Hochschule für Technik und Wirtschaft Dresden (FH)
Fachbereich Informatik/Mathematik

Diplomarbeit

im Studiengang Medieninformatik

Thema : **Anwendung objektorientierter Konzepte mit ActionScript 2 zur Realisierung wieder verwendbarer Lerninhalte – Design und Entwicklung des Kurses „Die interaktive Arbeit mit Flash MX 2004“ für das Bildungsportal Sachsen**

Eingereicht von : Torsten Rülke
Eingereicht am : 28.06.2006
Betreuer : Prof. Dr. Teresa Merino, Hochschule für Technik und Wirtschaft Dresden

Inhaltsverzeichnis

Einleitung

Der Einsatz elektronischer Medien im Bereich des Lernens ist nicht neu und wird in Form computerunterstützter Lösungen bereits seit über 30 Jahren praktiziert. Der große Durchbruch im E-Learning-Sektor erfolgte jedoch Mitte der neunziger Jahre mit der zunehmenden Ausbreitung des Internets sowie der stetigen Entwicklung neuer Informations- und Kommunikationstechnologien. Damit wurde es möglich, Lernangebote ohne hohen Zeit- und Kostenaufwand einer breiten Masse von Anwendern zur Verfügung zu stellen.

In den letzten Jahren kamen eine Reihe von Produkten aus den Bereichen Lernsoftware, Lermanagement- sowie Autorensysteme auf den Markt. Bei der Vielzahl an Lösungen ist es jedoch erstrebenswert, anstelle der proprietären Ansätze eine einheitliche Schnittstelle zu entwickeln, über die sich die Interoperabilität zwischen den Anwendungen sowie die Wiederverwendbarkeit von Lerninhalten gewährleisten lässt. Zu diesem Zweck bildeten sich verschiedene Standardisierungsinitiativen und begannen mit der Entwicklung einer Reihe von Spezifikationen und Standards. Mehrere dieser Richtlinien fließen in das Sharable Content Object Reference Model (SCORM) der ADL Initiative ein, das mittlerweile eine breite Unterstützung seitens der Hersteller im E-Learning-Sektor erfährt.

Der Einsatz von Internettechnologien stellt naturgemäß etwas andere Anforderungen an eine Lernsoftware, als dies bei lokal auf einem Rechner installierten der Fall ist. Die Firma Adobe Systems stellt mit Macromedia Flash ein leistungsfähiges Autorenwerkzeug zur Verfügung, das aktuell in der achten Version angeboten wird und nicht zuletzt aufgrund der Verwendung von Vektorgrafik sowie der Unterstützung der SCORM-Spezifikation für die Entwicklung netzbasierter Lösungen geeignet ist. Die interaktiven und multimedialen Fähigkeiten werden ergänzt bzw. erweitert durch die integrierte objektorientierte Skriptsprache ActionScript 2.0.

Diese Arbeit beschreibt das Design sowie die Entwicklung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“, das an der Hochschule für Technik und Wirtschaft (HTW) Dresden erstellt wurde. Es entstand im Rahmen eines Projektes für das Bildungsportal Sachsen, einem Internetportal der sächsischen Hochschulen für die Aus- und Weiterbildung, und wird seit dem Sommersemester 2006 begleitend zur Lehrveranstaltung „Entwicklungswerkzeuge für Multimediasysteme“ an der HTW Dresden eingesetzt.

Das erste Kapitel befasst sich mit der Erläuterung wichtiger Begriffe aus dem Bereich des E-Learning und gibt einen Überblick zu verschiedenen Standardisierungsinitiativen sowie der SCORM 1.2-Spezifikation. In den letzten Abschnitten wird das Autorenwerkzeug Macromedia Flash 8 einer näheren Betrachtung unterzogen.

Das zweite Kapitel beschreibt die Analyse- sowie die Designphase der Entwicklung des

Lernmoduls. In diese fallen beispielsweise das Bestimmen der Zielgruppe, die Festlegung der Lernziele sowie der zu einzusetzenden Medien, aber auch das funktionelle und visuelle Design.

Die letzten beiden Kapitel befassen sich hauptsächlich mit der technischen Entwicklung des Lernmoduls sowie der Implementierung der SCORM 1.2-Spezifikation. Im dritten Kapitel wird dabei auf die erstellten ActionScript 2.0-Klassen eingegangen sowie die komplette Flash-Struktur erläutert. Daraufhin folgt eine Beschreibung der Konfigurationsdateien sowie unterschiedlichen Ressourcentypen, die im Lernmodul zum Einsatz kamen. Den Abschluss bildet ein Überblick zur Erstellung neuer Lernobjekte.

Das vierte Kapitel beschreibt die Anpassung an SCORM sowie den Aufruf des Lernmoduls im Bildungsportal Sachsen. Außerdem wird auf die verschiedenen Tests eingegangen, die während und nach der Entwicklung des Programms stattfanden.

Der Arbeit wurden zwei separate Anhänge beigefügt. Dererste enthält eine komplette Beschreibung aller XML-Elemente der Konfigurationsdateien sowie eine Erläuterung der Klassen-Methoden und –Eigenschaften. Der zweite hingegen beinhaltet das zu Beginn der Produktionsphase erstellte Drehbuch inklusive des Basaltextes.

Außerdem wurde eine CD beigefügt, auf der die aktuelle Version des Lernmoduls gespeichert ist.

1 E-Learning und Flash

1.1 Begriffserklärungen

In diesem Abschnitt geht es um die Klärung wichtiger Begriffe, die zum besseren Verständnis der gesamten Arbeit beitragen.

1.1.1 E-Learning

Der Begriff „*E-Learning*“[1] steht für „*Electronic Learning*“ und bezeichnet die Anwendung von Informations- und Kommunikationstechnologien zur Unterstützung von Lehr- und Lernprozessen [Wikipedia 2006]. Er wird oft als Synonym für eine Reihe unterschiedlichster Lernformen verwendet. Dazu zählt neben dem Einsatz von Videobändern oder interaktivem Fernsehen zu Lernzwecken heutzutage insbesondere auch das computer- bzw. softwareunterstützte Lernen. So handelt es sich beim „*Computer Based Training*“ (CBT) um multimedial aufbereitete Lerninhalte, die entweder lokal auf der Festplatte gespeichert oder auf einem Wechseldatenträger (z.B. CD-ROM, DVD) bereitgestellt werden. Die Lernende hat somit die Möglichkeit, die Software flexibel und unabhängig von örtlichen und zeitlichen Gegebenheiten zu nutzen. Es besteht jedoch kein direkter Kontakt zu anderen Lernenden oder einem Lehrenden, so dass die Aneignung von Wissen in aller Regel komplett im Selbststudium stattfindet.

Mit der zunehmenden Bedeutung des Internets ging auch eine Weiterentwicklung des CBT hin zum so genannten „*Web Based Training*“ (WBT) einher. Es handelt sich dabei um die Nutzung netzbasierter Technologien (Internet, Intranet) zur Bereitstellung von Lerninhalten. Unter Einbeziehung neuer Wege der Kommunikation und Interaktion, wie beispielsweise Chats, Diskussionsforen oder Audio- und Videokonferenzen, besteht für den Lernenden die Möglichkeit, in direkten Gedankenaustausch mit Tutoren oder Mitlernenden zu treten. In der Fachliteratur wird oftmals eine Netzanbindung als wesentliches Merkmal von E-Learning-Anwendungen betrachtet.

Verstärkt zeichnet sich jedoch ein Trend hin zu einer Kombination aus Präsenzveranstaltungen und netz- bzw. computergestütztem Lernen ab. Diese Methode der Wissensvermittlung bzw. -aneignung wird „*hybrides Lernen*“ oder auch „*Blended Learning*“ genannt und hat zum Ziel, die sozialen Aspekte des Gruppenunterrichts sowie die individuelle Betreuung durch einen Lehrenden mit den Vorteilen digital bereitgestellter Lernanwendungen zu verknüpfen.

[1] Gebräuchlich sind auch die Schreibweisen eLearning, elearning, e-learning und e-Learning.

1.1.2 Netzbasierte Lernplattformen

Bei einer netzbasierten Lernplattform, auch *Lernmanagementsystem* oder *Learning Management System* (*LMS*) genannt, handelt es sich um eine serverseitig installierte Software, die selbst erstellte oder zugekaufte Lerninhalte in einer Datenbank verwaltet und den Lernenden zur Verfügung stellt. Das System ist darüber hinaus in der Lage, individuelle Lernprozesse (z.B. Auswahl der Kurse, Testergebnisse etc.) zu verfolgen und aufzuzeichnen. Über verschiedene asynchrone (z.B. Chats) und synchrone (z.B. Diskussionsforen) Kommunikationswerkzeuge wird Kursteilnehmern häufig die Möglichkeit geboten, Erfahrungen auszutauschen und in Gruppen zusammenzuarbeiten [Baumgartner&Häfele 2002, S.30].

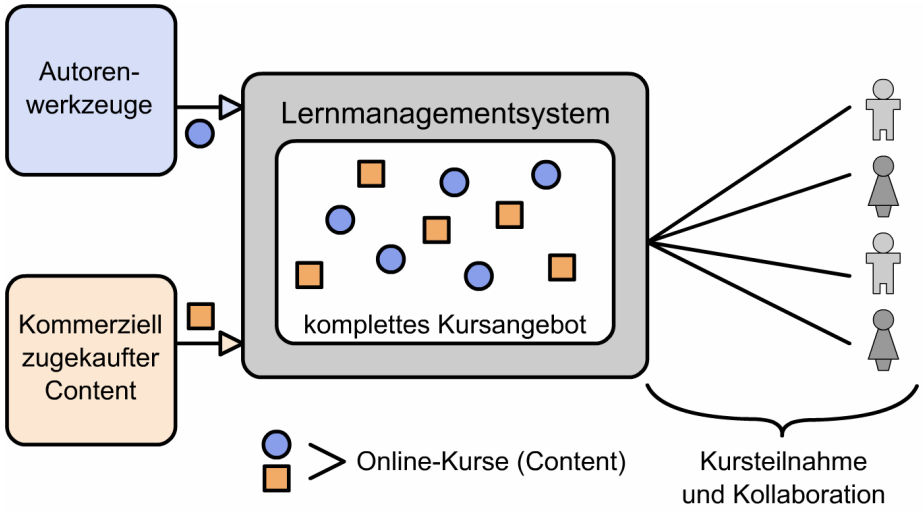

Abbildung 1.1: Schema eines Lernmanagementsystems nach [Baumgartner&Häfele 2002, S.30]

Schulmeister schlüsselt die Funktionen, über die ein Lernmanagementsystem verfügen sollte, wie folgt auf [Schulmeister 2003, S.10]:

- eine Benutzerverwaltung (Anmeldung mit Verschlüsselung)
- Administration der Kurse, Inhalte und Dateien
- die Zuweisung von Rollen (Administratoren, Tutoren, Studenten etc.) und eine Rechtevergabe mit differenzierten Rechten
- asynchrone und synchrone Kommunikationswerkzeuge
- Werkzeuge für das Lernen (Whiteboard, Kalender, Notiz- und Annotationsfunktionen etc.)
- Präsentation der Kursinhalte, Lernobjekte und Medien in einem netwerkfähigem Browser

Aus den aufgeführten Kriterien geht hervor, dass die Möglichkeit zur Erstellung neuer Lerninhalte, wie beispielsweise Aufgaben oder Übungen, nicht zwingend Bestandteil einer netz-

basierten Lernplattform ist. In der Praxis werden von den Herstellern – wenn überhaupt – oft auch nur rudimentäre Lösungen implementiert. Stattdessen kommen bei der Entwicklung von WBT-Anwendungen in den meisten Fällen externe Autorenwerkzeuge (siehe Abschnitt 1.1.4) zum Einsatz.

1.1.3 Learning Content Management System

Ein *Learning Content Management System* (LCMS) beherrscht die typischen Fähigkeiten eines LMS und erweitert diese um Funktionen zur Erstellung, Archivierung, Verwaltung, Wiederverwendung und Distribution von Lerninhalten.

Im Zusammenhang mit LCMS taucht häufig auch der Begriff des so genannten „*Lernobjekts*“ (LO) auf. Dabei handelt es sich um die kleinste sinnvolle Lerneinheit, in die ein Online-Kurs zerlegt werden kann. Demnach umschreiben Lernobjekte einzelne Bilder, Texte, Animationen oder auch einen einzelnen Test zur Lernerfolgskontrolle. Um die Recherchier- und Wiederverwendbarkeit solcher Elemente zu gewährleisten, müssen sie über zusätzliche Informationen eindeutig beschrieben werden. Man spricht in einem solchen Fall von *Metadaten*. Die wieder verwendbaren Lernobjekte, auch *Reusable Learning Objects* (RLO) genannt, können schließlich zu komplexeren Kurseinheiten zusammengefasst werden, wie die Abbildung 1.2 veranschaulicht:

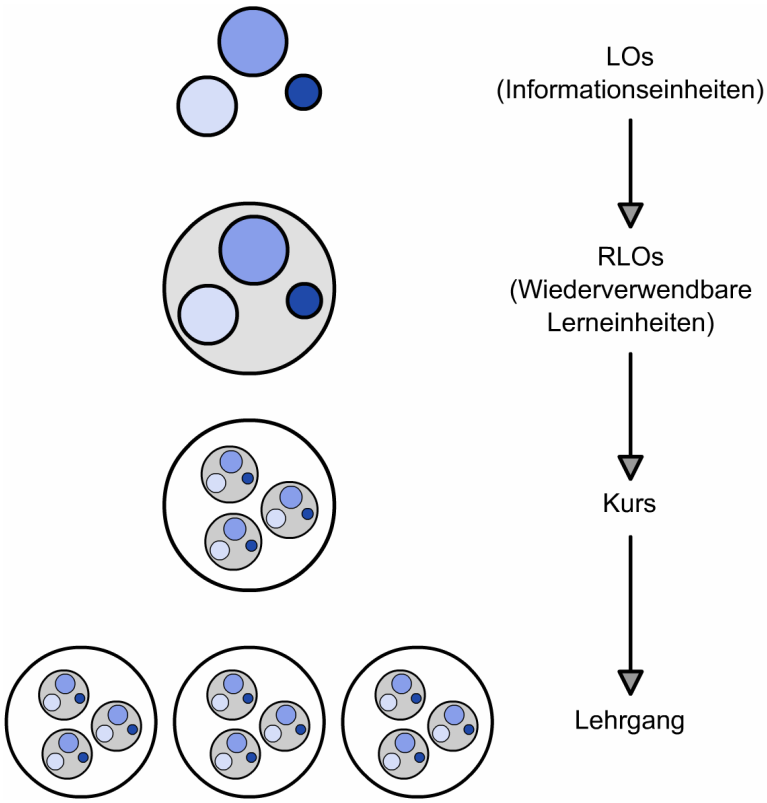

Abbildung 1.2: Das modulare Konzept der Lerninhalte nach [Baumgartner&Häfele 2002, S.42]

Learning Content Management Systeme gewähren WBT-Designern Zugriff auf bereits erstellte RLOs und ermöglichen es ihnen somit, in kürzester Zeit neue Lernangebote zu entwickeln. Dieses modulare Konzept erlaubt ein effizientes und kostengünstiges Arbeiten.
Einen weiteren Vorteil eines LCMS gegenüber einem LMS stellt die Möglichkeit einer personalisierten Zusammenstellung von Lerninhalten dar. Anhand einer so genannten *Skill-Gap-Analyse* werden zunächst eventuelle Wissenslücken („Skill Gaps“) des Lernenden aufgedeckt. Dies geschieht einerseits durch das System in Form verschiedener Tests (meist Multiple-Choice-Fragen), andererseits durch subjektive Beurteilungen der Kenntnisse durch Vorgesetzte und Kollegen. Anschließend werden die Daten ausgewertet und das Ergebnis der Analyse in einem speziellen Lernerprofil gespeichert. Dieses wird vom LCMS als Ausgangsbasis verwendet, um dem Lernenden fortan Vorschläge über hilfreiche Kursangebote zu unterbreiten, mit denen er seine spezifischen Wissenslücken schließen kann. Eine Aktualisierung des persönlichen Profils erfolgt, sobald der Kursteilnehmer über entsprechende neue Kenntnisse verfügt und diese anhand von Tests nachweisen kann [Baumgartner&Häfele 2002, S.45f].

1.1.4 Autorenwerkzeuge

Autorenwerkzeuge sind Softwaresysteme, deren Ziel es ist, die Entwicklung digitaler Lerninhalte so einfach und komfortabel wie möglich zu gestalten. Auf dem Markt existieren eine Reihe derartiger Produkte, die oftmals auf bestimmte Aufgabenbereiche (z.B. das Erstellen von Präsentationen, Simulationen, Animationen, Übungen etc.) spezialisiert sind. Je nach Komplexität erfordern solche Systeme einen mehr oder weniger hohen Einarbeitungsaufwand von den Autoren.

Anhand ihrer Leistungsfähigkeit lassen sich Autorenwerkzeuge in drei Gruppen unterteilen [Baumgartner&Häfele 2002, S.33f]:

- *Professionelle Autorensysteme*: Derartige Werkzeuge verfügen über eine integrierte Programmiersprache und erfordern im Allgemeinen einen hohen Einarbeitungsaufwand. Sie bieten den Entwicklern jedoch im Gegenzug häufig ein hohes Maß an kreativen Freiheiten.
- *WYSIWYG-HTML-Editoren*: WYSIWIG-Editoren („What You See Is What You Get“) unterstützen die Autoren digitaler Lerninhalte mittels einer grafischen Schnittstelle, über die sie wie gewohnt Texte schreiben und Animationen oder Bilder einfügen können. Die Arbeit erfolgt auf einer abstrahierten Ebene und hat den Vorteil, dass sich die Entwickler nicht mit den komplexen Details der zugrunde liegenden Aufzeichnungssprache (HTML, XML usw.) befassen müssen.
- *Rapid Content Development Tools*: Hierbei handelt es sich um Autorenwerkzeuge ohne hohen Einarbeitungsaufwand, mit deren Hilfe sich Lerninhalte auf schnelle und unkomplizierte Weise generieren lassen.

1.1.5 Standard und Spezifikation

Ein Standard ist ein veröffentlichtes und weitläufig akzeptiertes Regelwerk, das bestimmte (Mindest)eigenschaften von Produkten, Abläufen oder Methoden definiert. Die Erarbeitung der Richtlinien erfolgt durch so genannte *Standardisierungsinitiativen* (siehe Abschnitt 1.2.1). Grundsätzlich kann zwischen zwei verschiedenen Arten differenziert werden [Wikipedia 2006] [Montandon 2004, S.3]:

- *De-jure*[2]*-Standard*: Dabei handelt es sich um eine allseits juristisch anerkannte und anhand eines Normungsverfahrens beschlossene, allgemeingültige sowie veröffentlichte Regel zur Lösung eines Sachverhaltes. Als gebräuchliche Synonyme findet man im deutschsprachigen Raum auch die Begriffe „*Norm*“ oder „*akkreditierter Standard*“.
- *De-facto*[3]*-Standard*: Ohne durch ein nationales oder internationales Normungsverfahren zertifiziert worden zu sein, haben sich diese Standards beim Einsatz in der Praxis als sinnvoll und richtig erwiesen. Sie werden oft auch als so genannte *Industriestandards* bezeichnet.

Solange die von einer Standardisierungsinitiative vorgeschlagenen Richtlinien nicht durch ein Normungsverfahren offiziell anerkannt wurden und sich ihr Einsatz in der Praxis noch nicht durchgesetzt hat, werden sie als *Spezifikation* oder auch *Empfehlung* bezeichnet. Das Recht zur Akkreditierung eines Standards besitzen ausschließlich nationale sowie internationale Normierungsgremien (vgl. Tab. 1.1).

Nationale Normierungsgremien	**Internationale Normierungsgremien**
Deutsches Institut für Normung (DIN)	Institute of Electrical and Electronics Engineers, Inc. (IEEE)
American National Standards Institute (ANSI)	International Organization for Standardization (ISO)
	National Institute of Standards and Technology (NISO)
	Comité Européen de Normalisation (CEN)

Tabelle 1.1: Nationale und internationale Normierungsgremien, nach [Montandon 2004, S.3]

Im weiteren Verlauf dieser Arbeit wird der Begriff „Standard“ nur im Sinne von „de-jure-

[2] lateinisch für „durch das Recht“

[3] lateinisch für „durch die Tatsache“

Standard“ benutzt, die Begriffe „de-facto-Standard“ und „Spezifikation“ werden als Synonyme verwendet.

1.1.6 XML

Die *Extensible Markup Language* (Erweiterbare Auszeichnungssprache) ist ein vom World Wide Web Consortium (W3W) entwickelter Standard und kann als eine vereinfachte Form der Standard Generalized Markup Language (SGML) betrachtet werden. Mit XML wird ein plattformunabhängiges Textformat zur Verfügung gestellt, über das sich beliebige Daten innerhalb einer definierten Baumstruktur beschreiben lassen. Der Grundgedanke dabei ist die Trennung von Inhalt und Darstellung. Demnach kann beispielsweise für die Darstellung von Messdaten einmal als Balken- und einmal als Kreisdiagramm dieselbe XML-Datenquelle verwendet werden [Wikipedia 2006].

Die Struktur einer XML-Datei wird im Wesentlichen durch die *Elemente* (auch *Tags* genannt) sowie deren *Attribute* bestimmt. Darüber hinaus kann optional eine XML-Deklaration verwendet werden, um die Version, die Zeichenkodierung sowie die Verarbeitbarkeit ohne ein Datenschema (siehe weiter unten in diesem Abschnitt) zu spezifizieren. Die Definition folgt strikten Regeln:

- jedes geöffnete Element muss auch ein schließendes besitzen
- leere Elemente müssen auch als solche durch einen schließenden Schrägstrich gekennzeichnet werden
- ein Attribut setzt sich aus dem Namen, einem Gleichheitszeichen sowie einem in Anführungszeichen eingefassten Wert zusammen

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<person_element name_attribut="Max Mustermann">
  <wohnsitz_element>
    <ort_element>Musterstadt</ort_element>
    <plz_element>01234</plz_element>
  </wohnsitz_element>
  <sonstiges_element alter_attribut="30" />
</person_element>
```

Quellcode-Bsp. 1.1: Eine einfache XML-Struktur

Die Aufgabe, einen XML-Baum auszulesen und daraus eine für das umgebende System verständliche Struktur zu generieren, obliegt dem so genannten *Parser*.

Einen interessanten Aspekt der Verwendung von XML stellt die Möglichkeit dar, auf Basis dieses Standards neue Datentypen zu erstellen. Die Voraussetzung dazu bilden so

genannte *Schemasprachen*, wie beispielsweise *DTD* (Document Type Definition) oder *XSD* (XML Schema Definition). Mit ihnen lässt sich ein einheitliches Datenmuster anlegen, indem Elemente, Attribute und ihr Verhalten zueinander spezifiziert werden. Dies geschieht entweder in externer Form oder direkt in der XML-Struktur. XML eignet sich also zur Definition anderer Auszeichnungssprachen (z.B. XHTML) und wird deshalb auch als *Metasprache* bezeichnet.

Aufgrund der genannten Vorzüge wird dem XML-Standard auch im E-Learning-Bereich eine hohe Bedeutung zugemessen. So hat er sich mittlerweile als probates Mittel etabliert, um Lerninhalte in generalisierter Form zu beschreiben und somit die plattformübergreifende Recherchier- sowie Wiederverwendbarkeit selbiger sicherzustellen (siehe auch Abschnitt 1.2.2).

1.2 E-Learning-Standards

Ein wichtiges Kriterium bei der Beurteilung von Lernplattformen und Autorensystemen wird in zunehmendem Maße auch sein, welche Standards sie erfüllen. Im Gegensatz zu proprietären[4] Lösungen sichern einheitliche Richtlinien die Kompatibilität verschiedener Produkte untereinander und vereinfachen die Entwicklung sowie die Nutzung von Lernangeboten. Durch die Einhaltung von Standards und Spezifikationen lassen sich Lernressourcen mehrfach verwenden und zu neuen Online-Kursen kombinieren, ohne dass eine Anpassung an ein spezielles System erfolgen muss, was unter anderem zu einer Reduktion der anfallenden Kosten bei der Erstellung von Lerninhalten führt [Montandon 2004, S.6]. Der Lernende profitiert schließlich von einer steigenden Qualität sowie Quantität des Bildungsangebots im Netz.

1.2.1 Wichtige Standardisierungsinitiativen

In den letzten Jahren bildeten sich in den USA und in Europa verschiedene Initiativen, die sich mit der Entwicklung einheitlicher Richtlinien im Bereich des E-Learning beschäftigen. Viele der gewonnenen Erkenntnisse (Metadaten, Content-Packaging und –Sequencing etc.) fließen in die Arbeit an der SCORM-Spezifikation mit ein.

Ohne einen Anspruch auf Vollständigkeit zu erheben, sollen nachfolgend die wichtigsten Standardisierungsinitiativen genannt werden.

[4] Bei proprietären Lösungen handelt es sich um herstellerspezifische.

AICC

Das *Aviation Industry Computer Based Training Committee* ist ein internationaler Verband von Spezialisten auf dem Gebiet des technologiebasierten Lernens, der im Jahr 1988 durch die amerikanische Luftfahrtindustrie gegründet wurde. Es erarbeitet Richtlinien für die Entwicklung, Bereitstellung und Bewertung von softwareunterstützten Lerntechno-logien und veröffentlicht diese in den so genannten *AICC Guidelines and Recommendations* (AGR) [AICC].

Obwohl das Komitee im Bereich der Luftfahrtindustrie angesiedelt ist, kommen die Spezifikationen aufgrund ihrer Allgemeingültigkeit auch außerhalb zum Einsatz.

ARIADNE Foundation

Die *Alliance of Remote Instructional Authoring and Distribution Networks for Europe Foundation* ist eine europäische Organisation, die Mitglieder auch über die EU-Grenzen hinweg besitzt. Sie führt die Arbeit der Vorgängerprojekte ARIADNE und ARIADNE II fort, die während der Jahre 1996 bis 2000 stattfand und im Wesentlichen durch die Europäische Union (EU) sowie die Schweizer Regierung finanziert wurde. Das Hauptaugenmerk der ARIADNE Foundation richtet sich dabei auf die Pflege und Weiterentwicklung der im Laufe dieses Zeitraums entstandenen Werkzeuge und Methoden zur Produktion, Verwaltung und Wiederverwendung von computerbasierten Lerninhalten.

Als bedeutender Beitrag bei den Standardisierungsbemühungen im E-Learning-Bereich kann die Spezifikation eines einheitlichen Metadatenschemas angesehen werden, die in Kooperation mit dem IMS entwickelt wurde und 1998 die Grundlage bildete für den später offiziell anerkannten *IEEE Standard for Learning Objective Metadata* (LOM) [ARIADNE] [IMS].

IEEE LTSC

Das *Learning Technology Standards Committee* (LTSC) wurde 1996 vom *Institut of Electrical and Electronics Engineers* (IEEE) ins Leben gerufen. Seine Hauptaufgabe besteht in der Definition akkreditierter Standards und Richtlinien für die Entwicklung und Implementierung von softwareunterstützten Lehr- und Lernsystemen [Baumgartner&Häfele 2002, S.310][IEEE].

Eine wichtige Errungenschaft, insbesondere im Hinblick auf die SCORM-Spezifikation, stellt der bereits erwähnte LOM-Standard dar, mit dessen Hilfe sich Lernobjekte über XML-Schematas auf der Metaebene beschreiben lassen. Darüber hinaus basiert auch die SCORM-Laufzeitumgebung (Run-Time Environment, kurz RTE) seit der Version 1.3 auf Spezifikationen des IEEE LTSC [SCORM 2004a, S.3].

IMS

Das *Instructional Management Systems Global Learning Consortium* wurde 1997 gegründet und stellt einen Zusammenschluss mehrer internationaler Bildungs- und Regierungsorganisationen. Das Tätigkeitsfeld liegt mit Blick auf die verschiedenen Anforderungen im Bereich des E-Learning vor allem in der Erarbeitung neuer Spezifikationen, wie beispielsweise des schon erwähnten Metadatenschemas. Ziel dieser Anstrengungen ist es, eine möglichst hohe Interoperabilität zwischen Lernplattformen und Online-Kursen zu gewährleisten sowie die Recherchier- und Wiederverwendbarkeit von Lernangeboten sicherzustellen [IMS].

In die Entwicklung der SCORM-Spezifikation von ADL flossen neben dem Metadatenschema auch weitere Vorschläge das IMS mit ein, wie Richtlinien zum *Content Packaging* (seit SCORM 1.2, siehe Abschnitt 1.2.3) und zum so genannten *Simple Sequencing* (seit SCORM 2004) [SCORM 2004b, S.7].

ADL

Vom amerikanischen Verteidigungsministerium und vom Office of Science and Technology Policy (OSTP) wurde 1997 die *Advanced Distibuted Learning Initiative* ins Leben gerufen. Das erklärte Ziel dieses Gremiums ist es, qualitativ hochwertige und an individuelle Bedürfnisse anpassbare Bildungsangebote in Unabhängigkeit von zeitlichen und örtlichen Gegebenheiten zur Verfügung zu stellen. Diese Vision soll nach ADL mit Hilfe eines einheitlichen Referenzmodells realisiert werden, das es Lerninhalten ermöglichen soll, folgende Anforderungen zu erfüllen [SCORM 2001a, S.8]:

- *Accessibility*: Aufrufbarkeit der Lernmaterialien von verschiedenen Lokalitäten aus
- *Adaptability*: Anpassungsmöglichkeit von Lernobjekten an individuelle oder organisatorische Bedürfnisse
- *Affordability*: Steigerung der Effizienz und der Produktivität bei der Erstellung von Lernangeboten durch Zeit- und Kosteneinsparung
- *Durability*: Beständigkeit von Lernmaterialien gegenüber technischen Veränderungen oder Weiterentwicklungen, so dass es keines großen Investitionsaufwands bedarf
- *Interoperability*: plattform- und systemunabhängige Verwendung von Produkten verschiedener Hersteller, ohne dass spezielle Anpassungen erforderlich sind
- *Reusability*: Mehrfachnutzung von Lernobjekten in verschiedenen Anwendungen und Kontexten

Auf der Grundlage dieses Kriterienkatalogs wurde von ADL die SCORM-Spezifikation erarbeitet, auf die im Abschnitt 1.2.2 detaillierter eingegangen wird.

Nach anfänglich getrennten Bestrebungen um eine Standardisierung im E-Learnig-Bereich haben sich die einzelnen Gremien mittlerweile zu einem komplexen Kooperations-

netzwerk zusammengeschlossen (vgl. Abbildung 1.3).

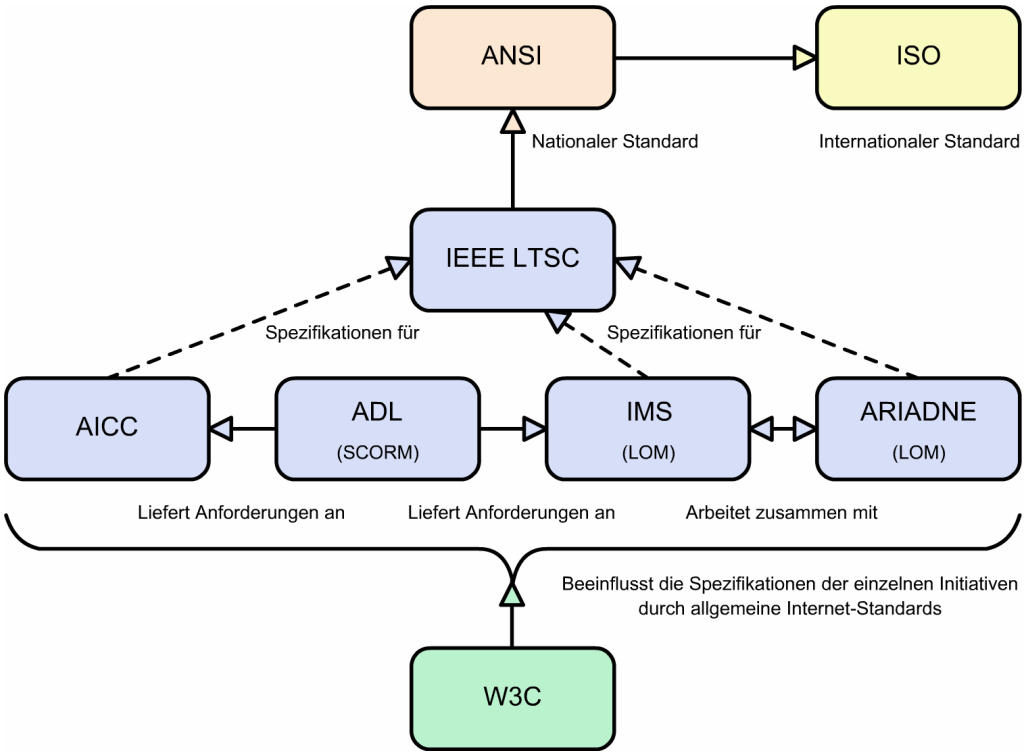

Abbildung 1.3: Das Kooperationsnetzwerk der Standardisierungsgremien nach [Baumgartner&Häfele 2002, S.32]

1.2.2 Die SCORM-Spezifikation

Die SCORM-Spezifikation wurde zu Beginn des Jahres 2000 veröffentlicht. Sie stellt ganz allgemein gesehen eine Schnittstelle zwischen Lernplattformen und –inhalten dar und hat sich mittlerweile als De-facto-Standard auf dem Markt etabliert. Lautete die vollständige Bezeichnung ursprünglich noch Sharable *Courseware* Object Reference Model, so wurde sie mit der Version 1.1 zu Sharable *Content* Object Reference Model abgeändert. Hinter dieser Umbenennung verbarg sich die Absicht, die Anwendbarkeit des Referenzmodells auf verschiedene Elemente in Unterebenen eines Kurses und nicht nur auf diesen als Ganzes zu verdeutlichen [SCORM 2001a, S.8].

Obwohl die Spezifikation seit Juli 2004 in der Version 2004 2nd Edition[5] vorliegt, befasst sich diese Arbeit lediglich mit SCORM 1.2. Dies resultiert aus der Tatsache, dass zum Zeitpunkt der Erstellung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“ das

[5] Die Kennzeichnung anhand einer Versionsnummer wird seit der Einführung von SCORM 2004 im Januar 2004 für jedes SCORM-Buch separat fortgeführt, und nicht mehr für die Spezifikation selbst. Im Falle einer Aktualisierung ändert sich demzufolge nur die Versionsnummer des betreffenden Buches, alle anderen bleiben unberührt. Zum gegenwärtigen Zeitpunkt liegen alle Bücher in der Version 1.3.1 vor.

vom Bildungsportal Sachsen verwendete LMS (Saba 3 Release 4) eine Unterstützung ausschließlich für diese Version bot. Ob und wann die Umstellung auf ein anderes System und damit gegebenenfalls auf die neue SCORM-Version erfolgen sollte, war damals nicht bekannt. Mit Beginn des Sommersemesters 2006 ist diese erfolgt, jedoch bietet das aktuell eingesetzte Lernmanagementsystem OLAT 4.0 ebenfalls nur eine Unterstützung der 1.2-Spezifikation [OLAT].

Die SCORM-Spezifikation wird in Form mehrerer Bücher veröffentlicht. Mit jeder neuen Version werden diese überarbeitet oder um weitere ergänzt. SCORM 1.2 umfasst folgende drei Bücher:

- *The SCORM Overview*: Dieses Buch verschafft einen Überblick zur ADL Initiative sowie zu SCORM.
- *The SCORM Content Aggregation Model*: In diesem Buch werden Richtlinien zum Aufbau und zur Beschreibung von Lerninhalten spezifiziert.
- *The SCORM Run-Time Environment*: Das dritte Buch definiert einheitliche Methoden zum Starten von Lerninhalten sowie zur Kommunikation und zum Datenaustausch zwischen Lernmanagementsystemen und Online-Kursen.

In Abbildung 1.4 werden neben den drei SCORM-Büchern auch die jeweils eingeflossenen Standards und Spezifikationen anderer Institutionen illustriert:

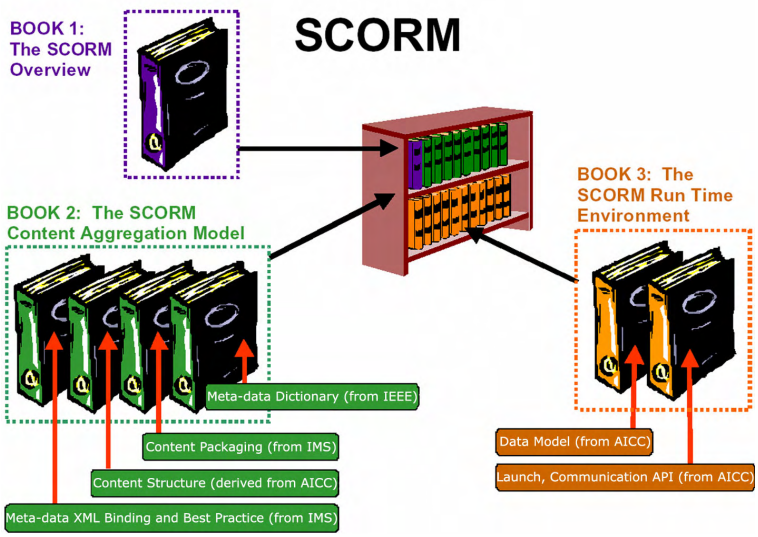

Abbildung 1.4: SCORM als eine Sammlung von Spezifikationen,
nach [SCORM 2001a, S.5]

Die aktuelle Version der SCORM-Spezifikation wurde um das Buch *Sequencing and Navigation* erweitert. Darin enthalten sind Richtlinien zur dynamischen Ablaufsteuerung von Lerninhalten sowie zur Navigation [SCORM 2004b, S.31].

1.2.3 SCORM Content Aggregation Model

Das Content Aggregation Model (CAM) beschreibt einzelne Ressourcen eines Lernangebots und spezifiziert darüber hinaus Methoden, anhand derer sich Lernobjekte zu komplex strukturierten Kursen zusammenfassen lassen. Darin fließt der Gedanke ein, Lerninhalte wieder verwendbar, eindeutig identifizier- und recherchierbar sowie kombinierbar zu gestalten. Darüber hinaus sollen sie flexibel sein und sich ohne zusätzliche Anpassung in verschiedene Lernmanagementsysteme integrieren lassen. Unter SCORM 1.2 besteht das CAM aus drei Teilen:

- Content Model
- Meta-Data
- Content Packaging

Content Model

Das SCORM Content Model beinhaltet die folgenden Elemente: *Assets*, *Sharable Content Objects* (SCO) sowie *Content Aggregations*.

Asset

Ein Asset ist die elementarste Form von Lerneinheiten und repräsentiert sämtliche elektronischen Daten, die sich in einem Internetbrowser darstellen oder abspielen lassen, wie beispielsweise Texte, Bilder, Klänge, Animationen oder auch JavaScript-Funktionen. Mehrere einzelne Assets können darüber hinaus zu einem neuen kombiniert werden.

Sharable Content Object

Die Grundlage eines SCOs bilden ein oder mehrere Assets bzw. auch Dateien, die nicht unter diese Definition fallen. Sharable Content Objects besitzen im Gegensatz zu Assets jedoch die Fähigkeit, über die SCORM-Laufzeitumgebung mit einem LMS zu kommunizieren und Daten auszutauschen. Das LMS stellt für diesen Prozess eine Schnittstelle (Application Programming Interface, kurz API) zur Verfügung. Das SCO muss in der Lage sein, diese zu lokalisieren, sowie mindestens die API-Methoden zum Initialisieren (`LMSInitialize("")`) und Beenden (`LMSFinish("")`) einer Kommunikation implementieren und aufrufen (siehe Abbildung 1.5).

Mit Blick auf die Mehrfachnutzung sollte ein SCO einen vom umgebenden Lernkontext möglichst unabhängigen, didaktisch sinnvollen Lerninhalt abbilden.

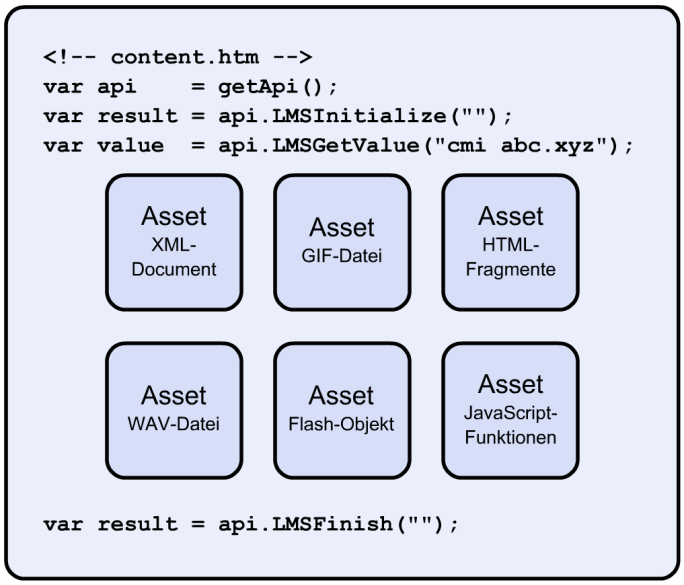

Abbildung 1.5: Typischer Aufbau eines SCO, nach [SCORM 2001b, S.5]

Content Aggregation

Die Content Aggregation („Inhaltszusammenfassung“) beschreibt eine inhaltliche Struktur, mit deren Hilfe sich einzelne Lernressourcen (Assets, SCOs) zu komplexeren Einheiten, wie beispielsweise Modulen, Kapiteln oder auch ganzen Kursen zusammenfügen lassen. Gleichzeitig wird die Reihenfolge festgelegt, in der die Lerninhalte dem Nutzer vom LMS präsentiert werden sollen. In SCORM 1.2 können darüber hinaus so genannte „*Prerequisites*“ (Vorbedingungen) definiert werden, anhand derer sich der Ablauf zwischen den einzelnen Lernressourcen kontrollieren lässt. Diese Möglichkeit wurde mit SCORM 2004 zugunsten eines deutlich leistungsfähigeren Sequenzierungs- und Navigationsmodells[6] wieder verworfen.

[6] spezifiziert im Buch *Sequencing and Navigation*

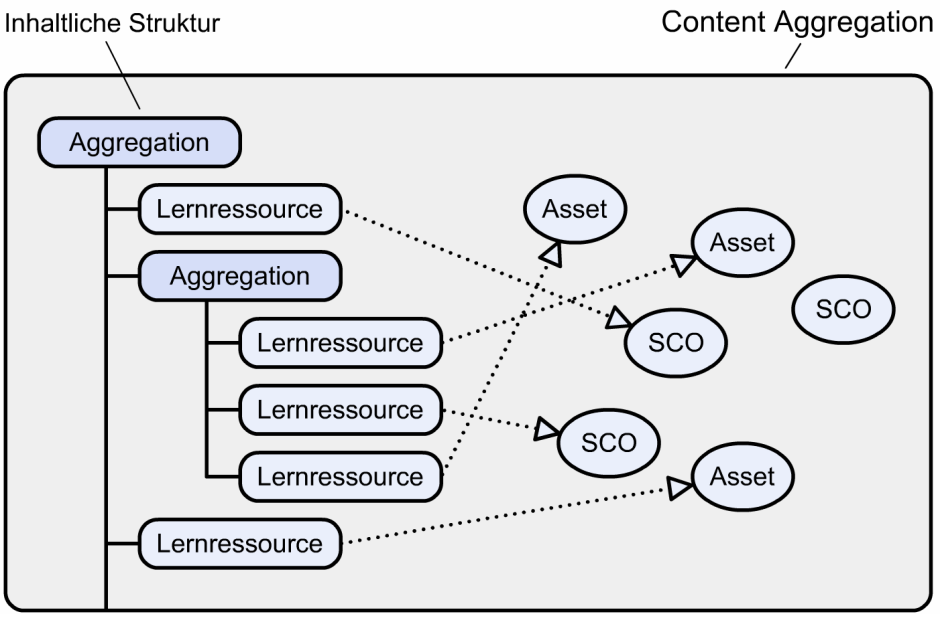

Abbildung 1.6: Content Aggregation, nach [SCORM 2001b, S.7]

Meta-Data

Sämtliche Elemente des Content Model lassen sich optional über zusätzliche XML-basierte Metadaten beschreiben. Anhand dieser Informationen sind eine spätere Recherche in Kursbibliotheken sowie die Integration der Lernressourcen in unterschiedliche Bildungsangebote möglich. In die Metadaten-Spezifikation flossen sowohl das *IMS Learning Ressource Meta-data Information Model* als auch die *IMS Learning Ressource XML Binding Specification* mit ein. ADL untergliedert sie in drei Teile:

- *Meta-data Information Model*: Dieses Modell beschreibt die Datenelemente (inklusive Datentyp und Häufigkeit), mit denen sich SCORM-konforme Metadatensätze anlegen lassen. Die Metadaten sind hierarchisch gegliedert und in mehrere Kategorien unterteilt.
- *Meta-data XML Binding*: In diesem Teil wird spezifiziert, wie die Metadatensätze innerhalb einer XML-Struktur repräsentiert werden.
- *Meta-data Application Profiles*: Die Profile legen fest, wie die Metadaten auf die einzelnen Elemente des Content Models anzuwenden sind. Dabei erfolgt grundsätzlich eine Differenzierung zwischen obligatorischen (*mandatory*) Datenelementen, die zwingend definiert werden müssen, sowie jenen, die wahlweise (*optional*) angegeben werden können.

Content Packaging

Das Content Packaging stellt eine Lösung dar, mit deren Hilfe sich digitale Lerninhalte in standardisierter Form zwischen verschiedenen Lernmanagementsystemen, Autorenwerkzeugen und Kursbibliotheken austauschen lassen. Es basiert direkt auf der *IMS Content*

Packaging Specification, wird jedoch unter SCORM um zusätzliche Elemente erweitert. Ein Content Package besteht hauptsächlich aus zwei Komponenten:

- Manifest-Datei
- physische Dateien, auf die innerhalb der Manifestdatei verwiesen wird

Ähnlich wie bei den Metadaten wird auch diese Spezifikation in drei Teile untergliedert:

- *Content Packaging Information Model*: Dieses Modell definiert die Datenelemente (inklusive Datentyp und Häufigkeit), mit denen sich SCORM-konforme Pakete beschreiben lassen.
- *Content Packaging XML Binding*: In diesem Teil wird spezifiziert, wie das Informationsmodell innerhalb einer XML-Struktur abgebildet werden kann.
- *Meta-data Application Profiles*: Die Profile definieren, wie sich die IMS Content Packaging Specification innerhalb des Kontexts von SCORM anwenden lässt. Daraus resultieren zwei verschiedene Paketarten: *Ressource Packages* (ohne Angaben zur inhaltlichen Struktur) sowie *Content Aggregation Packages*.

Die Manifest-Datei

Beim so genannten *Manifest* handelt es sich um eine XML-Datei, in der die inhaltliche Struktur des Pakets sowie Informationen zu den Ressourcen abgebildet werden. Die SCORM-Spezifikation legt fest, dass sie unter der Bezeichnung *imsmanifest.xml* im Stammverzeichnis des Content Package abgelegt wird.

Wie aus dem Quellcode-Beispiel 1.2 hervorgeht, ist eine Manifest-Datei in folgende Bereiche untergliedert:

- `<manifest>` : Beschreibung des kompletten Pakets sowie Definition optionaler „Submanifeste“
- `<metadata>` : Definition von Metadaten, entweder direkt im Manifest oder mittels Verlinkung zu einer externen Datei
- `<organizations>` : Beischreibung der inhaltlichen Struktur
- `<resources>` : Beschreibung der im Paket enthaltenen physischen Dateien

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="http://www.imsproject.org/xsd/imscp_rootv1p1p2"
      xmlns:imsmd="http://www.imsglobal.org/xsd/imsmd rootv1p2p1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:adlcp="http://www.adlnet.org/xsd/adlcp rootv1p2"
      identifier="MANIFEST-CB0E1885-9F17-29EE-9B91-BD3D161F6A71"
      xsi:schemaLocation="http://www.imsproject.org/xsd/imscp rootv1
           p1p2 imscp_rootv1p1p2.xsd
```

```
            http://www.imsglobal.org/xsd/imsmd_rootv1p2p1
            imsmd_rootv1p2p1.xsd
            http://www.adlnet.org/xsd/adlcp_rootv1p2
            adlcp_rootv1p2.xsd">
  <!-- Metadaten ------------------------------------------------------>
  <metadata>
    <schema>ADL SCORM</schema>
    <schemaversion>1.2</schemaversion>
    <adlcp:location>Metadata.xml</adlcp:location>
  </metadata>
  <!-- Organisationen ------------------------------------------------->
  <organizations default="ORG_1">
    <organization identifier="ORG_1" structure="hierarchical">
      <title>TITEL_1</title>
      <item identifier="ITEM_1" isvisible="true"
            identifierref="RESOURCE_1">
        <title>SCO_1</title>
      </item>
    </organization>
  </organizations>
  <!-- Ressourcen ----------------------------------------------------->
  <resources>
    <resource type="webcontent" adlcp:scormtype="sco"
              identifier="RESOURCE_1" href="MODULDATA/SCO_1.html">
      <file href="MODULDATA/SCO_1.html" />
    </resource>
  </resources>
</manifest>
```

Quellcode-Bsp. 1.2: Typische Manifest-Datei

Für den einfachen Austausch eines Content Package zwischen verschiedenen Systemen empfiehlt die ADL Initiative die Archivierung der Daten, z.B. im ZIP- oder JAR-Format[7]. Ein solches Archiv wird auch als *Package Interchange File* (PIF) bezeichnet.

[7] Ab SCORM 2004 wird das Format PKZip 2.04g vorgeschrieben.

