes, Magda Mourad, Harry Piccariello and Robby Robson: Digital rights management in e-learning — problem statement and terms of reference. In: *E-Learn 2003–World Conference on E-Learning in Corporate, Gov ernment, Healthcare, & Higher Education*, 2003.

[Dreyfus86] Hubert L. Dreyfus, Stuart E. Dreyfus and T. Anthanasiou: *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer.* The Free Press, 1986. ISBN: 0743205510.

[Dub99] Dublin Core Metadata Initiative: *Dublin Core Metadata Element Set, Version 1.1: Reference Description*, July 1999.

[Duval00] Eric Duval, E. Vervaet, B. Verhoeven, K. Hendrikx, K. Cardinaels, H. Olivié, E. Forte, F. Haenni, K. Warkentyne, M. Wentland-Forte and F. Simillion: Managing digital educational resources with the ariadne metadata system. In: *Journal of Internet Cataloging*, 3(2/3), 2000, pp. 145–171.

[Duval02] Erik Duval, Wayne Hodgins, Stuart Sutton and Stuart L. Weibel: Metadata principles and practicalities. In: *D-Lib Magazine*, 8(4), 2002.

[Duval03] Erik Duval and Wayne Hodgins: A lom research agenda. In: *WWW2003 - Twelfth InternationalWorldWideWeb Conference*, May 2003.

[Englander97] Robert Englander: *Developing Java Beans.* O'Reilly, 1997.

[Farrell04] Robert G. Farrell, Soyini D. Liburd and John C. Thomas: Dynamic assembly of learning objects. In: *Proceedings of the 13th international World Wide Web conference on Alternate Track Papers & Posters*, 2004, pp. 162–169.

[Farreres03] Javier Farreres: *The Dsssl Book: An XML/SGML Programming Language.* Kluwer Academic Publishers, 2003.

[Foerster95] Heinz von Foerster: *Die erfundene Wirklichkeit. Wie wissen wir, was wir zu wissen glauben.*, chapter Das Konstruieren einer Wirklichkeit. Paul Watzlawick and Peter Krieg, 1995.

[Freed96] N. Freed and N. Borenstein: Multipurpose internet mail extensions — (mime) part two: Media types. RFC 2046, Network Working Group, November 1996.

[Freitag02a] Burkhard Freitag: LMML — Eine XML-Sprachfamilie für eLearning Content. In: *Informatik bewegt: Informatik 2002 — 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI)*, 2002, pp. 349–353.

[Freitag02b] Burkhard Freitag, Christian Süß and Claus Dziarstek: Adaptation und Wiederverwendung von XML-basiertem eLearning-Content. In: *Informatik bewegt: Informatik 2002 — 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI)*, 2002, pp. 354–358.

[Friesen02] Norm Friesen, Jon Mason and Nigel Ward: Building educational metadata application profiles. In: *Proceedings of the International Conference on Dublin Core and Metadata for e-Communities 2002*, 2002, pp. 63–69.

[Gamma95] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides: *Desing Patterns: elements or reusable object-oriented software*. Addison Wesley, 1995.

[Gibbons02] Andrew S. Gibbons, Jon Nelson and Robert Richards: *The Instructional Use of Learning Objects*, chapter The Nature and Origin of Instructional Objects. AIT/AECT, 2002.

[Gill98] Tony Gill, Anne Gilliland-Swetland and Murtha Baca: *Introduction to Metadata: Pathways to Digital Information*. Getty Research Institute, 1998.

[Goland99] Y. Goland, E. Whitehead, A. Faizi, S. Carter and D. Jensen: Http extensions for distributed authoring — webdav. RFC 2518, Network Working Group, Februar 1999.

[Goldfarb91] Charles F. Goldfarb and Yuri Rubinsky: *SGML Handbook*. Oxford University Press, 1991.

[Gosling96] James Gosling, Bill Joy and Guy L. Steele: *The Java Language Specification*. Addison Wesley, 1996.