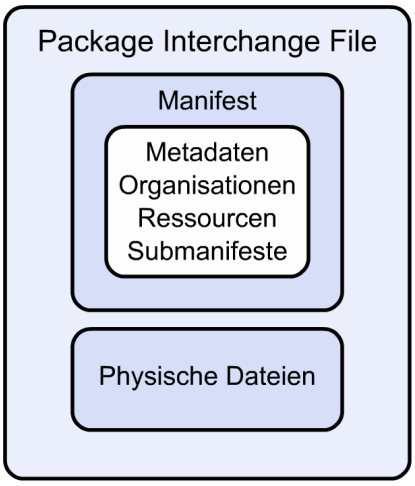

Abbildung 1.7: Stilisiertes Content Package,
nach [SCORM 2001b, S.111]

1.2.4 SCORM Run-Time Environment

Das Hauptanliegen der SCORM-Spezifikation ist es, Lerninhalte in wieder verwendbarer und interoperabler Form zur Verfügung zu stellen. Zu diesem Zweck ist es notwendig, eine einheitliche Lösung zu entwickeln, anhand derer Lerninhalte gestartet werden sowie mit Lernplattformen über eine vordefinierte Sprache kommunizieren können. Mechanismen, wie sich diese Funktionalität realisieren lässt, werden in der *Run-Time Environment* (RTE, zu Deutsch „Laufzeitumgebung“) spezifiziert, die sich aus folgenden Abschnitten zusammensetzt (siehe Abbildung 1.8):

- Launch
- Application Programming Interface
- Data Model

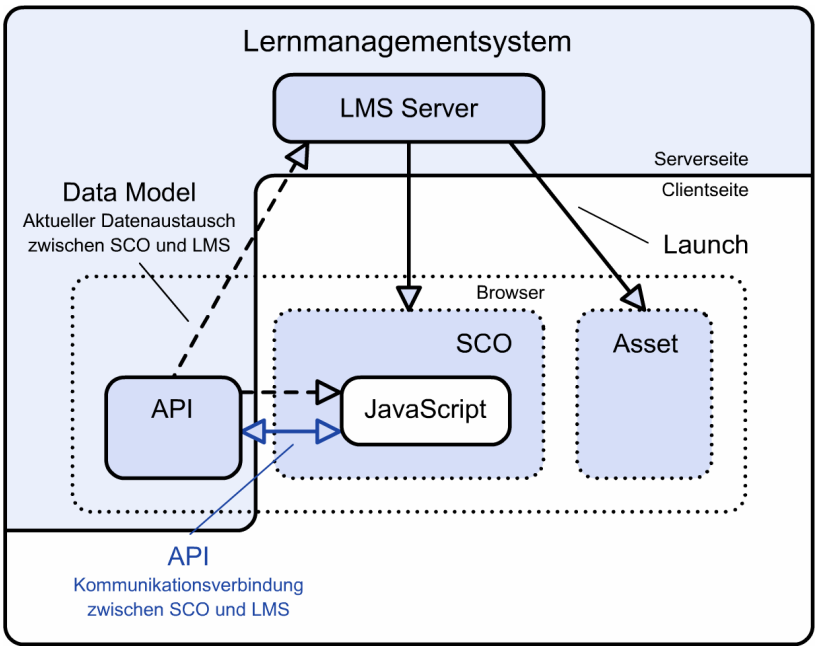

Abbildung 1.8: Komponenten der Laufzeitumgebung, nach [SCORM 2001c, S.3]

Launch

Der Launch-Mechanismus des RTE spezifiziert Richtlinien, wie netzbasierte Lerneinheiten (Assets, SCOs) in einer LMS-Umgebung gestartet und bereitgestellt werden sollen.

Die Interpretation und Verarbeitung der in einem SCORM-Paket definierten inhaltlichen Struktur obliegt ausschließlich dem LMS. Es ist dafür verantwortlich, dem Nutzer eines Bildungsangebots gegebenenfalls Werkzeuge für die Navigation bereitzustellen sowie die Abfolge zu steuern, in der die Lerninhalte präsentiert werden.

Weiterhin wird durch die RTE-Spezifikation vorgeschrieben, dass Assets und SCOs über das HTTP-Protokoll angezeigt werden. Allerdings ist nicht explizit definiert, ob der Start einer Ressource server- oder clientseitig initiiert wird. SCOs dürfen immer nur einzeln aktiv sein und nicht untereinander verlinkt werden.

API

Über die API eröffnet sich für SCO und LMS die Möglichkeit zur wechselseitigen Kommunikation. Wie bereits unter Abschnitt 1.2.3 erwähnt, muss zu diesem Zweck die Schnittstelle durch das LMS im Document Object Model (DOM) zur Verfügung gestellt (beispielsweise als Java-Applet) und anschließend durch das SCO lokalisiert werden.

Die Tabelle 1.2 bietet einen Überblick zu den verschiedenen Funktionen, die in der API implementiert sind. Die Kommunikation muss stets von Seiten des SCO initiiert werden.

Kategorie	Methoden	Bedeutung
Executive State	`LMSInitialize("")` `LMSFinish("")`	Methoden zum Starten und Beenden der Kommunikations-verbindung.

State Management	`LMSGetLastError ()`
	`LMSGetErrorString (errornumber)`
	`LMSGetDiagnostic (errornumber)`
		Methoden zur Fehlerbe-handlung.

Data Transfer	`LMSGetValue (datamodel_element)`
	`LMSFinish (datamodel_element, value)`
	`LMSCommit ("")`
		Methoden zum Lesen und Schreiben von Datenelementen sowie zur Übertragung zwischengespeicherter Werte an das LMS.

Tabelle 1.2: Methoden der SCORM-API, nach [SCORM 2001c, S.7ff]

Data Model

Das Datenmodell der SCORM-Version 1.2 basiert direkt auf dem *AICC CMI Data Model*[8], wurde jedoch um einige Elemente reduziert. Mit ihm lassen sich spezifische Informationen eines SCOs, wie beispielsweise Objektstatus, Bearbeitungsdauer oder auch Konfigurations-daten, in der Datenbank einer Lernplattform speichern und abrufen. Bestimmte Daten-elemente erfordern zwingend eine Implementierung durch das LMS, wohingegen sie bei anderen optional ist und – sollte von ihr abgesehen werden – die SCORM-Konformität nicht beeinträchtigt. Die Verwendung der Elemente seitens eines SCOs unterliegt indes keinen speziellen Richtlinien und ist rein optional.

Ein Element innerhalb des Datenmodells ist hierarchisch strukturiert und kann in folgende Bestandteile untergliedert werden:

- dem zu Grunde liegenden Datenmodell[9]
- dem eigentlichen, kategorisierten Elementnamen (z.B. `core.session_time` oder `objectives.score`)[10]

Der Zugriff auf einzelne Datenelemente erfolgt über die API-Methoden `LMSGetValue()` bzw. `LMSSetValue()`.

[8] Ab SCORM 2004 basiert die Laufzeitumgebung (RTE) auf IEEE-Spezifikationen.

[9] Derzeit existiert ausschließlich `cmi`, jedoch ist die SCORM-Laufzeitumgebung offen für eine Implementierung weiterer Datenmodelle.

[10] Zusammengesetzt ergibt sich daraus `cmi.core.session_time` bzw. `cmi.objectives.score`.

1.3 Das Autorensystem Macromedia Flash

Mit Macromedia Flash existiert in der Welt der Autorensysteme ein umfangreiches und leistungsfähiges Entwicklungswerkzeug, dessen Schwerpunkt auf der Erstellung multimedialer und interaktiver Inhalte liegt, die vorwiegend über das Internet angeboten werden. Die Komplexität kann dabei sehr unterschiedlich ausfallen und reicht von einfachen Animationen bis hin zu aufwendig gestalteten, ästhetischen sowie funktionalen Anforderungen gerecht werdenden Anwendungen, Präsentationen oder Spielen.

Das Softwareunternehmen *Macromedia* wurde 1992 durch die Fusion der Firmen *Macromind* und *Authorware* gegründet. Drei Jahre später erfolgte die Übernahme von *FutureWave* und damit auch der Rechte an deren Softwareprodukten *FutureSplash-Animator* und *–Player* (vektorbasiertes Animationsprogramm und dazugehöriges Wiedergabe-Plugin für den Browser). Die Programme wurden durch Macromedia unter den Namen *Flash* sowie *Shockwave Flash Player* weiterentwickelt und 1996 schließlich in der Version 1 veröffentlicht. Seitdem hat Flash mehrere Entwicklungsstadien durchlaufen und ist vom einfachen Animationsprogamm zu einem ausgereiften und vielseitig einsetzbaren Autorensystem herangewachsen. Ende 2005 wurde die derzeit aktuelle Version 8 in einer Basic- sowie einer Professional-Variante veröffentlicht, nachdem im April desselben Jahres die Übernahme von Macromedia durch die amerikanische Firma *Adobe Systems* bekannt gegeben wurde. Sämtliche Produkte werden jedoch weiterhin unter der Marke „Macromedia“ vertrieben [Wikipedia 2006].

Im verbleibenden Teil des ersten Kapitels dieser Arbeit soll das Autorensystem Flash 8 überblicksweise vorgestellt werden.

Durch die Verwendung vektorbasierter Technologien, anhand derer sich grafische Objekte über mathematische Formeln beschreiben lassen und nicht – wie bei Rastergrafiken üblich – durch eine separate Definition jedes einzelnen Bildpunktes, lässt sich die Dateigröße der Flash-Filme (siehe Abschnitt 1.3.2) vergleichsweise gering halten. Gerade dieser Aspekt ist von nicht geringer Bedeutung, prädestiniert er doch Flash-Inhalte für ihren Einsatz im Internet und bildet somit auch eine wichtige Voraussetzung für die Entwicklung netzbasierter Lernangebote. Darüber hinaus wird der Leistungsumfang des Autorensystems durch die Unterstützung zahlreicher Grafik-, Sound- und Videoimportformate erweitert.

Neben den verschiedenen Werkzeugen zum Erstellen und Modifizieren von Objekten wird den Entwicklern von Flash-Anwendungen mit der integrierten Skriptsprache *ActionScript* (AS, siehe auch Abschnitt 1.3.4) zusätzlich ein sehr leistungsfähiges Hilfsmittel zur Seite gestellt. Obwohl auch programmiertechnisch unerfahrene Autoren durch einen Hilfemodus in die Lage versetzt werden, eigene Skripte zu erstellen, so offenbart eine intensive Auseinandersetzung mit Actionscript jedoch Wege, auch wesentlich komplexere Aufgaben zu bewältigen. Durch die Kombination aus grafischer Schnittstelle und leistungsfähiger Programmiersprache lässt sich Flash eindeutig der Kategorie der profes-

sionellen Autorensysteme (vgl. Abschnitt 1.1.4) zuordnen.

1.3.1 Die Entwicklungsumgebung

Die Entwicklungsumgebung lässt sich flexibel an die individuellen Bedürfnisse und Vorstellungen des Nutzers anpassen. So können Bedienelemente (ausgenommen die Bühne) ein- oder ausgeblendet, hinzugefügt, entfernt sowie neu arrangiert werden. Einmal vorgenommene Einstellungen lassen sich als Layout abspeichern und sind auf diese Weise jederzeit verfügbar.

In der Standardeinstellung beinhaltet Arbeitsoberfläche von Flash eine zweckgemäße Auswahl der am häufigsten verwendeten Bedienelemente, die im Folgenden kurz erläutert werden sollen (vgl. auch Abbildung 1.9):

Bühne

Die Bühne stellt den sichtbaren Bereich einer Flash-Anwendung dar. Auf ihr lassen sich die verschiedenen Objekte (Grafiken, Texte usw.) platzieren und modifizieren.

Zeitleiste

Mit der Zeitleiste lässt sich der zeitliche Ablauf eines Projektes organisieren und steuern. Sie setzt sich im Wesentlichen aus den folgenden drei Bestandteilen zusammen:

- Die *Ebenen* ermöglichen eine hierarchische Anordnung der Objekte auf der z-Achse und legen somit fest, welche von diesen sich im Vorder- bzw. im Hintergrund befinden.
- Die *Bilder* repräsentieren einerseits die jeweiligen Zustände, die einzelne Objekte zu bestimmten Zeitpunkten einnehmen. Andererseits dienen sie auch als eine Art Container für ActionScript-Code.
- Der *Abspielkopf* gibt immer die aktuelle Position an und damit auch, was aktuell auf der Bühne dargestellt wird.

Eine sinnvolle Verwendungsmöglichkeit der Zeitleiste besteht auch darin, verschiedene Zustände zu definieren und diese im Bedarfsfall gezielt aufzurufen. Auf diese Weise lässt sich eine Flash-Anwendung gut und übersichtlich strukturieren.

Werkzeugleiste

In der Werkzeugleiste sind nahezu alle Mittel enthalten, mit denen sich Objekte erstellen und modifizieren lassen. Darüber hinaus enthält sie Werkzeuge zum Ausrichten und Skalieren der Bühnenansicht.

Bibliothek

In der Bibliothek werden sämtliche wieder verwendbaren Elemente eines Flash-Dokuments gespeichert. Darunter zählen die so genannten *Symbole* (Movieclips, Schaltflächen, Grafiken), die jeweils über eine eigene Zeitleiste mit beliebig vielen Ebenen verfügen, wie auch Komponenten, Schriftarten oder importierte Medien[11]. Die Elemente werden auf der Bühne in Form von *Instanzen* verwendet, deren Anzahl keiner Begrenzung unterliegt. Der Vorteil dieses Verfahrens besteht darin, dass die eigentlichen Elemente nur einmal abgespeichert werden müssen und die Dateigröße der Flash-Filme (siehe weiter unten) aufgrund dessen relativ gering bleibt.

Bibliotheksbestände lassen sich darüber hinaus von mehreren Flash-Projekten gemeinsam verwenden. Dadurch lässt sich die Effizienz gerade bei der Entwicklung größerer Projekte spürbar erhöhen, da Änderung nur einmal zentral vorgenommen werden müssen. Man spricht in diesem Fall von *Shared Libraries*[12]. Bedingt durch die Art der Bereitstellung der Bestände wird grundsätzlich zwischen zwei Bibliotheksarten unterschieden:

- die *Runtime Shared Library* stellt die in ihr enthaltenen Elemente anderen Projekten zur Laufzeit zur Verfügung
- auf die Bestände einer *Authoring Shared Library* kann während der Entwicklung zugegriffen werden

Eigenschafteninspektor

Der Eigenschafteninspektor gewährt Zugriff auf die am häufigsten verwendeten Parameter der Bühne, Zeitleiste, Werkzeuge oder Objekte. Darüber hinaus bietet er die Möglichkeit, die mit der aktuellen Version eingeführten grafischen Filter (z.B. Schlagschatten, Weichzeichnen usw.) auf Texte, Movieclips oder Schaltflächen anzuwenden.

Farbmischer

Mit dem Farbmischer lassen sich die Farben von Linien und Füllungen den individuellen Wünschen anpassen. Neben einzelnen Werten des Farbspektrums sind dabei auch komplexe Verläufe oder Bitmapmuster möglich.

Aktionen-Fenster

In diesem Fenster lässt sich ActionScript-Code einfügen und editieren. Programmiertechnisch unerfahrene Entwickler erhalten Unterstützung in Form von vorgefertigten Quellcode-Elementen sowie einer menügesteuerten Parametervergabe.

---

[11] Komponenten, Schriftarten und importierte Medien laufen in Flash häufig ebenfalls unter der Bezeichnung „Symbol".

[12] geteilte Bibliotheken

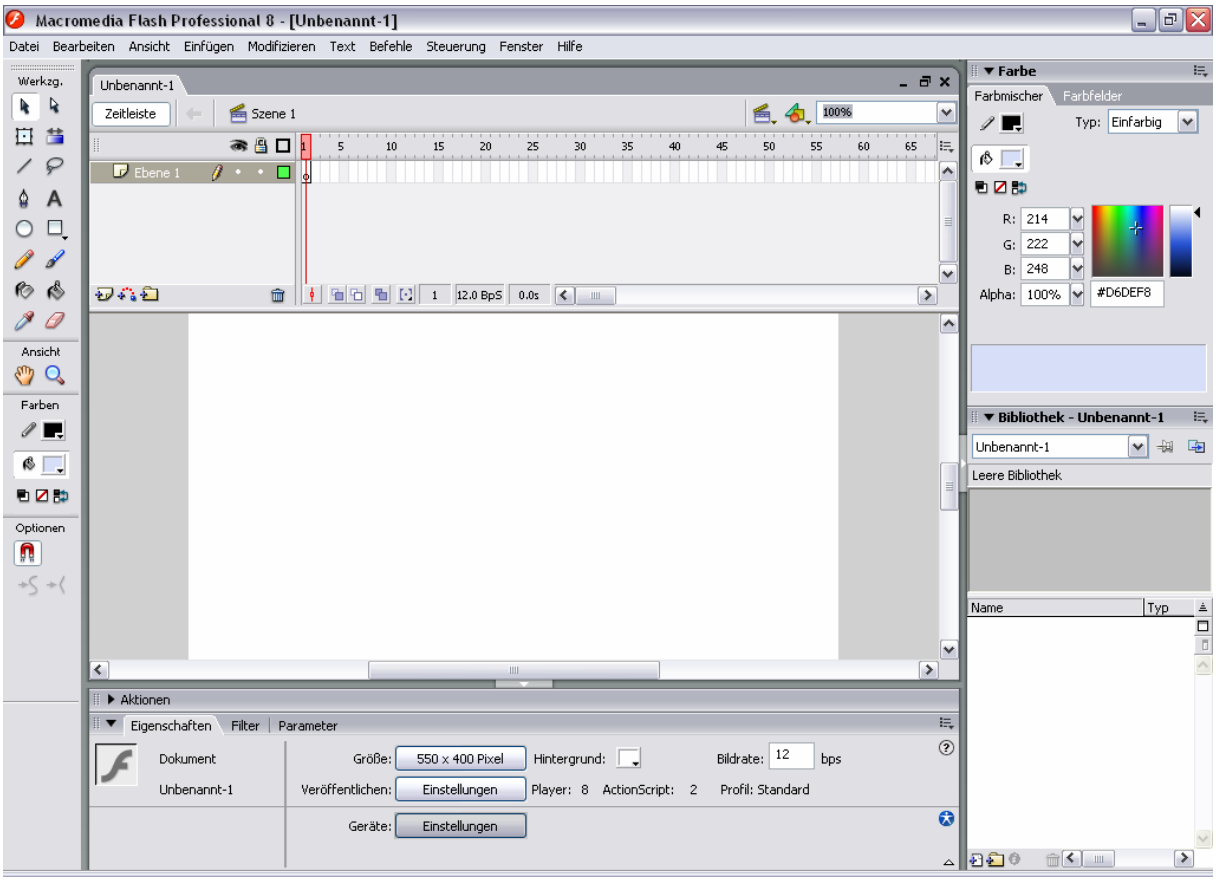

Abbildung 1.9: Die Entwicklungsumgebung von Macromedia Flash 8

Die in der Entwicklungsumgebung erstellten Projekte lassen sich als Dokumente mit der Dateierweiterung *fla* abspeichern. Sie enthalten lediglich die Quelldaten in Form von Objekten und ActionScript-Anweisungen, repräsentieren jedoch noch nicht den eigentlichen Flash-Film. Um die externe Wiedergabe zu ermöglichen, müssen die Daten zunächst in ein anderes Format kompiliert werden, wie es beispielsweise auch bei der Softwareentwicklung unter C++ oder Java der Fall ist. Zu diesem Zweck existiert im Autorensystem der Punkt *Veröffentlichen*, anhand dessen sich Filme im *Shockwave Flash Format*[13] erzeugen und auf Wunsch komprimieren lassen. Das Format stellt einen offenen Standard dar und erfährt als solcher mittlerweile auch Unterstützung durch andere Anwendungen.

Für die Wiedergabe der SWF-Dateien wird anschließend eine spezielle Software benötigt, der bereits erwähnte *Shockwave Flash Player*. Dieser existiert einerseits in Form von Browser-Erweiterungen (PlugIns) und ermöglicht somit die Darstellung von Flash-Inhalten im Netz, andererseits als eigenständige Applikation.

[13] Filme im Shockwave Flash Format besitzen die Dateierweiterung *swf*.

1.3.2 ActionScript 2.0

Wie bereits erwähnt, bezeichnet *ActionScript* die programminterne Skriptsprache von Flash. Über diese können unter anderem periphere und interne Ereignisse, wie beispielsweise Nutzereingaben oder Bildwechsel, erfasst und anschließend verarbeitet werden.

ActionScript hielt erstmals 1999 Einzug in das Autorensystem, das zu diesem Zeitpunkt in der vierten Version auf den Markt gebracht wurde. Ein Jahr darauf folgte die nächste Flash-Version und mit ihr auch eine Anpassung der Sprache an den *ECMAScript (ECMA-262) Edition 3*-Standard, auf dem auch JavaScript basiert. Obwohl natürlich Kenntnisse in einer der beiden anderen Sprachen keine Voraussetzung für die Beherrschung von ActionScript darstellen, so erweisen sie sich durchaus als hilfreich bei der Flash-Programmierung, gleichen sich doch oftmals Syntax und Semantik untereinander.

Wurde in Flash MX (Nachfolger der Version 5) der Befehlssatz um zahlreiche Elemente ergänzt und strenger an die ECMA-Normen gebunden, so war die Einführung von Flash MX 2004 im Jahr 2003 zugleich die Geburtsstunde von ActionScript 2.0 (AS2). Dabei handelt es sich um keine von Grund auf neue Sprache, sondern vielmehr um eine umfangreiche Überarbeitung und Erweiterung der Vorgängerversion. Der wohl wichtigste Aspekt im Zusammenhang mit der aktualisierten Sprachversion war die Implementierung eines neuen objektorientierten Klassenmodells, basierend auf dem noch nicht standardisierten Entwurf des ECMAScript Edition 4. Obwohl objektorientierte Programmierung (OOP) im Prinzip auch schon mit ActionScript 1.0 möglich war, mangelte es darin an traditionellen syntaktischen Strukturen, wie sie bei der Erstellung von Klassen in anderen Programmiersprachen häufig Verwendung finden[14]. Insbesondere denjenigen, die bereits Erfahrungen im Umgang mit Java gesammelt haben, wird die Einarbeitung in das neue Klassenmodell relativ leicht fallen, ist es doch stark an diese Programmiersprache angelehnt. Ebenso neu ist die Möglichkeit, eine *strikte Typisierung*[15] von Variablen vorzunehmen. Dadurch kann der Flash-Compiler beispielsweise falsche Wertzuweisungen erkennen oder auch Codehinweise anzeigen. In Flash 8 wurde der Umfang von ActionScript nochmals erweitert und beinhaltet neue Klassen, Methoden sowie andere Sprachelemente. Zudem wird die alte Syntax von ActionScript 1.0 weiterhin unterstützt, ebenso wie die Verwendung neuen Vokabulars unter selbiger, so dass ein Umstieg nicht zwingend erforderlich ist und der gewohnte Programmierstil oftmals beibehalten werden kann.

Es bliebe letztlich noch die Frage zu klären, wo der ActionScript-Code eines Flash-Dokuments platziert wird. Zum einen lässt er sich *intern* definieren[16], entweder direkt auf Movieclip- oder Schaltflächen-Instanzen oder in Schlüsselbildern. Die zweite Variante stellt

[14] In ActionScript 2 wurde beispielsweise das `class`-Schlüsselwort eingeführt zur Definition einer Klasse.

[15] Variablen lassen sich mit einem eindeutigen Datentyp deklarieren, z.B. `var x:Number`.

[16] AS2-Klassen *müssen* in externen (Text)dateien mit der Endung *as* definiert werden.

hierbei die bessere und effizientere Lösung dar, da sich der Quellcode zentral und übersichtlicher verwalten lässt. Eine weitere Möglichkeit besteht darin, Skripte *extern* in Textdateien zu definieren und über den `#include`-Befehl zum Zeitpunkt der Veröffentlichung eines Flash-Projektes einzubinden. Die Vorteile dieser Vorgehensweise liegen in einer leichteren Pflege des Quellcodes sowie der Mehrfachnutzung häufig verwendeter Programmsegmente. Die Dateierweiterung kann frei gewählt werden, wobei das häufig verwendete Kürzel *as* sowohl Vor- als auch Nachteile besitzt. Einerseits ist es seit der Version MX 2004 von Flash ein registrierter Datentyp und lässt sich automatisch in der Entwicklungsumgebung öffnen und editieren. Andererseits werden die Klassen von ActionScript 2.0 ebenfalls unter der Dateierweiterung *as* abgespeichert, was zu Missverständnissen führen kann. Vorbeugend sollte daher auf eine geeignete Verzeichnisstruktur bei der Entwicklung von Flash-Projekten geachtet werden, in der Klassen und normale Programmskripte klar voneinander abgegrenzt werden[17].

1.3.3 Entwicklungshilfen für Lernanwendungen

Autoren von interaktiven Lernanwendungen stehen in Flash verschiedene Entwicklungshilfen zur Verfügung. Neben der Möglichkeit, Inhalte AICC- oder SCORM-konform zu veröffentlichen (siehe Abschnitt 3.4.1), existieren auch vorgefertigte *Lerninteraktionen*, mit denen sich das erarbeitete Wissen eines Lernenden überprüfen lässt. Die Interaktionen basieren auf dem Komponentenmodell von Flash und können auf zwei Arten verwendetet werden: separat oder als Bestandteil einer Quiz-Vorlage.

Komponenten

Die Komponenten wurden mit Flash MX eingeführt und stellen eine Erweiterung des SmartClip-Konzepts aus Flash 5 dar. Sie bieten Entwicklern einfachen Zugriff auf häufig benötigte Funktionalität, wie beispielsweise Bildlaufleisten (*ScrollPane*-Komponente) oder Kombinationsfelder (*ComboBox*-Komponente). Die Komponenten unterlagen in Flash MX 2004 einer kompletten Überarbeitung und basieren nunmehr auf der Version 2 des Macromedia Komponentenmodells, das weitaus flexibler und leistungsfähiger ist als sein Vorgänger.

Eine Komponente ist vereinfacht gesagt ein mit einer AS2-Klasse verknüpfter Movieclip, dessen Parameter zum Zeitpunkt der Erstellung eines Flash-Dokuments festgelegt werden. Darüber hinaus besteht auch die Möglichkeit, Komponenten über ActionScript-Methoden, -Eigenschaften oder -Ereignisse zur Laufzeit individuell anzupassen. Den Entwicklern bietet dieses Konzept den Vorteil, komplexe Funktionen ohne hohen Programmieraufwand in ihre

[17] Eine solche Verzeichnisstruktur könnte beispielsweise zwei separate Ordner namens *classes* und *scripts* beinhalten.

Projekte zu integrieren. Neben der Verwendung vorgefertigter Komponenten erlaubt Flash zusätzlich auch die Erstellung eigener. Aufgrund der Möglichkeit zur Mehrfachnutzung in verschiedenen Flash-Dokumenten lassen sich Entwicklungsprozesse umfangreicher (Lern)anwendungen somit effizienter gestalten.

Lerninteraktionen

Die in Flash implementierten Lerninteraktionen wurden auf Basis der Komponentenarchitektur erstellt und sind Bestandteil einer so genannten *Allgemeinen Bibliothek*. Bei einer solchen handelt es sich um ein gewöhnliches FLA-Dokument, dessen Bibliotheksbestand anderen Projekten während deren Entwicklung zur Verfügung gestellt wird. Allgemeine Bibliotheken werden entweder separat hinzu geladen oder – wie im Fall der Lerninteraktionen – direkt in die Menüleiste von Flash integriert und von da aus aufgerufen. Für letztere Variante müssen die FLA-Dateien jedoch in einem speziell dafür vorgesehenen Verzeichnis abgespeichert werden, das sich plattformspezifisch unterscheidet. Anders als bei einer Authoring Shared Library verweisen die kopierten Bibliothekselemente jedoch nicht auf das Originalsymbol und sind dadurch auch nicht von einer eventuellen Änderung desselben betroffen.

Flash stellt in der aktuellen Version sechs verschiedene Arten von Lerninteraktionen zur Verfügung, die sich in Funktion und Aussehen anpassen lassen:

- *Drag and Drop*: Ein Objekt muss auf das dafür vorgesehene Feld gezogen werden, oder die Frage gilt als falsch beantwortet.
- *Fill In The Blank*: Ein vom Nutzer eingegebener Text wird mit einer oder mehreren zuvor festgelegten richtigen Antworten verglichen.
- *Hot Object*: Der Nutzer muss mit der Maus auf ein oder mehrere Objekte auf dem Bildschirm klicken, um die Frage richtig zu beantworten.
- *Hot Spot*: Der Nutzer muss mit der Maus auf ein oder mehrere Bildschirmbereiche klicken, um die Frage richtig zu beantworten
- *Multiple Choice*: Auf eine Frage muss der Nutzer eine Auswahl zwischen einer oder mehreren richtigen Antworten treffen.
- *True or False*: Der Nutzer muss entscheiden, ob eine aufgestellte Behauptung wahr oder falsch ist.

Jede Lerninteraktion kann Verfolgungs-Informationen an ein AICC-kompatibles LMS senden, wenn es mit der HTML-Vorlage *Flash mit AICC-Verfolgung* veröffentlicht wird. Darin sind sämtliche für die Kommunikation benötigten JavaScript-Funktionen bereits enthalten. Zur erfolgreichen Integration in eine SCORM-Umgebung müssen jedoch manuelle Anpassungen des betreffenden ActionScript-Quellcodes vorgenommen sowie eine entsprechende Veröffentlichungsvorlage gewählt werden.

Quiz

Mit den Quiz-Vorlagen, die anstelle eines neuen leeren Dokuments geöffnet werden können, stellt Flash Autoren bereits vorgefertigte Testszenarien für die Lernerfolgskontrolle zur Verfügung. Es existieren drei verschiedene Varianten, die sich lediglich in ihrer äußeren Form unterscheiden, jedoch nicht hinsichtlich der implementierten Funktionalität. Die Abbildung 1.10 zeigt ein Beispiel:

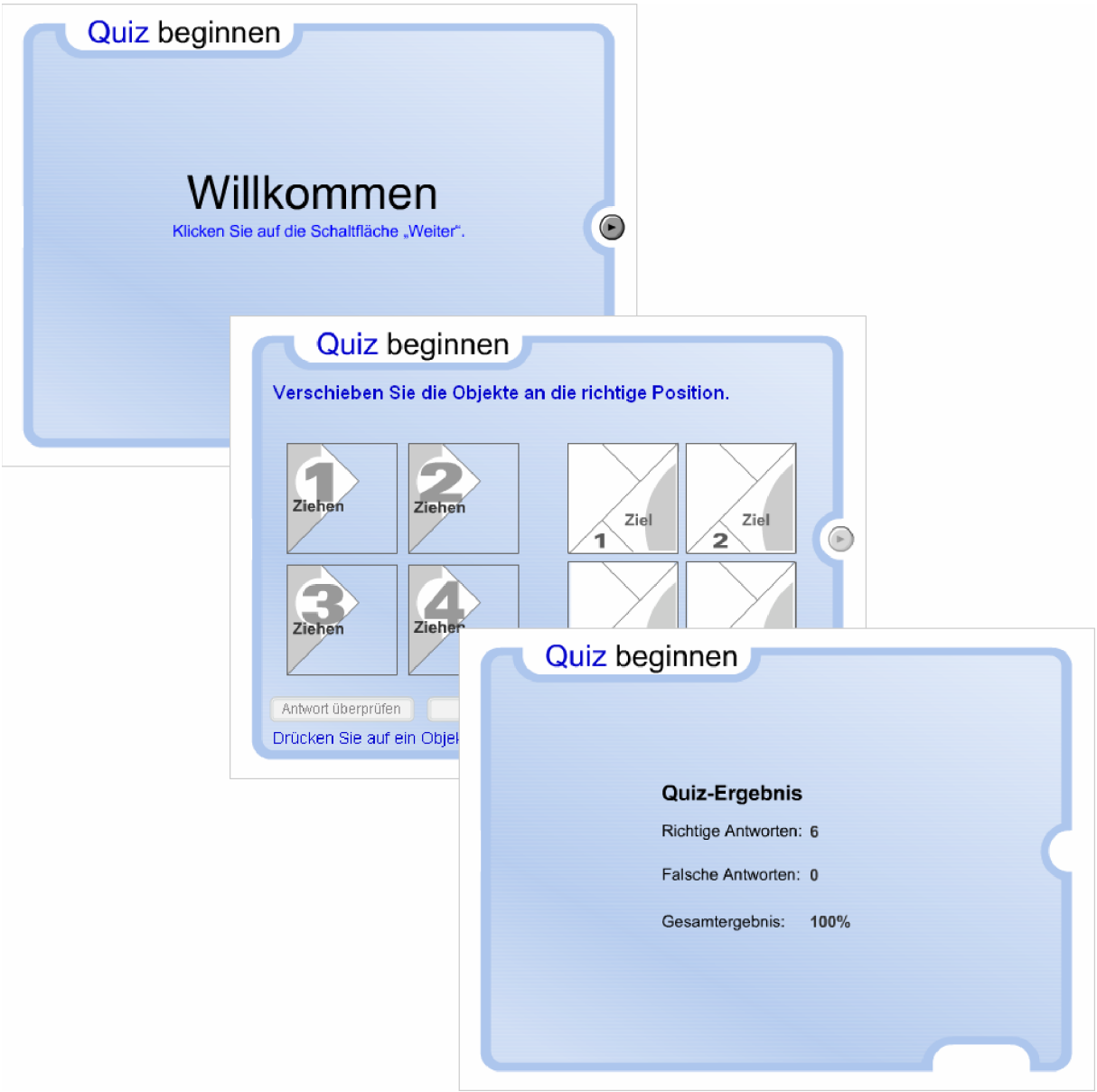

Abbildung 1.10: Beispiel-Bildschirme einer Quiz-Vorlage

Eine Quizvorlage setzt sich aus folgenden Bestandteilen zusammen:

- einer Begrüßungsseite
- sechs Lerninteraktionen jeweils unterschiedlichen Typs
- einer Ergebnisseite

- verschiedenen Navigationselementen
- ActionScript-Code zur AICC- sowie SCORM-Verfolgung

Die integrierten Lerninteraktionen lassen sich individuell anpassen, dem Nutzer zufallsgesteuert darbieten sowie um weitere ergänzen. Die erzielten Einzelergebnisse werden gesammelt und verschmelzen entsprechend ihrer festgelegten Gewichtung[18] zu einem Gesamtergebnis, das dem Nutzer auf der letzten Seite des Quiz präsentiert wird. In Kombination mit einer entsprechenden HTML-Vorlage kann ein Quiz auch zur Verwendung innerhalb AICC- oder SCORM-konformer Lernumgebungen veröffentlicht werden. Anschließend ist er auch in der Lage, Verfolgungs-Informationen an ein serverseitiges Lernmanagementsystem zu übertragen.

[18] Die Gewichtung ist ein definierbarer Parameter einer Lerninteraktion.

2 Konzeption und Gestaltung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“

2.1 Entwicklungsmodell für softwareunterstützte Lernanwendungen

Für die Entwicklung von softwareunterstützten Lernanwendungen existieren eine Reihe von untereinander oftmals nur geringfügig abweichenden Modellen, die allgemeine Methoden des klassischen Projektmanagements um spezifische Strukturen, Routinen und Regeln des E-Learning-Bereichs ergänzen. Schreiber dokumentiert ein derartiges Entwicklungsmodell und untergliedert es in drei Abschnitte: *Planung*, *Produktion* sowie *Postproduktion*. Innerhalb dieser unterscheidet er weiterhin zwischen verschiedenen Entwicklungsphasen, die größtenteils durch so genannte *Meilensteine* voneinander abgegrenzt werden. Dabei handelt es sich um zeitliche Prüfpunkte, bei denen bestimmte Ergebnisse vorliegen müssen, ohne die mit anderen Arbeiten des Projekts nicht begonnen oder fortgefahren werden kann [Schreiber 1998, S.47ff]. Die folgende Abbildung stellt die einzelnen Phasen überblicksweise dar und veranschaulicht die Zusammenhänge zwischen verschiedenen Entwicklungsschritten:

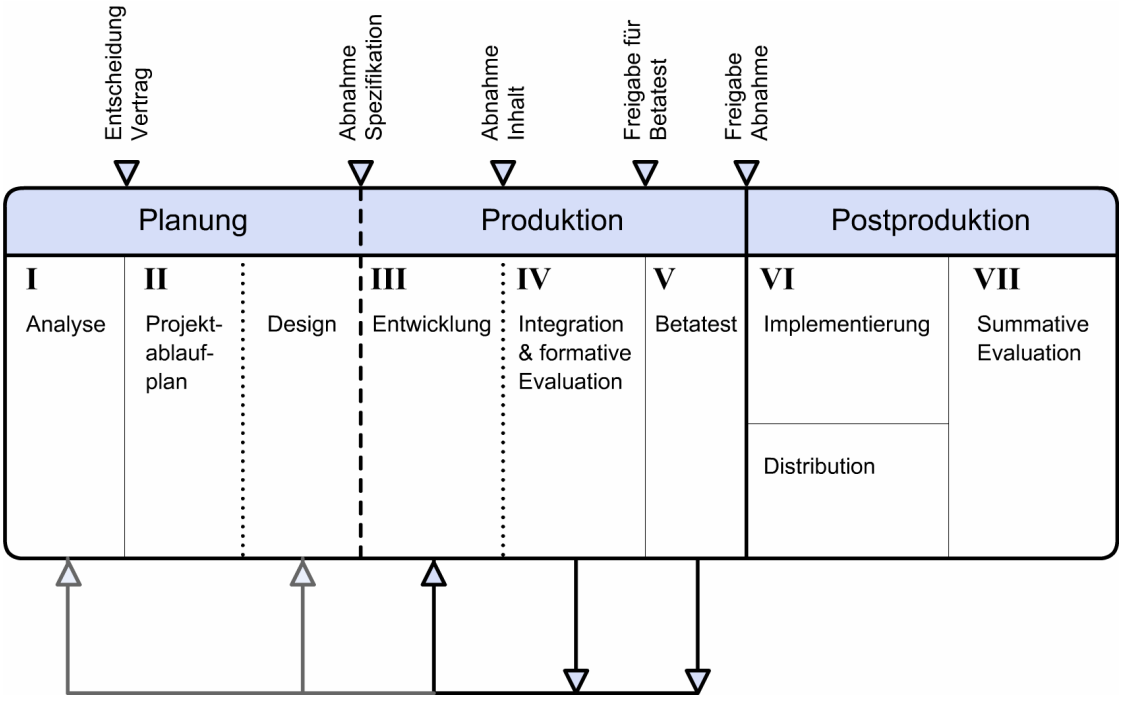

Abbildung 2.1: Das Entwicklungsmodell im Überblick, modifiziert nach [Schreiber 1998, S.84]

Am Anfang der *Planungsphase* stehen eine genaue Analyse der gegebenen Rahmenbedingungen sowie eine grobe Spezifikation des in der späteren Anwendung zu vermittelnden Wissens. Dabei sollten folgende Punkte einer genaueren Betrachtung unterzogen werden [Schreiber 1998, S.52ff] [Niegemann u.a. 2004, S.51ff]:

- *Formulierung des Themas*: Hierbei werden das Thema des Lernprogramms sowie ein eventueller Arbeitstitel festgelegt, außerdem erfolgt eine kurze Inhaltsbeschreibung.
- *Adressaten (Zielgruppe)*: In der Adressatenanalyse werden Daten zu vorhandenen Kompetenzen sowie weiteren personenspezifischen Merkmalen (Alter, Geschlecht, Bildungsstand usw.) der Zielgruppe erhoben.
- *Lernziel(e)*: Dieser Punkt umfasst die Festlegung von Kenntnissen und Fähigkeiten, die durch das Lernprogramm vermittelt werden sollen, sowie eine Spezifikation des dafür notwendigen Basiswissens.
- *Ressourcen*: Die Ressourcenanalyse befasst sich mit der Frage, welche Mittel (hard- und softwareseitige Materialien und damit verbundene Rechte, Personal, Geld, Zeit) für die Entwicklung zur Verfügung stehen bzw. benötigt werden.
- *Einsatzkontext*: Es sollte geklärt werden, wie (z.B. einzeln oder in der Gruppe) und unter welchen Bedingungen (Hardware, Software) die spätere Lernanwendung eingesetzt wird.

Mit den aus der Analyse gewonnenen Kenntnissen lässt sich anschließend der weitere Projektablauf planen. Dabei werden die im Einzelnen anstehenden Aufgaben auf die verschiedenen Mitglieder eines Entwicklungsteams verteilt sowie der jeweilige Bearbeitungszeitrahmen bestimmt.

Der nächste Schritt beschreibt die so genannte *Designphase*. Diese umfasst neben der Entwicklung eines didaktischen Konzepts zur Wissensvermittlung auch die Gestaltung einer optisch und funktionell ansprechenden Benutzungsoberfläche. Ein weiterer Aspekt, der nicht vernachlässigt werden sollte, ist der Entwurf einer geeigneten programminternen Struktur, mit der sich die vorgegebenen Ziele in effizienter Weise verwirklichen lassen. Den Abschluss einer Designphase bildet häufig die prototypische Umsetzung der erarbeiteten Spezifikationen, womit letztlich deren Beurteilung sowie Abnahme (Meilenstein) durch die dafür Verantwortlichen (Auftraggeber und/oder Projektleiter) begünstigt werden [Schreiber 1998, S.81ff].

Nach der Planung folgen die verschiedenen *Produktionsphasen*. In einem ersten Schritt werden zunächst die Komponenten einer Lernanwendung möglichst getrennt voneinander entwickelt. Dazu zählen unter anderem *Texte* (Basaltexte, Drehbücher etc.), *grafische Elemente* (Einzelabbildungen, Animationen etc.), *audiovisuelles Material* (Sprachaufnahmen, Videos etc.) sowie Bestandteile der *Benutzungsoberfläche*. Begleitend sollte auch eine Überprüfung hinsichtlich der Qualität seitens der Projektleitung stattfinden, um beispielsweise inhaltliche oder sprachliche Mängel der Texte rechtzeitig aufzudecken.

Nach der Abnahme der Komponenten, die als Meilenstein betrachtet werden kann, geht es im folgenden Schritt darum, die einzelnen Elemente innerhalb einer geeigneten Programmstruktur zu komplexeren Bildungseinheiten zusammenzufügen. Für gewöhnlich entstehen

während dieser Phase schon lauffähige Segmente des späteren Lernmoduls, die mit Hilfe so genannter *Pilottests* durch möglichst unabhängige Personen auf ihre technische, gestalterische sowie didaktische Qualität hin überprüft werden. Diese Prozedur wird auch als *formative Evaluation* bezeichnet und hilft dabei, frühzeitig etwaige Probleme zu erkennen und zu eliminieren.

Bevor das komplette Lernmodul seine finale Absegnung erhält und zur Veröffentlichung freigegeben wird, durchläuft es zunächst einen intensiven *Betatest* (Meilenstein). Dieser beinhaltet einerseits einen *technischen Test* unter verschiedenen Hard- und Softwarevoraussetzungen, andererseits einen so genannten *Feldtest*, bei dem mehrere Vertreter der Zielgruppe mit dem Programm konfrontiert werden, um Rückschlüsse hinsichtlich der pädagogischen Eignung und Bedienbarkeit zu erhalten. Wie bereits aus der Abbildung 2.1 hervorgeht, können die daraus resultierenden Ergebnisse im ungünstigsten Fall zu einer kompletten Überarbeitung des Designs oder gar einer wiederholten Analyse führen.

Nach Abschluss aller Tests und eventueller Anpassungen erfolgt die Freigabe des Lernmoduls, womit gleichzeitig auch der Abschnitt der *Postproduktion* beginnt. Dieser werden nach dem Modell von Schreiber die Einbindung des Programms in bestehende Lernorganisationen (Implementierung) bzw. Verfielfältigung, Endfertigung und Vertrieb des fertigen Produkts (Distribution) zugeordnet. In einer abschließenden *summativen Evaluation* werden vergleichende Untersuchungen hinsichtlich Lernerfolg und –dauer zu anderen Unterrichtsmedien (Seminare, Lehrbücher etc.) angestellt, die Effektivität des Produkts bewertet sowie eine Kosten-Nutzen-Analyse durchgeführt [Schreiber 1998, S.85ff].

Im weiteren Verlauf wird auf die Planung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“ eingegangen, bevor sich das fünfte Kapitel mit den Abschnitten Produktion und Postproduktion auseinandersetzt.

2.2 Analysephase

Das Lernmodul „Die interaktive Arbeit mit Flash MX 2004“ entstand im Auftrag des Bildungsportals Sachsen sowie der Hochschule für Technik und Wirtschaft (HTW) Dresden, vertreten durch Frau Prof. Dr. Teresa Merino. Bezüglich der Entwicklung wurden verschiedene technische, gestalterische und inhaltliche Vorgaben gemacht, so dass gewisse Rahmenbedingungen bereits gegeben waren, auf die in den folgenden Abschnitten eingegangen wird.

Da sämtliche Fragen bezüglich der Produktion ausschließlich in Zusammenarbeit mit Prof. Dr. Merino geklärt wurden, soll von ihr im weiteren Verlauf dieser Arbeit als der Auftraggeberin gesprochen werden.

2.2.1 Adressaten und Einsatzkontext

Die Lernanwendung soll einerseits begleitend zu der von der Auftraggeberin gehaltenen Lehrveranstaltung „Entwicklungswerkzeuge für Multimediasysteme“ im Studiengang Medieninformatik eingesetzt werden und den Studenten die Möglichkeit bieten, das in traditioneller Form vermittelte Wissen zu vertiefen. Andererseits wurde das Modul gemäß vertraglich festgelegter Bestimmungen für seine Verwendung innerhalb der SCORM-konformen Lernumgebung des Bildungsportals Sachsen konzipiert und steht dort sämtlichen registrierten Nutzern – hauptsächlich Studenten verschiedener Fachrichtungen sowie Dozenten – als so genanntes *Selbstlernmodul* zur Verfügung.

Aus den beiden Einsatzgebieten lassen sich folglich zwei Adressaten ableiten. Bei genauerer Betrachtung ist jedoch eindeutig festzustellen, dass die Studenten der Medieninformatik eine Teilmenge der möglichen Nutzer des BPS darstellen, weshalb Letztere somit auch als relevante Zielgruppe angesehen werden können. Ausgehend von dieser Festlegung lassen sich den Adressaten verschiedene Merkmale zuweisen. So ist bei diesen nicht zwingenderweise davon auszugehen, dass sie im Umgang mit Computern versiert sind. Ebenso wenig lassen sich bereits gesammelte Erfahrungen mit Flash oder diversen Programmiersprachen als selbstverständliches Basiswissen deklarieren. Es ist jedoch zu vermuten, dass bei den Personen der Zielgruppe keine grundlegend ablehnende Haltung in Hinblick auf softwareunterstütztes Lernen vorherrscht und dass ein gewisses Niveau an linguistischen Fähigkeiten nicht unterschritten wird.

Wie das Lernen mit dem Modul erfolgt, lässt sich nicht eindeutig auf eine Form begrenzen. In die eingangs erwähnte Lehrveranstaltung des Studiengangs Medieninformatik wird es als Bestandteil eines Blended Learning-Konzepts (vgl. Abschnitt 1.1) integriert. Daneben sollte jedoch auch bedacht werden, dass sich der Zugriff auf das Lernmodul über das Bildungsportal Sachsen orts- und zeitunabhängig gestaltet. Somit ist es einem einzelnen Lernenden beispielsweise auch möglich, die Software vom privaten netzgebundenen Rechner aus als reine WBT-Lösung zu nutzen.

Bezüglich der hard- und softwaretechnischen Voraussetzungen, unter denen die Lernanwendung nach Ihrer Fertigstellung lauffähig sein sollte, gab es seitens der Auftraggeberin genaue Spezifikationen. So wurde für die Netzanbindung eine Mindestbandbreite von einem Megabit je Sekunde (1 MBit/s) festgelegt. Diese Geschwindigkeit entspricht einer Standard-DSL-Verbindung (Digital Subscriber Line) und wurde in Folge einer Umfrage gewählt, die unter Studenten des Studiengangs Medieninformatik der HTW Dresden zu Beginn des Jahres 2005 stattfand. Selbige ergab, dass die meisten Befragten über einen entsprechend schnellen Internet-Zugang vom heimischen Arbeitsplatz aus verfügen. Daneben fallen die Bandbreiten der HTW-Computerlabore sowie der Studentenwohnheime nicht ins Gewicht, da sie mit jeweils 10 MBit/s ohnehin als ausreichend betrachtet werden können.

Die Arbeit mit der Lernanwendung sollte idealerweise in Kombination mit der Flash-

Entwicklungsumgebung erfolgen, die seit Einführung der Version MX 2004 eine ausschließliche Unterstützung für folgende Betriebssysteme bietet: Mac OS X ab Version 10.3, Windows 2000 sowie Windows XP. Beruhend auf dieser Tatsache galt für die Entwicklung des Lernmoduls die Anforderung, mindestens eine uneingeschränkte Lauffähigkeit unter den genannten Plattformen zu garantieren. Da es sich um eine netzbasierte Lösung handelt, spielt in diesem Zusammenhang natürlich auch die Frage nach den zu unterstützenden Internetbrowsern eine Rolle. So bestand seitens der Auftraggeberin die Forderung, eine problemlose Abarbeitung des Lernmoduls unter folgenden Software-Umgebungen sicherzustellen: dem *Apple Safari 1.x* für Macintosh, dem *Microsoft Internet Explorer 6.x* für Windows sowie der Windows-Version des *Mozilla Firefox 1.x*.