[Gourley02] David Gourley and Brian Totty: *HTTP, The Definitive Guide*. O'Reilly, 2002.

[Griffel98] Frank Griffel: *Componentware*. dpunkt-Verlag, 1998.

[Griffin97] Steve Griffin and Tom Wason: The year of metadata. In: *Educom Review*, 32(6), 1997, pp. 56–58.

[Hall00] Brandon Hall: *Learning Management Systems 2001: How to Choose the Right System for Your Organization*. brandon-hall.com, 2000.

[Hall03] Brandon Hall: *LMS 2003: Comparison of Enterprise Learning Management Systems*. brandon-hall.com, 2003.

[Hansch02] Matthias Hansch, Stefan Kuhlins and Martin Schader: XML Schema. In: *Informatik Spektrum*, 25(3), 2002, pp. 363–366.

[Haverkamp83] Anselm Haverkamp: *Theorie der Metapher*, chapter Die Metapher, von Max Black. Wissenschaftliche Buchgesellschaft, 1983.

[Heery00] Rachel Heery and Manjula Patel: Application profiles: mixing and matching metadata schemas. In: *Ariadne*, 25.

[Heery02] Rachel Heery, Pete Johnston, Dave Beckett and Damian Steer: The meg registry and scart: complementary tools for creation, discovery and re-use of metadata schemas. In: *Proceedings of the International Conference on Dublin Core and Metadata for e-Communities 2002*, 2002, pp. pp 63–69.

[Heery03] Rachel Heery, Pete Johnston, Csaba Fülöp and András Micsik: Metadata schema registries in the partially semantic web: the cores experience. In: *Proceedings of the 2003 Dublin Core Conference: Supporting Communities of Discourse and Practice - Metadata Research and Applications*, September/October 2003.

[Hjelm01] Johan Hjelm: *Creating the Semantic Web with RDF*. John Wiley & Sons, 2001.

[Hodgins00] Wayne Hodgins. Into the future - a vision paper. erhältlich bei Commission on Technology & Adult Learning of the American Society for Training & Development (ASTD) und National Governors' Association (NGA), February 2000.

[Hodgins02] Wayne Hodgins: The future of learning objects. In: *Proc. of the 2002 eTEE Conference*, August 2002, pp. 76–82.

[Howes98] T. Howes, M. Smith and F. Dawson: A MIME content-type for directory information. RFC 2425, Network Working Group, September 1998.

[Humbert05] Ludger Humbert: *Didaktik der Informatik*. Teubner, 2005. ISBN: 3835100386.

[Hunter01] Jane Hunter and Carl Lagoze: Combining rdf and xml schemas to enhance interoperability between metadata application profiles. In: *Proc. of the Tenth International Conference on World Wide Web*. ACM, 2001, pp. 457–466.

[Häfele03] Hartmut Häfele and Kornelia Maier-Häfele. Autorenwerkzeuge für Learning Content. Portal des bm:bwk bildung.at, 2003.

[Iannella01] Renato Iannella: Digital rights management (drm) architectures. In: *D-Lib Magazine*, 7(6), 2001.

[IEE] IEEE P1484.12: *Standard for Resource Description Framework (RDF) binding for Learning Object Metadata data model*.

[IEE02a] IEEE P1484.12: *Draft Standard for Learning Object Metadata*, 2002.

[IEE02b] IEEE P1484.12: *Standard for XML binding for Learning Object Metadata data model*, 2002.

[IMS01] IMS. IMS learning resource meta-data information model — version 1.2.1 final specification, 2001.

[IMS03a] IMS. IMS digital repositories interoperability, 2003.

[IMS03b] IMS. Ims learning resource meta-data best practice and implementation guide - version 1.2.1 final specification, 2003.

[IMS04a] IMS. IMS content packaging information model — version 1.1.4 final specification, 2004.

[IMS04b] IMS. IMS question and test interoperability, 2004.

[IMS05] IMS. Ims learner information package, 2005.

[Ingendahl71] Werner Ingendahl: *Der methaphorische Prozess. Methodologie zu seiner Erforschung und Systematisierung.* Pädagogischer Verlag Schwann, 1971.

[Int88] International Organization for Standardization (ISO): *ISO 639-1:2002. Codes for the representation of names of languages — Part 1: Alpha-2 code*, first edition, April 1988.

[Int97] International Organization for Standardization (ISO): *ISO 3166-1:1997. Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*, first edition, September 1997.

[Int98] International Organization for Standardization (ISO): *ISO 639-2:1998. Codes for the representation of names of languages — Part 2: Alpha-3 code*, first edition, November 1998.

[Int00] International Organization for Standardization (ISO): *ISO 8601:2000. Data elements and interchange formats — Information interchange — Representation of dates and times*, second edition, December 2000.

[Int02] International Organization for Standardization (ISO): *ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*, second edition, December 2002.

[Johnson04] Rod Johnson and Juergen Hoeller: *Expert One-on-One J2EE Development without EJB.* Wrox, 2004.

[Kerres98] M. Kerres: *Multimediale und telemediale Lernumgebungen - Konzeption und Entwicklung.* Oldenbourg Verlag, 1998.

[Klein99] Gary Klein: *Sources of Power: How People Make Decisions.* The MIT Press, 1999. ISBN: 0262611465.

[Klimsa93] Paul Klimsa: *Neue Medien und Weiterbildung. Anwendung und Nutzung in Lernprozessen der Weiterbildung.* Deutscher Studienverlag, 1993.

[Kohlhase00] Michael Kohlhase: Omdoc: An infrastructure for openmath content dictionary information. In: *Bulletin of the ACM Special Interest Group for Algorithmic Mathematics SIGSAM*, 2000.

[Kohlhase02] Michael Kohlhase: Omdoc: An open markup format for mathematical documents (version 1.1). Technical report, Carnegie Mellon University, 2002.

[Kolodner93] Janet Kolodner: *Case-Based Reasoning.* Morgan Kaufmann Publishers, 1993.

[Kortzfleisch99] Harald von Kortzfleisch, Ulrike Heller and Udo Winand: *Perspektiven der Medienwirtschaft. Kompetenz, Akzeptanz, Geschäftsfelder. Telekommunikation & Mediendienste 5*, chapter Das "Forum Virtuelle Lernwelten", pp. 51–73. Josef Eul Verlag, 1999.

[Krieg-Brückner04] Bernd Krieg-Brückner, Arne Lindow, Christoph Lüth, Achim Mahnke and George Russell: Semantic interrelation of documents via an ontology. In: *DELFI 2004 — Tagungsband der 2. e-Learning Fachtagung Informatik*, 2004, pp. 271–282.

[Leacock04] Tracey L. Leacock, Griff Richards and John C. Nesbit: Teachers need simple, effective tools to evaluate learning objects: Enter elera.net. In: *Seventh IASTED International Conference — Computers and Advanced Technologies in Education*, 2004, pp. 333–338.

[Lenz98] Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard and Stefan Wess (eds.): *Case-Based Reasoning Technology: From Foundations to Applications.* Springer Verlag, 1998.

[Letts02] Mike Letts: ADL and SCORM: Creating a standard model for publishing courseware. In: *Seybold Report - Analyzing Publishing Technologies*, 2(1), 2002, pp. 3–8.

[Louden94] Kenneth C. Louden: *Programmiersprachen. Grundlagen, Konzepte, Entwurf.* VMI Buch AG, 1994.

[Low02] Boon Low: Packaging educational content using IMS specifications. In: *VINE*, 127, pp. 40–46.

[Manola03] Frank Manola and Eric Miller. Rdf primer, December 2003.

[Martinez00] Margaret Martinez: *The Instructional Use of Learning Objects*, chapter Designing Learning Objects to Personalize Learning. AIT/AECT, 2000.