Darüber hinaus wurden vertraglich auch Richtlinien bezüglich der softwaretechnischen Entwicklung spezifiziert. Ein Punkt umfasste dabei die Einbindung des Lernmoduls in die Lernplattform des Bildungsportals Sachsen über die SCORM-Schnittstelle. Wie bereits unter Abschnitt 1.2.2 erwähnt, erforderte das zum damaligen Zeitpunkt im BPS integrierte Lernmanagementsystem eine Implementierung gemäß der SCORM-Spezifikation 1.2, da es ihm an der Unterstützung einer aktuelleren Version mangelte. Des Weiteren sollte das Lernmodul aufgrund der in Abschnitt 1.3 erläuterten Vorzüge mit Flash realisiert werden. Für die Entwicklung stand die Version MX 2004 (und damit auch der Flash Player 7) zur Verfügung, jedoch galt zunächst die Richtlinie, die Filme kompatibel zum Flash Player 6 zu veröffentlichen. Während der Design-Phase allerdings wurde dieses Konzept – in Übereinstimmung mit der Auftraggeberin – zugunsten der aktuelleren Version wieder verworfen, bot jene doch zahlreiche neue und effizientere Möglichkeiten zur Umsetzung der gestellten Anforderungen. Folgerichtig ist ein im verwendeten Internetbrowser installiertes Flash Player 7-Plugin zwingend notwendig, um die Lerninhalte in korrekter Art und Weise wiederzugeben.

2.2.2 Lernziele

Anhand von Lernzielen wird spezifiziert, welche Kenntnisse und Fähigkeiten dem Lernenden vermittelt werden sollen. Basierend auf Erkenntnissen der Pädagogischen Psychologie lassen sie sich in folgende Kategorien unterteilen [Kerres 2001, S.155ff][Schreiber 1998, S.20]:

- *Kognitive Lernziele*: Diese Kategorie bezieht sich auf geistige Fähigkeiten. So sollen das vermittelte Wissen (Fakten, Prinzipien, Prozeduren etc.) aus dem Gedächtnis abrufbar, komplexe Zusammenhänge durchschaubar und erarbeitete Kenntnisse in unterschiedlichen Kontexten anwendbar sein. Darüber hinaus können auch Tätigkeiten wie Analysieren, Synthetisieren sowie Beurteilen mit den kognitiven Lernzielen verknüpft werden.
- *Affektive Lernziele*: Diese Kategorie bezieht sich auf Änderungen von Gefühlen, Interessen, Einstellungen und Werten. Die Fähigkeit seitens des Lernenden, Werte beurteilen und einordnen zu können, soll sich in einer danach ausgerichteten

Verhaltensweise niederschlagen.

- *Psychomotorische Lernziele*: Diese Kategorie befasst sich mit der Beherrschung von Bewegungsabläufen, die eine Koordination unterschiedlicher (grob- und fein-)motorischer Subsysteme erfordern, wie etwa sportliche oder handwerkliche Tätigkeiten.

Softwareunterstützte Lernanwendungen befassen sich zumeist mit dem kognitiven Bereich, indem sie häufig klar strukturierbare Themen aufgreifen und darauf ausgelegt sind, fundamentales Wissen zu vermitteln und Grundfertigkeiten einzuüben [Schreiber 1998, S.20]. Diese Aussage lässt sich auch auf das im Rahmen dieser Arbeit entstandene Lernmodul „Die interaktive Arbeit mit Flash MX 2004“ anwenden. So soll der Lernende zunächst mit den verschiedenen Elementen der Arbeitsoberfläche sowie deren Besonderheiten vertraut gemacht werden. Außerdem besteht das Anliegen, verschiedene Begrifflichkeiten und Konzepte näher zu erläutern, die im Zusammenhang mit der Flash-Entwicklung häufig auftauchen. Des Weiteren soll der Lernende in die Lage versetzt werden, mit den erworbenen Kenntnissen Zeichnungen, interaktive Elemente und Animationen zu erstellen. Das Wissen über die multimedialen Fähigkeiten von Flash bezeichnet schließlich das letzte der soeben genannten *Groblernziele*. Seitens der Auftraggeberin wurden diesbezüglich vier Themenbereiche spezifiziert:

- Die Entwicklungsumgebung
- Bibliotheken und Symbole
- Mit Flash zeichnen und animieren
- Mit Medien arbeiten

Die Groblernziele wurden im Einklang mit dem zu vermittelnden Lehrstoff der Veranstaltung „Entwicklungswerkzeuge für Multimediasysteme“ (siehe Abschnitt 2.2.1) festgelegt und sollen Lernende in die Lage versetzen, ohne hohen Programmieraufwand Flash-Projekte zu erstellen.

Neben dem bisher erwähnten Lernmodul sah die ursprüngliche Planung noch ein weiteres vor. Mit diesem sollte den Nutzern der Einstieg in die ereignisorientierte Welt von Flash MX 2004 erleichtert werden. So waren die grob spezifizierten Lernziele darauf ausgelegt, grundlegende Konzepte sowie das objektorientierte Klassenmodell von ActionScript 2 zu vermitteln und dieses Wissen anhand praktischer Aufgaben zu festigen. Aufgrund zeitlicher Engpässe wurde jedoch die Entwicklung in Absprache mit der Auftraggeberin nach der Fertigstellung des Drehbuchs nicht weiter fortgesetzt. Daher wurde in dieser Arbeit auf eine nähere Betrachtung verzichtet und der Fokus ausschließlich auf das realisierte Lernmodul gerichtet. Es ist jedoch problemlos möglich, die spezifizierten Inhalte in die geschaffene Programmstruktur sowie das Oberflächendesign zu integrieren, so dass der Weiterentwicklung prinzipiell nichts im Wege steht.

2.3 Designphase – Didaktische Konzeption

Die folgenden Abschnitte befassen sich mit der Frage, wie sich basierend auf den während der Analysephase erhobenen Daten eine geeignete Lehrstrategie entwickeln lässt. Im Rahmen eines didaktischen Konzepts werden unter anderem die Lernziele näher spezifiziert und einer inhaltlichen Gliederung unterworfen. Daneben erfolgt auch eine Festlegung von Lernwegstrukturen, einzusetzenden Medien sowie Interaktionsformen.

2.3.1 Lehrstrategie

Lehrstrategien beschreiben konkrete Vorgehensweisen, *wie* sich Wissen in geeigneter Form vermitteln lässt. Im Bereich der Mediendidaktik stellt sich demnach die Frage, wie die verschiedenen Lerninhalte optimal aufbereitet und präsentiert werden können, um Lernprozesse anzuregen und die Wissensbildung beim Lernenden zu fördern. Es existieren verschiedene Ansätze, die auf behavioristischen, kognitivistischen oder auch konstruktivistischen Lerntheorien beruhen. Es kann jedoch nicht von *der* einen, richtigen Lehrmethode gesprochen werden. Vielmehr erscheint ein situationsbedingter und kombinierter Einsatz verschiedener Konzepte der Wissensvermittlung als günstigere Option [Kerres 2001, S.55ff].

Im Zusammenhang mit Lehrstrategien ist auch interessant, zwischen welchen Wissenstypen im didaktischen Design allgemein unterschieden wird [Kerres 2001, S.160ff, S.168ff]:

- *Deklaratives Wissen (was)*: Kenntnisse über Fakten, Ereignisse, Objekte etc.
- *Prozedurales Wissen (wie)*: beschreibt Fertigkeiten basierend auf der Konstruktion, Verknüpfung sowie Anwendung deklarativer Wissensbestände
- *Kontextuelles Wissen (wann und wo)*: umfasst situationsabhängige Problemlösestrategien, also wann und wo welches Wissen anzuwenden ist

Im nächsten Abschnitt wird deutlich, dass sich die verschiedenen Lernziele den einzelnen Wissenstypen zuordnen lassen. Ziel der Designphase war es letztlich, didaktisch geeignete Methoden zu finden, mit denen sich die spezifischen Kenntnisse und Fähigkeiten an den Lerner weiterreichen lassen.

2.3.2 Feinlernziele

Die durch die Auftraggeberin grob spezifizierten Lernziele (siehe Abschnitt 2.2.2) wurden zunächst eingehender untersucht, um in Hinblick auf die anvisierte Zielgruppe *Feinlernziele* herauszuarbeiten. Daraus resultierte eine Unterteilung in fünf Kapitel, denen sich die in folgender Tabelle aufgelisteten Lernziele zuordnen lassen:

Kapitel	**Feinlernziele**
Die Entwicklungsumgebung	Der Lernende soll mit dem Aufbau der Entwicklungs-umgebung und den grundlegenden Eigenschaften wesentlicher Elemente der Arbeitsoberfläche vertraut gemacht werden. Des Weiteren soll er sich detailliertes Wissen zum Umgang mit der Zeitleiste sowie den einzelnen Werkzeugen aneignen können.
Gruppen, Symbole, Instanzen und Bibliotheken	Der Lernende soll Kenntnisse erlangen, wie sich mehrere Objekte in *Gruppen* zusammenfassen lassen, was *Symbole* sind, welche Arten es davon gibt und wie sich von diesen *Instanzen* bilden lassen. Außerdem wird erläutert, wie das Bedienfeld *Bibliothek* gehandhabt wird und was unter *Allgemeinen Bibliotheken* und *Geteilten Bibliotheken* zu verstehen ist.
Mit Flash zeichnen	Der Lernende soll im Umgang mit dem *Farbmischer* geschult sowie mit verschiedenen Techniken des Zeichnens von komplexen Vektor-Formen vertraut gemacht werden. Dadurch lassen sich gleichzeitig Kenntnisse über den aufgabenspezifischen Einsatz von Werkzeugen vermitteln.
Mit Flash animieren	Der Lernende soll sich Wissen darüber aneignen, welche Zeitleisteneffekte zur Verfügung stehen und wie sie angewendet werden. Außerdem soll er die Fertigkeit erlangen, Bewegungs- und Form-Tweens sowie Pfad-Animationen zu erstellen.
Die Arbeit mit Medien	Es soll vermittelt werden, welche unterschiedlichen medialen Dateiformate in Flash für den Im- und Export zur Verfügung stehen. Darüber hinaus wird dem Lernenden erläutert, wie sich importierte Grafiken, Sounds und Videos verwenden und bearbeiten lassen. Des Weiteren wird der mit Flash MX 2004 eingeführte Videoimport-Assistent detailliert beschrieben. Außerdem soll der Lernende mit den umfangreichen Möglichkeiten des integrierten Textfeld-Werkzeugs sowie der Rechtschreibprüfung vertraut gemacht werden.

Tabelle 2.1: Kapitel und zugehörige Feinlernziele

Am Beispiel der Werkzeuge lässt sich veranschaulichen, wie die unter Abschnitt 2.3.1

genannten Wissenstypen durch die Lernziele repräsentiert werden. So werden einerseits grundlegende Informationen zu den einzelnen Elementen der Werkzeugleiste vermittelt (deklarativ). Speziell im Kontext des Zeichnens von komplexen Vektorformen werden andererseits auch die werkzeugspezifische Anwendung (prozedural) sowie der situationsbedingte Einsatz und das Zusammenspiel verschiedener Werkzeuge (kontextuell) erläutert.

2.3.3 Unterteilung in Lernobjekte und inhaltliche Gliederung

Nach der Aufschlüsselung der Feinlernziele wurde eine inhaltliche Struktur erstellt. Neben der Frage einer didaktisch sowie logisch sinnvollen *Gliederung* musste dabei auch die notwendige *Segmentierung* der Lerninhalte erörtert werden. Dies erforderte eine nähere Betrachtung des Einsatzkontextes.

Wie bereits erwähnt, soll das Lernmodul im Bildungsportal Sachsen innerhalb einer SCORM-konformen Laufzeitumgebung zur Verfügung gestellt werden. Die Spezifikation des Referenzmodells beinhaltet mit dem Content Aggregation Modell (vgl. Abschnitt 1.2.3) eine Lösung zum Erstellen und inhaltsbezogenen Strukturieren von startbaren Lernobjekten (Assets und SCOs). Dabei repräsentieren SCOs die wieder verwendbare Form von Lerninhalten und sind im Gegensatz zu Assets in der Lage, mit dem serverseitigen LMS zu kommunizieren und Lernerdaten auszutauschen. Aufgrund dieser Tatsache fiel die Entscheidung, das Lernmodul als eine Sammlung von SCOs zu entwickeln.

Hinsichtlich der Segmentierung der Lerninhalte wurde sich an den Empfehlungen seitens der ADL Initiative orientiert. Diese besagen, dass ein SCO eine in sich geschlossene und vom umgebenden Kontext unabhängige, kleinstmögliche Lerneinheit darstellt. Die Feinlernziele ließen sich auf diese Weise in logisch zusammengehörige Wissensbausteine zerlegen, wie beispielsweise Kenntnissen zu Bedienelementen der Flash-Oberfläche (Zeitleiste, Farbmischer etc.) oder auch themenbezogene Übungsaufgaben (Zeichnen eines Stilllebens, Verknüpfung einer Schaltfläche mit Sound etc.).

Beruhend auf den *sachlogischen Zusammenhängen*, die zwischen den verschiedenen Inhalten existieren, ließen sich die einzelnen Lernobjekte einer inhaltlichen Gliederung unterziehen. Dies lässt sich stellvertretend am Beispiel der Pfadanimation veranschaulichen: Es wäre didaktisch unklug, diese Art der Animation zu erklären, bevor nicht grundlegende Kenntnisse zum Umgang mit der Zeitleiste oder den zum Einsatz kommenden Werkzeugen vermittelt wurden.

Die Lernobjekte wurden auf einer zweiten Ebene als separate Unterkapitel zu den in Tabelle 2.1 aufgelisteten Hauptthemen angeordnet. Erweitert wurde die Inhaltsstruktur um eine Einleitung, ebenfalls in Form eines Lernobjekts. Darin werden der grundlegende Aufbau, die verschiedenen Bedienelemente sowie überblicksweise der Inhalt des Lernmoduls erläutert. Des Weiteren wurde jedem Hauptthema die Funktion zugeordnet, den Lernenden mit den jeweils angestrebten Feinlernzielen vertraut zu machen. Diese beiden Ergänzungen schienen

im Rahmen des Lernmoduls als sinnvoll, obwohl sie lediglich auf einer übergeordneten Ebene Wissen vermitteln und mit dem eigentlichen Lehrstoff nur bedingt etwas zu tun haben. Sie stellen demnach keine kontextunabhängigen, mehrfach nutzbaren Lerneinheiten dar, wurden allerdings der Einfachheit halber ebenfalls als SCOs konzipiert und nicht als Assets.

Abbildung 2.2 veranschaulicht die grundlegende inhaltliche Struktur des Lernmoduls. Die vollständige Gliederung der einzelnen Lernobjekte ist dem Anhang A dieser Arbeit zu entnehmen.

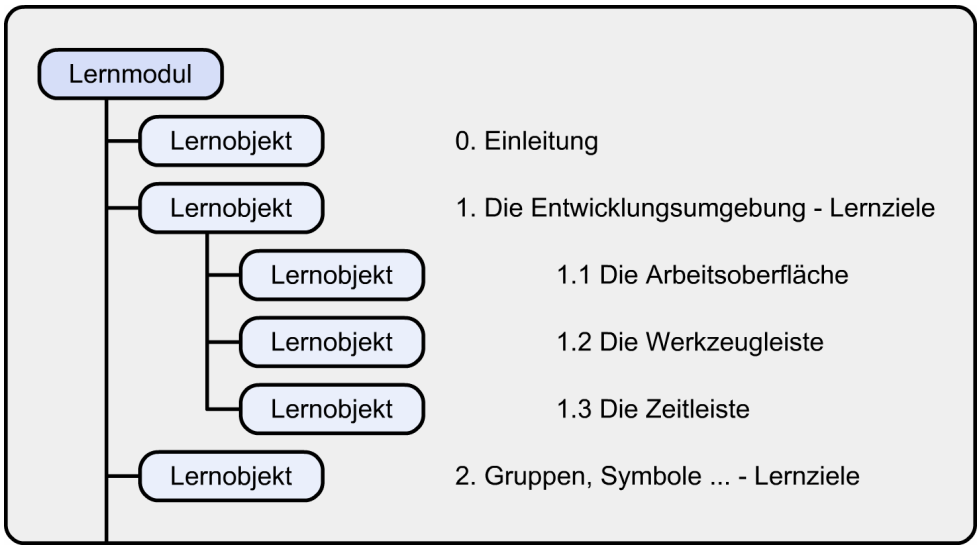

Abbildung 2.2: Die grundlegende Inhaltsstruktur des Lernmoduls

2.3.4 Lernwegstruktur

Lernwegstrukturen legen fest, wie sich der Nutzer durch den Lehrstoff bewegen kann. Dies betrifft die Navigation sowohl zwischen den verschiedenen Lernobjekten als auch innerhalb eines solchen. Letztere wird im Wesentlichen durch das Oberflächendesign sowie die Definition der Lerninhalte im Basaltext bzw. Drehbuch bestimmt und soll an dieser Stelle nicht näher erläutert werden. Die Navigation zwischen den einzelnen Lerneinheiten richtet sich stattdessen hauptsächlich nach dem Einsatzkontext. Dies lässt sich anhand der gegebenen Bedingungen verdeutlichen.

Die inhaltliche Gliederung der Lernobjekte wird bei SCORM in der so genannten Content Aggregation (vgl. Abschnitt 1.2.3) abgebildet. Wie bereits erwähnt, unterstützt das zum Zeitpunkt der Erstellung vom Bildungsportal Sachsen verwendete LMS, Saba 3 Release 4, lediglich die Spezifikations-Version 1.2. Im Gegensatz zur aktuellen SCORM-Version, die ein umfangreiches Modell zur Ablaufsteuerung der Inhalte und Navigation bereitstellt, beinhaltet diese lediglich die Option, anhand speziell festgelegter Vorbedingungen (Prerequisites) die Anzeige einer Lernressource zu kontrollieren. Jedoch wird durch die ADL Initiative eine Implementierung dieser Möglichkeit seitens des Lernmanagementsystems nicht

vorgeschrieben und obliegt ausschließlich der Entscheidung der Hersteller. Ebenso bleibt es dem LMS überlassen, welche Werkzeuge es für die Navigation bereitstellt (Auswahlmenü, Vor-/Zurück-Schaltflächen etc.).

Saba bot keine Unterstützung[19] von Prerequisites [Wagner 2005, S.88], weshalb von einer Implementation abgesehen wurde. Daher ist es dem Lernenden möglich, die Kurse in beliebiger Reihenfolge aufzurufen. Die inhaltliche Struktur kann aus diesem Grund nur eine Empfehlung für einen geeigneten Lernweg darstellen. Letztlich obliegt es jedoch der Eigenverantwortung des Lernenden, inwieweit er sich daran orientiert.

2.3.5 Einsatz von Medien

Multimediale Präsentationsformen stellen ein bedeutendes didaktisches Mittel dar, um komplexe Lerninhalte erfolgreich zu vermitteln. Im Hinblick auf die Informationsdarbietung kann zwischen einer *sprachlich-symbolischen* (Text in visueller oder akustischer Form), *bildlich-statischen* (Fotografie, Grafik) sowie *bildlich-dynamischen* (Video, Animation) Art unterschieden werden. Im Bereich des E-Learning ist insbesondere der kombinierte oder der wechselhafte Einsatz verschiedener Medien vorteilhaft, da sich so die Merkleistung des Rezipienten steigern lässt [Kerres 2001, S.179ff][Wendt 2003, S.187].

Die im Folgenden beschriebenen Medienarten wurden für die Entwicklung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“ ausgewählt.

Text

Text wird bei der Entwicklung multimedialer Lernangebote eine hohe Bedeutung beigemessen. Die Präsentation kann dabei auf verschiedene Art und Weise erfolgen [Kerres 2001, S.180]:

- dargestellt auf dem Bildschirm (visuell)
- als Vortrag in Form eines Audio-Tracks (auditiv)
- als Videodarstellung mit einem Sprecher (audiovisuell)

Die gebräuchlichste Form im E-Learning-Bereich stellt nach wie vor die erste Variante dar, die auch im Lernmodul vorrangig als Informationsträger eingesetzt wird. Für die Darstellung am Bildschirm gilt es jedoch einige Besonderheiten zu beachten. Im Gegensatz zu Printmedien lässt sich in digitaler Form präsentierter Text mühsamer lesen. Die Ursache dafür bilden beispielsweise Faktoren wie eine zu geringe Bildfrequenz (Flimmern)[20], die Bildschirmauflösung[21], ein schlechterer Kontrast zwischen Text und Hintergrund[22] sowie das für

[19] Das aktuell vom BPS verwendete OLAT 4 verarbeitet Prerequisites.

[20] Dieses Problem tritt nur bei CRT-Monitoren auf, TFTs betrifft es technologiebedingt nicht.

[21] Monitore erreichen zurzeit Auflösungen von 80 bis 100dpi (dots per inch), Printmedien hingegen je nach

das menschliche Auge ungewohnte Bildschirm-Querformat[23] [Naumann u.a 2005, S.19] [Wendt 2003, S.188f.].

Bei der Gestaltung von Textinhalten ist deshalb darauf zu achten, möglichst kurze Textblöcke mit geringer Spaltenbreite anzulegen. Oftmals ist es deswegen erforderlich, die zu vermittelnden Lerninhalte einer *Gewichtung* sowie *Reduktion* zu unterziehen und falls nötig über mehrere Informationsebenen (Hauptinformation, zusätzliche Information etc.) zu verteilen.

Umstritten ist die Verwendung von Laufleisten, mit denen das Rollen („Scrollen“) von Texten ermöglicht wird. Einerseits lässt sich auf diese Weise mehr Information auf einer Bildschirmseite unterbringen. Hingegen ist das Lesen größerer Textmengen anhand einer solchen Rollfunktion eher unangenehm und wirkt sich störend auf den Lernprozess aus.

Die folgende Abbildung demonstriert die Textgestaltung innerhalb des Lernmoduls am Beispiel des SCOs *Die Symbole*:

Druckverfahren von 300 bis zu mehreren tausend dpi.

[22] bezieht sich auf den Kontrast bei Printmedien

[23] TFT-Monitore verfügen teilweise über eine so genannte *Pivot-Funktion*, durch die sich der Bildschirm um 90° drehen lässt. Sie kann auch durch entsprechende Standfüße oder Wandhalterungen nachgerüstet werden.

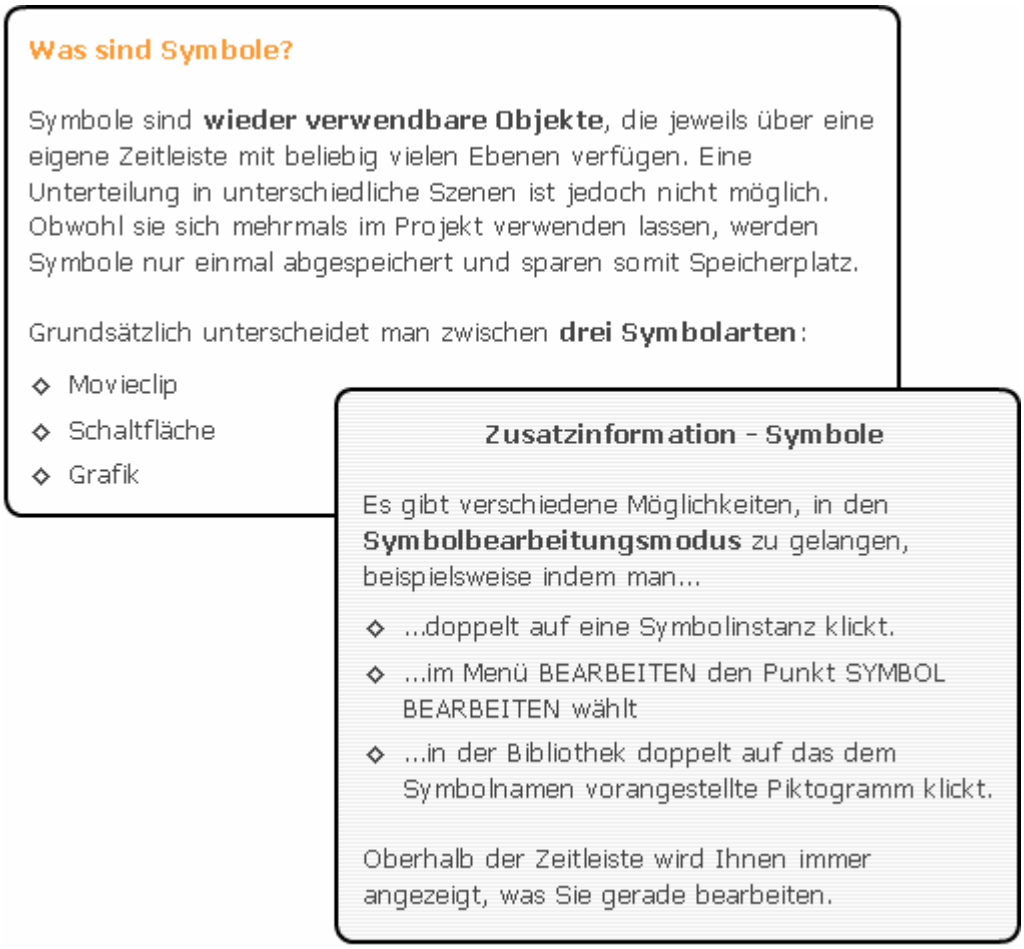

Abbildung 2.3: Textgestaltung im Lernmodul

Grafik

Eine Reihe von Studien belegen, dass Informationen aus Grafiken schneller und leichter entnommen werden können als aus Texten [Niegemann u.a. 2004, S.176]. Bilder werden daher in der Regel weiterführend, verstärkend oder auch alternativ zu Textinhalten eingesetzt. Es sollte dabei jedoch auch bedacht werden, dass Grafiken teilweise oberflächlicher wahrgenommen sowie verarbeitet werden als Texte und unter Umständen mehrdeutige Interpretationen zulassen. Daher gilt: Je aussagekräftiger und eindeutiger eine bildhaft präsentierte Information ist, desto weniger Begleittext ist erforderlich.

Bei bildlich-statischen Medien kann auch nach der Art der Informationsdarstellung unterschieden werden. *Schematisierte Grafiken* eignen sich dafür, komplexe Zusammenhänge auf einfache und anschauliche Weise abzubilden. *Fotorealistische Darstellungen* bieten dem Betrachter hingegen einen sehr konkreten und realitätsnahen Blick auf Personen, Objekte oder Situationen [Wendt 2003, S.190f].

Sinnvoll in den restlichen Lernkontext eingebundene, ansprechend gestaltete Bilder können die Aufmerksamkeit lenken und die Motivation des Nutzers anheben. Es sollte jedoch keine Grafik um ihrer selbst Willen verwendet werden. So kann das Nichtzeigen eines Bildes

unter Umständen besser sein, statt auf überflüssigen oder unpassenden Darstellungen zu beharren.

Im Hinblick auf die Verwendung bildlich-statischer Medien im Lernmodul „Die interaktive Arbeit mit Flash MX 2004“ wurde der Focus auf folgende Grafikformen gerichtet: *Flash-Vektorgrafiken* sowie *Bildschirmfotos* (Screenshots).

Flash-Vektorgrafik

Es liegt auf der Hand, bei der Entwicklung einer Lernanwendung zu und vor allem mit Flash dessen grafische Fähigkeiten auch illustrativ zu untermauern. So lassen sich beispielsweise die Funktionsweisen und Besonderheiten einzelner Zeichenwerkzeuge in einer Kombination von textlicher und bildlicher Information besser demonstrieren als in rein sprachlich-symbolischer Form. Ebenso kann mit Hilfe von grafischen Beispielen Wissen vermittelt werden, dessen textbasierte Umschreibung vergleichsweise umständlich wäre.

Durch die Verwendung von Flash-Grafikbeispielen sollten folglich das Verstehen einzelner Wissenseinheiten durch den Lernenden sowie die Merkleistung begünstigt werden. Nebenbei bot das Vektor-Format auch den Vorteil einer geringen Dateigröße.

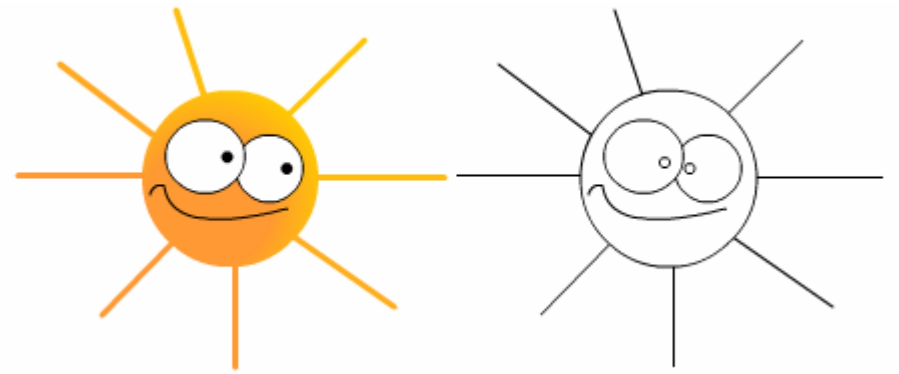

Abbildung 2.4: Illustration im Vektorformat

Bildschirmfotos

Bildschirmfotos stellen den Bezug zur Arbeitsoberfläche des Flash-Autorensystems her. Es handelt sich um Momentaufnahmen von bestimmten Zuständen der Entwicklungsumgebung, abgespeichert als Rastergrafiken. In das Design des Lernmoduls hielten Bildschirmfotos in folgender Form Einzug (vgl. auch Abbildung 2.5):

- *Piktogramme*: Viele Elemente der Arbeitsoberfläche von Flash werden in symbolhafter Form dargestellt, insbesondere Schaltflächen (z.B. Werkzeuge). Es schien daher angebracht, bei der Erwähnung oder Beschreibung selbiger innerhalb eines Textes zugleich das entsprechende Piktogramm darzustellen. Dem Lernenden soll es dadurch erleichtert werden, Symbolik und Semantik untereinander zuzuordnen.

- *Bildbeispiele*: Bildbeispiele wurden hauptsächlich dazu eingesetzt, Bearbeitungsstadien von Objekten auf der Bühne inklusive werkzeugspezifischer Anzeigen (Anfasser, Rahmen etc.) abzubilden. Eine rein vektorbasierte Umsetzung wäre mit einem erheblichen Mehraufwand verbunden gewesen und schien daher ungeeignet. Wo es angemessen erschien, wurden darüber hinaus sprachlich-symbolische Informationen um Abbildungen von Ausschnitten der Oberfläche ergänzt.
- *Schaltflächen*: Eine zentrale Rolle wurde Bildschirmfotos bei der Erstellung spezieller Schaltflächen zugewiesen. Ziel war es, dem Lernenden verschiedene Elemente der Arbeitsoberfläche als interaktive Objekte zu präsentieren, über die er erläuternde und weiterführende Informationen abrufen kann. In Abschnitt 2.4.3 wird näher auf die entwickelten Schaltflächen eingegangen.

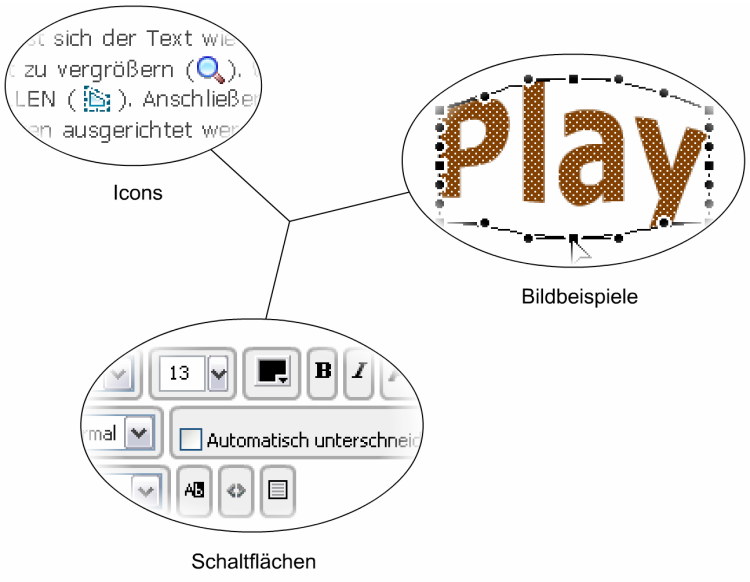

Abbildung 2.5: Verwendungsformen von Bildschirmfotos

Animationen

Animationen eignen sich insbesondere dazu, abstrakte und komplexe Sachverhalte anschaulich darzustellen, die sich mit Worten nur äußerst umständlich erklären ließen. Darüber hinaus können sie auch – ähnlich wie Grafiken – zum Zweck der Aufmerksamkeitslenkung und Motivation des Lernenden eingesetzt werden und fördern die Verstehens- und Merkleistung.

Für den Einsatz von Animationen im didaktischen Sektor gibt es einige grundlegende Aspekte zu beachten. Eine übermäßige Verwendung führt beispielsweise schnell zu einer Ablenkung des Lernenden und hindert somit den Prozess der Wissensaufnahme. Des Weiteren sollten Animationen nicht unnötig kompliziert gestaltet werden, es gilt das Prinzip „Weniger ist oft mehr“. Unter Umständen könnten Lernenden sonst Schwierigkeiten bekommen, relevante von unwesentlichen Informationen zu trennen. Dem kann beispiels-

weise auch in Form von Hervorhebungen bestimmter Objekte, in farblicher oder anderweitiger Art, begegnet werden.

Ein weiterer Punkt, der im Zusammenhang mit der Präsentation von Animationen betrachtet werden sollte, ist das Anbieten von Steuermöglichkeiten. Über diese hat der Lernende die Möglichkeit, den Abspielvorgang aktiv eigenen Bedürfnissen anzupassen und übernimmt somit nicht nur die Rolle des reinen Konsumenten. [Niegemann u.a 2004, S.140ff].

Für die Entwicklung des Lernmoduls wurden Flash-Animationen in zweierlei Hinsicht vorgesehen. Zum einen sollten sie das auf textlicher Ebene vermittelte Wissen um eine veranschaulichende Komponente bereichern. Dies galt insbesondere der Visualisierung verschiedener Animationskonzepte (Zeitleisteneffekte, Pfadanimation etc.) von Flash. Zum anderen sollten Animationen aber auch der Vermittlung prozeduralen sowie kontextuellen Wissens dienen, indem sie die Arbeit mit Flash MX 2004 anhand realistisch nachgebildeter und sprachlich unterstützter *Handlungsabläufe* demonstrierten. So ließen sich beispielsweise komplexe Zeichnungsvorgänge oder das Benutzen und Bearbeiten von Klangdaten auf eine leicht nachvollziehbare Weise veranschaulichen.

Dem Lernenden wurde – in Abhängigkeit vom Animations-Typ – eine mehr oder weniger umfangreiche Abspielsteuerung zur Seite gestellt. Weiterführende Informationen zum Thema „Animationen“ sind den Abschnitten 2.4.3 (Design) sowie 3.1.4 (Technische Entwicklung) zu entnehmen.

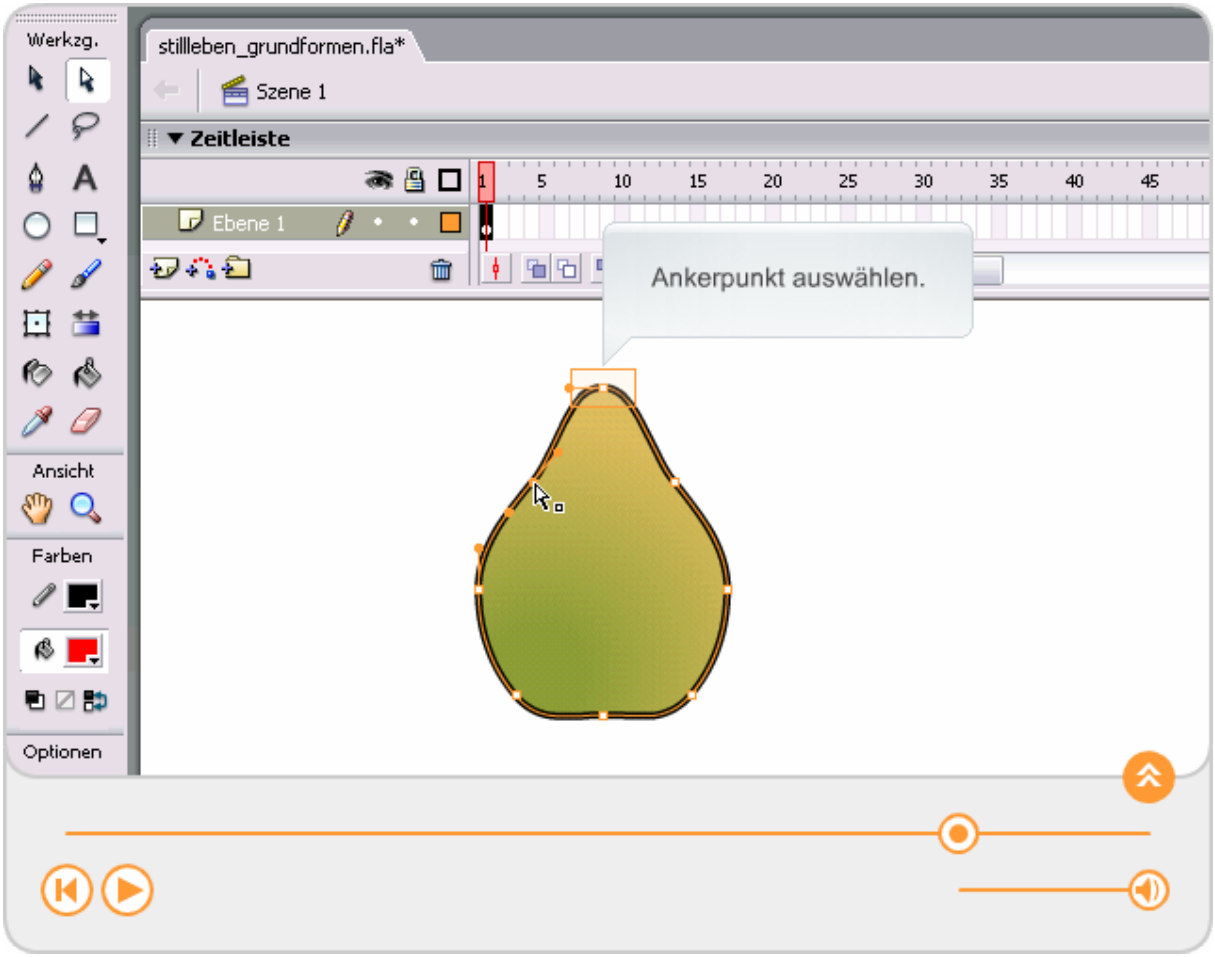

Abbildung 2.6: Erklärende Animation aus dem SCO *Workshop: Ein Stillleben zeichnen*

Interaktionsformen

Bei der Gestaltung softwareunterstützter Lernanwendungen ist unter *Interaktivität* die Eigenschaft des Programms zu verstehen, dem Nutzer diverse Steuerungs- und Eingriffsmöglichkeiten anzubieten. Auf diese Weise soll ein individualisiertes Lernen ermöglicht werden, indem sich die dargebotenen Informationen an die Vorkenntnisse, Interessen und Bedürfnisse des Lernenden anpassen lassen [Wikipedia 2006].

In der Fachliteratur wird häufig zwischen zwei unterschiedlichen Arten der Interaktivität gesprochen [Baumgartner&Häfele 2002, S.17]:

- *Steuernde Interaktivität*: Bezieht sich auf die Steuerung des Programmablaufs.
- *Didaktische Interaktivität*: Dient der Gestaltung des Lernerlebnisses und zur Erreichung von Lernzielen.

Eine steuernde Interaktivität umfasst somit die über das Bildungsportal Sachsen bereitgestellten Navigationsmöglichkeiten (vgl. Abschnitt 2.3.4) als auch die innerhalb eines Lernobjekts selbst enthaltenen Steuerfunktionen. Über Letztere kann beispielsweise zwischen einzelnen Seiten bzw. Lernschritten hin und her gewechselt oder der Zugriff auf verschiedene

Informationsebenen (themen- und objektspezifische Information, Tipps etc.) ermöglicht werden.

Didaktische Interaktionen bezeichnen hingegen Aktivitäten, mit denen sich Wissen vermitteln und abfragen lässt. Sie können entweder in programmgesteuerter Form oder als Kommunikation mit anderen Personen (Lernende und/oder Lehrende) auftreten. Als Beispiele ließen sich Lernerfolgskontrollen anhand von Aufgaben oder auch Chats nennen.

Der Schwerpunkt bei der Wahl der Interaktionsformen wurde für das Lernmodul „Die interaktive Arbeit mit Flash MX 2004“ auch den steuernden Bereich gelegt. So sollten einzelne Lernschritte abwechselnd angezeigt, verschiedene Informationsebenen abgerufen sowie Animationen dargestellt und gesteuert werden können.

Hinsichtlich der didaktischen Interaktivität wurde sich darauf beschränkt, Bearbeitungsmöglichkeiten für bestimmte präsentierte Inhalte anzubieten. Darüber hinaus besteht auch die Möglichkeit, mit anderen Personen zu kommunizieren und zu kollaborieren. Dazu bot bzw. bietet das Bildungsportal Sachsen beispielsweise ein Forum an, in dem sich Lernende und Lehrende untereinander austauschen können[24]. Andererseits kann davon ausgegangen werden, dass in der Lehrveranstaltung „Entwicklungswerkzeuge für Multimediasysteme“ neben dem Dozenten auch mehrere Studenten anwesend sind.

Auf eine Implementierung expliziter Lernerfolgskontrollen im Sinne kleiner Aufgaben am Ende eines Lernobjekts wurde dagegen aus zeitlichen und finanziellen Gründen verzichtet. Es bliebe auch die Frage nach den Maßnahmen offen, die nach einer Auswertung der Nutzereingaben zu erfolgen hätte. Wie bereits erwähnt, bot das vom Bildungsportal Sachsen zum damaligen Zeitpunkt verwendete LMS keine Unterstützung von Prerequisites und damit keine Möglichkeit der Einflussnahme auf die Präsentation von Lernobjekten.

2.4 Designphase – Gestaltung der Benutzungsoberfläche

Der Entwurf einer funktional und visuell ansprechenden Benutzungsoberfläche spielt eine nicht unbedeutende Rolle beim Softwaredesign. Es ist darauf zu achten, möglichst zweckgemäß und lerndienlich zu gestalten. Ein unausgewogenes, aufdringliches Bildschirmlayout behindert den Lernprozess nur unnötigerweise [Schreiber 1998, S.82f].

2.4.1 Gestalterische Vorgaben

Bezüglich der Gestaltung der Benutzungsoberfläche gab es seitens der Auftraggeberin ebenfalls verschiedene Richtlinien. Dazu stellte sie zwei kleinere, teils unfertige Lernmodule

[24] Unter OLAT stehen erweiterte Möglichkeiten zum Kommunizieren und Kollaborieren im Vergleich zu SABA zur Verfügung.

(vgl. Abbildung 2.7) zur Verfügung, die Studenten des Studiengangs Medieninformatik im Rahmen der von ihr gehaltenen Lehrveranstaltung „eLearning – Konzeption, Entwicklung, Anwendung“ realisiert hatten. Die Grundlage des von ihnen erarbeiteten Designs bildete das bereits ins Bildungsportal Sachsen implementierte Lernmodul „Interaktive Einführung in Arbeitsweisen der Kartographie“ des Fachbereichs Vermessungswesen/Kartographie der HTW Dresden [Wagner 2005, S.82].

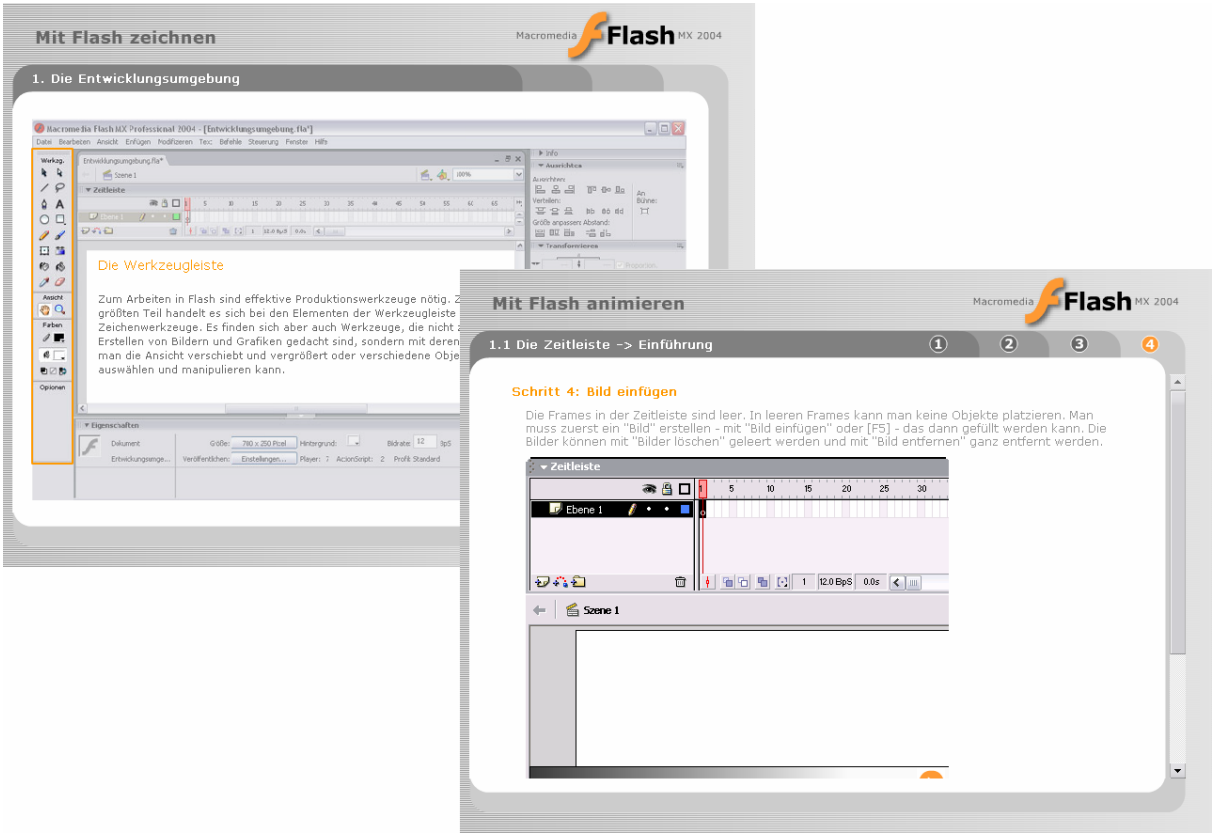

Abbildung 2.7: Ausschnitte aus den zwei Lernmodulen

Es galt die Anforderung, den Rahmen, wie er bei beiden Anwendungen verwendet wurde, so genau wie möglich in das Design des neu zu erstellenden Lernmoduls zu übertragen. Neben der farblichen Gestaltung betraf dies auch funktionale Vorgaben. So sollten umfangreiche Lerninhalte in mehrere Lernschritte segmentiert werden, zwischen denen sich mit Hilfe von Zahlen-Schaltflächen am oberen Rand navigieren lässt. Außerdem diente der linke obere Bereich zur Anzeige der aktuellen Position innerhalb des Lernmoduls. Farblich orientiert sich das Layout am Corporate Design der HTW Dresden (Logo, Internetauftritt etc.). Aufgrund seiner Signalwirkung wurde dem orange Farbton, wie er in Abbildung 2.7 deutlich zu erkennen ist, im Vorfeld eine besondere Bedeutung bei der Entwicklung interaktiver Elemente zugewiesen.

Darüber hinaus wurden durch die Auftraggeberin auch Richtlinien hinsichtlich der zu

verwendenden Schriftart (Verdana), Schriftgröße (12px) bzw. des einzuhaltenden Zeilenabstands (120%) festgelegt. Ebenfalls sollte die in den beiden Lernmodulen verwendete Symbolik bezüglich ihrer semantischen Eignung überprüft und gegebenenfalls übernommen werden.

Wie aus den gestellten Anforderungen ersichtlich wird, lag der gestalterische Schwerpunkt des Oberflächendesigns somit in der Entwicklung eines ergonomischen und didaktisch geeigneten Seitenlayouts.

In den folgenden Abschnitten wird auf die Benutzungsoberfläche der *finalen* Version des Lernmoduls eingegangen. Diese beinhaltet sämtliche Änderungen, die aufgrund gewonnener Erkenntnisse während der Produktion bzw. aus der formativen Evaluation erforderlich waren. Einzelheiten zu den verschiedenen Tests sind dem Abschnitt xxx zu entnehmen.

2.4.2 Funktionelles Design

Im Rahmen eines *funktionellen Designs* wird entschieden, über welche Funktionalität eine spätere Lernanwendung verfügen soll und auf welche Weise diese zu integrieren ist. Unerlässlich sind dabei natürlich tiefer gehende Kenntnisse über die in der verwendeten Entwicklungsumgebung zur Verfügung stehenden Möglichkeiten.

Es lässt sich prinzipiell eine Unterscheidung zwischen *globalen* sowie *lokalen Programmfunktionen* treffen.

Globale Programmfunktionen

Globale Funktionen stehen dem Lernenden programmweit zur Verfügung. Darunter lassen sich beispielsweise Navigationsstrukturen, Zusatz- und Hilfsfunktionen einordnen.

Bedingt durch den Einsatz des Lernmoduls im Bildungsportal Sachsen bedurften einige Fragen bezüglich des funktionalen Designs keiner weiteren Untersuchung. So wurden seitens des LMS bereits folgende Funktionen angeboten:

- die Visualisierung der Inhaltsstruktur als Auswahlmenü mit der Möglichkeit, einzelne Lernobjekte in beliebiger Reihenfolge zu starten
- die Anzeige statistischer Informationen hinsichtlich der Bearbeitung einzelner Inhalte
- ein Kommunikationsforum zum Wissensaustausch mit anderen Personen

Unter SABA wurden zu den bereits benutzten Lerneinheiten verschiedene Informationen dargestellt. Einige davon, wie beispielsweise Abschlussstatus und Bearbeitungsdauer, sind jedoch rein optionaler Natur und müssen durch ein Lernobjekt selbst an das LMS übermittelt werden.

Die eigentliche Frage, mit der es sich auseinanderzusetzen galt, war allerdings, welche

globalen Funktionen *innerhalb* der programmeigenen Benutzungsoberfläche angeboten werden sollten. Wie bereits erwähnt, wurde die *Navigation zwischen einzelnen Lernschritten* bereits in prinzipieller Form vorgegeben. So sollte diese anhand von bezifferten Schaltflächen erfolgen, dargestellt oberhalb des Seiteninhalts. Des Weiteren wurde festgelegt, dass die Darstellung der Navigation nur im Fall multipler Lernschritte erfolgt. Der Zugriff auf die einzelnen Seiten sollte von jeder Stelle innerhalb des Lernmoduls aus möglich sein und darüber hinaus keiner festen Lernwegstruktur unterliegen. Der letzte Punkt bedeutet nichts anderes, als dass der Nutzer die Lernschritte in beliebiger Reihenfolge aufrufen kann, also beispielsweise auch den viertem *vor* dem zweiten. Der Lernende hat somit die Möglichkeit, sich Informationen flexibel und bedürfnisgerecht anzeigen zu lassen.

Im Hinblick auf die Benutzerfreundlichkeit der Anwendung sollten weiterhin zu sämtlichen interaktiven Steuerelementen Hinweise angezeigt werden, welche Funktion bzw. Information jeweils mit ihnen verknüpft ist. Zur Realisierung dessen wurden kleine Textfenster (Tooltipps) vorgesehen, die mit leichter Verzögerung dargestellt werden, sobald der Mauszeiger reglos innerhalb des sensitiven Bereichs (der interaktiven Elemente) verharrt.

Lokale Funktionen

Als *lokal* werden im Rahmen dieser Arbeit diejenigen Funktionen betrachtet, die ausschließlich innerhalb eines Lernschritts zur Verfügen stehen. Zur Spezifikation selbiger wurden zunächst Überlegungen angestellt, wie das zu vermittelnde Wissen grundsätzlich präsentiert werden soll.

Die Verwendung von Laufleisten bietet den Vorteil, mehr Informationen auf einer Bildschirmseite unterbringen zu können. Jedoch werden die Aufnahme und kognitive Verarbeitung auf diese Weise präsentierter, umfangreicher Wissensmengen häufig als unangenehm empfunden. Laufleisten können somit den Lernprozess beeinträchtigen, weshalb auf ihre Verwendung innerhalb der Benutzungsoberfläche verzichtet wurde.

Stattdessen sollte eine Vermittlung der Kenntnisse und Fertigkeiten auf verschiedenen *Informationsebenen* erfolgen, unterteilt hinsichtlich ihrer praktischen Relevanz oder ihrem Abstraktionsgrad. So lässt sich beispielsweise zwischen wesentlichem und zusätzlichem Wissen unterscheiden, aber auch zwischen übergeordneter, abstrakter Information und detaillierten Erläuterungen zu einzelnen Bedienelementen oder Begriffen. Sämtliche unterhalb der Hauptebene liegenden Informationsschichten sollen optional mittels Schaltflächen abrufbar sein. Dem Anwender wird dadurch ein gewisses Maß an Kontrolle übertragen, und es bietet sich ihm die Möglichkeit, das Lerngeschehen aktiv zu beeinflussen und auf Wissenseinheiten nach Wunsch zuzugreifen.

Unter Abschnitt 2.3.5 wurde bereits erläutert, dass der Einsatz von Flash-Animationen unter anderem zur Demonstration von Handlungsabläufen vorgesehen war. Neben Audio-Daten mit den Sprechertexten beinhalten diese auch Bildschirmfotos, woraus eine

vergleichsweise hohe Dateigröße der veröffentlichten SWF-Filme resultiert. Aus diesem Grund wurde festgelegt, die Animationen nur im Bedarfsfall anzuzeigen, um einen Nutzer nicht schon beim Laden eines Lernschritts durch längere Wartezeiten zu demotivieren. Darüber hinaus sollte den Filmen eine Wiedergabesteuerung zur Seite gestellt werden, über die der Lernende den Abspielvorgang manipulieren kann.

Im folgenden Abschnitt wird beschrieben, wie mit Hilfe der spezifizierten Funktionen ein Oberflächendesign entwickelt wurde.

2.4.3 Visuelles Design

Zu Beginn der visuellen Gestaltung erfolgt gewöhnlich eine grundlegende Einteilung der Benutzungsoberfläche in verschiedene Funktionsbereiche. Nach Schreiber kann dabei für gewöhnlich zwischen folgenden drei Flächen unterschieden werden [Schreiber 1998, S.334f]:

- *Arbeitsbereich*: Im Arbeitsbereich werden die multimedial aufbereiteten Lerninhalte inklusive dazugehöriger lokaler Funktionen dargeboten. Er beansprucht den größten Teil der Oberfläche und lenkt gleichzeitig die meiste Aufmerksamkeit des Lerners auf sich.
- *Orientierungsbereich*: Dieser Bereich dient dem Nutzer zur Orientierung und veranschaulicht die Position innerhalb der Programmstruktur, an der die aktuellen Informationen abgerufen werden.
- *Steuerungsbereich*: Der Steuerbereich eignet sich zur Darstellung globaler Funktionen, wie beispielsweise Navigation oder Zusatzfunktionen.

Wie unter Abschnitt 2.4.1 beschrieben, waren Steuerungs- sowie Orientierungsbereich in ihrer prinzipiellen Gestaltung bereits vorgegeben. Hinsichtlich Letzterem musste jedoch der Umstand bedacht werden, dass es sich bei den einzelnen Lernobjekten um SCOs handelt. Es widerspräche dem Bestreben nach deren Wiederverwendbarkeit, als Orientierungshilfe die Position innerhalb des Lernmoduls mit Hilfe einer eindeutigen Kapitelnummer zu verdeutlichen. Sollte das Lernobjekt in einem anderen Kontext verwendet werden, so kann sich diese prinzipbedingt ändern. Aufgrund dessen wurden lediglich das Hauptthema (diese Angabe ist im Sinne der Mehrfachnutzung auch ein wenig fraglich, wurde jedoch in Absprache mit der Auftraggeberin zur besseren Orientierung eingefügt) sowie der Name des Lernobjekts (Unterpunkt im Auswahlmenü) angegeben.

Der gestalterische Schwerpunkt lag im Layoutdesign des Arbeitsbereiches. Zunächst wurde er grob in folgende drei Bereiche untergliedert:

- einen *Hauptbereich*, in dem grundlegende Informationen des jeweiligen Lernschritts visualisiert werden sowie Interaktionsmöglichkeiten, mit denen sich detaillierte Beschreibungen zu Flash-Bedienelementen, -Begriffen oder Arbeitsabläufen abrufen

lassen

- eine Fläche zur Darstellung vertiefender sowie zusätzlicher Informationen – im Folgenden *Zusatzbereich* genannt – die über die interaktiven Elemente der beiden anderen Bereiche abgerufen werden
- *Schaltflächen* zur Anzeige von Zusatzinformationen, nützlichen Tipps, Tastaturkürzeln sowie animierten Handlungsabläufen

Die folgende Abbildung veranschaulicht die grobe Unterteilung der Benutzungsoberfläche in Arbeits-, Orientierungs- und Steuerbereich:

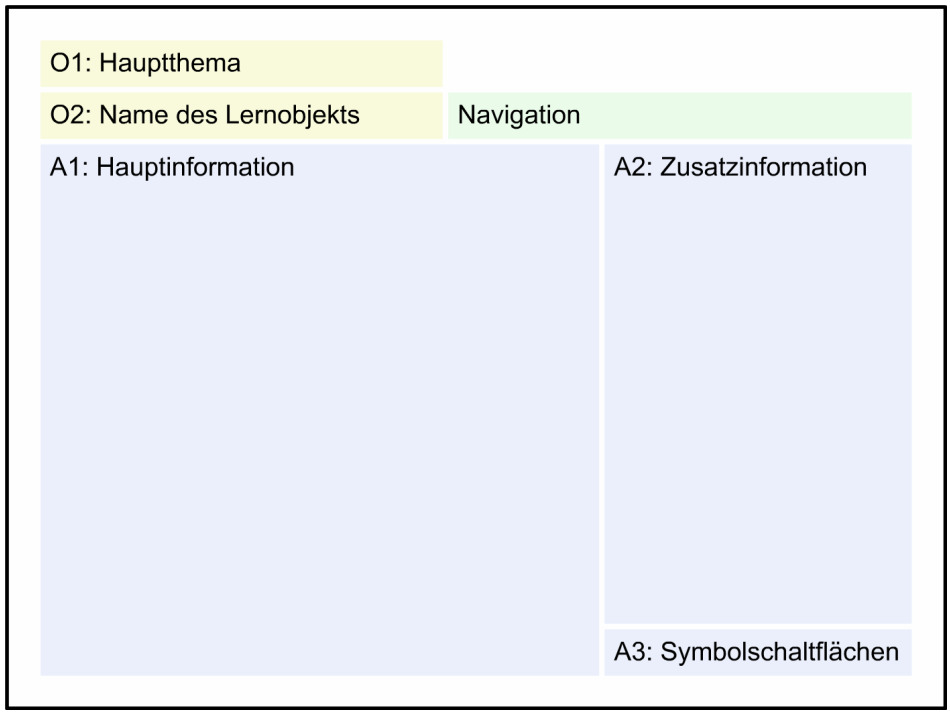

Abbildung 2.8: Unterteilung der Benutzungsoberfläche in Arbeits-, Orientierungs- und Steuerbereich

Der Zusatzbereich (und somit umgekehrt proportional auch der Hauptbereich) ist von variabler Breite. Das heißt, er kann an den jeweiligen Umfang des präsentierten Lerninhalts angepasst werden. Wird der Zusatzbereich innerhalb eines Lernschritts nicht benötigt, so lässt er sich auch komplett ausblenden. Dies ist genau dann der Fall, wenn neben den Hauptinformationen weder vertiefendes noch zusätzliches Wissen angeboten wird, d.h. keine lokalen Interaktionsfunktionen vorhanden sind.

Im Zuge der Layoutgestaltung wurde auch die ursprünglich vorgesehene Oberflächendimensionierung um jeweils 100 Bildpunkte erweitert. Daraus resultierten schließlich eine Breite von 900 Bildpunkten sowie eine Höhe von 700 Bildpunkten. Ziel dieser Maßnahme war es, mehr Raum für die zu vermittelnden Lerninhalte zu schaffen. Mit Blick auf die Tatsache, dass die meisten Computernutzer heutzutage mit einer Bildschirmauflösung von

1024 mal 768 Bildpunkten oder mehr arbeiten und die Lernobjekte vom LMS des Bildungsportals Sachsen in einem separaten Popup-Fenster geöffnet werden (und sich somit komplett auf dem Bildschirm darstellen lassen), bringt diese Erhöhung der Oberflächenausmaße keine relevanten Nachteile mit sich.

Während der Gestaltung durchlief das Aussehen der Benutzungsoberfläche mehrere Entwicklungsstadien. Abbildung 2.9 zeigt einen frühen prototypischen Entwurf, die Bildbeispiele 2.10 bis 2.12 hingegen Szenarien der finalen Version des Lernmoduls.

Abbildung 2.9: Frühe prototypische Umsetzung des Oberflächendesigns (800x600)

Abbildung 2.10: Selbes Szenario wie in Abbildung 2.9, jedoch im finalen Design (900x700)

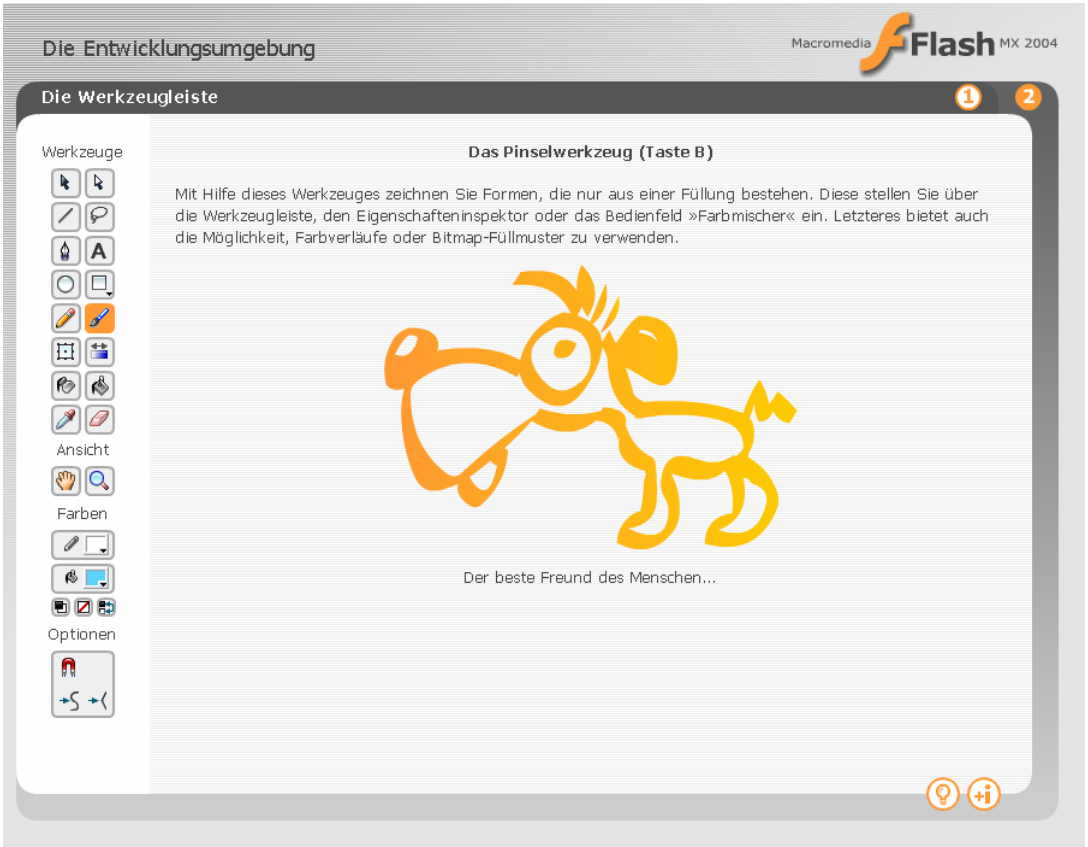

Abbildung 2.11: Beispiel für die variable Breite des Zusatzbereichs

Abbildung 2.12: Beispiel eines Lernschritts ohne Zusatzbereich

Beim Entwurf der Benutzungsoberfläche wurde versucht, die gestalterischen Vorgaben der Auftraggeberin konsequent weiterzuführen und in das restliche Design einfließen zu lassen. Dabei wurde sich an grundlegenden Gestaltungsprinzipien orientiert:

- *Modernes und ästhetisches, dabei ruhiges und ausgewogenes Design*: Die verwendeten Grautöne und das Orange stellen eine geschmackvolle und kontrastreiche Farbwahl dar und dominieren das gesamte Oberflächendesign. Die feinen Linien des Hintergrunds finden sich in dezenter Form im Zusatzbereich wieder und vermitteln einen gewissen edlen Eindruck. Zusammen mit der durchgängigen Verwendung abgerundeter Formen ergibt sich somit ein homogenes und ausgewogenes, gleichzeitig unaufdringliches und lerndienliches Erscheinungsbild.
- *Klare und übersichtliche Gliederung des Bildschirms*: Die verschiedenen Bereiche der Oberfläche sind klar voneinander getrennt und übersichtlich angeordnet. Interaktive Elemente lassen sich aufgrund ihrer Gestaltung leicht erkennen. Thematisch verwandte Funktionen – wie in diesem Fall die Schaltflächen für Tipps, Tastenkürzel etc. – wurden gestalterisch aufeinander abgestimmt und als Gruppe zusammengefasst.
- *Intuitive Bedienbarkeit*: Die verwendete Symbolik erleichtert dem Lernenden den Umgang mit dem Lernmodul, da die semantischen Zusammenhänge schnell und intuitiv erfasst werden können.

- *Konsistente Darstellungsform*: Durch das untereinander gleich bleibende Interaktionsverhalten verschiedener Schaltflächenelemente entsteht beim Lernenden schneller eine Vertrautheit im Umgang mit der Anwendung, als dies bei einer heterogenen Gestaltung der Fall wäre.

Im Folgenden sollen die einzelnen Elemente der Benutzungsoberfläche näher erläutert werden.

Navigationselemente

Die Vorgabe seitens der Auftraggeberin wurde zum einen dahingehend modifiziert, dass die Anzahl der dargestellten Registerkarten kongruent mit der Menge der Lernschritte ist (vgl. Abbildung 2.10 bis 2.12). Wie während des funktionalen Designs spezifiziert, wird die Navigation ausschließlich dann angezeigt, wenn sich das Lernobjekt in mehrere Seiten untergliedert.

Eine weitere Änderung betraf die Darstellung der einzelnen Schaltflächenzustände. Im nicht aktiven Zustand wird eine orangefarbene Ziffer auf weißem Untergrund abgebildet. Dadurch hebt sich die Schaltfläche im Vergleich zur ursprünglichen Variante – weiße Zahl auf dunkelgrauer Kreisfläche – stärker vom Hintergrund ab und ist deutlicher wahrzunehmen. Wird der Mauszeiger während des inaktiven Zustands in den sensitiven Bereich bewegt, so vergrößert sich etwas der Durchmesser der Schaltfläche. Durch einen Mausklick kann anschließend zum entsprechenden Lernschritt gewechselt werden. Im aktiven Zustand reagiert die Schaltfläche nicht mehr auf Mausereignisse (eine Seite kann demnach nicht mehrmals hintereinander geladen werden) und ändert sowohl die Farbe als auch erneut die Größe. Die folgende Abbildung soll das Interaktionsverhalten veranschaulichen:

Abbildung 2.13: Die Zustände Inaktiv ①+④,
Selektiert ② und Aktiv ③

Für jede nicht aktive Schaltfläche wird außerdem nach kurzer Verzögerung ein Tooltipp angezeigt, der über das Thema des jeweiligen Lernschritts informiert.

Lokale Interaktionsformen

Die im Folgenden beschriebenen Interaktionsformen dienen dem optionalen Abruf vertiefenden Wissens und werden im Hauptfeld des Arbeitsbereichs angeordnet. Es existieren zwei Typen: *Bild-* und *Textschaltflächen*. Als Darstellungsform wurden abgerundete Vierecke gewählt, um bereits vorhandene Designelemente aufzugreifen. Wie bereits erwähnt, erfolgt die Visualisierung der anhand dieser Schaltflächen abgerufenen Informationen im Zusatz-

bereich.

Bildschaltflächen kommen dann zum Einsatz, wenn Objekte oder Bereiche der Flash-Arbeitsoberfläche erläutert werden sollen (vgl. Abbildung 2.10). Als Grundlage dient dabei je Schaltflächengruppe ein teilweise transparentes Bildschirmfoto des betreffenden Bereichs. Die Anordnung einzelner Interaktionselemente zueinander orientiert sich weniger an gestalterischen Aspekten, sondern vielmehr an der realen Flash-Umgebung. Diese Vorgehensweise hat didaktische Hintergründe: Der Nutzer findet sich durch die wiederkehrende Optik schneller zurecht und kann erlerntes Wissen leichter übertragen bzw. anwenden.

Über Textschaltflächen kann stattdessen vertiefendes Wissen zu Begrifflichkeiten, Funktionalitäten sowie bestimmten Arbeitsabläufen abgerufen werden.

Das Interaktionsverhalten beider Schaltflächentypen ist identisch und durch farbliche Changierung der jeweiligen Zustände gekennzeichnet (siehe Abbildung 2.14). Der aktive Status wird konsequenterweise – wie durchgängig für alle Schaltflächentypen des Lernmoduls – anhand eines orange kolorierten Hintergrunds signalisiert. Durch diese konsistente Darstellungsweise und die hohe Aufmerksamkeitslenkung der Farbe ist für den Lernenden jederzeit leicht erkennbar, welche Informationen er gerade abruft.

Wie bei allen Schaltflächen wird auch in diesem Fall ein kleiner Tooltipp angezeigt, der Auskunft über die verknüpfte Information gibt.

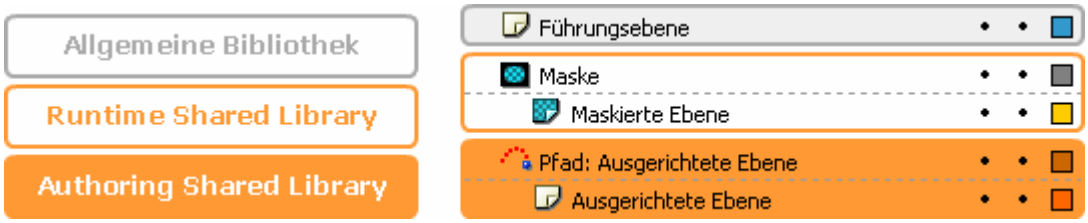

Abbildung 2.14: Text- (links) und Bildschaltflächen in den Zuständen (von oben nach unten) Inaktiv, Selektiert und Aktiv

Ebenfalls in die Gruppe der lokalen Interaktionen lassen sich die Schaltflächen für Zusatzinformationen, Tipps, Tastenkürzel sowie animierte Handlungsabläufe einordnen. Sie sind aufgrund ihrer thematischen Verwandtheit nebeneinander am rechten unteren Rand des Arbeitsbereichs platziert (siehe Abbildung 2.15). Ihre Darstellung ist abhängig vom Thema und erfolgt nach dem WYSIWYG-Prinzip (What You See Is What You Get): Nicht benötigte Schaltflächen werden auch nicht angezeigt, um keine Irritationen beim Lernenden hervorzurufen (vgl. Abbildung 2.10 und 2.11).

Abbildung 2.15: Die Anordnung der Symbolschaltflächen

Die folgende Tabelle gibt Aufschluss über Semantik sowie Funktionalität der einzelnen Symbolschaltflächen:

Symbol	Semantik	Funktion
	Animation	Laden einer Flash-Animation, die sprachlich kommentierte Handlungsabläufe visualisiert.
	Tastaturkürzel	Anzeige von Tastenkürzeln
	Nützliche Tipps	Anzeige von hilfreichen Tipps zum betreffenden Thema, die über Besonderheiten und Vereinfachungen informieren
	Zusätzliche Informationen	Darstellung von weiterführenden, interessanten Informationen

Tabelle 2.2: Erläuterung der Symbolschaltflächen

Das Interaktionsverhalten der Symbolschaltflächen ist identisch mit dem der Navigationsschaltflächen und damit Teil der konsistenten Darstellungsweise. Die mittels Mausklick abgerufenen Informationen werden im Zusatzbereich angezeigt. Animationen unterliegen jedoch einer gesonderten Präsentationsform, worauf im folgenden Text eingegangen werden soll.

Animationen

Für die Wiedergabe der „erklärenden“ Animationen wird der Zusatzbereich auf die Gesamtbreite der Arbeitsfläche ausgedehnt. Außerdem wird eine Filmsteuerung angezeigt, anhand derer der Lernende den Abspielvorgang beeinflussen kann. Gleichzeitig blendet das Lernmodul die normalen Symbolschaltflächen für die Wiedergabe aus. Stattdessen erscheint an gleicher Stelle eine neue Schaltfläche, über die der Bildschirm verlassen und der ursprüngliche Zustand wiederhergestellt werden können.

Die Gestaltung der Filmsteuerung sowie die zustandsbedingte Darstellung der einzelnen Interaktionselemente reihen sich auf konsequente Weise in das restliche Design der Benutzungsoberfläche ein. Zur Manipulation des Abspielvorgangs stehen dem Anwender folgende Funktionen zur Verfügung:

- *Starten/Pausieren der Wiedergabe*: Das Abspielen und Anhalten einer Animation erfolgt über eine Schaltfläche. Am Ende eines Films stoppt die Wiedergabe automatisch.

- *Zurückspulen zum Anfang*: Der Positionszeiger wird auf den Anfang der Animation gesetzt. Ein gegebenenfalls laufender Abspielprozess wird vom Beginn fortgesetzt.
- *Positionsschieberegler*: Mit dem Regler lässt sich eine Animation bei gedrückter Maustaste nach einer bestimmten Stelle durchsuchen. Ein laufender Abspielvorgang wird an der Position weitergeführt, an der die Taste wieder losgelassen wird.
- *Lautstärkeschieberegler*: Mit diesem Regler lässt sich bei gedrückter Maustaste die Lautstärke des wiedergegebenen Audiomaterials – in diesem Fall der Sprache – beeinflussen.
- *Abspielsteuerung ein-/ausblenden*: Über diese Schaltfläche lässt sich die Filmsteuerung anzeigen oder ausblenden.

Die für die verschiedenen interaktiven Elemente verwendete Symbolik entspricht geläufigen Standards und sollte den Anwender daher vor keine unüberbrückbaren Probleme stellen. Abbildung 2.16 veranschaulicht den Animations-Bildschirm, so wie er sich prinzipiell dem Lernenden präsentiert.

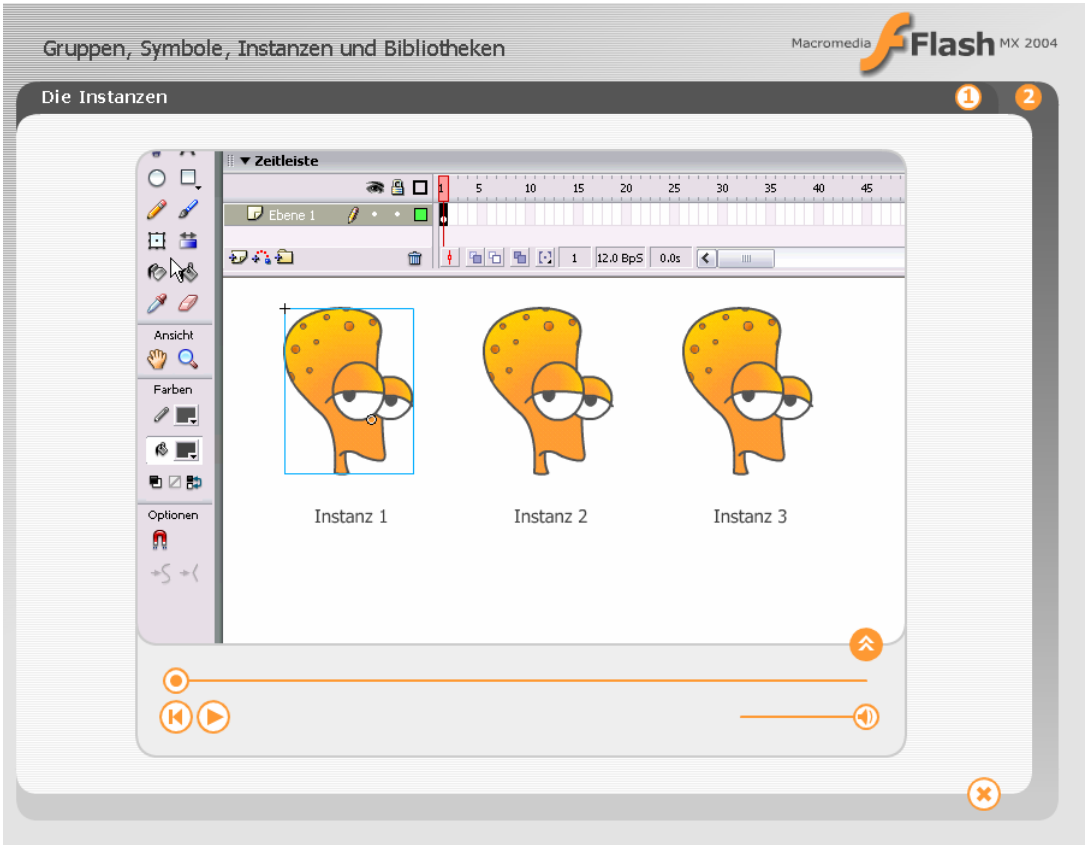

Abbildung 2.16: Der Animations-Bildschirm

Tooltipps

Wie bereits an anderen Stellen erwähnt, werden zu jeder Schaltfläche kleine Hinweisfenster angezeigt, so genannte *Tooltipps*. Dazu muss der Mauszeiger für kurze Zeit reglos im sensitiven Bereich verharren. Die Information eines Tooltipps ist sehr knapp und gibt über die jeweils verknüpfte Funktion einer Schaltfläche oder den dahinter verborgenen Lerninhalt Auskunft. Je nach Schaltflächentyp existieren bis zu zwei Tooltipps, einer für den inaktiven sowie ein weiterer für den aktiven Zustand[25].

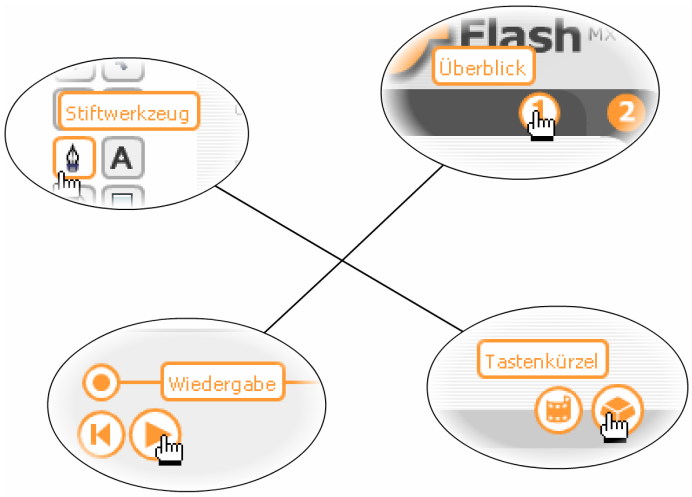

Abbildung 2.17: Tooltipps

Textgestaltung

Die Textgestaltung lässt sich grundsätzlich in zwei Bereiche untergliedern: das *visuelle* sowie das *sprachlich-inhaltliche* Design.

Die Darstellung des Textes auf dem Bildschirm wurde anhand eines so genannten *Styleguides* spezifiziert. Darin finden sich Angaben zu einzelnen Schriftarten und -formatierungen, wie sie für bestimmte Textinhalte verwendet werden sollten. So erfolgt die Darstellung der Überschrift eines Lernschritts beispielsweise mit Hilfe der Schriftart Tahoma und im orangefarbigen Fettdruck, wohingegen ActionScript-Code standardmäßig in dunkelgrauer Courier-Schriftart präsentiert wird. Nähere Informationen dazu sind dem entsprechenden Anhang zu entnehmen.

Die sprachliche sowie inhaltliche Aufbereitung des Textes richtete sich nach folgenden Zielsetzungen:

- *Personalisierung*: Durch die direkte Anrede des Nutzers wird dieser stärker in den Lernprozess einbezogen, was sich unter Umständen motivationsfördernd auswirken

[25] Im inaktiven Zustand einer Schaltfläche werden immer Tooltipps angezeigt, im aktiven hingegen nur für Bild-, Text- und Symbolschaltflächen sowie die *Wiedergabe/Pausieren*- und *Ein-/Ausblenden*-Taste der Wiedergabesteuerung.

kann.

- *Adressatenorientierter Sprachstil*: Die Hauptnutzer des Lernmoduls sind aller Voraussicht nach Studenten, von denen ein gewisses Maß an linguistischen Fähigkeiten erwartet werden kann. Dennoch sollen die Lerninhalte in einer einsteigerfreundlichen Weise präsentiert werden.
- *Reduktion*: Die Lerninhalte sollen nicht in ausschweifender Form beschrieben werden, sondern vor allem auf das Wesentliche beschränkt sein. Wie unter Abschnitt 2.3.5 erwähnt, lassen sich Texte auf dem Bildschirm schwerer erfassen, weshalb eine kurze und prägnante Darstellung notwendig ist.
- *Gewichtung*: Nicht alle Informationen sind von gleichwertiger Bedeutung für das Verständnis von Flash – es gibt Wesentliches ebenso wie rein Nützliches. Aufgrund dessen ist eine Aufteilung der Inhalte auf verschiedene Informationsebenen von Nöten.

2.4.4 Vorüberlegungen zur Umsetzung des Layouts mit Flash MX 2004

Während der gesamten Designphase bedarf es grundsätzlicher Überlegungen, ob und wie sich gestalterische, inhaltliche sowie funktionale Elemente innerhalb einer effizienten Programmstruktur abbilden lassen. Eine gute Softwarearchitektur zahlt sich auch später aus, wenn es um die Pflege, Wiederverwendung oder Erweiterung des betreffenden Produkts geht.

Grundsätzliche Überlegungen gingen in die Richtung, die Lernsoftware so flexibel und modular wie möglich zu gestalten sowie das spätere Hinzufügen neuer Lernobjekte so weit machbar von Flash abzukoppeln. Dazu wurden das Layoutdesign sowie die Möglichkeiten von Flash MX 2004 einer genaueren Betrachtung unterzogen.

Die Benutzungsoberfläche setzt sich bei näherem Hinsehen hauptsächlich aus verschiedenen grafischen Grundelementen zusammen. Dazu gehören Linien, Kreise sowie (abgerundete) Rechtecke. Geschlossene Formen besitzen darüber hinaus oftmals unifarbene oder farbverlaufsartige Füllungen. Daraus wird ersichtlich, dass sich die Benutzungsoberfläche leicht anhand von Vektorgrafiken realisieren lässt. Neben der Möglichkeit, diese in der Entwicklungsumgebung vorzufertigen, stellt Flash jedoch auch in ActionScript Zeichenfunktionen zur Verfügung. Der Vorteil einer Oberflächengenerierung auf Quellcode-Basis liegt darin, das Design zu einem späteren Zeitpunkt *schnell, leicht und flexibel* an geänderte Bedürfnisse anpassen zu können.

Ein weiterer Punkt ist die prinzipielle *Konfiguration des Lernmoduls sowie der einzelnen Lernobjekte*. Im Sinne der Flexibilität bietet es sich an, diese nicht fest in die interne Programmstruktur zu verankern, sondern in externer Form bereitzustellen. Mit dem XML-Objekt ist in Flash ein probates Mittel vorhanden, um Daten außerhalb der eigentlichen Anwendung in einer übersichtlichen Struktur (XML-Datei) zu definieren und während der Laufzeit des Programms zu laden.

Ein zentrales Thema ist auch die *Definition der Lerninhalte*. Insbesondere für die Textgestaltung bietet Flash ab der Version MX 2004 eine interessante Neuerung in Form einer Unterstützung von *Cascading Style Sheets* (*CSS*). Mit diesen lassen sich Textstile für HTML- oder XML-formatierte Texte definieren. Zusammen mit der Möglichkeit, sowohl entsprechende Texte als auch die CSS-Daten extern abzulegen und zur Laufzeit zu laden, lässt sich somit die Festlegung der schriftbasierter Inhalte komplett von Flash abkoppeln. Im Hinblick auf eine homogene Darstellung des Lernmoduls unter verschiedenen plattformspezifischen Bedingungen wurde darüber hinaus entschieden, sämtliche benötigten Schriftzeichen in die Flash-Anwendung einzubetten. Dadurch wird eine Unabhängigkeit von den jeweils zur Verfügung stehenden Systemschriftarten gewährleistet.

Weitere Elemente der Benutzungsoberfläche, wie beispielsweise Schaltflächen oder Filmsteuerung, sollten aufgrund ihrer visuellen und funktionalen Beschaffenheit ebenfalls so weit wie möglich mit ActionScript realisiert werden.

Das vorrangig verfolgte Ziel war es also, eine möglichst flexible, modulare und redundanzfreie Programmarchitektur zu entwickeln. Neben der Externalisierung von Daten, die spätere Anpassungen und Ergänzungen vereinfacht, zählen dazu auch ein bedarfsabhängiges Laden von Programmbausteinen sowie die Mehrfachnutzung von Datenbeständen.

2.5 Beginn der Produktionsphase

Nachdem während der Designphase die Fernlernziele spezifiziert und eine Gliederung erstellt wurden, konnte anschließend mit der Festlegung der eigentlichen Lerninhalte begonnen werden.

2.5.1 Basaltext

Innerhalb eines so genannten *Basaltextes* wurden zunächst die Inhalte in *vollständiger* und *redundanzfreier* Form abgebildet. Dabei galt es zu beachten, ausschließlich zielbezogene oder in anderer Weise für den Inhalt relevante Informationen in kurzer Form festzuhalten. Ausformulierte Textpassagen, Illustrationen, Beispiele etc. sind hingegen keine Bestandteile eines Basaltextes [Schreiber 1998, S.117].

Im Zuge der Inhaltsbeschreibung einzelner Lernobjekte erfolgte auch die jeweilige Aufgliederung des zu vermittelnden Wissens in Lernschritte. Der vollständige Basaltext ist dieser Arbeit als separater Anhang D beigefügt.

2.5.2 Drehbuch

Auf Grundlage des Basaltextes konnte mit der Erstellung des Drehbuchs begonnen werden. Dieses beinhaltet eine detaillierte Beschreibung sämtlicher Seiten in Form und Inhalt. Neben

der Ausformulierung der Textpassagen erfolgt auch eine konkrete Festlegung, welche weiteren Medientypen (Grafiken, Animationen, Klänge) an welcher Stelle verwendet werden. Des Weiteren gilt es, Interaktionsmöglichkeiten, Feedbackfunktionen sowie eventuelle Verknüpfungen einzelner Seiten untereinander aufzuzeigen [Schreiber 1998, S.119].

Die Erstellung des Drehbuchs für das Lernmodul „Die interaktive Arbeit mit Flash MX 2004“ erfolgte mit Hilfe so genannter Formblätter. Diese werden üblicherweise in drei Bereiche untergliedert: *Identifikation* (Angaben wie Autor, Datum, Kapitel- und Bildschirmnummern), *Präsentation/Layout* (von Texten, Grafiken, Interaktionsmöglichkeiten etc.) sowie *Anweisungen* (Angaben zu Darstellungsformen einzelner Elemente, Ablaufsteuerung etc.) [Schreiber 1998, S.121].

Die folgende Abbildung zeigt die Aufteilung der für die Erstellung des Drehbuchs verwendeten Formblätter.

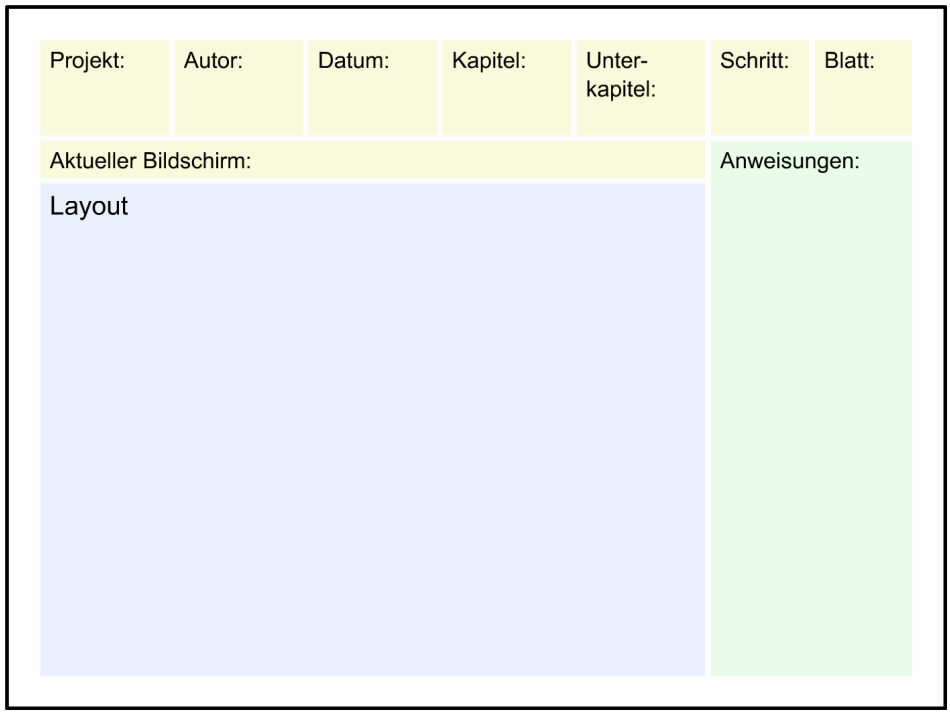

Abbildung 2.18: Aufbau eines Formblatts

Auf Grundlage des Drehbuchs wurden anschließend die einzelnen Lernobjekte entwickelt. Es ist als separater Anhang D dieser Arbeit beigefügt.

3 Technische Entwicklung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“

3.1 Erstellung der Klassen

Zu Beginn der technischen Entwicklung lag das Hauptaugenmerk auf der Erstellung der benötigten ActionScript 2-Klassen. Wie bereits unter Abschnitt 2.4.4 erwähnt, bestand ein wesentliches Ziel darin, die grafischen Elemente der Benutzungsoberfläche über ActionScript zu zeichnen und dadurch flexible Eingriffsmöglichkeiten in die visuelle Darstellung zu bieten.

Die verschiedenen Klassen wurden innerhalb von Paketordnern kategorisiert. So enthält beispielsweise das Verzeichnis *graphics/drawing* verschiedene Klassen zum Zeichnen grafischer Grundelemente, der Odner *loading* hingegen solche, die für das Laden von Dateien zuständig sind. Die Pakete selbst befinden sich im Pfad *common_sources/classes* und sind kein Bestandteil des finalen SCORM-Pakets, da sie ausschließlich für die Entwicklung relevant waren (siehe auch Abschnitt 3.2.1). Das Verzeichnis selbst wurde in der Flash-Umgebung als *globaler Klassenpfad*[26] registriert, um einen problemlosen Zugriff auf die Klassendefinitionen zu gewährleisten.

Bei der Entwicklung wurde von der Möglichkeit der Vererbung intensiv Gebrauch gemacht. Dies bezieht sich sowohl auf selbst erstellte Klassen als auch auf bereits in Flash integrierte, deren Funktionalität zum Bestandteil komplexerer Objektdefinitionen wurde.

Um dynamisch Instanzen von Klassen zu erzeugen, die von `MovieClip` erben, lässt sich in Flash nicht der `new`-Operator verwenden[27]. Eine Möglichkeit besteht darin, ein Movieclip-Symbol der Dokument-Bibliothek an die Klasse zu binden und von diesem über die Methode `MovieClip.attachMovie()` zur Laufzeit Objekte zu bilden. Eine weitere Variante existiert in Form der `Object.__proto__`-Eigenschaft[28]. Über diese können beispielsweise mittels `MovieClip.createEmptyMovieClip()` erzeugte Movieclips zur Laufzeit an entsprechende Klassen gebunden werden, wie das folgende Quellcode-Beispiel demonstriert:

[26] Klassenpfade sind Verzeichnisse, in denen Flash automatisch nach Definitionen der in einem Dokument verwendeten Klassen sucht. Neben globalen lassen sich auch lokale Klassenpfade registrieren, die ausschließlich auf Dokumentenebene Gültigkeit besitzen.

[27] Gleiches gilt beispielsweise auch für Klassen, die von `TextField` abgeleitet wurden.

[28] Alle Klassen in Flash erben von `Object`, wodurch deren Eigenschaften und Methoden praktisch immer zur Verfügung stehen.

```
import MyClasses.MyMovieclip;//von MovieClip erbende Klasse


createEmptyMovieClip("my_mc",1);
my_mc.__proto__ = new MyMovieclip();
```

Quellcode-Bsp. 3.1: Dynamisch erzeugte Instanz einer von `MovieClip` erbenden Klasse

Die soeben beschriebene Methode wurde für alle von `MovieClip` abgeleiteten Klassen des Lernmoduls verwendet.

Nahezu alle Klassen wurden anhand einer so genannten Klassenbibliothek eingebunden. Ausgenommen davon sind jene, die bereits von Beginn an zur Verfügung stehen müssen, wie beispielsweise die `Preloader`- und die `PreloaderView`-Klasse. Die Klassenbibliothek stellt eine gewöhnliche SWF-Datei dar, die zur Laufzeit des Programms nachgeladen wird. Dabei wurde von der Möglichkeit Gebrauch gemacht, den Kompiler durch die Angabe der Klassennamen *ohne* eine `import`-Anweisung[29] zu zwingen, die Klassen beim Veröffentlichen der Datei mit einzubinden, wie das folgende Quellcode-Beispiel veranschaulicht:

```
graphics.drawing.CircleDefinition;
graphics.drawing.RectangleDefinition;
graphics.drawing.PolygonDefinition;
…
```

Quellcode-Bsp. 3.2: Auszug aus dem Skript *class_library.as*

Nach dem Einlesen der Bibliothek im Initialisierungsfilm (siehe Abschnitt 3.2.2) stehen die enthaltenen Klassen auch allen nachgeladenen Filmen zur Verfügung und müssen somit nicht in jeden benutzenden Film separat eingebunden werden. Um jedoch eine Datentyp-Prüfung in *allen* benutzenden Flash-Dokumenten zu ermöglichen, die im Normalfall wiederum automatisch die Integration der betreffenden Klassen in veröffentlichte Datei zur Folge hätte, ist eine zusätzliche Maßnahme erforderlich. Durch die Verwendung so genannter *Exclude-XML-Dateien* besteht in Flash die Möglichkeit, Klassendefinitionen von der Kompilierung auszuschließen. Die XML-Struktur beinhaltet eine bestimmte Anzahl von `asset`-Elementen, die jeweils über ein `name`-Attribut verfügen. Diesem wird jeweils der vollständige Name der auszuschließenden Klasse zugewiesen. Das folgende Beispiel veranschaulicht den generellen Aufbau:

[29] Bei Verwendung der `import`-Anweisung bindet Flash die Klassen beim Kompilieren nur dann ein, wenn sie im Dokument benutzt werden, beispielsweise über einen `new`-Operator oder als Funktionsparameter.

```
<excludeAssets>
    <asset name="Vollständiger Klassenname 1" />
    <asset name="Vollständiger Klassenname 2" />
    …
</excludeAssets>
```

Quellcode-Bsp. 3.3: Genereller Aufbau einer Exclude-XML

Die XML-Datei muss im selben Verzeichnis wie das betreffende Flash-Dokument (FLA-Datei) gespeichert sein und als Bezeichnung den Namen desselben inklusive des Zusatzes *_exclude* zugewiesen bekommen. Das komplette Prinzip soll noch einmal am Beispiel der Datei *init.fla* sowie der dazugehörigen *init_exclude.xml* verdeutlicht werden:

```
import loading.Preloader;
import loading.PreloaderView;
import loading.XMLLoader;
import loading.LoadDefinition;
import TextField.StyleSheet;

//folgende Klassen werden nicht mit eingebunden (siehe exclude-XML), da
//sie in der Klassenbibliothek enthalten sind
import textdata.CheckableCSS;
```

Quellcode-Bsp. 3.4: `import`-Anweisungen im Quellskript (*init.as*) der *init.fla*

```
<excludeAssets>
   <asset name="textdata.CheckableCSS"/>
</excludeAssets>
```

Quellcode-Bsp. 3.5: Ausschluss der Klasse `CheckableCSS` über die Datei *init_exclude.xml*

Zu beachten ist bei dieser Methode, dass auch sämtliche nicht in Flash integrierten Klassen, von denen Instanzen in der auszuschließenden Klasse gebildet werden oder deren Funktionalität geerbt wird, ebenfalls mit angegeben werden müssen.

In den folgenden Abschnitten werden die für das Lernmodul erstellten Klassen kurz vorgestellt. Zur Verdeutlichung der Paketzugehörigkeit sowie der Verknüpfungen untereinander wird als Bezeichnung jeweils die komplette `class`-Definition angegeben. Eine genaue Beschreibung der jeweiligen Klassenmethoden ist dem entsprechenden Anhang zu entnehmen.

3.1.1 Klassen zum Vorausladen von Daten

Durch die modulare und flexible Architektur des Lernmoduls müssen zur Laufzeit des Programms eine Reihe von externen Daten geladen werden. Um den Nutzer während dieser Zeit nicht im Unklaren darüber zu lassen, was gerade passiert, wurde eine *grafische Ladefortschrittsanzeige* implementiert. Diese gibt Auskunft über den aktuellen Ladestatus und zeigt an, welche Informationen gerade eingelesen werden. Realisiert wurde die Anzeige mit Hilfe verschiedener Klassen, die im Folgenden kurz vorgestellt werden sollen.

`interface loading.Loadable`

Die Schnittstelle `Loadable` deklariert die drei Methoden `loadFile()`, `getBytesLoaded()` sowie `getBytesTotal()`. Klassen, die diese Schnittstelle implementieren, können als *ladendes Objekt* an die Klasse `Preloader` übergeben werden.

`class loading.LoadDefinition`

Diese Klasse wurde speziell zur Übergabe an Preloader entwickelt und enthält Informationen, welche Dateien über welche Objekte geladen werden sollen. Dem Konstruktor werden dazu folgende Parameter übergeben:

- das ladende Objekt
- die URL der einzulesenden Datei
- maximal drei Flash-Parameter
- eine optionaler Parameter als Array[30]

Beim *ladenden Objekt* sollte es sich um Instanzen solcher Klassen handeln, die über die Methoden `getBytesLoaded()` sowie `getBytesTotal()` verfügen. Dazu gehören neben Flash-internen wie beispielsweise `MovieClip` oder `LoadVars` auch solche, in denen die Schnittstelle `Loadable` implementiert wurde. Andere Klassen werden durch `PreLoader` nicht verarbeitet.

`interface loading.PreloaderViewInterface`

Die Schnittstelle dient dazu, Ladefortschrittsanzeigen in der `Preloader`-Klasse zu registrieren und für den Empfang von Statusinformationen einzurichten. Ausschlaggebend für Letzteres ist die Deklaration der Methode `renderView()`, die als Parameter den Ladefortschritt in Prozent sowie Angaben über die geladene und totale Byteanzahl übergeben bekommt.

Dieses Prinzip wird auch *Observer Pattern* (*Beobachter-Muster*) genannt. Es stellt eine Form des objektorientierten Designs dar und bedeutet, dass ein Subjekt (in diesem Fall eine

[30] Ein Array bzw. `Array`-Objekt bezeichnet ein ein- oder mehrdimensionales Feld.