[Mason00] Jon Mason, Graham Adcock and Albert IP: Modeling information to support value-adding: Edna online. In: *WebNet Journal: Internet Technologies, Applications & Issues*, 2(3), 2000, pp. 38–45.

[McDaniel03] Mason McDaniel and M. Hossain Heydari: Content based file type detection algorithms. In: *Proceedings of the 36th Hawaii International Conference on System Sciences - 2003*, 2003.

[Metzinger99] Thomas Metzinger: *Subjekt und Selbstmodell.* Mentis-Verlag, 1999. ISBN: 3897850818.

[Meyer97] Bertrand Meyer: *Object-Oriented Software Construction.* Prentice Hall, 1997.

[Meyer02] Eric A. Meyer: *On CSS.* New Riders Publishing, 2002.

[Milligan00] Colin Milligan: The role of virtual learning environments in the online delivery of staff development. Technical Report 44, Joint Information Systems Committee, 2000.

[Minsky94] Marvin L. Minsky: *Mentopolis.* Klett-Cotta, 1994.

[Mintert02] Stefan Mintert: *XML & Co. Die W3C-Spezifikationen für Dokumenten- und Datenarchitektur.* Addison-Wesley, 2002.

[Mosley05] Pauline Mosley: A taxonomy for learning object technology. In: *J. Comput. Small Coll.*, 20(3), 2005, pp. 204–216.

[Nagamori01] Mitsuharu Nagamori, Thomas Bakery, Tetsuo Sakaguchi and Tetsuo Sakaguchi Mitsuharu Nagamori, Thomas Bakery: A multilingual metadata schema registry based on rdf schema. In: *Proc. Int l. Conf. on Dublin Core and Metadata Applications 2001*, 2001, pp. pp 209–212.

[Najjar03] Jehad Najjar, Stefaan Ternier and Erik Duval: The actual use of metadata in ariadne: an empirical analysis. In: *3rd Annual Ariadne Conference*, 2003.

[Negroponte96] Nicholas Negroponte: *Being Digital.* Vintage, 1996.

[Nejdl02] Wolfgang Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér and T. Risch: Edutella: A p2p networking infrastructure based on rdf. In: *Proc. of the Eleventh International Conference on World Wide Web.* ACM, 2002, pp. 604–615.

[Nesbit04] John C. Nesbit, Tracey L. Leacock and Cindy Xin: Learning object evaluation and convergent participation: Tools for professional development in e-learning. In: *Seventh IASTED International Conference — Computers and Advanced Technologies in Education*, 2004, pp. 339–344.

[Neven02] Filip Neven and Erik Duval: Reusable learning objects: a survey of lom-based repositories. In: *Proc. of ACM Multimedia.* ACM, 2002, pp. 291–294.

[Newman03] Tabetha Newman: Scorm force. In: *Conspectus*, pp. 18–20.

[Niegemann04] Helmut M. Niegemann, Silvia Hessel, Dirk Hochscheid-Mauel, Kristina Aslanski, Markus Deimann and Gunther Kreuzberger: *Kompendium E-Learning.* Springer-Verlag, 2004.

[Nilsson03] Mikael Nilsson, Matthias Palmér and Jan Brase. The lom rdf binding - principles and implementation. Paper to 3rd Annual Ariadne Conference, Katholieke Universiteit Leuven, Belgium, November 2003.

[Obj03] Object Management Group: *Unified Modeling Language (UML), version 1.5*, formal/03-03-01 edition, 2003.

[Padberg02a] Kathrin Padberg and Sabine Schiller: Web-based drills in maths using a computer algebra system. In: *Proc. ED-MEDIA 2002–World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2002.

[Padberg02b] Kathrin Padberg and Andreas Sorgatz: Webbasierte Übungselemente mit MuPAD. In: *Computeralgebra in Lehre, Ausbildung und Weiterbildung III*, 2002.