`Preloader`-Instanz) bei einer Statusänderung alle registrierten Beobachter (die Ladefortzschrittsanzeigen mit PreloaderViewInterface-Implementierung) über selbige benachrichtigt.

`class loading.PreloaderView implements loading.PreloaderViewInterface`

Diese Klasse implementiert die soeben erwähnte Schnittstelle und generiert die für das Lernmodul verwendete grafische Ladefortschrittsanzeige. Selbige präsentiert sich in Form eines äußeren Kreises, bestehend aus Linie und Füllung, sowie einem inneren Kreissegment, dessen Größe an die geladene Prozentzahl angepasst wird. Darüber hinaus besteht optional die Möglichkeit, den Prozentwert sowie einen beliebigen Ladetext anzuzeigen. Die folgende Abbildung zeigt drei Beispiele:

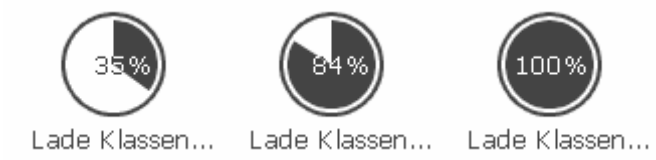

Abbildung 3.1: Verschiedene `PreloaderView`-Zustände

Die Kreisgrafiken werden über die ActionScript-Zeichenfunktionen `moveTo()` sowie `lineTo()` erstellt. Die Rundung entsteht dabei durch Linien, die jeweils im Abstand von 3,6 Grad gezeichnet werden (entspricht einem Prozent). Zur Bestimmung der jeweiligen Zielkoordinaten eines Liniensegments werden Kosinus- bzw. Sinus-Werte mit dem Kreisradius multipliziert und entsprechend zu den x- bzw. y-Werten des Kreiszentrums hinzuaddiert. Zur Verdeutlichung soll noch einmal das folgende Quellcode-Beispiel dienen. Es zeigt die Methode drawForm(), die unter anderem in Folge eines `renderView()`-Aufrufs durch den Preloader automatisch gestartet wird:

```
private function drawForm (clip:MovieClip,
                           x:Number,
                           y:Number,
                           r:Number,
                           deg1:Number,
                           deg2:Number,
                           line_col:Number,
                           fill_col:Number,
                           line:Boolean,
                           closed:Boolean):Void
```

```
{
      with (clip)
      {
            clear();

            beginFill(fill_col);

            if (line)
            {
                  lineStyle(2, line_col);
            }

            var x1:Number = x + Math.cos(Math.PI/180 * deg1) * r;
            var y1:Number = y - Math.sin(Math.PI/180 * deg1) * r;

            //ein Kreis wird aus Linien zusammengesetzt,
            //wobei die Schrittweite 3.6° beträgt (1%)
            var steps:Number = (deg1 - deg2) / 3.6;

            //bei geschlossenem Kreissegment in der
            //Kreismitte mit dem Zeichnen beginnen
            if (steps!=100 && closed)
            {
                  moveTo(x,  y);
                  lineTo(x1, y1);
            }

            moveTo(x1, y1);

            //cosinus- und sinus-Werte zum Berechnen neuer
            //Koordinaten verwenden
            for (var i=1; i<=steps; i++)
            {
                  lineTo(x + Math.cos(Math.PI/180 * (deg1-i*3.6)) * r,
                         y - Math.sin(Math.PI/180 * (deg1-i*3.6)) * r);
            }
            //falls es sich um ein geschlossenes Kreissegment
            //handelt, verläuft die letzte Linie wieder zur Mitte
```

```
            if (steps!=100 && closed)
            {
                  lineTo(     x, y);
            }

            endFill();
      }
}
```

Quellcode-Bsp. 3.6: Die Methode `drawForm()` zum Zeichnen der Kreis(segment)e

Die Klasse bietet zahlreiche gestalterische Konfigurationsmöglichkeiten. So lassen sich Kreisgrößen und –farben frei bestimmen oder auch die Texte individuell zu formatieren.

`class loading.Preloader`

Die `Preloader`-Klasse wurde entwickelt, um den Status nachgeladener Daten zu überwachen und Programmablauf dahingehend zu steuern, dass erst nach dem vollständigen Einlesen an einer gewünschten Stelle mit der Abarbeitung fortgefahren wird. Nach der Initialisierung eines Preloaders und der Registrierung aller Fortschrittsanzeigen lassen sich Objekte grundsätzlich mit Hilfe zweier Methoden prüfen: `loadData()` sowie `checkProgress()`. Der wesentliche Unterschied zwischen beiden besteht darin, dass die erste Methode ein oder mehrere `LoadDefinition`-Objekte übergeben bekommt und der Ladeprozess für die darin enthaltenen Objekte direkt in der Klasse initialisiert wird. `checkProgress()` hingegen überprüft nur den Ladefortschritt der übergebenen Objekte[31]. Das folgende Quellcode-Beispiel zeigt einen Ausschnitt aus `loadData()`:

```
public function loadData (objects:Object,
                          callFunc:String,
                          param:Object ):Void
{
      …
      for(var i=0; i<objects.length ; i++)
      {
            if(objects[i] instanceof LoadDefinition)
            {
                  loadDef = objects[i];
```

[31] Die an `checkProgress()` übergebenen Objekte müssen die Methoden `getBytesTotal()` sowie `getBytesLoaded()` implementieren, um geprüft werden zu können.

```
            }
            else …

            //LoadDefinition-Objekt auswerten
            obj         = loadDef.getLoadObject();
            url         = loadDef.getLink();
            f_param1    = loadDef.getFlashParam1();
            f_param2    = loadDef.getFlashParam2();
            opt_param   = loadDef.getOptParam();

            if(obj instanceof MovieClip)
            {
                  //opt_param >> überschriebene loadMovie-Methode verw.
                  if(opt_param != undefined)
                  {
                        obj.loadMovie(url, f_param1, opt_param)
                  }
                  //Klasse "MovieClipLoader" verwenden
                  else
                  {
                        mcArray.push(objects[i]);
                  }
            }
            else if(obj instanceof Loadable)
            {
                  obj.loadFile(url, opt_param);
            }
            …
            checkArray.push(obj)
      }
      …
      //Objekte prüfen
      this.startCheckTimer(   checkArray.slice(),
                              callFunc,
                              param);
}
```

Quellcode-Bsp. 3.7: Auszug aus der Methode `loadData()`

In der Regel werden übergebene `MovieClip`-Objekte mit Hilfe der in Flash MX 2004 neu eingeführten Klasse `MovieClipLoader` geladen. Anhand dieser lässt sich neben dem Ladestatus zusätzlich auch prüfen, wann ein Movieclip vollständig initialisiert wurde und zur Verwendung bereitsteht.

Sowohl an `loadData()` wie auch an `checkProgress()` können optional ein Funktionsname sowie ein Parameter-Array übergeben werden. Diese Funktion dient als Sprungziel und wird automatisch nach dem vollständigen Einlesen aller Daten aufgerufen.

```
private function intervalCheckSize(…)
{
      …
      if (this.m_nPercent == 100)
      {
            …
            if (callFunc) eval(callFunc)(param);
      }
}
```

Quellcode-Bsp. 3.8: Aufruf der Sprungfunktion

Der Ladestatus selbst wird intervallgesteuert überwacht. Dabei wird jedes Mal der prozentuale Fortschritt aller einzulesenden Daten insgesamt berechnet, bevor anschließend die registrierten View-Klassen über die Methode `updateViews()` aktualisiert werden.

```
private function updateViews(total:Number, loaded:Number)
{
      for (var i=0; i<this.m_aViews.length; i++)
      {
            this.m_aViews[i].renderView(this.m_nPercent, total, loaded);
      }
}
```

Quellcode-Bsp. 3.9: Aktualisierung der Ladefortschrittsanzeigen

3.1.2 Grundlegende Klassen zum Zeichen der grafischen Elemente

Die im folgenden Abschnitt beschriebenen Klassen dienen dem Definieren und Zeichen von Vektorformen und bilden eine wesentliche Grundlage für die Realisierung des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“.

class graphics.drawing.LineStyle

Mit dieser Klasse wird der Stil einer zu zeichnenden Linie beschrieben. Die an den Konstruktor übergebenen Parameter orientieren sich an der `MovieClip`-Methode `lineStyle()` von Flash MX 2004. Das heißt, es können Linienstärke, -farbe sowie -transparenz festgelegt werden. In der Ende 2005 erschienenen achten Version von Flash wurden die Möglichkeiten zur Gestaltung von Linien noch stark erweitert. Beispielsweise lassen sich nunmehr auch Farbverläufe definieren oder der optische Übergang zwischen angrenzenden Liniensegmenten festlegen.

class graphics.drawing.FillStyle

Über diese Klasse lässt sich eine *unifarbene Füllung* für Vektorformen definieren. Wie zuvor richten sich die übergebenen Parameter nach der für die spätere Auswertung verwendeten `MovieClip`-Methode, in diesem Fall `beginnFill()`. Es ist somit möglich, Farbe und Transparenz der Füllung festzulegen.

class graphics.drawing.GradientFillStyle

Diese Klasse orientiert sich an der `MovieClip`-Methode `beginGradientFill()` von Flash MX 2004. Mit ihr können *radiale oder lineare Farbverläufe* spezifiziert werden. In Flash 8 wurden die Gestaltungsmöglichkeiten noch erweitert.

class graphics.drawing.FormDefinition

Die Klasse `FormDefinition` bildet die Grundlage für nahezu alle mit ActionScript gezeichneten Vektorgrafiken (ausgenommen die Ladefortschrittsanzeige). Mit ihr lassen sich mehr oder weniger komplexe Formen aus einzelnen Linien und Kurven zusammensetzen. Dazu existieren die zwei Methoden `addLine()` sowie `addCurve()`, die im folgenden Quellcode-Beispiel dargestellt werden:

```
public function addLine(x1:Number, y1:Number, x2:Number, y2:Number,
                        lineStyleObject:LineStyle):Void
{
      this.m_aCoordinates.push({_x1:x1, _y1:y1, _x2:x2, _y2:y2});

      this.m_aLineStyles.push(this.getLineStyle(lineStyleObject));
}

public function addCurve(x1:Number, y1:Number, x2:Number, y2:Number,
                         xcp:Number, xcp:Number,
                         lineStyleObject:LineStyle):Void
```

```
{
      this.m_aCoordinates.push({_x1:x1, _y1:y1, _xcp:xcp, _xcp:xcp,
                                _x2:x2, _y2:y2});
      this.m_aLineStyles.push(this.getLineStyle(lineStyleObject));
}
```

Quellcode-Bsp. 3.10: Aktualisierung der Ladefortschrittsanzeigen

Wie aus dem Beispiel hervorgeht, kann für jedes Segment ein separater Linienstil spezifiziert werden. Ebenso wurde die Möglichkeit implementiert, die Stile aller Linien im Nachhinein anzupassen, wovon insbesondere bei der Realisierung verschiedener Schaltflächenzustände Gebrauch gemacht wurde.

`class graphics.drawing.CircleDefinition extends graphics.drawing.FormDefinition`

Diese Klasse wurde von Formdefinition abgeleitet und stellt die Möglichkeit zur Verfügung, Kreis(segment)e oder Ring(segment)e[32] zu definieren. Sie fand im Lernmodul insbesondere beim Erstellen der Navigations-, Symbol- sowie Abspielsteuerungsschaltflächen Verwendung.

`class graphics.drawing.LinePatternDefinition extends graphics.drawing.FormDefinition`

Der Zweck dieser Klasse ist es, ein Muster bestehend aus einer beliebigen Anzahl parallel zueinander verlaufender Linien zu definieren. Die Koordinaten der Basislinie sowie x- und y-Versatz lassen sich individuell bestimmen. Die Klasse wurde verwendet, um die Linien im Hintergrund und im Zusatzbereich bzw. Animationsbildschirm zu zeichnen.

`class graphics.drawing.RectangleDefinition extends graphics.drawing.FormDefinition`

Bei dieser Klasse handelt es sich um eine der am häufigsten verwendeten im gesamten Lernmodul. Das Hauptanliegen bei der Entwicklung war es, auf einfache Art und Weise die Definition abgerundeter Rechtecke zu ermöglichen, wie sie in der grafischen Benutzungsoberfläche häufig anzutreffen sind. So werden beispielsweise Elemente des Hintergrunds, wie Register oder Bühne, sowie Bild- und Textschaltflächen mit Hilfe der Klasse `RectangleDefinition` erstellt. Darüber hinaus lassen sich auch Rahmen zeichnen, bestehend aus einem äußeren und einem inneren Rechteck. Diese Möglichkeit wurde jedoch

[32] Ein Ring(segment) bezeichnet hier eine Form bestehend aus zwei Kreis(segmentent)en mit unterschiedlichem Radius.

nur zum Zwecke der späteren Wiederverwendung hinzugefügt, jedoch für die Realisierung der Oberfläche nicht verwendet.

class graphics.drawing.SplineDefinition extends graphics.drawing.FormDefinition

Die an die Methode `defineSpline()` übergebenen Koordinaten innerhalb eines Arrays werden von `SplineDefinition` automatisch zu einer – auf Wunsch geschlossenen – Vektorform verbunden. Die Klasse entstand aus der Absicht heraus, die relativ einfach gehaltenen Symbole der Abspielsteuerung über ActionScript zu zeichnen und diese nicht als vorgefertigte Grafiken in einer Bibliothek abzulegen.

class graphics.drawing.FormDraw

Mit Hilfe der Klasse `FormDraw` lassen sich erstellte `FormDefinition`-Objekte in Movieclips zeichnen. Dazu werden die gewünschten Instanzen zunächst an die Methode `addformDefinition()` übergeben, bei Bedarf inklusive eines `FillStyle`- oder `GradientFillStyle`-Objekts. Anschließend lassen sich die Vektorformen über `drawForm()` in einem oder mehren Movieclips zeichnen. Dabei werden innerhalb einer Schleife die übergebenen Definitionen und Füllstile ausgewertet und in entsprechende ActionScript-Befehle übertragen. Das soll im folgenden Quellcode-Beispiel veranschaulicht werden:

```
public function drawForm(clips:Object, ignoreLines:Boolean):Void
{
  …
  for (var j=0; j<this.m_aFormDefinitions.length; j++)
  {
      //Referenzen auf Koordinaten, Linienstile, Füllstile etc.
      //bilden
      …
      for (var z=0; z<clips.length; z++)
      {
            …
            //Füllung zeichnen
            if( fillStyle )
            {
                  if(fillStyle instanceof GradientFillStyle)
                  {
                    clips[z].beginGradientFill(fillStyle.getFillType(),
```

```
                                        fillStyle.getColors(),
                                        …);
            }
            else if(fillStyle instanceof FillStyle)
            {
                  clips[z].beginFill(…);
            }
      }

      for (var i=0; i<coordinates.length; i++)
      {
            coord       = coordinates[i];
            ls          = lineStyles[i];

            //linie sichtbar?
            if(ignoreLines != true && ls)
            {
                  if(ls.getThickness()    != lsOld.getThickness()||
                     ls.getColor()        != lsOld.getColor()    ||
                     ls.getAlpha()        != lsOld.getAlpha() )
                  {
                        clips[z].lineStyle(…);
                  }
                  else
                  …

                  if(coord._cp1!=undefined) //kurve
                        clips[z].curveTo(…);
                  else //linie
                        clips[z].lineTo(…);
                  …
```

Quellcode-Bsp. 3.11: Auswertung übergebenen Formdefinitionen und Füllstile in der Klasse `FormDraw`

3.1.3 Klassen zum Erstellen spezieller Movieclips

Die im Folgenden beschriebenen Klassen sind von `MovieClip` abgeleitet und bilden die Grundlage für eine Reihe weiterer AS2-Klassendefinitionen, speziell im Zusammenhang mit interaktiven Steuerelementen.

`dynamic class graphics.objects.FramedClip extends MovieClip`

`FramedClip` ist eine von `MovieClip` abgeleitete Basisklasse, die ihre Eigenschaften und Methoden komplexeren Klassen vererbt. Obwohl in der Regel alle auf dynamischen Klassen[33] basierenden Unterklassen ebenfalls dynamisch sind, so wurde dies im Fall von `MovieClip` Flash-intern verhindert. Um dennoch wie gewohnt mit `FramedClip` arbeiten zu können, enthält der Klassenkopf zusätzlich das Schlüsselwort `dynamic`.

Im Konstruktor der Klasse werden drei leere Movieclips erstellt, die für verschiedene Elemente vorgesehen sind. In der obersten Ebene kann ein Rahmen, bestehend aus einem übergebenen `FormDefinition`-Objekt, gezeichnet werden. Diese Vektorform ist gleichzeitig ausschlaggebend für eine Farbfüllung, die auf Wunsch in der untersten Ebene erzeugt wird. In den mittleren Movieclip lassen sich Inhalte aus der Bibliothek oder externe SWF-Dateien laden. Verfügen diese über durchsichtige Bereiche, ist die Hintergrundfüllung zu erkennen. Alternativ lassen sich die beiden unteren Ebenen auch vertauschen, was im Zusammenhang mit semitransparenten Füllstilen unter Umständen sinnvoll sein kann.

Mit Blick auf die beiden für das Lernmodul vorgesehenen Abspielsteuerungen wurde eine Methode namens `stopDuringLoad()` erstellt. Diese verhindert das automatische *Streaming* (Abspielen während des Ladens) aktuell ladender SWF-Inhalte, indem innerhalb einer `onEnterFrame`-Funktion der betreffende Movieclip stetig gestoppt wird. Darüber hinaus lässt sich anhand eines übergebenen booleschen Parameters festlegen, ob der Abspielvorgang nach einem erfolgreichen Einlesen gestartet werden soll.

```
public function stopDuringLoad(playAfterLoad:Boolean):Void
{
  var theClass = this;
  this.m_mcContentHolder.onEnterFrame = function()
  {
    theClass.m_mcContent.stop();

    theClass.m_bContentIsLoaded=(theClass.m_mcContent.getBytesTotal() &&
                                 theClass.m_mcContent.getBytesTotal() ==
                                 theClass.m_mcContent.getBytesLoaded())

    if(theClass.m_bContentIsLoaded)
    {
      playAfterLoad ? theClass.m_mcContent.gotoAndPlay(1) :
                      theClass.m_mcContent.gotoAndStop(1);
```

[33] Dynamische Klassen erlauben das nachträgliche Hinzufügen von Eigenschaften und Methoden zur Laufzeit.

```
      delete onEnterFrame;
    }
  }
}
```

Quellcode-Bsp. 3.12: Auszug aus `stopDuringLoad()`

`class graphics.objects.MaskedClip extends graphics.objects.FramedClip`

Diese Klasse erweitert `FramedClip` dahingehend, dass der Inhalt durch die Form maskiert und somit ausschließlich innerhalb der Grenzen dargestellt wird. Dazu wird eine entsprechende schwarze Füllung in einem separaten Movieclip gezeichnet, bevor dieser mit Hilfe der `MovieClip`-Methode `setMask()` als Maske zugewiesen wird. Durch die Möglichkeit, den Inhalt zu verschieben[34], lässt sich der maskierte Ausschnitt individuell festlegen.

`class graphics.objects.TextClip extends graphics.objects.MaskedClip`

Die Klasse `MaskedClip` wurde nochmals erweitert um die Möglichkeit der Darstellung von Text. Falls dieser mit eingebetteten Schriftzeichen angezeigt wird (siehe Abschnitt 3.2.3), so lässt er sich ebenfalls maskieren. Die Größe des Textfeldes kann in Abhängigkeit von dem an `setTextFieldProperties()` übergebenen `_autoSize`-Parameter automatisch an den Inhalt angepasst werden. Zudem besteht die Möglichkeit, die Schrift mittig an einem zuvor spezifizierten Punkt auszurichten (siehe Quellcode-Beispiel 3.12). Alternativ lässt sich das Textfeld jedoch auch traditionell positionieren und dimensionieren.

```
public function centerTextField(x:Number, y:Number):Void
{
      …
      this.m_nTimerID = setInterval(this, "center", 20, x, y);
}

private function center(x:Number, y:Number):Void
{
      var tf = this.m_Textfield;

      if( tf.textWidth != 0 && tf.textHeight != 0 )
```

[34] geerbt von `FramedClip`

```
        {
                this.m_Textfield._x = int(x-tf._width/2);
                this.m_Textfield._y = int(y-tf._height/2);

                        this.removeTimer();
        }
}
```

Quellcode-Bsp. 3.13: Automatisches Zentrieren des Textes an einem spezifiziertem Punkt

Die Schrift lässt sich flexibel formatieren, entweder anhand eines `TextFormat`- oder eines `StyleSheet`-Objekts.

3.1.4 Klassen zur Steuerung von Animationen

Die folgenden Klassen wurden entwickelt, um die für das Lernmodul vorgesehenen Animationen (vgl. Abschnitt 2.3.5) anzuzeigen und zu steuern.

`class animation.SimplePlayer`

Diese einfache Abspielsteuerung mit lediglich einer *Wiedergabe/Pausieren*-Schaltfläche wurde entwickelt, um kleine Beispielanimationen wiedergeben und bei Bedarf anhalten zu können. Es lassen sich bei der Initialisierung verschiedene Parameter festlegen, wie Linien- und Füllstile oder die Skalierung der Schaltfläche bei einem Maus-Rollover-Ereignis. Zur Darstellung des Films enthält die Klasse eine Mitgliedsvariable des Typs `MaskedClip`. Von einer Vererbung wurde abgesehen, da verschiedene öffentliche Methoden in `SimplePlayer` nicht zur Verfügung stehen sollen. Stattdessen kapselt die Klasse benötigte Dienste in gleichnamigen Methoden und macht diese dem Anwender zugänglich, wie das nachfolgende Beispiel zeigt. Man spricht in einem solchen Fall auch von einer *Delegation*.

```
…
public function stopDuringLoad(playAfterLoad:Boolean):Void
{
	this.m_mcMovie.stopDuringLoad(playAfterLoad);
}

public function getContent():MovieClip
{
	return this.m_mcMovie.getContent();
}
…
```

Quellcode-Bsp. 3.14: Delegation von Aufgaben an das `MaskedClip`-Objekt

Die Schaltfläche der Abspielsteuerung wurde ausschließlich mit Hilfe der unter Abschnitt 3.1.2 beschriebenen Zeichen-Klassen generiert. Beim Erstellen einer Instanz von `SimplePlayer` lässt sich zudem festlegen, an welcher Kante des Wiedergabefensters die Schaltfläche platziert wird. Standardmäßig erfolgt die Darstellung – wie auch im Lernmodul – rechts unten. Die Ereignisbehandlung, die das wechselnde Starten und Pausieren der Animation bei Klicken und anschließendem Loslassen der Maustaste beinhaltet, wurde direkt in die Klasse implementiert.

Für die Schaltfläche lassen sich darüber hinaus Tooltipps definieren, die beim Hineinfahren des Mauszeigers in den sensitiven Bereich sowie im Fall eines Abspielstatus-Wechsels angezeigt werden. Realisiert wird diese Funktionalität mit Hilfe einer `Tooltip`-Mitgliedsvariable. Die Ausrichtung des Hinweisfensters richtet sich dabei immer in umgekehrter Weise nach der Schaltflächenposition: Befindet sich das Steuerelement beispielsweise im *linken* Bereich, so wird der Tooltip *rechts* über ihr dargestellt.

Prinzipiell gelten für den Aufbau des abgespielten Flash-Films dieselben Regeln wie bei der Klasse `ComplexPlayer` und werden deshalb im nachfolgenden Textabschnitt mit beschrieben.

`class animation.ComplexPlayer`

Obwohl diese Klasse nicht von `SimplePlayer` erbt, so ist sie doch recht ähnlich aufgebaut. Sie dient im Lernmodul dazu, die Animationen mit kommentierten Handlungsabläufen darzustellen. Der wesentliche Unterschied zu `SimplePlayer` besteht darin, dass ungleich flexiblere Eingriffsmöglichkeiten in das Wiedergabegeschehen geboten werden. So lässt sich beispielsweise die Animation auf der Suche nach einer gewünschten Stelle durchsuchen oder die Lautstärke von enthaltenem Audiomaterial beeinflussen (vgl. auch Abschnitt 2.4.3).

Die Steuerelemente sind in einer Konsole platziert, die je nach Bedarf an der oberen oder

unteren Kante des Wiedergabefensters angezeigt werden kann. Über eine spezielle Schaltfläche besteht zusätzlich die Möglichkeit, die Steuerkonsole aus- bzw. einzublenden. Dazu wird sie mit Hilfe einer `onEnterFrame()`-Funktion einfach hinter das Wiedergabefenster bzw. wieder heraus gefahren.

Der Positionsschieber besitzt zwei Funktionen, wie sie auch aus anderen Mediensteuerungen bekannt sind: Zum einen zeigt er die aktuelle Abspielposition an, indem er bei während der Wiedergabe automatisch mitläuft, zum anderen lässt sich mit seiner Hilfe eine gewünschte Stelle innerhalb der Animation aufsuchen. Die jeweils benötigte Position[35] wird mit folgender (der Schaltfläche direkt zugeordneter) Funktion berechnet:

```
b.getPos = function(search_to_movie:Boolean)
{
  var frames = theClass.getContent()._totalframes - 1;
  //Breite der Positions-Skala
  var w      = b._parent.scale._width;
  var pos;

  if(search_to_movie) //Abspielkopf an Reglerposition ausrichten
  {
    pos = Math.round(b._x * frames / w) + 1;
  }
  else //Reglerposition an Wiedergabe anpassen
  {
    pos = Math.round((theClass.getContent()._currentframe-1)*w/frames);
  }
  return pos;
}
```

Quellcode-Bsp. 3.15: Anpassung des Films oder des Positionsschiebereglers

Die Beschaffenheit der Klasse `ComplexPlayer` bedingt einen bestimmten Aufbau der Filme, um diese ordnungsgemäß wiedergeben und steuern zu können. Die Flash-Animationen müssen so erstellt werden, dass das Geschehen direkt und komplett in der *Hauptzeitleiste* des Dokuments stattfindet. Alternativ können animierte Abläufe auch in *Grafiksymbolen*[36] gekapselt und in die Hauptzeitleiste eingefügt werden. Verzichtet werden sollte hingegen auf ein dynamisches Laden von Inhalten zur Laufzeit sowie auf die Nutzung animierter

[35] x-Koordinate oder Schlüsselbild

[36] Grafiksymbolen werden synchron zur Hauptzeitleiste wiedergegeben.

Movieclip-Symbole, da diese asynchron abgespielt werden. Ebenso ist die Verwendung von Audiomaterial in Form von *Streaming-Sounds*[37] zu empfehlen. Dahinter verbirgt sich die Tatsache, dass Flash diese wie Grafiksymbole synchron zur Zeitleiste wiedergibt, in die sie eingefügt wurden. Auf diese Weise wird ein Gleichlauf von visuellen und akustischen Inhalten gewährleistet.

Beim Erstellen einer Instanz von `ComplexPlayer` lassen sich neben der bereits erwähnten Position der Steuerkonsole beispielsweise auch Linien- und Füllstile, Schaltflächenradien oder die Größe des Wiedergabefensters bestimmen. Es ist somit eine durchaus flexible Anpassung an die Programmumgebung möglich.

3.1.5 Klassen zum Erstellen der Benutzungsoberflächenelemente

Mit den im folgenden Abschnitt beschriebenen Klassen lassen sich verschieden Objekte der Benutzungsoberfläche, wie beispielsweise Hintergrund oder interaktive Steuerelemente, generieren. Die Wiederverwendbarkeit einiger Klassen ist jedoch fraglich, da sie sehr auf die Bedürfnisse des Lernmoduls zugeschnitten wurden. Es lässt sich auf diese Weise jedoch der Programmcode des Flash-Systemkerns (siehe Abschnitt 3.2.2) entlasten und übersichtlicher gestalten, was im Fall einer späteren Pflege bzw. Erweiterung von Vorteil ist.

`class moduleObjects.ModuleBackground`

Mit Hilfe der unter Abschnitt 3.1.2 beschriebenen Zeichenklassen generiert die Klasse `ModuleBackground` der Hintergrund der Benutzungsoberfläche. Dieser setzt sich in hierarchischer aufsteigender Reihenfolge zusammen aus:

- einer Füllung
- einem waagerechten Linienmuster
- einer gewünschten Anzahl von Registern
- einer Bühne
- zwei Textfeldern für die Hauptüberschrift sowie den Namen des Lernobjekts

Die grafische Darstellung lässt sich flexibel an die jeweiligen Bedürfnisse anpassen. So können beispielsweise die Farben und Größen der Bühne und des Hintergrunds ebenso festgelegt werden wie die Formatierung der Textfelder.

Die Anzahl der darzustellenden Register wird beim Erzeugen einer Klassen-Instanz festgelegt. Sie werden in Form übereinander gezeichneter `RectangleDefinition`-

[37] Der Begriff Streaming-Sound besitzt in Flash zweierlei Bedeutungen: Zum einen bezeichnet er eine mit Hilfe der `Sound`-Klasse dynamisch geladene MP3-Datei, die bereits während des Ladens abgespielt wird. Zum anderen – und auf diese Bedeutung bezieht sich der Text – ist damit ein in der Zeitleiste eingefügtes Sound-Symbol gemeint, das mittels entsprechendem Parameter synchron zu dieser wiedergegeben wird.

Objekte realisiert. Dabei erfolgt eine farbliche Abstufung, die sich aus übergebenen Farbwerten für das erste Register und die Unterkante des Bühnenrahmens bzw. das letzte Register[38] ergibt. In der Methode `getColorStep()` werden dazu die Rot-, Grün- und Blauwerte beider Farben extrahiert. Anschließend wird jeweils die Differenz gebildet, durch die maximale Registeranzahl dividiert und mit der im Parameter `step` übergebenen Registernummer multipliziert. Der daraus resultierende Wert wird zum entsprechenden Farbanteil addiert, bevor die neue Farbe aus den Anteilen zusammengesetzt und zurückgegeben wird. Dies soll im Quellcode-Beispiel 3.14 veranschaulicht werden:

```
function ModuleBackground(…)
{
  …
  //Register erstellen
  for (var i=1; i<=number_of_sites; i++)
  {
    r.defineRect(…);

    f.addFormDefinition(r, new FillStyle(
                                    this.getColorStep(number_of_sites-i)));
  }
…
public function getColorStep (step:Number):Number
{
  var c1:Number  = this.m_nRegisterColorTop;
  var c2:Number  = this.m_nRegisterColorBottom;
  var max:Number = this.m_nMaxRegister;

  var r1,r2,g1,g2,b1,b2;

  //RGB-Werte extrahieren
  r1 = c1 >> 16;
  r2 = c2 >> 16;

  g1 = (c1 & 0x00ff00)  >> 8;
  g2 = (c2 & 0x00ff00)  >> 8;
```

[38] Die maximale Anzahl an Registern lässt sich ebenfalls über einen Parameter festlegen.

```
  b1 = c1 & 0x0000ff;
  b2 = c2 & 0x0000ff;

  //RGB-Werte des Schritts berechnen
  r1 += ((r2-r1) / max) * step;
  g1 += ((g2-g1) / max) * step;
  b1 += ((b2-b1) / max) * step;

  return (r1 << 16) + (g1 << 8) + b1;
}
```

Quellcode-Bsp. 3.16: Berechnung der farblichen Zwischenwerte

Ist die tatsächliche Anzahl der Register geringer als die maximal festgelegte, so unterscheidet sich der Farbwert des letzten Registers von dem der Unterkante. Da zwischen beiden jedoch eine optische Verbindung besteht (rechte Rahmenkante), wurde ein Farbverlauf generiert und somit der Übergang fließend gestaltet.

Die beiden Textfelder wurden so konzipiert, dass die Formatierung über `StyleSheet`-Objekte erfolgt. Zudem lässt sich festlegen, ob eingebettete Schriftzeichen verwendet werden sollen oder nicht.

`class moduleObjects.ModulePage`

Diese recht einfach gehaltenene Klasse generiert auf Wunsch ein Textfeld für den Hauptbereich sowie den Zusatzbereich, ebenfalls inklusive eines Textfelds. Sämtliche Elemente werden in einem speziell zu diesem Zweck übergebenem MovieClip-Objekt erzeugt. Beim Erstellen einer Instanz lässt sich neben der Position und den Abmessungen der Seite auch festlegen, welchen Abstand die Textfelder an ihren vier Kanten zum Rand besitzen, ob sie auswählbar sind und ob eingebettete Schriftzeichen verwendet werden sollen.

Der Zusatzbereich kann optional erstellt werden. Dazu existieren die beiden öffentlichen Methoden `makeLinePatternSecField()` und `makeFilledSecField()`. Wie aus deren Bennenung hervorgeht, ist es entweder möglich, als Hintergrund ein Muster aus (horizontalen) Linien zu erzeugen, oder aber die Fläche komplett mit einer beliebigen Füllung zu versehen. Letztere Variante verhilft unter Umständen zu einer besseren Lesbarkeit und wurde deshalb implementiert. Die Position des Zusatzbereichs – links, rechts, oben oder unten auf der Seite – kann ebenfalls frei gewählt werden. Im Anschluss erfolgt automatisch eine Anpassung der Textfeldgröße des Hauptbereichs an die veränderte Situation.

Die HTML-formatierten Texte selbst werden über `LoadVars`-Objekt geladen. Dazu werden der Methode `loadText()` eine boolesche Variable zur Bestimmung des Textfelds, die URL der Datei sowie ein Objekt der `StyleSheet`-Klasse übergeben, anhand dessen der

eingelesene Text später formatiert wird. Um den Ladeprozess innerhalb der `Preloader`-Klasse initialisieren zu können, wurde zusätzlich die Methode `getLoadObject()` implementiert, die lediglich die `LoadVars`-Instanz erstellt und anhand der übergebenen Parameter konfiguriert, ohne mit dem Einlesen der Daten zu beginnen.

Alternativ zum Laden einer HTML-Datei kann eine entsprechend formattierte Zeichenkette auch direkt dem gewünschten Textfeld zugewiesen werden.

Des Weiteren besteht die Möglichkeit, über die Klassenmethoden Container-Movieclips zu generieren, die sich beispielsweise zur Aufnahme grafischer Ressiourcen verwenden lassen[39].

`class moduleObjects.ModuleAnimationPage`

Diese Klasse erzeugt den Animationsbildschirm (ausgenommen der *Schließen*-Schaltfläche), wie er für das Lernmodul vorgesehen ist. In einem an den Konstruktor übergebenen Container-Movieclip lassen sich ein Hintergrund, ein Textfeld sowie ein leerer Movieclip erstellen, der wiederum als Basis für eine Instanz der Klasse `ComplexPlayer` dient.

Der Hintergrund lässt sich über ähnlich benannte Methoden an die Gestaltung des Zusatzbereichs anpassen. Ebenso ist das Prinzip des Textfeld-Konfigurierens und Ladens von HTML-Inhalten mit dem der Klasse `ModulePage` identisch.

Die Abspielsteuerung wird in der Klasse lediglich anhand der übergebenen Parameter initialisiert und positioniert. Jede weitere Nutzung der Instanz muss jedoch extern erfolgen, indem auf diese über die Methode `getPlayer()` eine Referenz gebildet wird.

`dynamic class moduleObjects.NavigationButton extends MovieClip`

Diese Klasse erbt von `MovieClip` und wurde entwickelt, um die kreisförmigen Navigationsschaltflächen (siehe Abschnitt 2.4.3) zu generieren. Sie beinhaltet zu diesem Zweck eine Instanz des Typs `TextClip`. Mit Hilfe von an die Methode `setButtonProperties()` übergebenen `LineStyle`- und `FillStile`-Instanzen sowie einem `StyleSheet`-Objekt lassen sich die beiden Zustände *Inaktiv* und *Aktiv* flexibel konfigurieren. Zusätzlich kann die Schaltfläche skaliert werden, wenn der Mauszeiger in den sensitiven Bereich hineinfährt.

Die Behandlung von Mausereignissen wurde nicht fest in die Klasse implementiert. Es existieren lediglich *Pseudo-Ereignisroutinen* in Form von Methoden mit den Bezeichnungen `onRollOverAction()`, `onRollOutAction()` sowie `onReleaseAction()`. Der Vorteil dieser Vorgehensweise besteht darin, dass die eigentlichen Ereignisroutinen (z.B. `onRelease()`) weiterhin zur Verfügung stehen und je nach Flash-Anwendung zusätzlichen Quellcode beinhalten können. Um die Methoden der Klasse zu verwenden,

[39] Die Movieclips werden in der Ebenenhierarchie oberhalb der Textfelder platziert.

müssen diese nur über den Zeiger `this`[40] innerhalb der jeweiligen Ereignisroutine aufgerufen werden.

```
public function onReleaseAction():Void
{
	this.setActive();
	this.scaleBack();
}
public function onRollOverAction():Void
{
	this.scaleUp();
}
public function onRollOutAction():Void
{
	this.scaleBack();
}
```

Quellcode-Bsp. 3.17: Das Prinzip der Pseudo-Ereignisroutinen

`class moduleObjects.NavigationButtonBar`

Diese Klasse dient der weiteren Vereinfachung des Systemkern-Programmcodes, indem sie automatisch eine gewünschte Anzahl `NavigationButton`-Instanzen generiert und diese waagerecht nebeneinander als Navigationsleiste anordnet. Die Schaltflächen lassen sich beim Erstellen über `configButtons()` gleichzeitig auch konfigurieren.

Das Prinzip der Pseudo-Ereignisroutinen wird hier in übergeordneter Form fortgesetzt. Die Klasse definiert entsprechende Methoden, in denen die Routinen eines `NavigationButton`-Objekts aufgerufen werden. Darüber hinaus sorgt jedoch zusätzlicher Programmcode dafür, dass das Schaltflächenverhalten weiter angepasst wird. So werden die in der `onReleaseAction()`-Methode enthaltenen Anweisungen nur ausgeführt, wenn es sich um eine inaktive Schaltfläche handelt. In diesem Fall wird eine eventuell aktive Schaltfläche zurückgesetzt und der Status der aktuellen geändert. Für eine aktive Schaltfläche wird auch keine Hand dargestellt, wenn sich der Mauszeiger im sensitiven Bereich befindet.

```
public function onReleaseAction(button:NavigationButton):Void
{
	if( !button.getStatus() )
```

[40] Der Zeiger this verweist immer auf das aktuelle Objekt oder die aktuelle Movieclip-Instanz. Im speziellen Fall wäre dies das `NavigationButton`-Objekt, in dessen Ereignisroutine es verwendet wird.

```
        {
                //Aktive Schaltfläche zurücksetzen
                if(this.m_mcActiveButton != null)
                {
                        this.m_mcActiveButton.setInactive();
                        this.m_mcActiveButton.useHandCursor = true;
                }

                this.m_mcActiveButton = button;
                //Pseudo-Ereignisroutine der Schaltfläche aufrufen
                button.onReleaseAction();
                button.useHandCursor = false;
        }
}
```

Quellcode-Bsp. 3.18: `onReleaseAction()`-Methode

`dynamic class moduleObjects.LabeledButton extends MovieClip`

Diese Klasse erbt von MovieClip und erzeugt eine rechteckige, auf Wunsch abgerundete Textschaltfläche mit Hilfe eines `TextClip`-Objekts. Der Aufbau der Klasse ähnelt in seinen Grundzügen dem von `NavigationButton`. Ein Unterschied besteht darin, dass als dritter der *Selektiert*-Zustand konfiguriert werden kann. Da eine Skalierung der Schaltfläche als Resultat des visuellen Designs nicht vorgesehen war, wurde eine entsprechende Möglichkeit nicht implementiert.

Bei einem Wechsel des Schaltflächenzustands wird die Beschriftung mit der entsprechenden CSS-Klasse formatiert und anschließend das Textfeld mittig in der Schaltfläche ausgerichtet.

```
private function setLabel(styleClass:String):Void
{
        var s:String = "<span class='" + styleClass + "'>" + this.m_sLabel
                        + "</span>";

        this.m_mcTextClip.setText(s);
        this.m_mcTextClip.centerTextField(this.m_nTextX, this.m_nTextY);
}
```

Quellcode-Bsp. 3.19: Beschriftung formatieren und zuweisen, das Textfeld anschließend ausgerichten

`class moduleObjects.LabeledButtonContainer`

Diese Klasse erzeugt in einem an den Konstruktor übergebenen Container-Movieclip eine beliebige Anzahl von Textschaltflächen. Dazu existiert eine Methode namens `addButton()`, der neben den relativen Koordinaten, den Abmessungen sowie der Beschriftung unter anderem auch Identitäts-Nummer übergeben wird. Anhand dieser lässt sich später über die Methode `getButton()` eine Referenz auf eine bestimmte Schaltfläche bilden.

Mit der Methode `setAllInactive()` wird die gerade aktive Schaltfläche zurückgesetzt. Dies ist dann wichtig, wenn verschiedene Instanzen der Klasse erzeugt werden, global jedoch immer nur maximal eine Schaltfläche aktiv sein soll.

`dynamic class moduleObjects.ScreenshotButton extends MovieClip`

Diese Klasse erzeugt mit Hilfe einer `MaskedClip`-Instanz eine Bildschaltfläche und ist vom Aufbau her nahezu identisch mit `LabeledButton`. Als Bildinhalt wird ein Objekt der Bibliothek verwendet, das über den ihm zugewiesenen und an die Klasse übergebenen *Verknüpfungsbezeichner*[41] angesprochen wird.

`class moduleObjects.ScreenshotButtonContainer`

Diese Klasse ist praktisch identisch mit `LabeledButtonContainer`, nur dass in diesem Fall Bildschaltflächen ertsellt werden.

`dynamic class moduleObjects.SymbolButton extends MovieClip`

Der prinzipielle Aufbau dieser Klasse orientiert sich an dem von `NavigationButton`, ist das Schaltflächenverhalten bei beiden doch praktisch identisch. Die Grafik wird mit Hilfe eines `MaskedClip`-Objekts realisiert und nutzt vorgefertigte Elemente der Dokument-Bibliothek zur Darstellung des Symbols. Diese lässt sich in den Zuständen *Aktiv* und *Inaktiv* jeweils in einer frei wählbaren Farbe darstellen. Die dafür benötigte Methode wurde in der Klasse `FramedClip`[42] definiert und aus `drawButton()` heraus aufgerufen:

```
//Methode von SymbolButton
private function drawButton():Void
{
      var _status:String = this.m_bIsActive? "2"      : "1";
      …
      if(this["m_nSymbolColor" + _status])
```

[41] Ein Verknüpfungsbezeichner kann bestimmten Symbolarten in der Dokument-Bibliothek zugewiesen werden, um von diesen zur Laufzeit mittels ActionScript Instanzen zu erstellen.

[42] `MaskedClip` erbt von `FramedClip`

```
            this.m_mcMaskedClip.setContentColor
                  (this["m_nSymbolColor" + _status]);
}

//aufgerufene Methode aus FramedClip
public function setContentColor(col:Number):Void
{
      var c = new Color(this.m_mcContentHolder);
      c.setRGB(col);
}
```

Quellcode-Bsp. 3.20: Zuweisen einer Farbe über eine `FramedClip`-Methode

`class moduleObjects.SymbolButtonBar`

Nach dem Prinzip von `NavigationButtonBar` erzeugt diese Klasse aus einer beliebigen Anzahl von `SymbolButton`-Instanzen eine waagerechte Symbolleiste. Die Schaltflächen reagieren jedoch im Gegensatz zu denen der Navigationsleiste auch im aktiven Zustand auf Mausereignisse (über Pseudo-Ereignisroutinen). Die Klasse wurde verwendet, um die Symbolleiste des Zusatzbereichs zu erzeugen.

`class moduleObjects.SimplePageObjects`

Diese Klasse erzeugt innerhalb eines an den Konstruktor übergebenen Container-Clips Objekte, mit denen sich die verschiedenen grafischen Ressourcen des Lernmoduls darstellen lassen. Dazu zählen im Einzelnen:

- Bilder
- Download-Schaltflächen, die mit einer URL verknüpft sind
- Interaktionen
- Beispielanimationen

Für alle Elemente existieren `create`-Methoden, mit denen die benötigten Objekte erzeugt und konfiguriert werden. Im Fall von *Bildern* und *Download-Schaltflächen* werden einfache Movieclips erstellt. Über die `onRelease()`-Ereignisroutine wird für die Schaltfläche zusätzlich angegeben, dass die zuvor übergebene und als Eigenschaft zugewiesene Datei-URL aufgerufen werden soll.

Zur Darstellung von *Interaktionen* wird eine Instanz von `MaskedClip` verwendet. Als Rahmen wird ein – auf Wunsch abgerundetes – Rechteck gemäß der an die Methode übergebenen Parameter gezeichnet.

Beispiel-Animationen werden mit Hilfe der `SimplePlayer`-Klasse dargestellt. Der `createAnimation`-Methode lassen sich neben den eigentlichen Konfigurationsdaten

optional auch die zur Darstellung von Tooltipps notwendigen Parameter übergeben.

Der Ladeprozeß der einzelnen Ressourcen wird nicht initialisiert. Stattdessen wird für jedes erzeugte Objekt gleichzeitig auch eine `LoadDefinition`-Instanz erstellt, in der die benötigten Daten gespeichert werden. Jede Instanz wird einem Array hinzugefügt, das eine Mitgliedsvariable der Klasse ist, und dieses anschließend von der betreffenden `create`-Methode zurückgegeben. Alternativ lässt sich auf das Array über `getLoadDefinitions()` zugreifen. Die `LoadDefinition`-Objekte sind letztlich für die Übergabe an eine `Preloader`-Instanz vorgesehen.

Wie ersichtlich wird, ist diese Klasse sehr speziell und wurde auch nur aus Gründen der Programmstrukturierung entwickelt.

3.1.6 Klassen zum Laden der XML-Konfigurationsdateien

Die folgenden Klassen wurde entwicklelt, um einfache Variablendefinitionen auf XML-Basis laden und innerhalb einer hierarchischen Objektstruktur abbilden zu können. Nähere Informationen zu den einzelnen erstellten XML-Dateien sind dem Abschnitt 3.3 sowie den betreffenden Spezifikationen zum Aufbau im Anhang dieser Arbeit entnehmen.

`dynamic class util.DBoolean extends Boolean`
`dynamic class util.DNumber extends Number`
`dynamic class util.DString extends String`

Die einzige Erweiterung dieser drei Klassen im Vergleich zu den Flash-Originalen liegt in der `dynamic`-Deklaration. Zweck dieser Maßnahme war es, Objekte der Klassen untereinander als Eigenschaften zuzuweisen, um auf diese Weise die Element- und Attribut-Beziehungen der XML-Struktur leicht abbilden zu können.

`class loading.XMLLoader extends XML`

Der Konstruktor der Klasse dient lediglich dazu, verschiedene Mitgliedsvariablen mit Standardwerten zu initialisieren. Die zentrale Rolle bei der Anwendung von `XMLLoader` spielt hingegen die überschriebene Methode `load()` der `XML`-Klasse.

Da die Werte der XML-Elemente und –Attribute standardmäßig als Objekte vom Typ `String` zurückgegeben werden, erschien es sinnvoll, eine Möglichkeit zur Umwandlung in numerische bzw. boolesche Variablen zu implementieren. Ausgangspunkt dessen sind die `Number`-Klasse sowie die `isNaN`-Funktion (*is Not a Number – ist keine Zahl*) von Flash. Mit ihrer Hilfe wird geprüft, ob ein Wert in einem korrekten Zahlenformat (z.B. `2`, `2.4`, `2e+3`, `0xff00ee`) vorliegt. Ist dem so, wird automatisch eine numerische Variable daraus erzeugt. Anderenfalls wird geprüft, ob die Zeichenkette `"true"` oder `"false"` lautet und demzufolge nach einem booleschen Wert verlangt. Die Datentypumwandlung erfolgte mit Hilfe der Klassen `DNumber`, `DBoolean` sowie `DString` aus den bereits erwähnten Gründen. Das

folgende Quellcode-Beispiel veranschaulicht diese Vorgehensweise:

```
private function generateVariable (varName:String,
                                   varValue:String, obj:Object):Void
{
      var val:Object;
      //Wert auf numerische Konformität überprüfen
      //true = NaN = keine Zahl
      if (isNaN(Number(varValue)))
      {
            var s:String = varValue.toLowerCase();
            //überprüfen, ob es sich um eine boolesche Variable
            //handelt
            if(s == "true" || s == "false")
            {
                  val = new DBoolean(s == "true" ? true : false);
            }
            else
            {
                  val = new DString(varValue);
            }
      }
      //false = Zahl
      else
      {
            val = new DNumber(varValue);
      }
      …
}
```

Quellcode-Bsp. 3.21: Datentyp-Umwandlung von Werten

Beim Aufruf von `load()` wird außerdem eine neue `Object`-Instanz für die Mitgliedsvariable `m_oVariables` angelegt, in der die spätere Datenstruktur gespeichert wird. Nach dem erfolgreichen Einlesen der Daten[43] wird die Methode `checkChildNode()` aufgerufen und der XML-Baum Schritt für Schritt rekursiv ausgewertet. Ihr werden das zu

[43] Der Ladestatus wird anhand der überschriebenen Ereignisroutine `onLoad()` ermittelt. Diese wurde als `private` deklariert und ist deshalb von außen nicht mehr zugänglich.

prüfende XML-Element[44] der Struktur sowie das aktuelle (Unter)Objekt übergeben, in dem die Klasse die ausgelesenen Daten platziert. Beim erstmaligen Aufruf sind dies das `XMLLoader`-Objekt[45] selbst und die in `load()` erstellte `Object`-Instanz.

Die Methode `checkChildNode()` besteht hauptsächlich aus einer `for`-Schleife, die so lange durchlaufen wird, wie Unterelemente für das übergebene XML-Element vorhanden sind. Besitzt eines dieser ebenfalls Unterelemente[46], so wird die Methode `checkCildNode()` rekursiv aufgerufen. Derartige Elemente werden außerdem in Form einer neuen `Object`-Instanz der jeweils übergeordneten hinzugefügt[47]. Besitzt ein Unterelement hingegen einen eigenen Wert bzw. Textknoten[48], so wird es an die Methode `generateVariable()` gesendet. Zuletzt erfolgt eine Prüfung hinsichtlich eventuell definierter Attribute. Im folgenden Quellcode-Beispiel wird der gesamte Prozess noch einmal veranschaulicht:

```
private function checkChildNode(child:XMLNode, obj:Object):Void
{
      for (var j=0; j<child.childNodes.length; j++)
      {
        subChild = child.childNodes[j];
        //Element hat selbst Unterelemente
        if(subChild.firstChild.firstChild)
        {
            this.checkChildNode(subChild,
                                this.generateObject(subChild.nodeName,
                                                    obj));
        }
        //Element besitzt einen Wert/Textknoten
        else if (subChild.firstChild.nodeValue)
        {
            //Variable aus Wert des Elements generieren
            this.generateVariable(subChild.nodeName,
                                  subChild.childNodes.join(""),
```

[44] `XMLNode`-Objekt

[45] `XMLLoader` erbt von der `XML`–Klasse, die ihrerseits von `XMLNode` abgeleitet ist.

[46] Wird durch `subChild.firstChild.firstChild` ermittelt.

[47] Ausgenommen davon ist der Wurzelknoten, dessen Daten (Attribute, Wert) direkt in `m_oVariables` platziert werden.

[48] Textknoten inklusive Formatierung (z.B. …) werden als ein Wert ausgelesen.

```
                                    obj);
            }
            //Element nach Attributen durchsuchen
            this.checkForAttributes(subChild, obj[subChild.nodeName]);

        }
}
```

Quellcode-Bsp. 3.22: Auswerten des XML-Baums

XML-Attribute werden in der Objektstruktur dem dazugehörigen Element als Eigenschaften zugewiesen, ebenso wie Elemente, die hierarchisch eine Ebene tiefer liegen. Als Variablenname wird jeweils die Bezeichnung des XML-Elements bzw. -Attributs verwendet. Sollten sich auf einer Ebene der XML-Struktur Elemente gleichen Namens befinden oder eine Elementbezeichnung mit der eines Attributs des übergeordneten Elements übereinstimmen, so wird aus diesen automatisch ein `Array`-Objekt erstellt. Dies soll an folgendem Beispiel verdeutlicht werden:

```
<wurzel_element>
   <elementX wert1="1">
      <wert1>2</wert1>
      <wert2><b>Ein</b> Text</wert2>
   </elementX>
   <elementX>
      <wert>true</wert>
   </elementX>
   <elementY wert="Hallo">Welt</elementY>
</wurzel_element>
```

Quellcode-Bsp. 3.23: Beispiel-XML-Datei

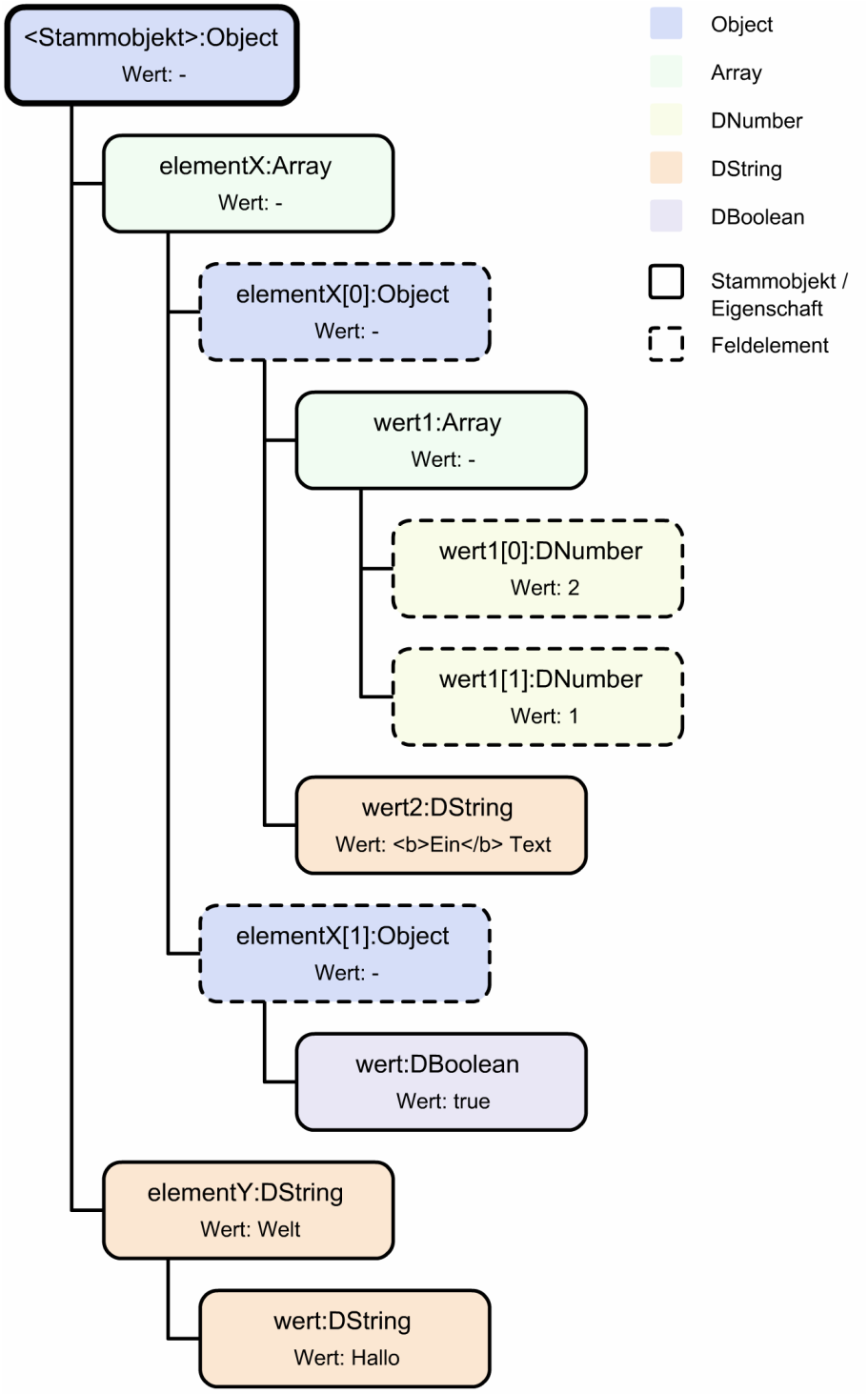

Abbildung 3.2: Objektstruktur nach dem Auslesen der XML-Datei

Nachdem der komplette XML-Baum ausgewertet und die entsprechenden Variablen im klasseneigenen Stammobjekt gespeichert wurden, lässt sich eine Referenz auf dieses über die Methode `getObject()` bilden. Wöllte man anschließend beispielsweise auf das `wert1`-Attribut des ersten `elementX`-Elementes der XML-Struktur zugreifen, so müsste Folgendes im Quelltext eingegeben werden:

```
<Stammobjekt_Referenz>.elementX[0].wert1[1]
```

Der Attribut-Wert befindet sich deshalb an letzter Stelle im Array, weil er *nach* dem `wert1`-Element ausgelesen wird.

Des Weiteren wurde die `getBytesLoaded()`-Methode der XML-Superklasse überschrieben, so dass von der zurückgegebenen Bytezahl stets der Wert Eins subtrahiert wird, solange die geladene XML-Struktur nicht vollständig ausgewertet wurde.

3.1.7 Klasse zum Anzeigen von Tooltipps

Die im foldenen Abschnitt beschriebene Klasse wurde entwickelt, um für verschiedene interaktive Elemente kleine Tooltipps anzeigen zu können.

`class util.Tooltip`

Ein Tooltip wird in Form eines abgerundeten Rechtecks mit Hilfe der unter Abschnitt 3.1.2 beschriebenen Zeichenklassen dargestellt. Dieses lässt sich über die Methode `setBackground()` konfigurieren. Zur Formatierung der Textausgabe stehen verschiedene Möglichkeiten zur Verfügen, beispielsweise kann dies mit Hilfe eines `StyleSheet`-Objekts geschehen.

Für die Anzeige des Tooltipps kann der Anwender beispielswiese festlegen, wie groß das Intervall bis zur Darstellung ist oder ob diese nur erfolgt, wenn der Mauszeiger reglos an seiner aktuellen Position verharrt. Letztere Möglichkeit bewirkt, dass ein laufender Timer gelöscht und ein neuer gestartet wird. Dieser Prozess wird so lange fortgesetzt, bis der Mauszeiger die erforderliche Zeitspanne an Ort und Stelle verbleibt. Das folgende Quellcode-Beispiel soll diesen Vorgang verdeutlichen:

```
private function startTimer(cancel_on_mouseMove:Boolean):Void
{
	this.setTimer();

	if(cancel_on_mouseMove)
	{
		var theClass = this;

		this.m_mcTooltip_mc.onMouseMove = function()
		{
			theClass.clearTimer();
			theClass.setTimer();
		}
```

```
        }
}
```

Quellcode-Bsp. 3.24: Mausabfrage zur Unterbrechung eines laufenden Intervalls

Darüber hinaus besteht die Möglichkeit, zwischen zwei Formen der Darstellung zu wählen: Einem statischen Hinweisfenster (`showLockedTooltip()`) und einem, das sich mit dem Mauszeiger in einem spezifizierten Abstand mitbewegt (`showMovingTooltip()`).

Entfernt wird ein Tooltipp manuell über die Methode `removeTooltip()`.

3.1.8 Klasse zum Laden von CSS-Dateien

Die im Folgenden beschriebene Klasse wurde zum Laden von CSS-Dateien über eine `Preloader`-Instanz entwickelt.

`class textdata.CheckableCSS extends TextField.StyleSheet implements loading.Loadable`

Die Klasse `CheckableCSS` wurde von `StyleSheet` abgeleitet und erweitert diese um die Möglichkeit, der `Preloader`-Klasse als ladendes bzw. zu prüfendes Objekt übergeben zu werden. Dazu wurde die Schnittstelle `Loadable` implementiert. In der Methode `loadFile()` wird eine Mitgliedsvariable mit einer neuen `LoadVars`-Instanz initialisiert. Nach dem vollständigen Einlesen der CSS-Daten werden diese der Klasse mittels `parseCSS()` übergeben.

```
public function loadFile(css_file:String, param:Array):Void
{
      var CLASS = this;

      if(this.m_oLoadVars) delete this.m_oLoadVars;

      this.m_oLoadVars = new LoadVars();

      this.m_oLoadVars.onData = function(src:String)
      {
            //CSS-Daten zuweisen
            CLASS.parseCSS(src);
      }

      this.m_oLoadVars.load(css_file)
```

```
}

public function getBytesLoaded():Number
{
	return this.m_oLoadVars.getBytesLoaded();
}

public function getBytesTotal():Number
{
	return this.m_oLoadVars.getBytesTotal();
}
```

Quellcode-Bsp. 3.25: Loadable-Implementation in CheckableCSS

3.1.9 Klassen zur Fehlerbehandlung

Die im Folgenden beschriebenen Klassen zur Fehlerbehandlung wurden nur rudimentär implementiert, sollen aber der Vollständigkeit halber mit erwähnt werden. Beide sind von der AS2-Klasse `Error` abgeleitet

`class errorhandling.TypeError extends Error`

Dieser spezielle Fehler sollte mittels `throw`-Anweisung geworfen werden, wenn ein Objekt einen falschen Datentyp aufweist. Die Standard-Fehlerausgabe (`message`-Variable) der `Error`-Klasse wurde überschrieben, wie das folgende Quellcode-Beispiel zeigt:

```
class errorhandling.TypeError extends Error
{
	public var message:String = "Invalid datatype of object."

	public function TypeError(e:String)
	{
		super(e);
	}
}
```

Quellcode-Bsp. 3.26: `TypeError`-Klasse

`class errorhandling. UndefinedObjectError extends Error`

Im Unterschied zur vorherigen Klasse lässt sich mit dieser der Fehler ausgegeben, dass ein Objekt nicht definiert wurde.

3.2 Generelle Struktur des Lernmoduls

In den folgenden Abschnitten soll näher auf die physischen und logischen Strukturen des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“ eingegangen werden. Das Hauptaugenmerk bei der Entwicklung selbiger lag auf der Umsetzung der im Vorfeld angestellten und unter Abschnitt 2.4.4 beschriebenen Überlegungen.

3.2.1 Verzeichnisstruktur

Beim Erstellen der Verzeichnisstruktur wurde versucht, die zwei wesentlichen Datenbereiche des Lernmoduls eindeutig voneinander abzugrenzen: spezifische Lernobjekt-Daten sowie gemeinsam genutzte bzw. nutzbare Ressourcen (Systemkern, Grafiken etc.). Ebenso erschien es sinnvoll, die Quelldaten (z.B. FLA-Dateien, (Klassen)Skripte, PNG-Dateien) in separaten Ordnern zu speichern. Durch die Möglichkeit von Flash, die eigentlichen Filme (SWF-Dateien) jeweils in einem gesonderten Pfad zu veröffentlichen, lassen sich auf diese Weise die für den späteren Ablauf benötigten Programmteile von den ausschließlich für die Entwicklung relevanten trennen.

Ein weiterer Punkt betrifft die Auslagerung von Quellcode in AS-Dateien (vgl. Abschnitt 1.3.2). Obwohl im speziellen Fall des Lernmoduls nicht zwingend erforderlich, hilft eine solche Vorgehensweise insbesondere bei der Entwicklung größerer Flash-Projekte. So lässt sich die Effizienz steigern, indem von verschiedenen Filmen gemeinsam benutzte Programmteile externalisiert und zentral verwaltet werden.

Im Folgenden soll die Verzeichnisstruktur des Lernmoduls erläutert werden. Dabei ist anzumerken, dass Ordner, die lediglich Quelldaten beinhalten (Endung *_sources*), kein Bestandteil des finalen SCORM-Paketes sind. Nachstehende Verzeichnisse und Unterverzeichnisse wurden erstellt:

- *chapter_X_Y*: Ordner mit spezifischen Lernobjekt-Daten (X=Hauptkapitel, Y=Unterpunkt); enthalten jeweils die HTML-Datei zum Starten, eine Konfigurations-XML, SCORM-Metadaten sowie folgende Unterverzeichnisse:
  - *page1…pageX*[49]: enthalten die XML-Konfigurationsdaten sowie die HTML-Texte für die einzelnen Lernschritte; die Textdaten wurden der Übersichtlichkeit halber ebenfalls in Unterverzeichnissen (*common* , *buttons*) abgelegt
- *common*: auf diesen Ordner wird von sämtlichen Lernobjekten zugegriffen; er enthält SWF-Dateien (Systemkern, Klassenbibliothek, Logo) sowie das folgende Unterverzeichniss:
  - *config*: beinhaltet XML- sowie CSS-Dateien für das Lernmodul und die

[49] Die Bezeichnung ist für alle Lernobjekte vorgeschrieben und setzt sich immer aus *page* und der Lernschrittnummer zusammen.

Ladefortschrittsanzeige

- *common_sources*: enthält die Flash-Quelldateien für das *common*-Verzeichnis und folgende Unterordner:
  - *scripts*: ausgelagerter AS2-Quellcode der in *common_sources* gespeicherten FLA-Dateien
  - *classes*: enthält die AS2-Klassen, gespeichert unter verschiedenen Paket-ordnern
- *javascript*: enthält JS-Dateien für die Einbindung des Lernmoduls in eine SCORM-konforme Lernumgebung und zum Laden der Datei *init.swf* aus dem *common*-Verzeichnis
- *resources*: enthält alle sonstigen Ressourcen des Lernmoduls in folgenden Unterordnern:
  - *big_animations*: Animationen, die Handlungsabläufe verdeutlichen
  - *download_buttons*: Download-Schaltflächen
  - *downloads*: Dateien, die heruntergeladen werden können
  - *interactions*: Interaktionen
  - *pictures*: Grafiken
  - *screenshots*: Hintergrundgrafiken für Bildschaltflächen
  - *small_animations*: Beispielanimationen
  - *symbols*: Grafiken für die Symbolschaltflächen
- *resources_sources*: enthält die Quellen für den *ressources*-Ordner

3.2.2 Prozess beim Laden eines Lernobjekts

Wie aus dem vorangegangenen Abschnitt bereits ersichtlich wurde, besteht ein komplettes Lernobjekt nicht aus einem einzelnen Flash-Film, sondern setzt sich aus verschiedenen Teilfilmen, XML-, HTML- sowie JavaScript-Dateien zusammen. Diese Vorgehensweise bietet einerseits den Vorteil, Programmsegmente zentral zu verwalten und je nach Bedarf auch mehreren Lernobjekten zur Verfügung zu stellen. Andererseits begünstigt die Externalisierung von Konfigurationsdaten und Textinhalten sowohl die Pflege als auch die Erweiterung des Lernmoduls. Die Daten werden zur Laufzeit des Programms mit Hilfe der `Preloader`-Klasse nacheinander geladen. Gleichzeitig wird der Anwender mit Hilfe einer Ladefortschrittsanzeige über den jeweils aktuellen Status informiert.

Abbildung 3.3 veranschaulicht den Ladeprozess eines Lernobjekts und stellt die einzelnen Abhängigkeiten zwischen der verschiedenen Programmteilen dar. Es lässt sich eine grobe Untergliederung in drei Abschnitte vornehmen: Initialisierung des Lernmoduls, Generierung von Hintergrund und Navigation sowie Aufbau eines Lernschritts.

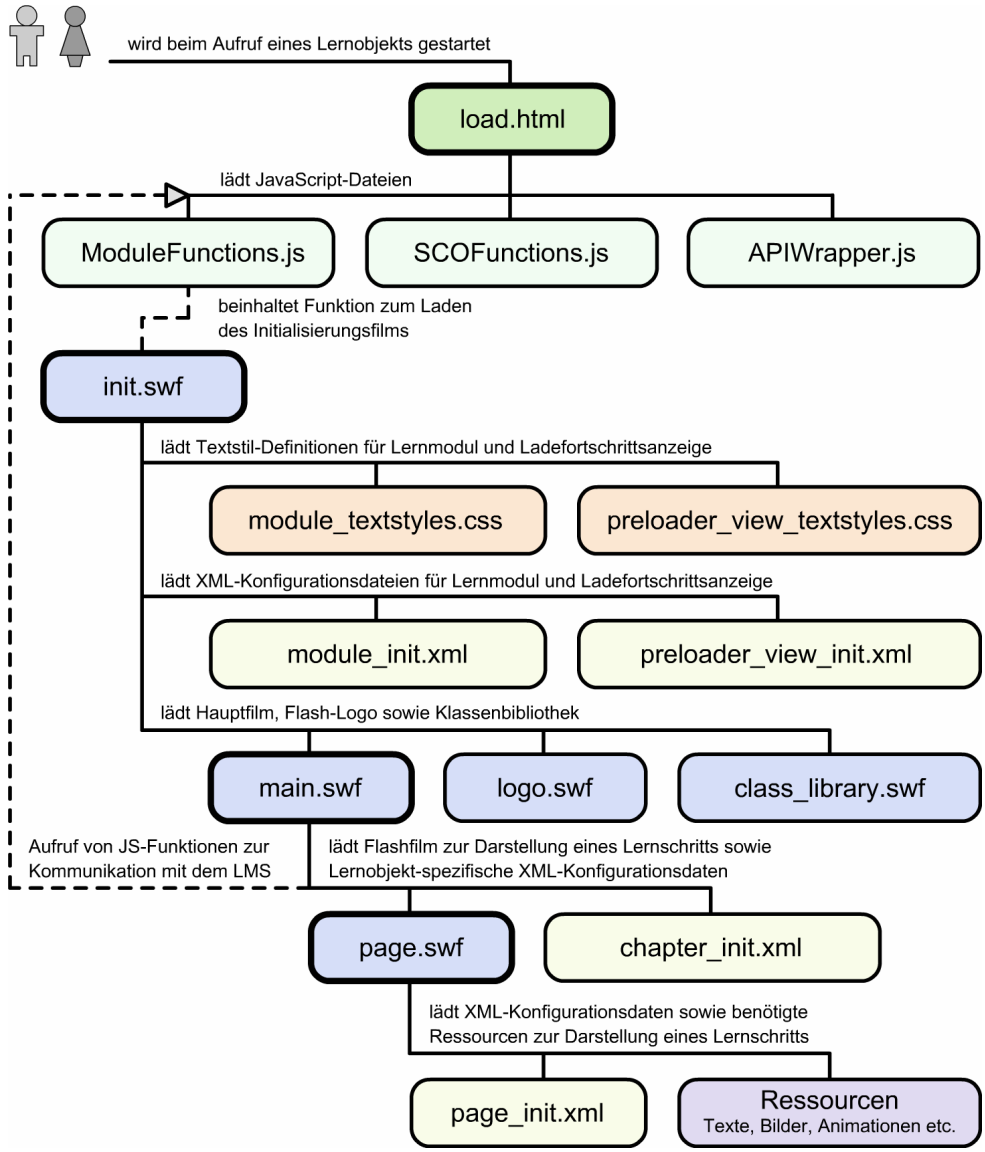

Abbildung 3.3: Ladeprozess eines Lernobjekts

Die Vorgehensweise bei Erstellung bzw. Programmierung der einzelnen Anwendungsbausteine orientierte sich an der *Top-down-Methode*: Ausgehend von der höchsten Ebene der Flash-Struktur (*init.swf* im Ordner *common*) wurden nach und nach die hierarchisch untergeordneten Programmelemente erzeugt. Nachdem das Grundgerüst des Lernmoduls fertig gestellt war und die funktionalen Anforderungen erfüllte, begann die Produktion der eigentlichen Lernobjekte durch Erstellung der jeweiligen Konfigurationsdaten und multimedialen Ressourcen.

Im folgenden Text soll näher auf verschiedene Komponenten eingegangen werden, die am Ladeprozess beteiligt sind.

load.html

Über die Datei *load.html* wird ein einzelnes Lernobjekt aufgerufen. Sie verfügt über keinerlei relevanten Inhalt und dient lediglich als eine Art Container für JavaScript-Programmcode.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
<title>FLASH INTERAKTIV</title>

<script type="text/javascript" language="JavaScript">
        mainpath="../"
</script>
<script type="text/javascript"
        language="JavaScript"
        src="../javascript/APIWrapper.js">
</script>
<script type="text/javascript"
        language="JavaScript"
        src="../javascript/SCOFunctions.js">
</script>
<script type="text/javascript"
        language="JavaScript"
        src="../javascript/ModuleFunctions.js">
</script>

</head>
</html>
```

Quellcode-Bsp. 3.27: Die Datei *load.html*

Die Datei ist für jedes Lernobjekt genau einmal vorhanden und prinzipiell identisch aufgebaut. Der Quellcode unterscheidet sich lediglich dann, wenn SCOs in unterschiedlichen Verzeichnisebenen platziert werden. In diesem Fall müssen die jeweiligen JavaScript-Links sowie die Variable `mainpath` modifiziert werden, um in korrekter Weise auf den Hauptordner zu verweisen. Die Variable ist wichtig, damit später der Flash-Film *init.swf* in der Verzeichnisstruktur gefunden und geladen werden kann.

Der Speicherpfad der *load.html* spielt auch eine Rolle beim Zugriff auf die spezifischen Lernobjekt-Daten. Er wird vom Flash-Systemkern verwendet, um das jeweils aktuelle *Arbeitsverzeichnis* zu ermitteln.

JavaScript-Dateien

Die Dateien *APIWrapper.js* sowie *SCOFunctions.js* dienen ausschließlich der Integration des Lernmoduls in eine SCORM 1.2-konforme Lernumgebung. Sie werden in ihrer grundlegenden Form von ADL für die lizenzfreie Nutzung zum Download bereitgestellt und sind ebenso in den Beispieldateien der kostenlos verfügbaren *Sample RTE 1.2.2* (Sample Runtime Environment) enthalten. In Abschnitt 3.4.1 dieser Arbeit wird näher auf die in den beiden JavaScript-Dateien enthaltenen Funktionen eingegangen, mit denen die Einbindung in eine SCORM-Laufzeitumgebung ermöglicht wird. An dieser Stelle soll deshalb lediglich der Programmcode betrachtet werden, der für das Laden des obersten Flash-Films (*init.swf*) verantwortlich ist und sich in der Datei *ModuleFunctions.js* befindet.

Nachdem alle Skripte durch die *load.html* geladen wurden, erfolgt innerhalb der soeben erwähnten JavaScript-Datei automatisch der Aufruf der Funktion `loadFlashContent()`. Die zentrale Aufgabe dieser ist es, mit Hilfe der `write`-Methode des `document`-Objektes dynamischen HTML-Code zu erzeugen. Die betreffenden Programmzeilen wurden der HTML-Veröffentlichungsvorlage *Flash mit FSCommand* von Flash entnommen und enthalten neben VBScript (nähere Informationen dazu im Abschnitt 3.4.1) auch ein `object`- und ein `embed`-Element. Letztere werden generiert, um das Laden des Flash-Films sowohl auf ActiveX-fähigen Browsern[50] als auch auf allen anderen[51] zu ermöglichen.

```
//dynamisch erzeugter HTML-Code zum Laden des Flash-Inhalts
function loadFlashContent()
{
      var useGetURL = navigator.platform.indexOf("Win") == -1 ||
                      navigator.userAgent.indexOf("Opera") != -1;

      var movie_id = "flashfilm";

      //an den Flashfilm zu übergebende Variablen
      var flashVars = "IS_LMS=" + checkForLMS.toString() +
                      "&USE_GETURL=" + useGetURL.toString() +
                      "&WORKPATH=" + location.toString();
```

[50] über das `object`-Element, z.B. Microsoft Internet Explorer

[51] über das `embed`-Element, z.B. Firefox

```
//falls das Lernmodul in ein LMS integriert wurde, werden
//zusätzliche Variablen gesetzt und die SCORM-Schnittstelle
//initialisiert
if(checkForLMS)
{
      …
      flashVars += "&LESSON_LOCATION=" + ll_vp[0] +
                   "&VISITED_PAGES=" + ll_vp[1];
}
…

////////////////////////////////////////////////////////////////
//dynamisches Einbinden des Flashfilms ////////////////////////
////////////////////////////////////////////////////////////////

document.write('<body bgcolor="#e3e3e3" … >');
//ActiveX
document.write('<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-
                                  444553540000"
                        codebase="http://fpdownload.macromedia.com
                                  /pub/shockwave/cabs/flash/
                                  swflash.cab#version=7,0,0,0"
                        … >');

document.write('<param name="movie"
                       value="' + mainpath + 'common/init.swf">');
…
//Variablenübergabe in param-Tag
document.write('<param name="flashvars"
                       value="' + flashVars + '" />');

//alternative Variante für nicht ActiveX-fähige Browser
document.write('<embed src="' + mainpath + 'common/init.swf"
                       …
                       flashvars="' + flashVars + '"
                       …/></embed>');
```

```
        document.write('</object></body>');
}
var checkForLMS = true;
loadFlashContent();
```

Quellcode-Bsp. 3.28: Auszug aus der Funktion `loadFlashContent()`

Der HTML-Code wurde dahingehend erweitert, dass beim Laden gleichzeitig Variablen an die Datei *init.swf* übergeben werden. Realisiert wird dies mit Hilfe der Eigenschaft `flashvars` (auch `FlashVars`), die sowohl in einem separaten `param`-Element als auch in Form eines `embed`-Attributs definiert werden sollte, um die Unterstützung auf allen Browsern zu gewährleisten. Im geladenen Flash-Film werden automatisch gleichnamige Eigenschaften des Typs `String` erzeugt und in der obersten Ebene[52] platziert. Sollen mehrere Variablen übergeben werden, so müssen diese durch ein UND-Zeichen (‚&') miteinander verkettet werden.

Die Datei *init.swf* empfängt beim Start bis zu fünf Variablen, deren Verwendungszweck in nachfolgender Tabelle kurz beschrieben wird:

Variable	**Bedeutung**
`IS_LMS`	Anhand der booleschen Variable `checkForLMS` innerhalb der *ModulFunctions.js* wird festgelegt, ob ein Lernobjekt nach der API-Instanz im DOM suchen soll. Da sich das Lernmodul grundsätzlich auch außerhalb einer SCORM-konformen Lernumgebung einsetzen lässt, kann mit der Übergabe des Wertes als `IS_LMS` die Lernerdaten-Übermittlung an ein Lernmanagement-system eingeschaltet oder unterbunden werden.
`USE_GETURL`	In Abhängigkeit vom verwendeten Internetbrowser wird die Kommunikation von Flash nach JavaScript entweder über die Funktion `getURL` oder mittels `fscommand` realisiert. Nähere Informationen dazu sind dem Abschnitt 3.4.1 zu entnehmen.
`WORKPATH`	Die `WORKPATH`-Variable enthält den Pfad der *load.html* und damit das Arbeitsverzeichnis des gerade aktiven Lernobjekts.

[52] `_root` bzw. `_level0`

LESSON_LOCATION	Diese Variable enthält die Nummer des Lernschritts, bei dem das betreffende Lernobjekt zuletzt verlassen wurde. Bei einem erstmaligen Aufruf eines SCOs wird die Zahl Eins übergeben. Weitere Informationen sind im Abschnitt 3.4.1 zu finden.
VISITED_PAGES	Dem Flash-Film werden zu Beginn die Nummern der schon bearbeiteten Lernschritte übergeben. Dies ist notwendig, um den Abschlussstatus eines SCOs zu bestimmen. Weitere Informationen können dem Abschnitt 3.4.1 entnommen werden.

Tabelle 3.1: An die Datei *init.swf* übergebene Variablen

init.swf (Quellen: *init.fla*, *init.as* sowie *init_exclude.xml*)
Die Datei *init.swf* bildet hierarchisch die oberste Ebene in der dem Lernmodul zugrunde liegenden Flash-Struktur. Zur Laufzeit eingelesene XML-Konfigurationsdaten dienen als Basis, um zu einem späteren Zeitpunkt der Programmabarbeitung die verschiedenen Elemente der Benutzungsoberfläche zu generieren bzw. konfigurieren. Darüber hinaus werden weitere Flash-Filme – *class_library.swf* (Klassenbibliothek) sowie *main.swf* (Hauptfilm) – zur Laufzeit nachgeladen.

Neben den mittels JavaScript übergebenen Variablen wurden weitere definiert (siehe Tabelle 3.2). Da innerhalb einer Flash-Struktur alle geladenen Filme aufeinander zugreifen können, stehen die Eigenschaften prinzipiell allen untergeordneten SWFs zur Verfügung. Um die übergreifende Verwendung bestimmter Variablen zu verdeutlichen, wurden bei der Namensvergabe *Großbuchstaben* gewählt.

Variable	**Bedeutung**
_global.INIT_CLIP	Eine globale Referenz auf den Film *init.swf*.
_global.STAGE_XW _global.STAGE_YW	In den beiden Variablen, die der Einfachheit halber global definiert wurden, werden Bühnenbreite sowie -höhe der *init.swf* gespeichert. Diese Maße dienen zur Dimensionierung der Benutzungsoberfläche.
PATH	Die Variable enthält den Pfad der *init.swf*.
PRELOADER	Ein Objekt der Klasse `Preloader`, über das Initialisierungsdaten sowie Inhalte des Hauptbereichs geladen werden.
PRELOADER_VIEW	Enthält eine Ladefortschrittsanzeige zum Visualisieren des Status von PRELOADER.

INFO_PRELOADER	Ein Objekt der Klasse Preloader, über das sämtliche Inhalte des Zusatzbereichs geladen werden.
INFO_PRELOADER_VIEW	Enthält eine Ladefortschrittsanzeige zum Visualisieren des Status von INFO_PRELOADER.
CSS_DATA	Ein Objekt der Klasse CheckableCSS zum Einlesen der Textstile *module_textstyles.css*.
MODULE_INIT	Ein Objekt, das die eingelesenen und aufbereiteten Daten der Konfigurationsdatei *module_init.xml* enthält.
CHAPTER_INIT	Ein Objekt, das die eingelesenen und aufbereiteten Daten der Lernobjekt-spezifischen Konfigurationsdatei *chapter_init.xml* enthält.
config_path	Die Variable enthält den Pfad der Modul-Konfigurations-daten und ergibt sich aus PATH+"config/"
plView_init	Ein Objekt, das die eingelesenen und aufbereiteten Daten der Konfigurationsdatei *preloader_view_init.xml* enthält.
plView_styles	Ein StyleSheet-Objekt zum Einlesen der Textstile *preloader_view_textstyles.css*.
module_cfg_loader	Ein Objekt der Klasse XMLLoader zum Laden und Aufbereiten der Konfigurationsdatei *module_init.xml*.
chapter_cfg_loader	Ein Objekt der Klasse XMLLoader zum Laden und Aufbereiten der Konfigurationsdatei *chapter_init.xml*.

Tabelle 3.2: Variablen der *init.swf*

Die PATH-Variable wird mit Hilfe der MovieClip-Eigenschaft _url ermittelt (die oberste Ebene eines SWF, also _root, ist ebenfalls vom Typ MovieClip). Dieser Aufruf wird von den verschiedenen Browsern jedoch unterschiedlich ausgewertet. So gibt der Internet Explorer die URL in der Schreibweise mit umgekehrten Schrägstrich (‚\') zurück, wohingegen andere Produkte die gebräuchlichere Variante mit dem normalen Schrägstrich (‚/') liefern. Aus diesem Grund wurde die Funktion checkPath() erstellt. Sie wandelt sämtliche Pfadangaben in die normale Schrägstrich-Schreibweise um und returniert anschließend einen Teilstring, der die neue URL ohne den Dateinamen enthält (siehe Quellcode-Bsp. 3.29).

```
checkPath = function(path:String):String
{
      if(path==undefined) return null;

      var index_char;
      var old_index     = 0;
```

```
        var newPath       = "";

        while(true)
        {
              //umgekehrten Schrägstrich suchen
              index_char = path.indexOf('\\', old_index);

              if(index_char != -1)
              {
                 //umgekehrten Schrägstrich ersetzen
                 newPath += path.substring(old_index, index_char) + '/';
                 old_index = index_char + 1;
              }
              else
              {
                 if (newPath != "") path = newPath;
                 break;
              }
        }
        //Pfad ohne Dateinamen zurückliefern
        return path.substr(0, path.lastIndexOf('/')+1);
}
```

Quellcode-Bsp. 3.29: Die Funktion `checkPath()` innerhalb der *init.swf*

Die `PATH`-Variable wird an verschiedenen Stellen des Quellcodes verwendet und macht es zwingend erforderlich, die Anordnungen der einzelnen Dateien des *common*-Verzeichnisses sowie dessen selbst innerhalb der Lernmodul-Ordnerstruktur beizubehalten. Anderenfalls lassen sich unter Umständen einzelne Daten nicht auffinden oder auch der Hauptpfad (im Lernmodul definiert als hierarchisch dem *common*-Ordner direkt übergeordnete Verzeichnis-Ebene) nicht korrekt bestimmen.

Eine Hauptfunktion der Datei *init.swf* ist das Bereitstellen zweier *Preloader-Instanzen* inklusive der dazugehörigen *Ladefortschrittsanzeigen* (vgl. Tabelle 3.2, siehe auch Abschnitt 3.1.1), über deren Klassenmethoden weitere Komponenten des Systemkerns, Konfigurationsdaten sowie multimediale Ressourcen geladen werden. Die grafische Statusanzeige lässt sich mit Hilfe der Dateien *preloader_view_init.xml*[53] sowie *preloader_view_textstyles.css* konfigurieren. Eine genaue Beschreibung der darin enthaltenen

[53] Die Datei *preloader_view_init.xml* wird mit einem Objekt der `XMLLoader`-Klasse geladen.

Datenelemente kann dem separaten Anhang C entnommen werden, der dieser Arbeit beiliegt. Nach dem erfolgreichen Einlesen der Konfigurationsdaten werden in einem zweiten Schritt die `Preloader`- und `PreloadView`-Instanzen erzeugt und eingerichtet (siehe Quellcode-Bsp. 3.30). Die für die Textdarstellung benötigten Schriftzeichen sind in der *init.swf* eingebettet (siehe Abschnitt 3.1.3).

```
//Laden der Konfiguration für die PreloaderView-Instanzen
loadPreloaderViewConfig = function()
{
…
}

loadPreloaderViewTextstyles = function()
{
…
}

//Preloader initialisieren und anzeigen
initializePreloader = function ()
{
      //MovieClips für PreloaderView-Instanzen erstellen
      var mc:MovieClip = createEmptyMovieClip ("plView_holder",
                                               1001);

      mc.createEmptyMovieClip("preloaderView1", 1);
      mc.createEmptyMovieClip("preloaderView2", 2);

      //Referenzen auf Objekte mit Konfigurationsdaten
      var c_obj = plView_init.colors;
      var s_obj = plView_init.sizes;
      var p_obj = plView_init.coordinates;

      //Preloader für Hauptbereich erstellen
      PRELOADER         = new Preloader();
      //Ladefortschrittsanzeige erstellen
      PRELOADER_VIEW    = new PreloaderView (mc.preloaderView1,
                                             c_obj.color1,
                                             c_obj.color2,
```

```
                                                s_obj.outerCircleRadius,
                                                s_obj.innerCircleRadius);

        //View bekanntmachen
        PRELOADER.registerView(PRELOADER_VIEW);

        //Preloader für Zusatzbereich erstellen
        INFO_PRELOADER          = new Preloader();
        …

        var view:Array = new Array(PRELOADER_VIEW, INFO_PRELOADER_VIEW);

        //Textfelder formatieren
        for (var i=0; i<view.length; i++)
        {
              view[i].setPercentByStyleSheet(p_obj.percentX,
                                             p_obj.percentY,
                                             "percent1",
                                             "percent2",
                                             plView_styles);

              view[i].setLoadTextByStyleSheet(p_obj.loadTextX,
                                              p_obj.loadTextY,
                                              s_obj.loadTextWidth,
                                              s_obj.loadTextHeight,
                                              "loadtext",
                                              plView_styles);
        }

        PRELOADER_VIEW.resetView();
        PRELOADER_VIEW.setPos(STAGE_XW/2, STAGE_YW/2-10);
        PRELOADER_VIEW.setLoadText("  Initialisiere...");

        INFO_PRELOADER_VIEW.hideView();
        INFO_PRELOADER_VIEW.resetView();

        loadClasses();
}
```

```
…
loadPreloaderViewConfig();
```

Quellcode-Bsp. 3.30: Initialisierung der Preloader(Views)

Nach der Initialisierung der Preloader werden nacheinander folgende Daten geladen:

- Klassenbibliothek *class_library.swf*
- Konfigurationsdaten *module_init.xml* sowie *chapter_init.xml*
- Textstile *module_textstyles.css*
- Hauptfilm *main.swf*

Beim Einlesen wird nach einem einheitlichen Prinzip vorgegangen, das auch in den beiden nachgeladenen Filmen *main.swf* sowie *page.swf* wiederkehrt: Nachdem der Fortschrittsanzeige eine Zeichenkette übergeben wurde, die den während des Ladevorgangs anzuzeigenden Text enthält, erfolgt der Aufruf von `Preloader.loadData()`. Der Methode werden die *ladenden Objekte*, die *URLs der einzulesenden Dateien* sowie bei Bedarf zusätzliche *Ladeparameter* mit Hilfe von `LoadDefinition`-Instanzen übermittelt. Daneben dient die Angabe einer *Funktion als String-Literal* dem Preloader als Sprungadresse, an der nach erfolgreichem Einlesen der Daten die Abarbeitung automatisch fortgesetzt wird.

Das folgende Quellcode-Beispiel veranschaulicht am Beispiel der Klassenbibliothek den Einsatz des Preloaders sowie der Ladefortschrittsanzeige:

```
//Klassen-Bibliothek laden
loadClasses = function()
{
      var mc:MovieClip = createEmptyMovieClip("classLibrary", 1);

      PRELOADER_VIEW.setLoadText("  Lade Klassen...");

      PRELOADER.loadData(new LoadDefinition(mc, //ladendes Obj.
                              PATH + "class_library.swf"),//URL
                        "INIT_CLIP.loadXML"); //Sprung-Funktion
}

//XML-Daten laden
loadXML = function()
{
      module_cfg_loader  = new XMLLoader();
      chapter_cfg_loader = new XMLLoader();
```

```
        …
}
…
_global.INIT_CLIP = this;
```

Quellcode-Bsp. 3.31: Laden der Klassenbibliothek mit anschließendem Sprung zur Funktion `loadXML()`

Für die *init.swf* wurde eine Abspielgeschwindigkeit von 25 Bildern pro Sekunde festgelegt. Da es sich um den obersten Film in der Flash-Hierarchie handelt, ist diese Angabe *maßgebend* für alle untergeordneten SWF-Dateien. Demzufolge werden alle Animationen ebenfalls mit dieser Bildrate wiedergegeben, unabhängig davon, mit welcher Geschwindigkeit sie normalerweise aufwarten.

main.swf (Quellen: *main.fla*, *main.as* sowie *main_exclude.xml*)
Der Hauptfilm ist verantwortlich für das Zeichnen des Hintergrunds, das Laden des Flash-Logos sowie die Bereitstellung der Navigations-Schaltflächen, anhand derer sich die einzelnen Lernschritte aufrufen lassen. Die Konfiguration der einzelnen Elemente erfolgt auf Basis der beiden Objekte `MODULE_INIT` sowie `CHAPTER_INIT`.

Neben verschiedenen Referenzen auf Variablen des Initialisierungsfilms werden innerhalb der *main.swf* weitere Eigenschaften definiert:

Variable	Bedeutung
`_global.MAIN`	Eine globale Referenz auf den Film *main.swf*.
`PAGE_PATH`	Eine `String`-Variable, die den aktuellen Pfad mit den Lernobjekt-spezifischen Daten enthält.
`CHAPTER_COMPLETED`	Eine boolesche Variable, die den Abschlussstatus eines Lernobjekts angibt. Sie ist nur innerhalb einer SCORM-konformen Lernumgebung relevant.
`tooltip`	Ein Objekt der Klasse `Tooltip` zum Anzeigen von Schaltflächen-Hinweisen.
`TOOLTIP_MC`	Container-Movieclip, in dem alle Tooltipps gezeichnet werden.
`navigation_bar`	Die Navigationsleiste des Lernmoduls als Objekt der Klasse `NavigationBar`.
`pages`	Diese Variable enthält die Anzahl der Lernschritte.

`background_mc`
`navigation_mc`
`page_mc`
`logo_mc`
	Container-Movieclips für den Hintergrund, die Navigationsleiste, den Lernschritt sowie das Logo.

PATH PRELOADER PRELOADER_VIEW INFO_PRELOADER INFO_PRELOADER_VIEW CSS_DATA MODULE_INIT CHAPTER_INIT	Referenzen auf gleichnamige Variablen des Initialisierungsfilms.

Tabelle 3.3: Variablen der *main.swf*

Die Abarbeitung des Hauptfilms wird durch einen Aufruf der Funktion `initialize()` über die *init.swf* gestartet. Darin wird der Film zunächst mit der Anweisung `_visible=false` unsichtbar geschaltet, um einen schrittweisen Bildaufbau zu unterdrücken. Anschließend erfolgt die Erstellung der in Tabelle 3.4 erwähnten *Container-Movieclips* mit Hilfe der Methode `createEmptyMovieClip()`, bevor mit der Initialisierung des `Tooltip`-Objektes fortgefahren wird.

Wie bereits erwähnt, werden für alle im Lernmodul vorkommenden Schaltflächen kleine Hinweise, so genannte Tooltipps, angezeigt. Verantwortlich dafür ist in den meisten Fällen die Funktion `showTooltip()` des Hauptfilms, der neben dem eigentlichen Hinweis auch die y-Position sowie die vertikale und horizontale Ausrichtung (ausgehend vom Registrierungspunkt des Movieclip-Containers, in dem die Zeichenvorgänge stattfinden) übergeben werden. Der Tooltipp wird statisch an der x-Position des Mauszeigers generiert, sobald dieser für kurze Zeit reglos innerhalb eines sensitiven Bereichs verharrt. Wird der Cursor wieder herausgefahren oder die Maustaste gedrückt, verschwindet auch der aktuelle Hinweis (durch Aufruf der Funktion `removeTooltip()`). Je nach Schaltfläche werden zustandsbedingt verschiedene Tooltipps dargestellt.