[Pawlowski01] Jan M. Pawlowski: *Das Essener-Lern-Modell (ELM): Ein Vorgehensmodell zur Entwicklung computerunterstützter Lernumgebungen.* PhD thesis, Universität Essen, 2001.

[Pawson02] Dave Pawson: *XSL-FO.* O'Reilly, 2002.

[Powers03] Shelley Powers: *Practical RDF.* O'Reilly, 2003.

[Qin04] Jian Qin and Naybell Hernández: Ontological representation of learning objects: building interoperable vocabulary and structures. In: *Proceedings of the 13th international World Wide Web conference on Alternate Track Papers & Posters*, 2004, pp. 348–349.

[Rashid02] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan and John Riedl: Getting to know you: Learning new user preferences in recommender systems. In: *Proceedings of the 7th international conference on Intelligent user interfaces*, 2002, pp. 127–134.

[Ray01] Erik T. Ray: *Learning XML — Guide to Creating Self-Describing Data*. O'Reilly, 2001.

[Rehberg03] Bettina Rehberg and Ulrich Rehberg: Efficient development of multimedia applets for elearning. In: *Proc. 21st ICDE World Conference on Open Learning & Distance Education*, 2003.

[Reigeluth80] Charles M. Reigeluth, M.D. Merrill, B.G. Wilson and R.T. Spiller: The elaboration theory of instruction: A model for structuring instruction. In: *Instructional Science*, 9, pp. 125–219.

[Reigeluth83] Charles M. Reigeluth and F.S. Stein: *Instructional Design Theories and Models: An Overview of Their Current Status*, chapter The Elaboration Theory of Instruction. Lawrence Erlbaum, 1983.

[Reigeluth99] Charles M. Reigeluth: *Instructional Design — Theories and Models. A New Paradigm of Instructional Theory*, chapter The Elaboration Theory: Guidance for Scope and Sequence Decisions, pp. 425–453. Lawrence Erlbaum, 1999.

[Roisin98] Cécile Roisin: Authoring structured multimedia documents. In: *SOFSEM '98: Theory and Practice of Informatics: 25th Conference on Current Trends in Theory and Practice of Informatics*, 1998, pp. 222–239.

[Saddik00] Abdulmotaleb El Saddik, Amir Ghavam, Stephan Fischer and Ralf Steinmetz: Metadata for smart multimedia learning objects. In: *Proc. of the Australasian conference on Computing education*, December 2000, pp. 87–94.

[Saddik01] Abdulmotaleb El Saddik, Stephan Fischer and Ralf Steinmetz: Reusability and adaptability of interactive resources in web-based educational systems. In: *Journal on Educational Resources in Computing (JERIC)*, 1(4), 2001.

[Santos04] Osvaldo A. Santos and Fernando M. S. Ramos: Proposal of a framework for internet based licensing of learning objects. In: *Comput. Educ.*, 42(3), 2004, pp. 227–242.

[Schiller02] Sabine Schiller and Luise Unger: Math-kit: a multimedia project for learning and teaching mathematics. In: *Proceedings 10th Meeting of European Women in Mathematics*, 2002, pp. 383–386.

[Schulmeister00] Rolf Schulmeister: Selektions- und Entscheidungskriterien für die Auswahl von Lernplattformen und Autorenwerkzeugen. Technical report, Österreichisches Bundesministeriums für Bildung, Wissenschaft und Kultur (bm:bwk), 2000.

[Schulmeister01] Rolf Schulmeister: *Virtuelle Universität — Virtuelles Lernen*. Oldenbourg, 2001.

[Schulmeister03] Rolf Schulmeister: *Lernplattformen für das virtuelle Lernen*. Oldenbourg, 2003.

[Schwabe01] Gerhard Schwabe, Norbert Streitz and Raine Unland (eds.): *CSCW-Kompendium Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten* . Springer, 2001.