```
showTooltip = function(str:String,
                       y:Number,
                       h_align:String,
                       v_align:String)
{
    var obj:Object = MODULE_INIT.tooltips;

    tooltip.setTooltipByStyleSheet( TOOLTIP_MC,
                                    "tooltip_mc",
                                    str,
                                    … );
```

```
    //statischer Tooltipp, ausgerichtet an der Position
    //des Mauszeigers (x-Parameter = null)
    tooltip.showLockedTooltip( obj.sizes.delay,
                               null,
                               y,
                               h_align,
                               v_align,
                               true);
}
```

Quellcode-Bsp. 3.32: Funktion der *main.swf* zum Anzeigen von Tooltipps

Ausgenommen von dieser Methode der Tooltipp-Darstellung sind die einzelnen Bedienelemente der Abspielsteuerungen für Animationen. Die betreffenden Klassen verfügen selbst über eine entsprechende Funktionalität und erübrigen daher eine externe Lösung.

Im Anschluss an die Initialisierung des `Tooltip`-Objektes wird der Hintergrund der Benutzungsoberfläche mit Hilfe der Klasse `ModulBackground` (siehe Abschnitt 3.1.5) gezeichnet. Der durch die gestalterische Vorgabe spezifizierte waagerechte Farbverlauf (vgl. Abbildung 2.7) wurde dahingehend modifiziert, dass nunmehr *diagonal* von einem dunklen Grauton (links oben) zu einem etwas helleren (rechts unten) übergeblendet wird. Über ein Objekt der Klasse `GradientFillStyle` (siehe Abschnitt 3.1.2) wurde dabei die Verlaufsmatrix derart festgelegt, dass der untere sowie der rechte Rand einen einheitlichen hellgrauen Farbton besitzen. Dieselbe Farbzuweisung erfolgte auch im Fall des Linienmusters sowie für den Hintergrund der *load.html*, so dass Lernmodul und HTML-Seite an den Grenzen fließend ineinander übergehen.

Die Abrundung der Rechtecke, aus denen sich Bühne und Register zusammensetzen, wurde auf einen Wert von 20 Bildpunkten festgelegt. Dieser lässt sich – wie alle Lernobjekt-unabhängigen Konfigurationsdaten – in der Datei *module_init.xml* optional ändern.

Die Überschriften, Hauptthema sowie Unterpunkt, sind neben der Anzahl der Lernschritte und den Tooltipps für Navigationsschaltflächen Bestandteil des `CHAPTER_INIT`-Objektes und werden als Parameter beim Erstellen des Hintergrundes übergeben. Die notwendigen Schriftzeichen für beide dynamischen Textfelder sind in der Datei *main.swf* eingebettet.

Der nächste Abarbeitungsschritt des Hauptfilms ist abhängig von der Anzahl der Lernschritte. Bei einer einzelnen Seite wird direkt das *Logo* geladen, das sich als Datei *logo.swf* ebenfalls im *common*-Verzeichnis befindet. Ist stattdessen eine Unterteilung des SCOs in mehrere Lernschritte vorgesehen, so wird zunächst die Funktion `generateNavigation()` zwecks Erstellens der Navigationsschaltflächen aufgerufen. Darin wird der *main.swf*-Variable `navigationBar` eine neue Instanz der Klasse

`NavigationButtonBar` (siehe Abschnitt 3.1.5) zugewiesen, deren Konstruktor den leeren Movieclip `navigation_mc` sowie dessen neue Koordinaten auf der Bühne übergeben bekommt. Im Anschluss daran werden mit Hilfe der Methode `configButtons()` die einzelnen Schaltflächen als Instanzen der Klasse `NavigationButton` erstellt und eingerichtet.

Das folgende Quellcode-Beispiel befasst sich mit der Funktion `setButtonEventHandlers()`, in der jeweils Referenzen auf die einzelnen Schaltflächeninstanzen gebildet werden, bevor im Anschluss die Behandlung von Mausereignissen folgt:

```
generateNavigation = function()
{
      …
      setButtonEventHandlers();
}
////////////////////////////////////////////////////////

setButtonEventHandlers = function()
{
      var i:Number, mc:NavigationButton;

      var obj:Object = MODULE_INIT.buttons.navigation;

      for(i=1; i<=pages; i++)
      {
            //Referenz auf eine Navigationsschaltfläche bilden
            mc = navigationBar.getButton(i);

            //Eigenschaft, um den Abschlussstatus des SCOs zu prüfen;
            //gibt an, ob eine Seite bereits aufgerufen wurde
            mc.visited = false;

            //Name des Verzeichnisses, in dem sich die
            //Konfigurations- und Textdaten eines Lernschritts
            //befinden
            mc.path = "page" + i;

            mc.onRollOver = mc.onDragOver = function()
```

```
            {
                  //Aufruf der Pseudo-Behandlungsroutine
                  MAIN.navigationBar.onRollOverAction(this);

                  //Tooltipp anzeigen, wenn die Schaltfläche
                  //nicht aktiv ist
                  if(!this.getStatus())
                  {
                        MAIN.showTooltip( … );
                  }
            }
            mc.onRollOut = mc.onDragOut = function()
            {
                  MAIN.navigationBar.onRollOutAction(this);
                  MAIN.removeTooltip();
            }
            mc.onRelease = function()
            {
                  if ( !this.getStatus() )
                  {
                        …
                        MAIN.removeTooltip();
                        MAIN.navigationBar.onReleaseAction(this);

                        //Seiteninhalt laden
                        MAIN.loadPage(this.path)
                  }
            }
      }
      loadLogo()
}
```

Quellcode-Bsp. 3.33: Ereignisbehandlung der Navigationsschaltflächen

Wenn eine Schaltfläche gedrückt und wieder losgelassen wird (`onRelease()`), erfolgt eine Abfrage ihres Zustands, damit die gerade aktive Seite nicht unnötigerweise neu geladen wird. Ebenso verhindert der aktive Zustand die Anzeige eines Tooltipps. Ist die Schaltfläche hingegen inaktiv, so wird der Seiteninhalt durch einen Aufruf der Funktion `loadPage()` des Hauptfilms geladen. Die übergebene Zeichenkette enthält den Namen des entsprechenden

Unterordners[54], in dem sich die Konfigurations- sowie Textdaten eines Lernschritts befinden.

Im Anschluss an die Erstellung der Navigationsleiste folgt die Abarbeitung der *main.swf* wieder einem einheitlichen Schema, indem wie bei einem einzelnen Lernschritt als nächstes das Logo geladen wird. Nach dem Einlesen wird selbiges positioniert und der Hauptfilm wieder sichtbar geschaltet.

Abbildung 3.2 veranschaulicht die Elemente der grafischen Benutzungsoberfläche, wie sie typischerweise durch den Hauptfilm gezeichnet werden, am Beispiel des Lernobjekts *Die Symbole*:

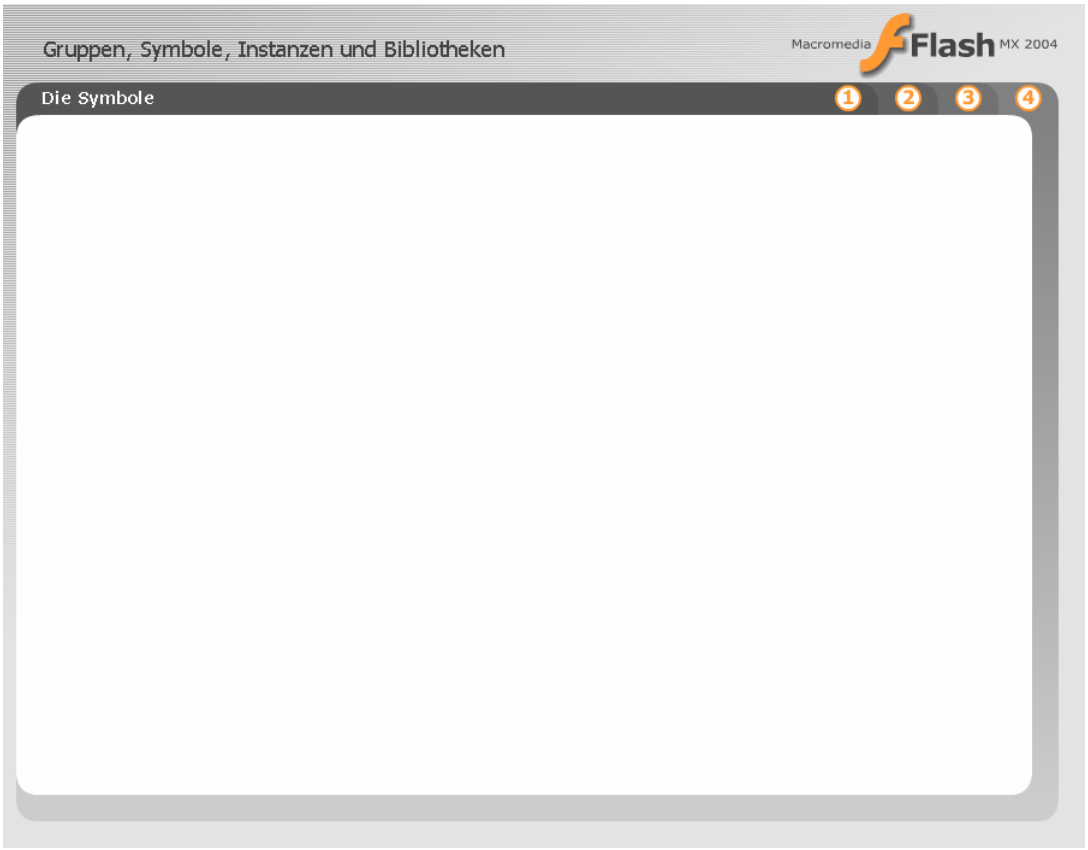

Abbildung 3.4: Hintergrund mit Überschriften, Navigationsschaltflächen und Flash-Logo

Über die Funktion `loadFirstPage()` wird in einem letzten Schritt die Startseite geladen. Sollte es sich um den ersten Aufruf des Lernobjekts handeln oder läuft das Lernmodul nicht in einer SCORM-Umgebung, so werden standardmäßig die Daten des ersten Lernschritts eingelesen. Wurde ein *mehrseitiges* SCO hingegen mindestens einmal gestartet, so präsentiert sich dem Anwender der Lernschritt im Startbildschirm, in dem er die Anwendung zuletzt verließ. Eine entsprechende Information wurde als Variable `LESSON_LOCATION` dynamisch an die *init.swf* übergeben. Gleichzeitig wird auch die betreffende Navigationsschaltfläche in

[54] `"page1"`, `"page2"` usw., bezieht sich auf die Unterordner eines *chapterX*-Verzeichnisses

den *Aktiv*-Zustand versetzt.

Das folgende Quellcode-Beispiel zeigt Ausschnitte aus den beiden Funktionen `loadFirstPage()` sowie `loadPage()`:

```
loadFirstPage = function()
{
	//ein Lernschritt
	if(pages<2)
	{
		loadPage("page1");
	}
	//mehrere Lernschritte
	else
	{
		…
		//Position, an der das Lernobjekt zuletzt verlassen
		//wurde;
		//bei erstmaligem Aufruf oder mangelnder
		//SCORM-Unterstützung ist dieser Wert 1
		var loc:Number = INIT_CLIP.IS_LMS == "true" ?
				parseInt(INIT_CLIP.LESSON_LOCATION) : 1;

		var mc:NavigationButton = navigationBar.getButton(loc);

		navigationBar.onReleaseAction(mc);

		loadPage(mc.path);
	}
}

loadPage = function(path:String)
{
	//Preloader(Views) zurücksetzen
	…
	PAGE_PATH = INIT_CLIP.WORKPATH + path + "/";

	PRELOADER_VIEW.setLoadText("  Lade Seite...");
	PRELOADER.loadData (new LoadDefinition(page_mc,
```

```
                                                    INIT_CLIP.PATH+"page.swf"),
                                "MAIN.page_mc.init");
}
```

Quellcode-Bsp. 3.34: Die Funktionen `loadFirstPage()` und `loadPage()`

Darüber hinaus enthält die Datei *main.swf* verschiedene Funktionen, um Lernerdaten an ein serverseitiges LMS zu übermitteln und den Abschlussstatus des SCOs zu ermitteln. Nähere Informationen dazu sind dem Abschnitt 3.4.1 zu entnehmen.

page.swf (Quellen: *page.fla*, *page.as* sowie *page_exclude.xml*)
Der Film *page.swf* wird aus der `loadPage()`-Funktion des Hauptfilms heraus geladen und ist verantwortlich für die Darstellung des Seiteninhalts. Eine Hauptfunktion ist demzufolge auch die Verarbeitung der Konfigurationsdaten aus der Datei *page_init.xml*.

Neben verschiedenen Referenzvariablen enthält der Film weitere Eigenschaften, die in folgender Tabelle kurz erläutert werden:

Variable	**Bedeutung**
`_global.PAGE`	Eine globale Referenz auf den Film *page.swf*.
`page_cfg_loader`	Ein Objekt der Klasse `XMLLoader` zum Laden und Aufbereiten der Konfigurationsdatei *page_init.xml*.
`page_init`	Ein Objekt, das die eingelesenen und aufbereiteten Daten der Konfigurationsdatei *page_init.xml* enthält.
`mainpage_mc` `animationpage_mc`	Übergeordnete Movieclips, die sämtliche Daten des Haupt- bzw. des Animationsbildschirms enthalten.
`screenshot_mcs` `labeled_mcs`	Arrays, die Container-Movieclips für Bild- und Textschaltflächen enthalten. Die Anzahl richtet sich jeweils nach den entsprechenden Elementen innerhalb der *page_init.xml*.

`page_mc`
`simpleobjects_mc`
`symbols_mc`
`animation_mc`
`exitsymbol_mc`
	Container-Movieclips, die an entsprechende Klasseninstanzen übergeben werden.

`exit_button`	Die *Schließen*-Schaltfläche des Animationsbildschirms als Objekt der Klasse `SymbolButton`.
`modulePage`	Ein Objekt der Klasse `ModulePage`.
`moduleAnimationPage`	Ein Objekt der Klasse `ModuleAnimationPage`.

`screenshot_container` `labeled_container`	Arrays für Container-Klassen zum Generieren der Screenshot- sowie Textschaltflächen-Instanzen.
`symbol_bar`	Ein Objekt der Klasse `SymbolButtonBar` zur Darstellung der Symbolschaltflächen.
`symbol_IDs`	Ein Array mit den Verknüpfungsnamen der vier Symbol-Movieclips.
`symbol_nodeNames`	Ein Array mit den Element-Bezeichnungen innerhalb der *page_init.xml*, unter denen der Inhalt definiert wurde.

`pageX`
`pageY`
`pageW`
`pageH`
	Die Koordinaten und Dimensionen einer Seite (entsprechen dem weißen Bereich der Benutzungs-oberfläche).

`infoW` `infoH`	Die Abmessungen des Zusatzbereichs.
`*_path`	Sämtliche `_path`-Variablen enthalten Pfadangaben zu den verschiedenen Ressourcen (Bilder, Animationen etc.).
`btninf_cur_loading`	In der Variable werden die aktuellen Informationen gespeichert, anhand deren der Inhalt für eine Schaltfläche geladen wird.
`cur_anim_url`	Die Variable speichert die URL einer geladenen Animation.
`PATH`	Referenz auf die *main.swf*-Eigenschaft `PAGE_PATH`.

`PRELOADER`
`PRELOADER_VIEW`
`INFO_PRELOADER`
`INFO_PRELOADER_VIEW`
`CSS_FDATA`
`MODULE_INIT`
	Referenzen auf gleichnamige Variablen des Initialisierungsfilms.

Tabelle 3.4: Variablen der *page.swf*

Nachdem die Datei *page.swf* vollständig geladen wurde und zur Abarbeitung bereitsteht, springt der Preloader automatisch zur Funktion `initialize()`. Ähnlich wie bei der gleichnamigen Funktion des Hauptfilms werden darin zunächst die Seite unsichtbar geschaltet sowie die in jedem Fall benötigten Container-Movieclips[55] erstellt. Im Anschluss daran liest

[55] bezieht sich in diesem Fall auf `mainpage_mc` und `page_mc`

die Datei *page.swf* die Konfigurationsdaten des Lernschritts[56] aus und speichert die generierte Objektstruktur in der Variable `page_init`.

Die Basiselemente einer Seite – Haupt- und Zusatzbereich sowie die dazugehörigen Textfelder – lassen sich mit Hilfe der Klasse `ModulPage` erstellen, von der eine neue Instanz für die Variable `modulPage` gebildet wird. Der Zusatzbereich wird nur dargestellt, wenn innerhalb der Datei *page_init.xml* die Elemente `info_width` oder `info_height` an entsprechender Stelle definiert wurden (siehe auch Elementbeschreibung im entsprechenden Anhang). Letzteres Element lässt die Möglichkeit offen, den Zusatzbereich alternativ im unteren Abschnitt der Seite über die gesamte Breite darzustellen. Von dieser Variante wurde jedoch bei den entwickelten Lernobjekten kein Gebrauch gemacht.

Im folgenden Quellcode-Beispiel, das Ausschnitte aus der betreffenden Funktion `drawBackground()` zeigt, sollen die erläuterten Schritte noch einmal verdeutlicht werden:

```
drawBackground = function()
{
    page_init = page_cfg_loader.getObject();
    //Referenzen
    …
    //Bühnenkoordinaten und -ausmaße errechnen
    …
    //ein leeres Textfeld für den Hauptbereich wird über
    //den Konstruktor der Klasse "ModulPage" erstellt
    modulePage = new ModulePage(…);

    //Breite bzw. Höhe des Zusatzbereichs (es sollte nur
    //ein Element definiert werden)
    infoW = page_init.common.sizes.info_width;
    infoH = page_init.common.sizes.info_height;

    //Zusatzbereich generieren
    if (infoW>0 || infoH>0)
    {
      //bei Angabe von "info_width" wird der Bereich
      //rechts angelegt, sonst unten
      var align:String  = infoW>0 ? "right" : "bottom";
      var size:Number   = infoW>0 ? infoW   : infoH;
```

[56] enthalten in der Datei *page_init.xml*

```
        //Linienmuster
        if(o_obj.scanlineInfofield.valueOf())
        {
            modulePage.makeLinePatternSecField(…);
        }
        //vollständig gefüllte Fläche
        else
        {
            modulePage.makeFilledSecField(…);
        }

        //Preloader für Zusatzbereich mittig ausrichten
        …
    }
    checkForButtonGraphics();
}
```

Quellcode-Bsp. 3.35: Haupt- und Zusatzbereich generieren

Im Zusatzbereich wird in der Standardeinstellung das dezente Linienmuster des Hintergrunds fortgeführt. Dies lässt sich bei Bedarf jedoch über das Element `scanlineInfofield` in der Datei *module_init.xml* abschalten und durch eine einheitliche Fläche ersetzen. Dadurch kann die Lesbarkeit geringfügig erhöht werden.

Im nächsten Schritt wird geprüft, welche Schaltflächenarten der aktuelle Lernschritt beinhaltet, um je nach Bedarf leere Container-Movieclips zu erstellen. Darüber hinaus erfordern Bild- sowie Symbolschaltflächen das Laden zusätzlicher Flash-Filme, in denen die benötigten Grafiken eingebettet wurden. Dabei handelt es sich um die Dateien *info_buttons_symbols.swf* (Symbole im Zusatzbereich), *anim_screen_exit_symbol.swf* (*Schließen*-Symbol des Animationsbildschirms) oder den entsprechenden Hintergrund für Bildschaltflächen aus dem Ordner *ressources/screenshots*. Die Grafiken sind jeweils in der Bibliothek als Movieclips abgelegt und werden in den verarbeitenden Klassen über ihren Verknüpfungsbezeichner angesprochen.

```
checkForButtonGraphics = function()
{
  //Array mit LoadDefinition-Objekten für Schaltflächen-Grafiken
  var mcsToLoad:Array = new Array();
```

```
//Zusatzfeld wurde gebneriert
if (modulePage.getSecondaryTextField() != undefined)
{
    //Clip mit Schaltflächen-Symbolen (für Zusatzbereich) laden
    if (page_cfg_loader.checkForString("<info_")
    {
          …
    }
    //Container-Clip für Animationsbildschirm erzeugen
    //und SWF-Datei mit 'Schließen'-Symbol laden
    if (page_cfg_loader.checkForString("<info_anim>"))
    {
          …
    }
}
var i:Number;

//Screenshot-Clips laden
if (page_init.screenshot_buttons)
{
    screenshot_mcs = new Array();

    //Objekt in Array umwandeln >> falls lediglich ein
    //<screenshot_buttons>-Element definiert wurde, erstellt
    //die Klasse XMLLoader automatisch den Typ 'Object', nicht
    //'Array'
    if( !(page_init.screenshot_buttons instanceof Array) )
    {
          page_init.screenshot_buttons = new Array
                                        (page_init.screenshot_buttons);
    }

    //für jedes definierte <screenshot_buttons>-Element wird ein
    //Container-Clip erstellt und die SWF-Datei mit dem
    //Hintergrund-Bild geladen
    for (i=0; i<page_init.screenshot_buttons.length; i++)
    {
          screenshot_mcs.push (mainpage_mc.createEmptyMovieClip
```

```
                                        ("screenshot_mc" + i,
                                         mainpage_mc.getNextHighestDepth()));

            mcsToLoad.push(new LoadDefinition(screenshot_mcs[i],
                screenshot_path + page_init.screenshot_buttons[i].link));
        }
    }

    if (page_init.labeled_buttons)
    {
        labeled_mcs = new Array();

        if( !(page_init.labeled_buttons instanceof Array) )
        {
              page_init.labeled_buttons = new Array
                                            (page_init.labeled_buttons);
        }
        for (i=0; i<page_init.labeled_buttons.length; i++)
        {
                  labeled_mcs.push (mainpage_mc.createEmptyMovieClip
                                        ("labeled_mc" + i,
                                        mainpage_mc.getNextHighestDepth()));
        }
    }
    //Schaltflächen-Grafiken laden
    …
}
```

Quellcode-Bsp. 3.36: Container-Clips generieren und Schaltflächen-Grafiken laden

Anhand der erstellten Container-Clips wird ermittelt, welche Schaltflächen zu initialisieren sind. Da die strukturellen Unterschiede zwischen Bild- und Textschaltflächen eher marginaler Natur sind, lassen sich beide problemlos mit denselben Funktionen, `generateMainButtons()` sowie `makeMainButton()`, konfigurieren. In Abhängigkeit vom Schaltflächentyp werden entweder `ScreenshotButtonContainer`- oder `LabeledButtonContainer`-Instanzen erstellt und den betreffenden Arrays (vgl. Tabelle 3.4) hinzugefügt. Alle Schaltflächen eines Containers besitzen mindestens einen *Abstand von fünf Bildpunkten* zueinander. Dieser Wert lässt sich jedoch im Gegensatz zu den meisten anderen (z.B. der Abrundung der Schaltflächen an den Ecken) nicht über die Datei

module_init.xml einstellen, sondern resultiert aus einer entsprechenden Koordinatendefinition[57] innerhalb der betreffenden *page_init.xml*-Dateien. Darüber hinaus besteht für einen kompletten Schaltflächen-Container die Möglichkeit, diesen automatisch mittig im Hauptfeld auszurichten, indem die Koordinate der betreffenden Achse nicht spezifiziert wird. Zum Zentrieren werden anschließend die Gesamtbreite bzw. -höhe aller dazugehörigen Schaltflächen errechnet, um daraus mit den Funktionen `centerX()` bzw. `centerY()` die neue Position zu berechnen (vgl. nachfolgendes Quellcode-Beispiel).

```
generateMainButtons = function(areScreenshotButtons:Boolean,
                                        pos:Number)
{
      var scr:Boolean = areScreenshotButtons;
      …
      if(scr)
      {
            screenshot_container.push(new ScreenshotButtonContainer (…));
      }
      else
      {
            labeled_container.push(new LabeledButtonContainer(…));
      }
      var minmax:Object = new Object();
      var minX:Number, maxX:Number, minY:Number, maxY:Number;

      var x:Number = button_obj.coordinates.x;
      var y:Number = button_obj.coordinates.y;

      var o:Object;

      for (var i in button_obj.buttons)
      {
            o = button_obj.buttons[i]

            if(x==undefined)
            {
```

[57] Die Koordinaten einer Schaltfläche werden relativ zu denen des übergeordneten Container-Movieclips angegeben.

```
                    //minimale und maximale x-Position
                    //der aktuellen Schaltfläche
                    minX = o.coordinates.x;
                    maxX = o.coordinates.x + o.sizes.w;

                    //x-Grenzen anpassen
                    if(minmax.minX==undefined)
                    {
                          minmax.minX = minX;
                          minmax.maxX = maxX;
                    }
                    else if(minX < minmax.minX)
                    {
                          minmax.minX = minX;
                    }
                    else if(maxX >minmax.maxX)
                    {
                          minmax.maxX = maxX;
                    }
              }
              if(y==undefined){ … }

              makeMainButton(o, i, pos, button_obj.id);
        }
        //wenn x oder y nicht definiert wurden, wird der Begrenzungsrahmen
        //der Schaltflächen im Hauptfeld jeweils zentriert
        if( x==undefined ) x = centerX(minmax.maxX - minmax.minX,
                                       true);
        if( y==undefined ) y = centerY(minmax.maxY - minmax.minY,
                                       true);

        var ctn:Object = scr ? screenshot_container[pos] :
                               labeled_container[pos];

        ctn.setPos(x,y);
}
```

Quellcode-Bsp. 3.37: Bild- und Textschaltflächen generieren und ggf. mittig ausrichten

Die folgende Abbildung soll die bisherigen Schritte veranschaulichen, ist in dieser Form jedoch aufgrund des Sichtbarkeitsstatus der *init.swf* im Normalfall nicht wahrnehmbar:

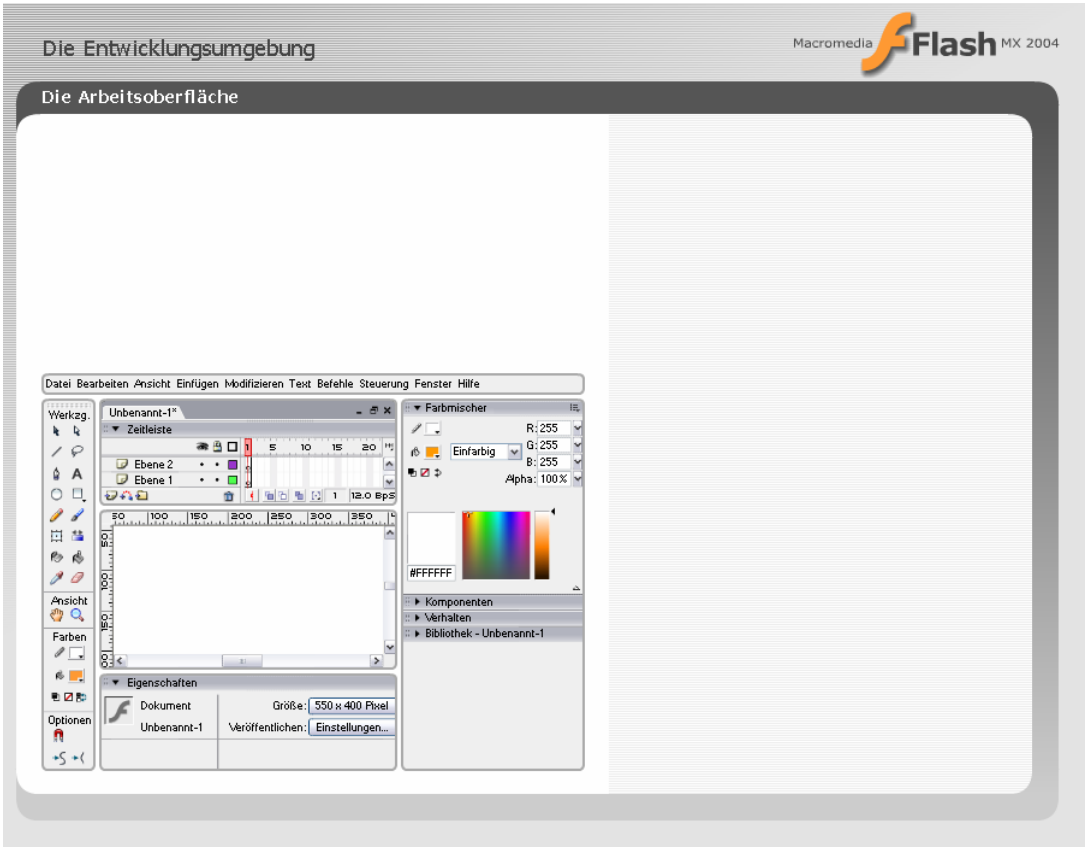

Abbildung 3.5: Zusatzbereich und Bildschaltflächen (auf der x-Achse zentriert)

Die Symbolschaltflächen des Zusatzbereichs werden mit Hilfe der Klasse `SymbolButtonBar` erzeugt. Ihre Sichtbarkeit wird innerhalb der *page_init.xml* anhand von `info`-Elementen[58] festgelegt und über die Funktion `activateSymbolButtons()` gesteuert. Diese wird in der `onRelease()`-Routine einer jeden Bild- bzw. Textschaltfläche sowie beim Laden der Hauptinformationen aufgerufen. Die Informationen der Symbolschaltflächen stellen demnach *Zusatzwissen* zum gerade aktiven Schaltflächenelement bzw. zum Text des Hauptfeldes dar. Das nachfolgende Quellcode-Beispiel zeigt die soeben erwähnte Funktion:

[58] Ein `info`-Objekt definiert den Inhalt des Zusatzbereichs und enthält unter Umständen die Elemente `info_anim`, `info_key`, `info_hint` und/oder `info_plus`, deren Präsenz die Sichtbarkeit der Symbolschaltflächen bestimmt.

```
activateSymbolButtons = function(info:Object)
{
	var mc:SymbolButton;

	for (var i=0; i< symbol_IDs.length; i++)
	{
		//in Abhängigkeit vorhandener "info_[symbolname]"-Elemente
		//werden entsprechend Symbole dargestellt oder verdeckt
		if( info[symbol_nodeNames[i]] )
		{
			mc = symbol_bar.getButton(symbol_IDs[i]);
			//Information zuweisen, die beim Aktivieren
			//bzw. Deaktivieren geladen werden
			mc.own_info	= info[symbol_nodeNames[i]];
			mc.common_info	= info;

			symbol_bar.setVisibilityByButton(mc, true);
		}
		else
		{
			symbol_bar.setVisibilityByName(symbol_IDs[i],
							false);
		}
	}
	//Schaltflächen zurücksetzen
	symbol_bar.setAllInactive();
}
```

Quellcode-Bsp. 3.38: Steuerung der Sichtbarkeit für die Symbolschaltflächen im Zusatzbereich

Falls innerhalb der *page_init.xml* das Element `info_anim` aufgefunden wird (vgl. Quellcode-Beispiel 3.32), erfolgt die Erstellung eines – zunächst nicht sichtbaren – Animationsbildschirms. Dieser besteht hauptsächlich aus einem Objekt der Klasse `ModuleAnimationPage`, das der gleichnamigen *page.swf*-Variable zugewiesen wird. Der Hintergrund entspricht visuell dem Design des Zusatzbereichs, erstreckt sich jedoch über die gesamte Seitenbreite. Ebenso wurden das grafische Erscheinungsbild sowie das Schaltflächenverhalten der Abspielsteuerung dem umgebenden Design des Lernmoduls angepasst. Die Größe des Wiedergabefensters wurde auf 640 mal 410 Bildpunkte festgelegt, lässt sich jedoch flexibel über die Datei *module_init.xml* modifizieren. Darüber hinaus enthält der

Animationsbildschirm ein separates *Schließen*-Symbol, gebildet über eine Instanz der Klasse `SymbolButton`. Es wurde an derselben Stelle wie das Symbol für zusätzliche Informationen platziert und dient der Rückkehr zur Hauptseite.

Die nachfolgende Abbildung stellt den Animationsbildschirm dar, wie er mit Hilfe der Funktion `generateAnimationPage()` erzeugt wird:

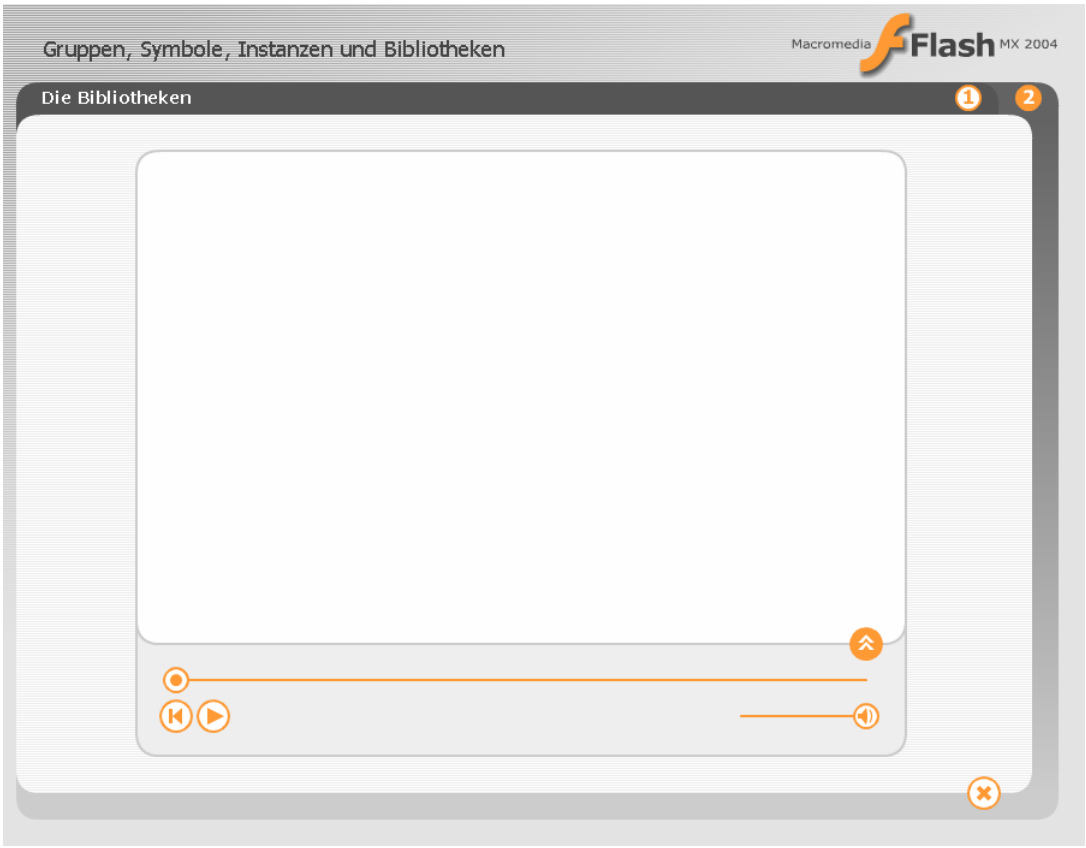

Abbildung 3.6: Der Animationsbildschirm

Der Animationsbildschirm wird über die entsprechende Symbolschaltfläche des Zusatzbereichs aufgerufen. Ob ein so genanntes *Streaming*[59] des Films stattfindet, hängt von der Eigenschaft `streaming` der *module_init.xml* ab. Besitzt dieser den Wert `true`, so beginnt der Abspielvorgang der Animation, sobald genügend Daten gepuffert wurden. Anderenfalls wird der abzuspielende Film mit Hilfe des `PRELOADER`-Objekts geladen. Der Nutzer erhält zusätzlich zur normalen Prozentanzeige genaue Informationen darüber, wie groß der gesamte Speicherbedarf der SWF-Datei ist und wie viele Kilobytes bereits eingelesen wurden. Dies erschien sinnvoll, da gerade Nutzer langsamerer Internetverbindungen mit erhöhten Wartezeiten rechnen müssen und auf diese Weise leichter entscheiden können, ob

[59] Beim Streaming wird ein Film bereits während des Herunterladens abgespielt, sobald genügend Daten gepuffert wurden.

sie den Ladevorgang abbrechen – bewegt sich der Dateiumfang der Filme doch teilweise im Megabyte-Bereich.

Standardmäßig wurde das Streaming ausgeschaltet, um auch Anwender zu berücksichtigen, die über keine breitbandige Internetanbindung verfügen. Bei diesen hätte die Option in der Praxis meist einen unstetigen Abspielvorgang und damit ein beeinträchtigtes Lernerlebnis zur Folge.

Das folgende Quellcode-Beispiel zeigt Ausschnitte aus der Funktion `showAnimationPage()`, die beim Laden des Animationsbildschirms aufgerufen wird:

```
showAnimationPage = function(info:Object)
{
      //Referenz auf das ComplexPlayer-Objekt bilden
      var player:ComplexPlayer = moduleAnimationPage.getPlayer();

      //Sichtbarkeit der Bildschirme wechseln
      mainpage_mc._visible          = false;
      animationpage_mc._visible     = true;
      …
      //Animation wird nur geladen, falls sich der Inhalt geändert hat
      //bzw. kein Film geladen ist
      if(!player.isContentLoaded() || info.link != cur_anim_url)
      {
            cur_anim_url = info.link;
            player.createEmptyContent();//leeren Container erzeugen

            if(MODULE_INIT.other.streaming.valueOf())//Streaming
            {
                  player.loadContent(big_anim_path + info.link, true);
            }
            else
            {
                  PRELOADER_VIEW.showKB(true); //Kilobytes anzeigen
                  …
                  player.stopDuringLoad();     //kein Streaming

                  PRELOADER.loadData(new LoadDefinition
                                          (player.getContent(),
                                          big_anim_path+info.link),
```

```
                                        "PAGE.afterAnimationIsLoaded");
            }
      }
}
```

Quellcode-Bsp. 3.39: Anzeige des Animationsbildschirms und Laden des betreffenden Flashfilms

Das Informationsobjekt einer Schaltfläche wird während des Ladevorgangs temporär in der Variable `btninf_cur_loading` gespeichert. Erfolgt ein Abbruch durch den Aufruf des Animationsbildschirms, so wird beim Verlassen desselben über die *Schließen*-Schaltfläche der betreffende Inhalt automatisch neu geladen:

```
//Aufruf durch den 'Schließen'-Button
showMainPage = function()
{
      …
      if(btninf_cur_loading) loadButtonContent(btninf_cur_loading);
}
…
loadButtonContent = function(info:Object)
{
      var a:Array = getLoadDefinitions(info, false);

      if (a.length>0)
      {
            //Informationsobjekt wird temporär gespeichert und nach
            //einem erfolgreichen Ladevorgang wieder gelöscht
            btninf_cur_loading = info;
            …
      }
}
```

Quellcode-Bsp. 3.40: Wiederaufnahme eines abgebrochenen Ladevorgangs

Sämtliche Schaltflächen eines Lernschritts, ausgenommen derer der Abspielsteuerung, bekommen dieselben Behandlungsroutinen zur Verarbeitung von Mausereignissen zugewiesen. Die je nach Typ unterschiedliche Funktionalität ergibt sich aus speziellen `if`-Abfragen, die direkt in den Routinen platziert sind. Dies soll am Beispiel der `onRelease()`-Funktion veranschaulicht werden:

```
setButtonEventHandlers = function(mc:MovieClip)
{
	…
	mc.onRelease = function()
	{
		…
		//'Schließen'-Schaltfläche
		if(this == PAGE.exit_button)
		{
			PAGE.showMainPage();
		}
		//(Zusatzinfo)Symbol-, Bild- und Textschaltlächen
		else
		{
			var info:Object;

			if (this == this.container.getActiveButton())
			{
				info = this.common_info;
				PAGE.showTooltip(…);
			}
			else
			{
				info = this.own_info;
				//'Animation'-Schaltfläche
				if(this == symbol_bar.getButton(symbol_IDs[0]))
				{
					PAGE.showAnimationPage(info);
					return;
				}
				PAGE.showTooltip(…);
			}

			this.container.onReleaseAction(this);
			//Bild- und Textschaltflächen
			if(!(this instanceof SymbolButton))
			{
				PAGE.activateSymbolButtons(info);
```

```
                        PAGE.setContainersBack(this.container);
                    }
                    PAGE.loadButtonContent(info);
                }
            }
}
```

Quellcode-Bsp. 3.41: Die `onRelease()`-Behandlungsroutine

Nachdem alle Schaltflächen bzw. der Animationsbildschirm initialisiert wurden, folgt im nächsten Schritt das Laden der Hauptinhalte, bevor die Seite endgültig zur Anzeige gebracht wird. Die einzulesenden Informationen (Texte, Grafiken, Interaktionen etc.) werden mit Hilfe der Funktion `getLoadDefinitions()`[60] ermittelt und als ein Array von `LoadDefinition`-Objekten zurückgeliefert. Über den an die Funktion übergebenen booleschen Parameter `isMainField` lässt sich festlegen, welchem Textfeld[61] die zu ladenden HTML-Daten später zugeordnet werden sollen. Wurde für ein Informationsobjekt kein Textlink spezifiziert, so wird das betreffende Textfeld gelöscht.

Weitere Ressourcen, wie Grafiken, Interaktionen, Beispielanimationen oder Downloadmöglichkeiten, werden mit Hilfe der Klasse `SimplePageObjects` hinzugefügt (vgl. Abschnitt 3.1.5). Um welche es sich letztlich handelt, entscheiden die unterhalb eines `special`-Elements in der *page_init.xml* eingetragenen Daten. Die Auswertung in der *page.swf* über zwei `for`- sowie eine `switch`-Anweisung wird im folgenden Quellcode-Beispiel dargestellt:

```
loadButtonContent = function(info:Object)
{
  …
  // Ressourcen
  if(info.special)
  {
      …
      var objects:SimplePageObjects = new SimplePageObjects(content_mc);
      //Ressourcentypen bestimmen
      for (var j in info.special)
      {
            type = info.special[j];
```

[60] Über die Funktion werden auch die einzulesenden Daten für Schaltflächen ermittelt.

[61] bezieht sich auf die Textfelder im Haupt- und Zusatzbereich

```
            //Ressourcendefinition verarbeiten
            for (var i in type)
            {
              element = type[i];
              …
              switch(j)
              {
                  case "pictures"   : objects.createPicture(…);
                                      break;

                  case "downloads"  : objects.createDownload(…);
                                      break;

                  case "interactions" : …
                                        objects.createInteraction(…);
                                        break;

                  case "animations" : …
                                      objects.createAnimation(…);
                                      break;
              }
            }
      }
      //LoadDefinition-Arrays zusammenfügen und zur Verarbeitung durch
      //den Preloader zurückgeben
      return filesToLoad.concat(objects.getLoadDefinitions());
  }
  …
}
```

Quellcode-Bsp. 3.42: Die Auswertung des `special`-Elements

3.2.3 Schriftarten

In Flash existieren grundsätzlich zwei Möglichkeiten zur Darstellung von Schrift: Einerseits durch die Verwendung von Schriftarten, die auf dem System des Nutzers installiert sind (so genannte *System-* oder *Geräteschriftarten*), andererseits durch das *Einbetten* der benötigten Schriftzeichen in einen Flash-Film. Die erste Variante bietet den Vorteil einer geringeren Dateigröße, resultiert jedoch möglicherweise in einer veränderten und unerwünschten Darstellung, wenn die betreffende Schriftart auf dem Zielrechner nicht zur Verfügung steht

und durch den Flash-Player automatisch durch eine andere ersetzt werden muss. Das Einbetten von Schriftkonturen hingegen gewährleistet eine einheitliche Darstellung in Unabhängigkeit vom verwendeten System. Es ist jedoch zu beachten, dass für jeden verwendeten Stil (normal, fettgedruckt, kursiv, fettgedruckt und kursiv) auch separate Zeichen in einen Flash-Film integriert werden müssen.

War es bei früheren Flash-Versionen lediglich möglich, Schriftzeichen in vektorieller Form einzubetten, so wurde mit Flash MX 2004 die Option eingeführt, stattdessen eine *pixelbasierte Variante* (in Flash auch *Bitmaptext* genannt) gewünschter Größe zu erstellen. Auf diese Weise werden die Zeichen dem Pixelraster des Bildschirms angepasst, was insbesondere bei geringen Schriftgrößen zu einer besseren Lesbarkeit führt, da keine zusätzliche Kantenglättung (Anti-Aliasing) erfolgt. Aus dieser Variante resultieren jedoch auch Zeichenabstände jenseits einer optimalen Darstellung. In Verbindung mit *dynamischen Textfeldern*, wie sie im Lernmodul zum Einsatz kommen, ergibt sich unter Flash MX 2004 ein weiterer Nachteil: Die automatische Unterschneidung der Schriftzeichen, auch als *Kerning* bezeichnet, wird abgeschaltet. Dies führt insbesondere bei der ursprünglich für das Lernmodul vorgesehenen Schriftart *Verdana* zu optisch deutlich wahrnehmbaren Unregelmäßigkeiten bei bestimmten Zeichenabständen. Aus diesem Grund wurde auf die ähnliche Schriftart *Tahoma* ausgewichen, deren Schriftbild unter diesen Bedingungen homogener wirkt. Die folgende Tabelle enthält Bildschirmfotos der Schriftdarstellung im Flash-Player und soll diesen Umstand nochmals verdeutlichen:

	Tahoma	***Verdana***
Geräteschriftart	Ein Flash-Beispieltext	Ein Flash-Beispieltext
eingebettet, normal	Ein Flash-Beispieltext	Ein Flash-Beispieltext
eingebettet, fettgedruckt	**Ein Flash-Beispieltext**	**Ein Flash-Beispieltext**

Tabelle 3.5: Textdarstellung ohne und mit eingebetteten Schriftzeichen

In Flash 8 *für Windows* wurde das Manko der fehlenden Unterschneidung bei dynamischen Textfeldern beseitigt. Das Kerning lässt sich nunmehr im Eigenschafteninspektor oder mit Hilfe der `Textformat`-Klasse einstellen. Die auf diese Weise erzeugten Flash-Filme können auch auf Flash-Playern dargestellt werden, die nicht unter Windows laufen. Ebenso lässt sich mit der aktuellen Version die Kantenglättung für eingebettete Schriftzeichen differenziert einstellen. Da zum Zeitpunkt der Entwicklung des Lernmoduls lediglich die Vorgängerversion von Flash zur Verfügung stand, konnte auf diese Möglichkeiten nicht zurückgegriffen werden.

Neben der *Tahoma* wurden außerdem Zeichen der Schriftarten *Courier New* sowie *Wingdings* verwendet. Zudem basieren die Zahlen der Navigationsschaltflächen auf *vektoriell* eingebetteten *Tahoma*-Schriftzeichen.

Die Schriftart *Wingdings* lässt sich im Flash-Player ausschließlich über die Methode des Einbettens darstellen. Dies hatte auch zur Folge, dass die anderen Schriftarten, obwohl sie auf den meisten Systemen installiert sind, ebenfalls mit integriert werden mussten. Dahinter verbirgt sich die Tatsache, dass innerhalb eines Textfeld-Objekts *entweder* Geräteschriftarten *oder* eingebettete Schriftarten benutzt werden können, jedoch nicht beide Methoden gleichzeitig. Da der Textinhalt des Lernmoduls stellenweise eine Kombination mehrerer Schriftarten erfordert, wurde auf die Nutzung von Systemschriftarten komplett verzichtet.

Das Einbetten der Schriftkonturen erfolgte anhand von Textfeld-Objekten, die – integriert in Movieclips – in der Bibliothek des betreffenden Flash-Dokuments platziert wurden. Auf diese Weise lassen sich nur die jeweils benötigten Glyphen einbinden, wodurch die Dateigröße geringer ausfällt im Vergleich zur alternativen Variante mit so genannten *Schriftartsymbolen*, bei denen der komplette Zeichensatz integriert wird. Um derart eingebettete Alias-Text-Schriftkonturen mittels ActionScript zur Laufzeit auf Textinhalte von dynamischen Textfeldern oder Eingabetextfeldern anzuwenden, beispielsweise über `TextFormat`- oder `StyleSheet`-Objekte, reicht jedoch die normale Bezeichnung der Schriftart nicht aus. Stattdessen muss der Name verwendet werden, unter dem der Zeichensatz *intern* in Flash registriert wurde. Dieser setzt sich wie folgt zusammen:

***Schriftartname_Schriftgröße*pt_st**

Eine Alias-Text-Variante der *Tahoma* in der Größe 13 ließe sich demnach über die Bezeichnung *Tahoma_13pt_st* in ActionScript verwenden. Eingebettete Zeichen von Schriftartsymbolen lassen sich hingegen einfach über einen *Verknüpfungsbezeichner* verwenden, der in der Dokument-Bibliothek dem Symbol zugewiesen wurde.

3.3 Konfiguration des Lernmoduls und Erstellung der Ressourcen

3.3.1 Definition von Variablen auf XML-Basis

Flash bietet mit der `XML`-Klasse eine Möglichkeit, im XML-Format (vgl. Abschnitt 1.1.6) gespeicherte Daten zur Laufzeit einer Anwendung zu laden. Die enthaltenen Strukturen werden ausgelesen und in einer der Programmumgebung verständlichen Form zur Verfügung gestellt. Dieser Prozess wird im Allgemeinen auch mit dem Begriff *Parsing* umschrieben.

Auf einzelne Elemente und Attribute der XML-Struktur lässt sich mittels Klasseneigenschaften, die von `XMLNode` geerbt wurden, zugreifen. Dazu gehören beispielsweise `firstChild`[62], `childNodes`[63], `nodeValue`[64] sowie `attributes`[65]. Alle Elemente

[62] erstes Element eines übergeordneten Knoten

eines XML-Baums werden ebenfalls durch Objekte des Datentyps `XMLNode` repräsentiert und lassen sich somit auf identische Weise auslesen.

Die `XML`-Klasse bietet auch die Möglichkeit, Dokumenttyp-Definitionen (DTD) zu einzulesen. Selbige werden in der `String`-Eigenschaft `docTypeDecl` gespeichert. Es handelt sich bei dem in Flash implementierten jedoch nicht um einen validierenden Parser, das heißt, eine Überprüfung der XML-Struktur auf Schema-Konformität findet nicht statt. Aus diesem Grund wurde bei der Erstellung der einzelnen XML-Dateien für das Lernmodul auf eine DTD verzichtet.

Beim Erzeugen der Konfigurationsdaten kam die lizenzfreie Software *Microsoft XML Notepad 1.5 Beta* zum Einsatz. Bei dieser handelt es sich um einen kleinen Editor, mit dem einfache XML-Dateien schnell und unkompliziert erstellt und bearbeitet werden können (siehe Abbildung 3.7).

Ein Lernobjekt des Moduls greift prinzipiell auf vier XML-Dateien zu, deren innere Struktur jeweils ausschließlich aus Elementen zusammensetzt und auf die im Folgenden näher eingegangen werden soll. Anhand dieser Dateien sind individuelle Anpassungen an geänderte Rahmenbedingungen schnell und ohne spezielle Flashkenntnisse möglich, ebenso wie das Hinzufügen neuer Lernobjekte. Geladen wurden sämtliche Konfigurationen mit der bereits unter Abschnitt 3.1.5 erwähnten, von `XML` abgeleiteten Klasse `XMLLoader`. Eine ausfühliche Referenz zu den jeweils definierbaren Elementen ist dieser Arbeit als Anhang C beigefügt.

preloader_view_init.xml

Über diese Datei wird die visuelle Darstellung der beiden `PreloaderView`-Instanzen eingestellt, die zur Anzeige des Ladefortschritts verwendet werden. Es lassen sich verschiedene Farbwerte, Kreisradien sowie Textfeldabmessungen und -koordinaten definieren.

module_init.xml

Die in dieser Datei spezifizierten Variablen dienen der grundlegenden Initialisierung des Lernmoduls bzw. der Benutzungsoberflächenelemente. So erfolgt einerseits beispielsweise der Zugriff auf die verschiedenen Ressourcen anhand der unterhalb des `directories`-Elements festgelegten relativen Pfade, andererseits lassen sich die unterschiedlichen interaktiven Schaltflächenelemente individuell und unabhängig voneinander über entsprechende Kindelemente von `buttons` konfigurieren. Die folgende Abbildung zeigt die geöffnete Datei im Editor:

---

[63] ein `Array`-Objekt mit allen Kindelementen

[64] Wert eines Elements als `String`

[65] eine `Object`-Instanz mit allen Attributen eines Elements

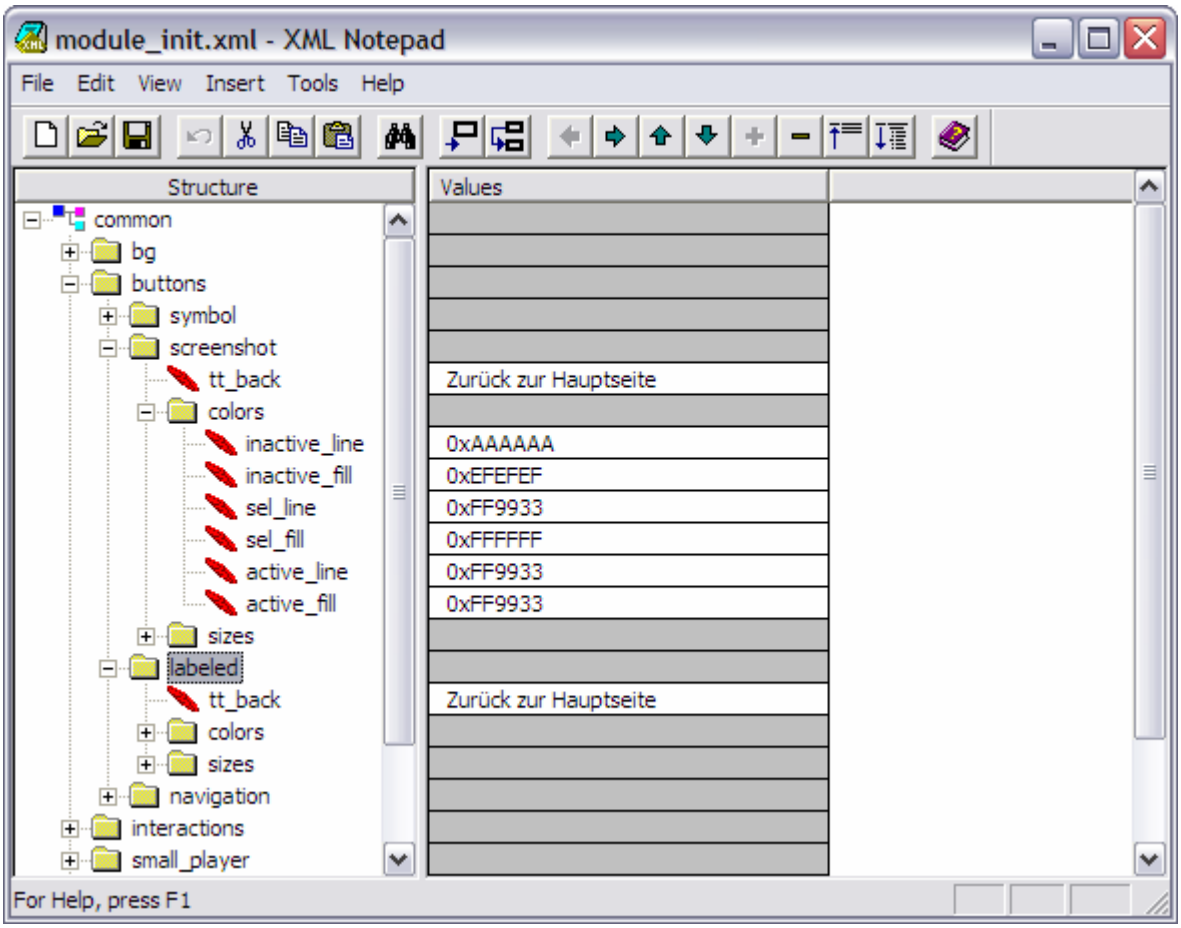

Abbildung 3.7: Die Datei *module_init.xml* im XML Notepad

Wie bereits an anderer Stelle erwähnt, ist die Formatierung der Textinhalte kein Bestandteil der *module_init.xml*, sondern lässt sich über die Stylesheet-Datei *module_textstyles.css* festlegen.

Die Konfigurationsdaten werden in der Datei *init.swf* eingelesen und in einer Objektstruktur namens MODULE_INIT gespeichert. Über diese greifen die beiden anderen Filme des Systemkerns anschließend auf die einzelnen Werte zu (vgl. Abschnitt 3.2.2).

chapter_init.xml

Diese Datei befindet sich im Stammverzeichnis eines jeden SCOs *(chapterX*-Ordner) und beinhaltet grundlegende Angaben zum Lernobjekt. Dazu gehören die beiden Hauptüberschriften, die entsprechend den Namen des übergeordneten Kapitels bzw. Themas sowie die Bezeichnung des Lernobjekts selbst angeben. Des Weiteren sind die Anzahl der Lernschritte sowie die Tooltipps[66] der Navigationsschaltflächen definiert.

[66] Die Tooltipps brauchen nur im Fall mehrerer Lernschritte definiert werden.

page_init.xml

Die in der Datei *page_init.xml* enthaltenen Daten werden benötigt, um einen einzelnen Lernschritt zu generieren. Sie ist deshalb auch in multipler Ausführung vorhanden, und zwar in jedem *pageX*-Verzeichnis eines jeweiligen Lernobjekts. Die Struktur untergliedert sich auf der ersten Ebene unterhalb des Wurzelelements in folgende drei Bestandteile:

- einem `common`-Element, unter dem der Teil an Konfigurationsdaten definiert ist, bei dem es sich nicht um eine zu Bild- oder Textschaltflächen gehörende Information handelt,
- optional einem oder mehreren `screenshot_buttons`-Elementen zur Beschreibung von Bildschaltflächen sowie bei Bedarf
- einem oder mehreren `labeled_buttons`-Elementen, mit denen sich Textschaltflächen einrichten lassen.

Obwohl ein Weglassen des `common`-Elements zu keinem Fehler bei der Abarbeitung führt, würde es einen Bruch im generellen Seitenlayout bedeuten, da die Textinformation des Hauptfelds inklusive der Lernschritt-Überschrift entfällt. Es sollte daher immer definiert werden und mindestens über ein `htmlLink`-Element[67] verfügen. Mit diesem wird der *relative Pfad* (bezogen auf das *pageX*-Verzeichnis) der HTML-Datei angegeben, die als Textinformation im Hauptbereich dargestellt werden soll.

Angaben zu den Inhalten des Zusatzbereichs werden unterhalb eines `info`-Elements platziert. Je nach dessen Anordnung innerhalb der XML-Struktur lassen sich Zusatzinformationen den Haupttext betreffend oder Text-/Bildschaltflächen-bezogene Inhalte festlegen. Durch die Definition von hierarchisch untergeordneten `info_`-Elementen[68] werden später im Lernobjekt die jeweiligen Symbolschaltflächen dargestellt.

Ressourcen wie Bilder, Interaktionen, Beispielanimationen oder Downloads werden als kategorisierte Unterelemente von `special` konfiguriert und lassen sich frei auf der Oberfläche positionieren. Wird eine Koordinate[69] nicht angegeben, so erfolgt automatisch eine mittige Ausrichtung auf der betreffenden Achse des Bereichs[70], dem die Ressource zugeordnet wurde. Dasselbe Prinzip wurde auch bei der Positionierung der Bild- bzw. Textschaltflächen-Container verwendet. (vgl. Abschnitt 3.2.2).

Das folgende Quellcode-Beispiel zeigt einen Ausschnitt aus der *page_init.xml*, die für den ersten Lernschritt des SCOs *Gruppen, Symbole, Instanzen und Bibliotheken* erstellt wurde, und soll die typische XML-Struktur verdeutlichen:

---

[67] Der Name des Elements ist identisch für alle anderen HTML-Verknüpfungen.

[68] `info_anim`, `info_key`, `info_hint`, `info_plus`

[69] x- oder y-Koordinate

[70] Haupt- oder Zusatzbereich