[Schöning03] Harald Schöning: *XML und Datenbanken*. Carl Hanser Verlag, 2003.

[Searle86] John R. Searle: *Geist, Hirn und Wissenschaft : die Reith lectures 1984*. Suhrkamp, 1986.

[Seifert80] Walter Seifert: *Sprachbetrachtung und Kommunikationsanalys*, chapter Didaktik rhetorischer Figuren: Metapher als Unterrichtsgegenstand, pp. 129–138. Königstein/Taunus, 1980.

[Shackelford02] Bill Shackelford: A scorm odyssey. In: *T+D*, 56(8), 2002, pp. 31–35.

[Shannon04] Bill Shannon, Mark Hapner, Vlada Matena, James Davidson, James Davidson and Larry Cable: *Java 2 Platform, Enterprise Edition: Platform and Component Specifications*. Addison Wesley, 2004.

[Shore85] John Shore: *Sachertorte Algorithm and Other Antidotes to Computer Anxiety*. Viking Press, 1985.

[Simon01] Bernd Simon: *E-Learning an Hochschulen: Gestaltungsräume und Erfolgsfaktoren von Wissensmedien*. Josef Eul Verlag, 2001.

[Simpson02] John E. Simpson: *XPath and XPointer*. O'Reilly, 2002.

[Skinner54] Burrhus F. Skinner: The science of learning, and the art of teaching. In: *Harvard Educational Review*, 24(2), 1954, pp. 86–97.

[Slein98] J. Slein, F. Vitali, E. Whitehead, U.C. Irvine and D. Durand: Requirements for a distributed authoring and versioning protocol for the world wide web. RFC 2291, Network Working Group, Februar 1998.

[South02] Joseph B. South and David W. Monson: *The Instructional Use of Learning Objects*, chapter A University-wide System for Creating, Capturing, and Delivering Learning Objects. AIT/AECT, 2002.

[Spiro88] Rand J. Spiro, Richard L. Coulson, Paul J. Feltovich and Michael J. Jacobson: Cognitive flexibility theory: Advanced knowledge acquisition in ill-structured domains. In: *Proceedings of the 10th Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, 1988.

[Spiro91] Rand J. Spiro, Paul J. Feltovich, Michael J. Jacobson and Richard L. Coulson: Cognitive flexibility, constructivism, and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains. In: *Educational Technology*, 31(5), 1991, pp. 24–33.

[Stevens94] W. Richard Stevens: *TCP/IP Illustrated, Vol.1 : The Protocols*. Addison-Wesley, 1994.

[Stevens96] W. Richard Stevens: *TCP/IP Illustrated, Vol.3 : TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*. Addison-Wesley, 1996.

[Sun99] Sun Microsystems, Inc.: *WebNFS Client SDK*, 1999.

[Sun01] Sun Microsystems, Inc.: *Java 2 Platform, Standard Edition, v 1.3.1, API Specification*, 2001.

[Sweet85] Richard E. Sweet: The mesa programming environment. In: *SIGPLAN Notices*, 20(7), 1985, pp. 216–229.

[Szyperski98] Clemens Szyperski: *Component Software: beyond object-oriented software*. Addison Wesley, 1998.

[Tanenbaum97] Andrew S. Tanenbaum and Albert S. Woodhull: *Operating Systems: Design and Implementation — 2nd ed.* Prentice Hall, 1997.

[Tate04] Bruce A. Tate and Justin Gehtland: *Better, Faster, Lighter Java*. O'Reilly, 2004.

[Teege02] Gunnar Teege and Peter Breitling: Targeteam: Adaptierbare Lehrinhalt auf Basis on XML und XSLT. In: *Informatik bewegt: Informatik 2002 — 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI)*, 2002, pp. 364–368.

[Thiere03a] Bianca Thiere, Gudrun Oevel and Kathrin Padberg: Mathematics in engineering education with math-kit. In: *Proc. of 7th Baltic Region Seminar on Engineering Education*, Septembe 2003.

[Thiere03b] Bianca Thiere, Kathrin Padberg and Gudrun Oevel: Learning mathematics through a multimedia construction kit. In: *Proc. SITE2003*, March 2003, pp. 24–29.

[Tidwell01] Doug Tidwell: *XSLT*. O'Reilly, 2001.

[Tulodziecki96] G. Tulodziecki, W. Hagemann, B. Herzig, S. Leufen and C. Mütze: *Neue Medien in den Schulen: Projekte-Konzepte-Kompetenzen*. Verlag Bertelsmann Stiftung, 1996.

[Turan04] Nurdan Turan and Yıldız Sünneli: Entwicklung einer Software-Komponente zur Integration bestehender Lehr- und Lernmaterialien und deren Metadaten in das mαth-kit-System. Master's thesis, Universität Hamburg, 2004.

[Turner03] James Turner and Kevin Bedell: *Struts*. Addison-Wesley, 2003.

[Unger02] Luise Unger, Gudrun Oevel and Bärbel Mertsching: Web-based teaching and learning with math-kit. In: *Proc. 2th International Conference on the Teaching of Mathematics*, 2002.

[Unger04] Luise Unger, M. Bauch, A. Baudry, M. Bungenstock, B. Mertsching, G. Oevel, K. Padberg and B. Thiere: math-kit — Ein multimedialer Baukasten für die Mathematikausbildung im Grundstudium. In: *Softwaretechnik-Trends*, 24(1), 2004, pp. 62–71.

[Vitali99] Fabio Vitali: Versioning hypermedia. In: *ACM Comput. Surv.*, 31(4es), 1999, pp. 24.

[Vlist02] Eric van der Vlist: *XML Schema*. O'Reilly, 2002.

[Vollmann04] Marc Vollmann: Modellierung und Implementation eines Werkzeugs mit Methoden fallbasierten Schließens zur generischen Anbindung an schlagwortbasierte Wissenssysteme. Master's thesis, Universität Hamburg, 2004.

[Walsh02a] Norman Walsh: The docbook document type - committee specification 4.2. Technical report, OASIS, 2002.

[Walsh02b] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc., 2002. Version 2.0.8.

[Wessner00] Martin Wessner and Hans-Rüdiger Pfister: Points of cooperation: Integrating cooperative learning into web-based courses. In: *Proceedings of NTCL2000 International Workshop on New Technologies for Collaborative Learning*, 2000, pp. 33–41.

[Wiley99] David A. Wiley. The post-lego learning object. Homepage, 1999.

[Wiley00a] David A. Wiley: *Learning object design and sequencing theory*. PhD thesis, Department of Instructional Psychology and Technology Brigham Young University, 2000.

[Wiley00b] David A. Wiley, Mimi Recker and Andy Gibbons. A reformulation of the issue of learning object granularity and its implications for the design of learning objects. Homepage, 2000.

[Wiley02] David A. Wiley: *The Instructional Use of Learning Objects*, chapter Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. AIT/AECT, 2002.

[Wolff82] Gerhart Wolff: *Metaphorischer Sprachgebrauch*. Reclam, 1982.

[Wollowski02] Michael Wollowski: Xml based course websites. In: *E-LEARN 2002–World Conference on E-Learning in Corp., Govt., Health., & Higher Ed.*, 2002, pp. 1043–1048.

[Wozniak94] Robert H. Wozniak: *Reflex, habit and implicit response: The early elaboration of theoretical and methodological behaviourism*, chapter Behaviourism: the early years. Routledge/Thoemmes Press, 1994.

[Wright95] Gary R. Wright and W. Richard Stevens: *TCP/IP Illustrated, Vol.2 : The Implementation*. Addison-Wesley, 1995.

[Züllighoven98] Heinz Züllighoven, Dirk Bäumer and Wolf-Gideon Bleek: *Das objektorientierte Konstruktionshandbuch*. Dpunkt Verlag, 1998.