```
<page_init>
   <common>
      <htmlLink>common/common.html</htmlLink>
      <sizes>
         <info_width>380</info_width>
      </sizes>
      <info>
         <htmlLink>common/common_info.html</htmlLink>
         <info_key>
            <htmlLink>common/common_info_key.html</htmlLink>
         </info_key>
      </info>
   </common>
   <screenshot_buttons>
      <link>bibliothek.swf</link>
      <id>screenshot</id>
      <coordinates>
         <y>297</y>
      </coordinates>
      <buttons>
         <optionen>
            <tooltip>Optionsmenü</tooltip>
            <coordinates>
               <xm>294</xm>
               <ym>1</ym>
               <x>290</x>
               <y>0</y>
            </coordinates>
            <sizes>
               <w>22</w>
               <h>18</h>
            </sizes>
            <info>
               <htmlLink>buttons/optionen_info.html</htmlLink>
               <info_hint>
                  <htmlLink>buttons/optionen_info_tip.html</htmlLink>
               </info_hint>
            </info>
```

```
            </optionen>
            <vorschau>…</vorschau>
            …
        </buttons>
    </screenshot_buttons>
</site_init>
```

Quellcode-Bsp. 3.43: Typische Struktur einer *page_init.xml*

3.3.2 Erstellen der Textinhalte

Wie bereits mehrfach erwähnt, erfolgte die Definition der Textinhalte mit Hilfe gewöhnlicher HTML-Dateien. Der Vorteil dieser Vorgehensweise liegt hauptsächlich darin begründet, dass die Texte ohne Flash-Kenntnisse jederzeit angepasst und erweitert werden können.

Durch die Einführung der `StyleSheet`-Klasse mit Flash MX 2004 war es möglich, externe CSS-Dateien zur Laufzeit zu laden und mit diesen die einzelnen Textbausteine zu formatieren. Flash unterstützt dabei eine Reihe von Eigenschaften, die Bestandteil der *CSS1-Spezifikation des W3-Konsortiums* sind und als eine Art Untergruppe betrachtet werden können. Stellvertretend genannt seien die Angaben `color`, `font-family` oder auch `font-weigth`. In der aktuellen Version von Flash wurde der Umfang um die Eigenschaften `letter-spacing` sowie `kerning`[71] erweitert.

Neben der Datei *preloader_view_stylesheets.css*, mit der die Textdarstellung der Ladefortschrittsanzeige formatiert wird, existiert außerdem die Datei *module_stylesheets.css*. Die darin enthaltenen Stil-Festlegungen sind verantwortlich für das restliche Schriftlayout des Lernmoduls. Dieses bezieht sich neben den eigentlichen HTML-Dateien auch auf die Textdarstellung bei Navigations- und Textschaltflächen, Tooltipps, Hauptthema und Lernobjektnamen. Beispielsweise wird mit der Stil-Klasse *maintitle* die Überschrift im Hauptbereich eines Lernschritts fettgedruckt, linksbündig und in dem im Lernmodul übergreifend verwendeten Orangeton dargestellt. Schriftart und –größe richten sich hingegen nach dem Stil des übergeordneten `body`-Elements. In der folgenden Tabelle werden sämtliche Stil-Definitionen inklusive ihrer jeweiligen Verwendung aufgelistet. Stil-Klassen sind an einem vorangestellten Punkt erkennbar:

Stil	Verwendung
`.percent1` `.percent2`	Stile für die Prozentangabe in der Ladefortschrittsanzeige; beide Definitionen sollten zur korrekten Darstellung bis auf die Farbe identisch sein

[71] Die Eigenschaft `kerning` ist kein Bestandteil CSS1-Spezifikation, sondern stellt eine spezielle Erweiterung für Flash 8 dar.

`.loadtext`	Stil des Ladetextes der Ladefortschrittsanzeige
`.tooltip`	Textformatierung der dargestellten Tooltipps
`.chapter`	Stil der Hauptüberschrift bzw. des Themas eines SCOs
`.subchapter`	Textformatierung der Zweitüberschrift bzw. des Lernobjekt-Namens, der auf Höhe der Register dargestellt wird
`.naviButtonInactive` `.naviButtonActive`	Stile für die jeweiligen Zustände der Navigationsschaltflächen

`.labeledButtonInactive`
`.labeledButtonSelected`
`.labeledButtonActive`
	Stile für die jeweiligen Zustände der Textschaltflächen

`body`	Der Stil von `body`, bei dem es sich um das Wurzelelement aller HTML-Seiten handelt, dient zur Standardformatierung der Textinhalte von Haupt- und Zusatz-bereich.
`a:hover` `a:visited`	Die Formatierung der Ankertexte wird lediglich für die Verlinkung zu Macromedia in der Einleitung des Lernmoduls verwendet.
`.maintitle`	Stil für die Überschrift im Hauptbereich
`.infotitle`	Stil für die Überschrift im Zusatzbereich
`.symbols`	Stil, der die verwendete Schriftart für bestimmte Symbole festlegt (z.B. ⌘,◆).
`.listpoint`	Stil eines Listenpunkts
`.listing`	Einrückung des Texts in der ersten Ebene einer Liste
`.listing2`	Einrückung des Texts in der zweiten Ebene einer Liste
`.quellcode`	Stil für Quellcode-Angaben
`.keyword`	Stil für hervorgehobene AS2-Schlüsselwörter oder Ähnliches

Tabelle 3.6: Stildefinitionen in *preloader_view_textstyles.css* sowie *module_textstyles.css*

Eine beliebige Anpassung der Schriftstile ist jedoch nicht ohne weiteres möglich, da die verwendeten Schriftzeichen für das Lernmodul eingebettet wurden, größtenteils mit der Alias-Text-Option (vgl. Abschnitt 3.2.3). Eine entsprechende Modifikation der Stil-Definitionen zieht demzufolge nach sich, dass die nicht mehr benötigten Schriftkonturen ausgetauscht oder die neuen einfach hinzugefügt werden. Dies betrifft je nach Stil die Filme *init.swf*, *main.swf* und *page.swf*.

Für die HTML-Dateien wurden einige Richtlinien festgelegt. Mangels einer

Unterstützung der CSS-Eigenschaft `line-height` seitens Flash wurde das `textformat`-Element verwendet, um den Zeilenabstand zu spezifizieren. Dessen `leading`-Attribut besitzt standardmäßig den Wert 2 (Pixel). Bei Aufzählungen wurde der Zeilenabstand jedoch erhöht, um ein besseres optisches Gesamtbild zu erhalten. So beträgt der Wert vor dem ersten Listenpunkt 10, zwischen zwei Punkten hingegen 7 (siehe Quellcode-Bsp. 3.44).

Das `li`-Element zur Darstellung von Listen wertet der Flash-Player zwar aus, jedoch wurde eine Unterstützung gängiger CSS-Formatoptionen wie `list-style-type` oder `list-style-image` seitens Macromedia nicht implementiert. Aus diesem Grund – und mit der Absicht einer individuelleren Darstellung – wurde von einer Verwendung abgesehen. Stattdessen wurden Aufzählungen aus verschiedenen `p`-Elementen generiert und in gewünschter Weise formatiert. Der Abstand zwischen Text und Aufzählungspunkt wird mit Tabstopps realisiert.

Das folgende Quellcode-Beispiel veranschaulicht den typischen Aufbau einer HTML-Seite am Beispiel der Datei *common.html*, deren Inhalt im Hauptbereich des ersten Lernschritts vom SCO *Die Symbole* dargestellt wird. Die Umsetzung im Lernmodul wird in der anschließenden Abbildung gezeigt.

```
<body><textformat leading="2">

      <p class="maintitle">Was sind Symbole?</p>
      <br />

      <p>Symbole sind <b>wieder verwendbare Objekte</b>, …</p>
      <br />

      <!--Abstand zu nachfolgender Zeile erhöhen-->
      <textformat leading="10">
      <p>Grundsätzlich unterscheidet man zwischen <b>drei
Symbolarten</b>:</p>
      </textformat>

      <!--Abstand zwischen den Aufzählungspunkten erhöhen-->
      <textformat leading="7">
      <p><span class="listpoint">◇</span>Movieclip</p>
      <p><span class="listpoint">◇</span>Schaltfläche</p>
      </textformat>

      <p><span class="listpoint">◇</span>Grafik</p>
```

```
      <br />

      <p>Zusätzlich verfügt Flash über sogenannte…</p>
      <br />

      <p>Symbole werden in der so genannten Bibliothek gespeichert.</p>

</textformat>
</body>
```

Quellcode-Bsp. 3.44: Typische Struktur einer HTML-Seite

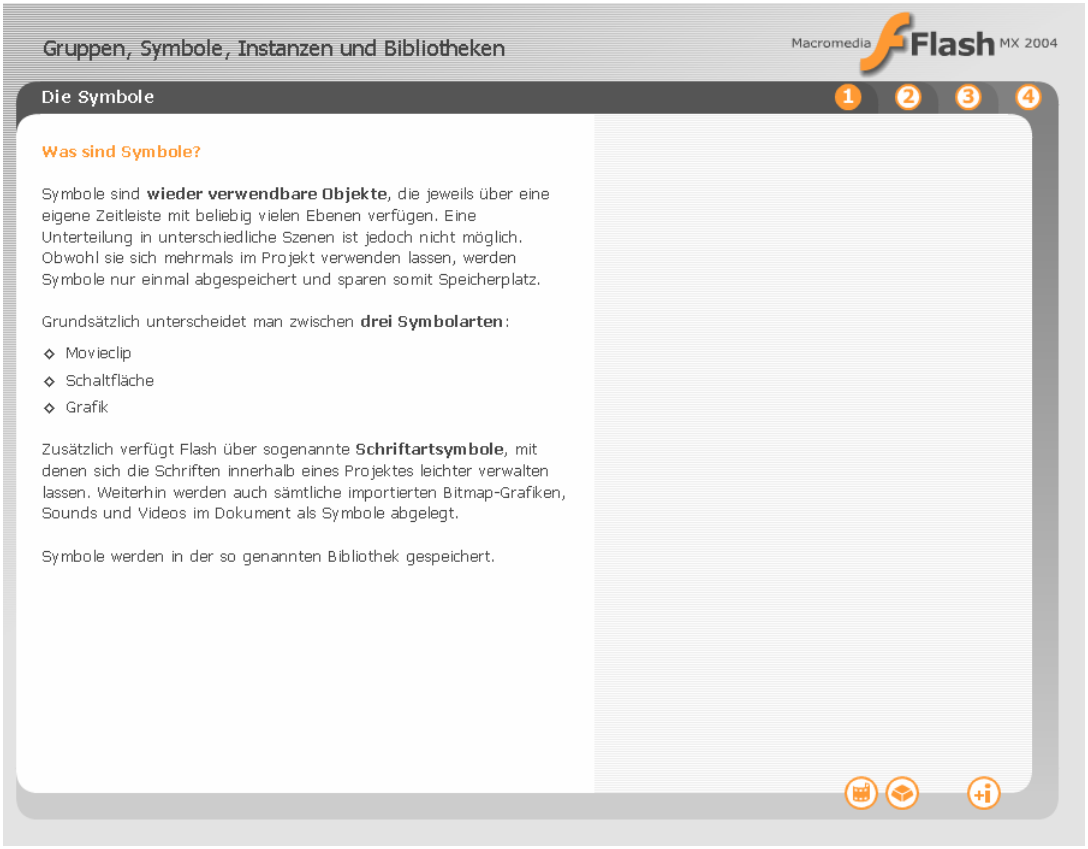

Abbildung 3.8: Die Umsetzung der HTML-Daten im Lernmodul

In der folgenden Tabelle sind spezielle Symbole bzw. Zeichen aufgelistet, die bei der Gestaltung der Texte Lernobjekt-übergreifend verwendet wurden:

HTML	Zeichen	Stil-Klasse	Verwendung
`&#x7a;`	⌘	symbols	Zur Darstellung von Tastenkürzeln; signalisiert entsprechende Taste auf Apple MacIntosh-Tastaturen

`&#x25ca;`	◇	listpoint	Aufzählungspunkt der ersten Ebene; lässt sich auch direkt in den HTML-Code einfügen
`&#x77;`	◆	symbols	Aufzählungspunkt der zweiten Ebene
`&#x3c;`	<	quellcode keyword	öffnende eckige Klammer bei Quellcode-Angaben
`&#x3e;`	>	quellcode keyword	schließende eckige Klammer bei Quellcode-Angaben
`&#x2026;`	…	body	wird je nach Bedarf (direkt) verwendet

Tabelle 3.7: Spezielle Symbole bzw. Zeichen

3.3.3 Erstellung der grafischen Ressourcen

Der folgende Abschnitt beschäftigt sich mit der Herstellung der Bilder, Interaktionen, Animationen sowie Downloadmöglichkeiten. Einzelne Ressourcen werden teilweise von mehreren Lernobjekten verwendet.

Bilder

Wie bereits den Abschnitten 3.1.5 und 3.2.2 entnommen werden konnte, erfolgt das Laden der Bildressourcen zur Laufzeit des Lernprogramms. Mit Hilfe der `SimplePageObjects`-Klasse werden dazu leere Movieclips erstellt und inklusive der URL als `LoadDefinition`-Objekt an den Preloader übergeben. Die verwendeten Grafiken sind pixelbasierter und vektorieller Natur, wobei letztere ausschließlich mit den Zeichenwerkzeugen von Flash generiert wurden.

Bis auf wenige Ausnahmen enthalten alle Bilder *transparente Bereiche*. Realisiert wurde dies bei Rastergrafiken mit Hilfe des *PNG-Grafikformats* (Portable Network Graphic), das im Gegensatz zum JPEG-Format Alphakanäle unterstützt. Da es in Flash MX 2004 – im Unterschied zur aktuellen Version – noch nicht möglich war, in entsprechendem Format vorliegende Grafiken zur Laufzeit zu laden, musste der Umweg über den Import in ein Flash-Dokument und anschließenden Export als SWF-Datei gegangen werden. Beim Kompilieren des Films lassen sich die Rastergrafiken mit dem JPEG-Algorithmus komprimieren.

Der Import von Bildern ist für das Lernmodul auch unter einem anderen Aspekt wichtig. Wie bereits erwähnt, kann durch das Weglassen einer Koordinate in der *page_init.xml* das automatische Ausrichten von Ressourcen auf der betreffenden Achse erreicht werden. Dazu müssen Raster- bzw. Vektorgrafiken auf der Bühne der Hauptzeitleiste jedoch mittig zum Koordinatenursprung platziert werden. Diesen Umstand soll die nachfolgende Abbildung veranschaulichen. Die darin zu erkennende Grafik soll auf der x-Achse zentriert werden, der weiße Bereich stellt die Bühne der Flash-Entwicklungsumgebung dar:

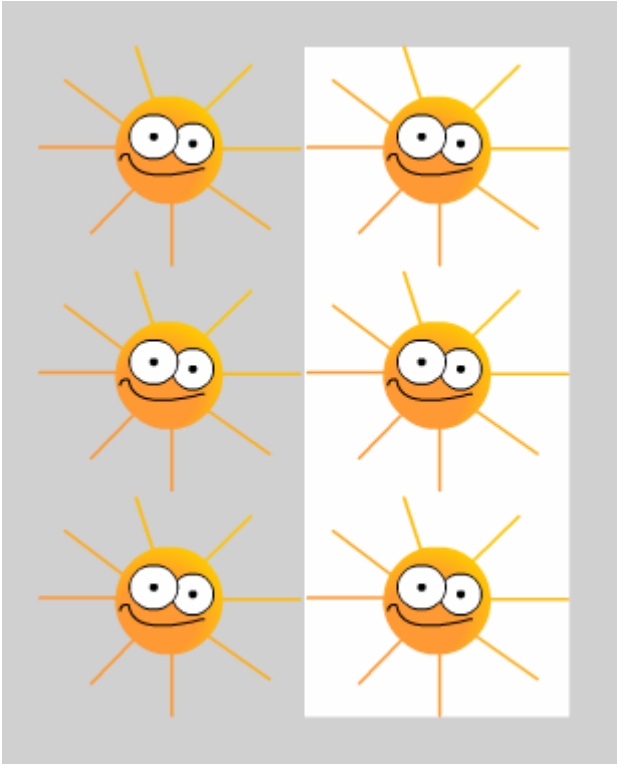

Abbildung 3.9: Anordnung einer Grafik auf der Hauptbühne
für die automatische Ausrichtung auf der x-Achse

Um eine mittige Ausrichtung auf der y-Achse zu gewährleisten, müssten die sechs dargestellten Sonnen demzufolge nach oben verschoben werden.

Einige wenige JPEG-Grafiken, bei denen keine Zentrierung notwendig bzw. erwünscht ist, werden direkt und ohne den Umweg eines Im-/Exports zur Laufzeit geladen.

Hintergrundgrafiken für Bildschaltflächen

Die Basis der Hintergründe für die Bildschaltflächen bilden Bildschirmfotos der Flash-Entwicklungsumgebung. Diese wurden anschließend mit Adobe Photoshop bearbeitet, so dass die standardmäßig grauen Bereiche der Oberfläche transparent erscheinen. Durch die Speicherung im PNG-Format ließen sie sich anschließend korrekt in Flash importieren.

Im Flash-Dokument wurde eine Instanz der Grafik in ein Movieclip-Symbol eingefügt. Die später in der Datei *page_init.xml* definierten `xm`- sowie `ym`-Werte[72] beziehen sich auf das Koordinatensystem des Movieclips und legen die linke obere Ecke des Bildausschnitts fest, der in einer Schaltfläche dargestellt werden soll. Das Symbol befindet sich nicht auf der Bühne, sondern ist lediglich Bestandteil der Dokument-Bibliothek. Über einen so genannten *Verknüpfungsbezeichner*, der sich dem Movieclip-Symbol vergeben lässt, wird in der Klasse `ScreenshotButton` eine Instanz erzeugt.

[72] Das *m* steht für *Maske.*

In den folgenden beiden Abbildungen werden eine Grafik sowie die auf ihrer Grundlage erstellten Bildschaltflächen veranschaulicht. Die transparenten Bereiche des Originals sind orange dargestellt.

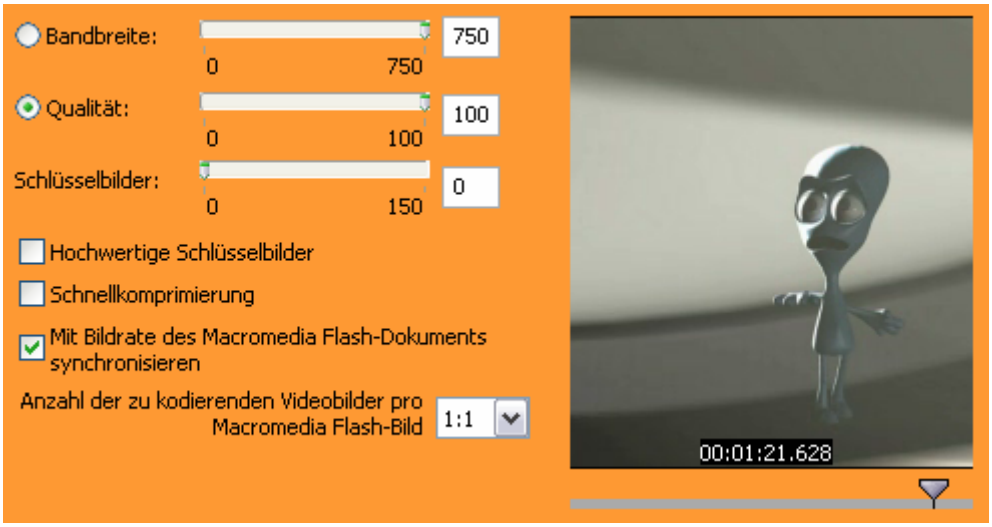

Abbildung 3.10: PNG-Grafik (transparente Bereiche sind orange dargestellt)

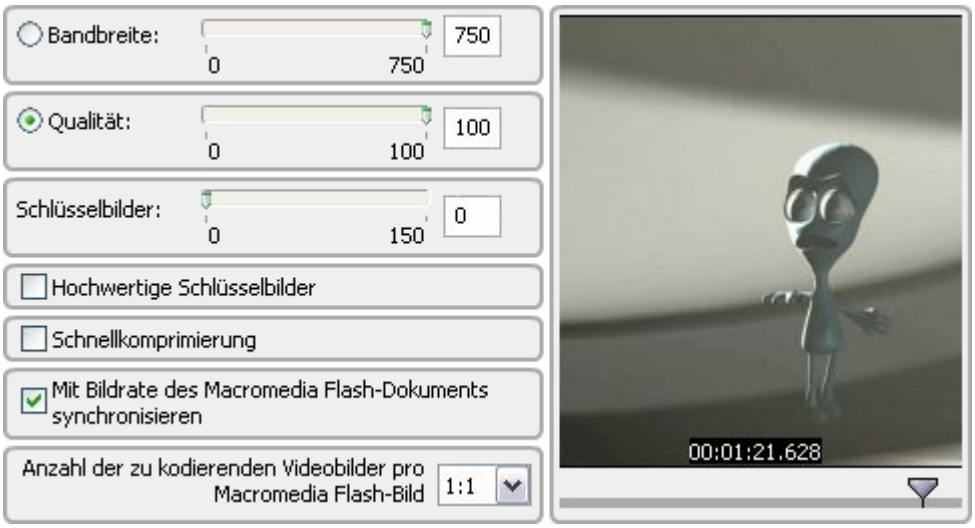

Abbildung 3.11: Bildschaltflächen, die auf Basis der Grafik aus Abbildung 3.10 erstellt wurden

Interaktionen

Bei den Interaktionen handelt es sich um kleine Flash-Filme, in denen der Lernende verschiedene Tätigkeiten, wie Maus- oder Tastatureingaben, ausführen kann, um eine bestimmte Aktion auszulösen. Sie dienen der Verdeutlichung von textbasierten Inhalten und sollen die Motivation des Lerners anregen.

Die Darstellung der Interaktionen erfolgt über ein `MaskedClip`-Objekt (vgl. Abschnitt 3.1.3), dessen Form durch eine Instanz der `RectangleDefinition`-Klasse (vgl. Abschnitt 3.1.2) bestimmt wird. Die Linie besitzt den einheitlich im Lernmodul verwendeten

Orangeton. Auf diese Weise wird die Aufmerksamkeit gelenkt und signalisiert, dass es sich um ein interaktives Objekt handelt und nicht um eine reine Grafik.

Höhe und Breite einer Interaktion werden durch die in der *page_init.xml* eingetragenen Werte bestimmt und sollten demnach auf den zu ladenden Flash-Film abgestimmt werden.

Die folgende Abbildung verschaulicht eine Interaktion, wie sie im Lernobjekt *Die Symbole* zur Demonstration von Schaltflächenzuständen verwendet wird. Sie reagiert auf Mausereignisse und stellt je nach Zustand ein anderes Bild dar.

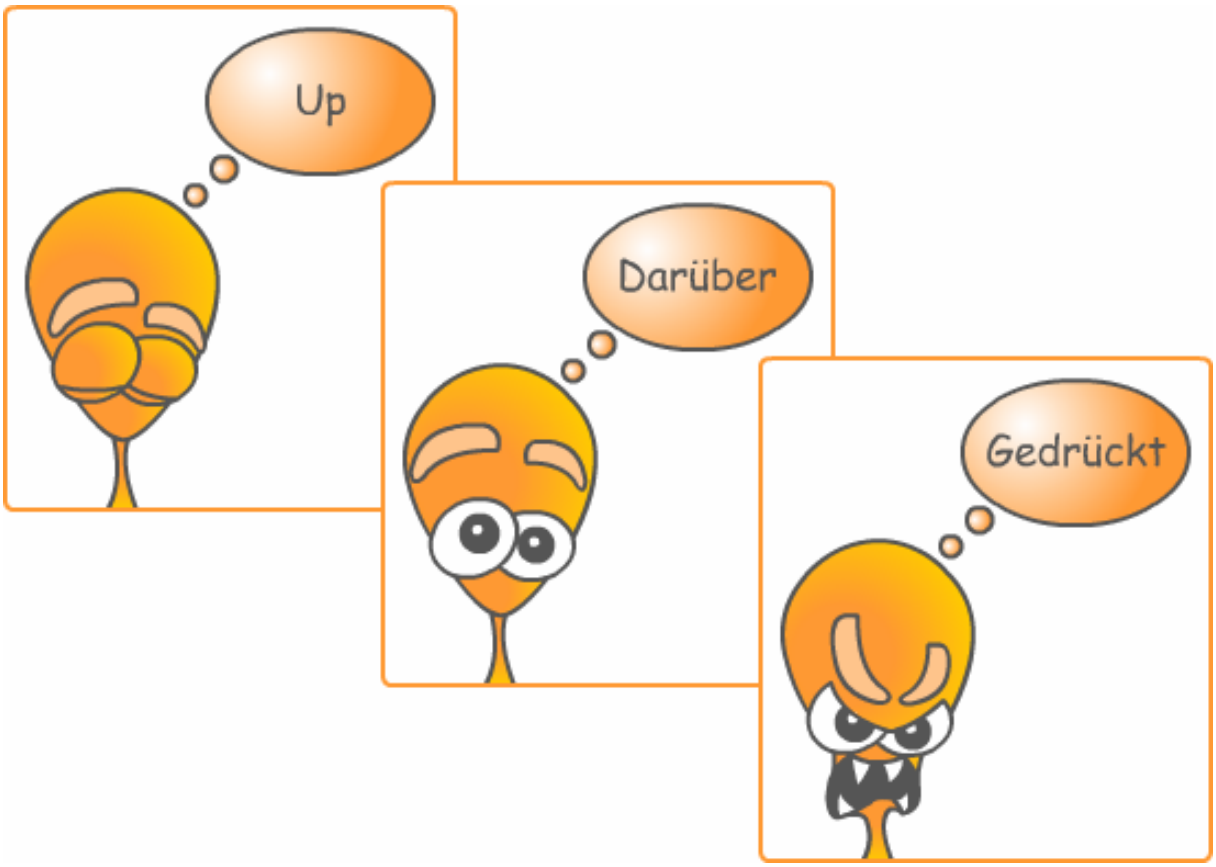

Abbildung 3.12: Interaktion zur Veranschaulichung von Schaltflächenzuständen

Animationen

Wie bereits mehrfach erwähnt, wurden zwei Arten von Animationen entwickelt: veranschaulichende Beispielanimationen und solche, die Handlungsabläufe in sprachlich unterstützter Form erklären. Der notwendige Aufbau eines Flash-Dokuments, damit ein veröffentlichter Film problemlos mit den beiden Klassen `SimplePlayer` sowie `ComplexPlayer` wiedergegeben werden kann, wurde bereits in Abschnitt 3.1.4 eingehend erläutert und soll an dieser Stelle keiner weiteren Beschreibung unterliegen.

Beispielanimationen

Diese Animationen wurden komplett mit den zur Verfügung stehenden Mitteln von Flash MX 2004 erstellt. Ihr Einsatz erfolgt dann, wenn es um die Veranschaulichung bestimmter Flash-Funktionen (z.B. Zeitleisteneffekte) geht, oder aber begleitend zu verschiedenen Workshops, in denen der Lernende mit den unterschiedlichen Animationstechniken vertraut gemacht werden soll.

Die folgenden beiden Bildbeispiele zeigen eine Animation, wie sie für das Lernobjekt *Workshop: Eine Pfad-Animation erstellen* entwickelt wurde. Die erste Abbildung veranschaulicht das betreffende Dokument in der Entwicklungsumgebung mit aktivierter Zwiebelschalenfunktion[73], die zweite hingegen die Darstellung im Lernobjekt mit Hilfe der `SimplePlayer`-Klasse.

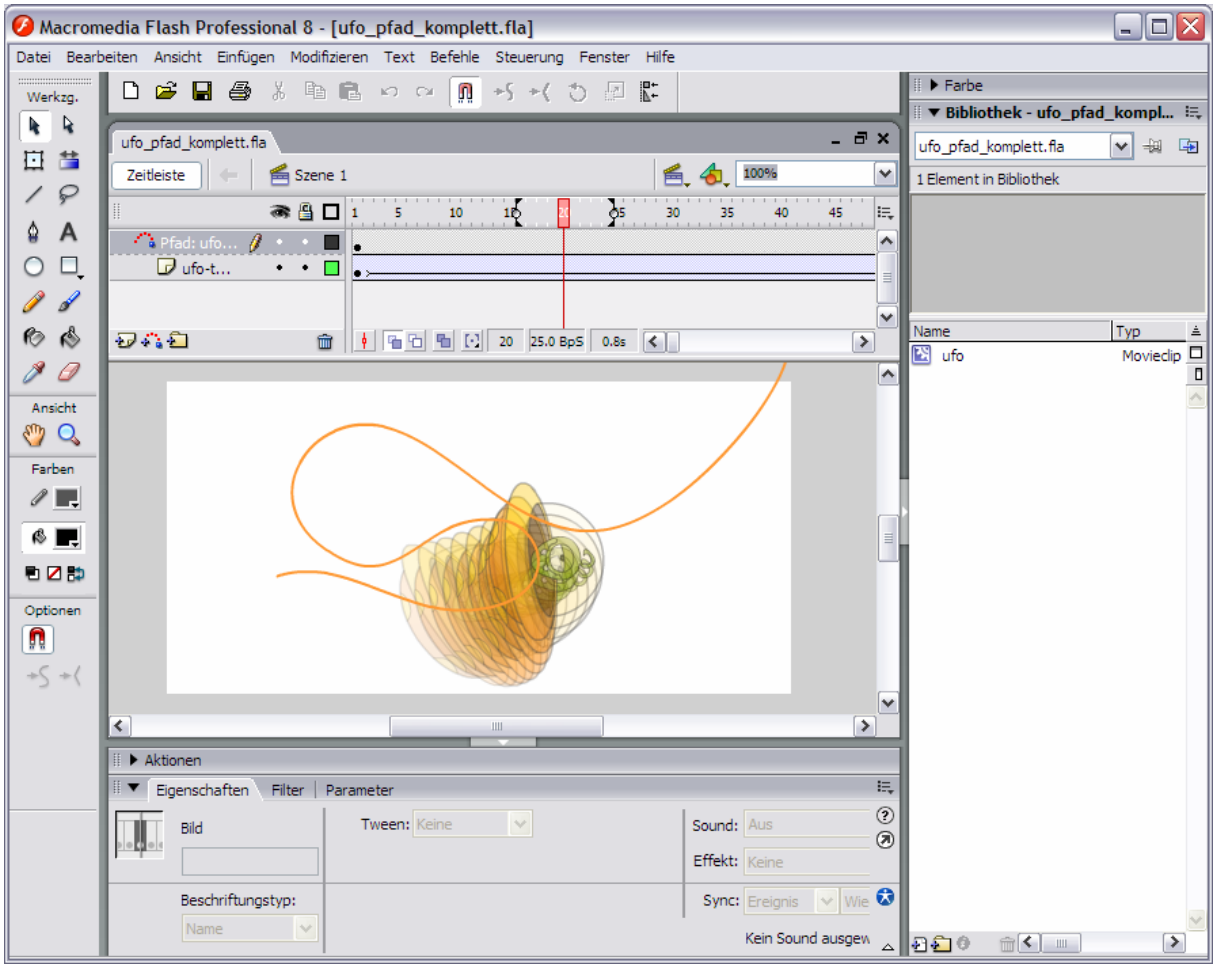

Abbildung 3.13: Eine erstellte Pfadanimation in der Entwicklungsumgebung

[73] Mit Hilfe der Zwiebelschalenfunktion der Flash-Entwicklungsumgebung lassen sich mehrere aufeinanderfolgende Zeitleistenbilder gleichzeitig anzeigen.

Abbildung 3.14: Die Darstellung der Animation im Lernobjekt

Höhe und Breite des Wiedergabefensters lassen sich wie bei den Interaktionen in der Datei *page_init.xml* festlegen und richten sich *nicht* nach der Bühnengröße der geladenen Animation.

Erklärende Animationen

Diese Animationen wurden unter Zuhilfenahme der Software *Macromedia Captivate* erstellt. Bei dieser handelt es sich um ein Programm, mit dem sich unter anderem die in beliebigen Anwendungen ausgeführten Aktionen aufzeichnen und als selbstlaufende Demos oder interaktive Präsentationen zusammenstellen lassen. Neben verschiedenen anderen Formaten besteht auch die Möglichkeit, diese als SWF-Dateien zu speichern. Jedoch liegen die Filme programmbedingt nicht in einer Form vor, die durch die Player-Klassen korrekt wiedergegeben werden kann. Mit der Installation von Captivate wird jedoch ein auf dem System vorhandenes Flash MX 2004[74] dahingehend erweitert, dass es in die Lage versetzt wird, Captivate-Projektdateien (CP-Dateien) zu importieren. Dabei werden die in Captivate erstellten Effekte (Mausbewegungen (nur linear), Überblendungen etc.) in *Flash-Zeitleistenanimationen* umgewandelt und können bei Bedarf nachbearbeitet werden.

Für das Lernmodul wurden demzufolge zunächst die benötigten Flash-Handlungsabläufe mit Captivate aufgezeichnet und anschließend mit Flash dahingehend modifiziert, dass sie problemlos unter der Wiedergabesteuerung laufen. Die Geschwindigkeit wird beim Import automatisch an die in Captivate eingestellte angepasst und beträgt 25 Bilder je Sekunde.

Für die Sprachaufnahmen der erklärenden Texte wurde eine professionelle Sprecherin engagiert. Die Aufzeichnung erfolgte im Tonstudio des SAEK Dresden und lag anschließend in Form einer Audio-CD mit 78 Titeln vor. Nach der Umwandlung der Sprachdaten ins WAV-Format (16Bit, 44.1kHz) wurden sie den jeweiligen Flash-Dokumenten über die

[74] Bei Flash 8 ist die Erweiterung zum aktuellen Zeitpunkt nur manuell möglich.

Import-Funktion hinzugefügt. Die Verwendung in Form von *Streamsounds* (vgl. Abschnitt 3.1.4) gewährleistet, dass die Sprache bei Nutzung des Positionsschiebereglers der Wiedergabesteuerung immer synchron abgespielt wird.

Bis auf wenige Ausnahmen, die ein Stereo-Signal zwingend erforderten, wurden die Audiodaten beim Veröffentlichen der Flash-Animationen mit Hilfe des internen MP3-Algorithmus in ein Mono-Signal mit 128kbps (Kilobit per Second) Bitrate komprimiert. Dadurch ließ sich die Ausgabegröße der SWF-Dateien bei guter Sprachqualität auf ein relativ geringes Maß reduzieren.

Unterstützt werden die Animationen außerdem durch kleine Texthinweise und orangefarbene Markierungsfelder, die teilweise automatisch während der Aufzeichnung, teilweise auch manuell im Anschluss daran durch und mit Captivate generiert wurden. Sie sollen den Lernenden beim Aufnehmen und Verstehen des Gezeigten behilflich sein und werden in folgendem Bildbeispiel dargestellt:

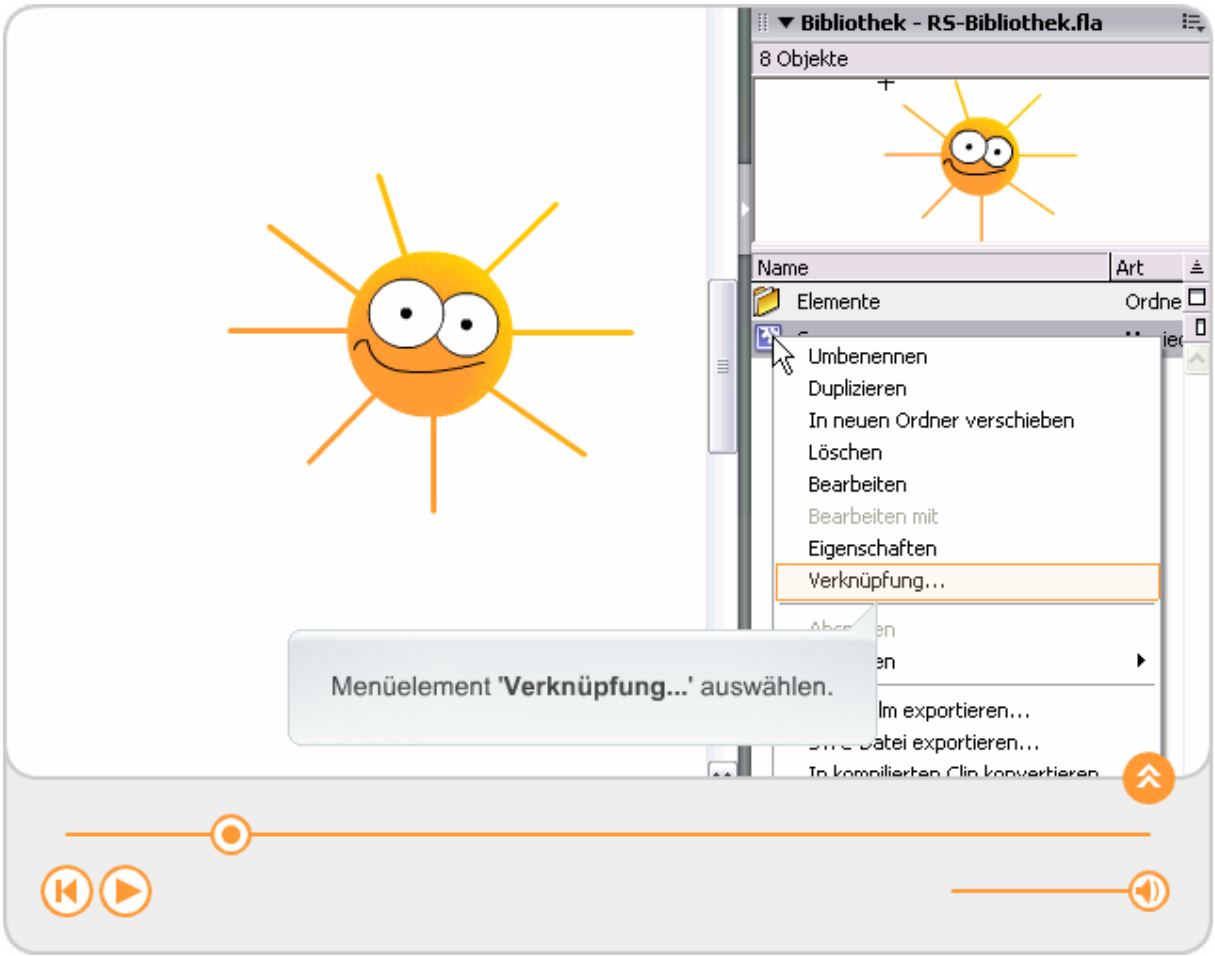

Abbildung 3.15: Text- und Markierungsfeld

Downloads

Downloads bestehen aus zwei Komponenten: einer Schaltfläche und der verknüpften Datei. Bei den Schaltflächen handelt es sich um Standard-SWF-Dateien, die zur Laufzeit in einen leeren Movieclip geladen werden. In der Ereignisroutine `onRelease()` wird dieser angewiesen, eine zuvor als Eigenschaft gespeicherte Adresse, die den Pfad der Datei angibt und in der *page_init.xml* festgelegt wird, mittels `getURL()` aufzurufen und damit den Downloadprozess zu initialisieren.

Für das Lernmodul wurden zwei Schaltflächengrafiken entwickelt, deren grundlegendes Design durch die Auftraggeberin bereits vorgegeben war. Sie dienen dazu, PDF- oder FLA-Dokumente herunterzuladen, die jeweils komprimiert als ZIP-Archive bereitgestellt wurden.

Abbildung 3.16: Download-Schaltflächen

3.4 Hinzufügen eines neuen Lernobjekts

Aus den vorangegangenen Abschnitten des dritten Kapitels ging bereits hervor, dass für die Erstellung eines neuen Lernobjekts der Kontakt mit der Flash-Entwicklungsumgebung auf ein Minimum reduziert wurde. Stattdessen müssen im Wesentlichen XML- sowie HTML-Dateien generiert werden. An dieser Stelle wird daher auch ausdrücklich auf die im separaten Anhang C befindlichen Beschreibungen zu den definierbaren Elementen verwiesen.

Im folgenden Text soll zusammengefaßt erläutert werden, wie sich prinzipiell ein neues Lernobjekt dem Modul hinzufügen lässt.

Verzeichnis des Lernobjekts und Start-Datei anlegen

Im ersten Schritt sollte das Verzeichnis erstellt werden für die Lernobkjekt-spezifischen Daten. Die Position innerhalb der Ordnerstruktur des Lernmoduls sowie die Bezeichnung lassen sich dabei frei wählen. Als nächstes wird die HTML-Datei zum Starten des Lernobjekts benötigt (siehe Abschnitt 3.2.2). Diese wird entweder von einem bereits vorhandenen SCO kopiert oder neu erstellt. Die im Lernmodul verwendete Bezeichnung *load.html* ist frei gewählt und lässt sich nach Belieben ändern. Wichtig ist, dass sich die Datei immer in dem zuvor erstellten Ordner befindet und intern korrekt auf den Stammordner verwiesen wird (`mainpath`-Variable, JavaScript-Dateien). Würden sich also beispielsweise die Lernobjekt-Daten im Pfad *[Stammordner]/scos/sco1* befinden, so müsste die `mainpath`-Variable den Wert `"../../"` zugewiesen bekommen.

chapter_init.xml sowie Unterverzeichnisse anlegen

Im nächsten Schritt wird die Datei *chapter_init.xml* angelegt (vgl. Abschnitt 3.3.1 bzw. betreffenden Anhang). In ihr werden die beiden Überschriften (Thema und SCO-Bezeichnung), die Anzahl der Lernschritte sowie die Tooltipps für Navigationsschaltflächen festgelegt. Für ein Lernobjekt, das unter dem Thema *Kurs1* läuft, als Bezeichnung *Lernobjekt1* zugewiesen bekommt und über zwei Lernschritte verfügt, könnte die Datei *chapter_init.xml* beispielsweise wie folgt aussehen:

```
<chapter_init>
   <chapter>Kurs1</chapter>
   <subchapter>Lernobjekt1</subchapter>
   <pages>2</pages>
   <tooltips>
      <tooltip1>Lernschritt1</tooltip1>
      <tooltip2>Lernschritt2</tooltip2>
   </tooltips>
</chapter_init>
```

Quellcode-Bsp. 3.45: Beispiel einer *chapter_init.xml*

Abbildung 3.17: Darstellung im Lernobjekt

Die Bezeichnung des Wurzelelements spielt keine Rolle und lässt sich frei wählen. Für jeden Lernschritt wird anschließend ein separater Unterordner erzeugt. Dessen Name ist vorgeschrieben und setzt sich immer aus *page* sowie der jeweiligen Lernschrittnummer zusammen. Im soeben dargestellten Beispiel wären dies also *page1* sowie *page2*.

page_init.xml sowie Ressourcen anlegen

Dieser Schritt bei der Entwicklung eines neuen Lernobjekts stellt den wesentlichen Teil dar und erfordert den meisten Zeitaufwand. In welcher Reihenfolge die Daten erstellt werden, spielt prinzipiell keine Rolle. Gerade bei den SWF-Ressourcen für Bildschaltflächen ist es jedoch sinnvoll, zunächst diese zu entwickelen, bevor die einzelnen interaktiven Elemente in der Datei *page_init.swf* über Koordinaten- und Maßangaben definiert werden. Welche Vorgaben für die Erstellung grafischer Ressourcen gelten, wurde bereits in Abschnitt 3.3.3 ausführlich erläutert und soll deshalb an dieser Stelle nicht noch einmal erwähnt werden.

Im Folgenden wird der Aufbau der *page_init.xml* anhand einer fiktiven Struktur erläutert.

Die Erklärung wird in drei Abschnitte untergliedert.

```
<page_init>
   <common>
      <htmlLink>common/common.html</htmlLink>
      <sizes>
         <info_width>400</info_width>
      </sizes>
      <info>
         <htmlLink>common/common_info.html</htmlLink>
      </info>
      <special>
         <downloads>
            <datei1>
               <link>fla_download_button.swf</link>
               <file>archiv1.zip</file>
               <coordinates>
                  <y>580</y>
               </coordinates>
            </datei1>
         </downloads>
      </special>
   </common>
   …
</page_init>
```

Quellcode-Bsp. 3.46: `common`-Abschnitt einer *page_init.xml*

Der `common`-Abschnitt spezifiziert die allgemeine Information eines Lernschritts. Mit dem ersten Element `htmlLink` wird festgelegt, welche HTML-Datei die Hauptinformation enthält, also den Textinhalt, der später im Hauptbereich dargestellt wird und ständig sichtbar ist. Die relative Pfadangabe bezieht sich auf das *pageX*-Verzeichnis. Die für dieses Beispiel im Unterordner *common* angelegte *common.html* ist sehr einfach aufgebaut und sieht wie folgt aus:

```
<body>
<textformat leading="2">
      <p class="maintitle">Überschrift von Lernschritt 1</p>
      <br />
```

```
      <p>Hauptinformation…</p>
</textformat>
</body>
```

Quellcode-Bsp. 3.47: *common.html*

Die Wertzuweisung beim `leading`-Attribut ist wichtig, damit ein optisch ausgewogenes Textbild entsteht.

Das Element `info_width` der *page_init.xml* legt fest, dass ein Zusatzbereich von 400 Bildpunkten Breite rechts auf der Seite dargestellt wird. Diese Angabe (alternativ auch `info_height`) ist *zwingend* notwendig, wenn Inhalte im Zusatzbereich dargestellt werden sollen. Damit sind alle Daten gemeint, die unterhalb eines `info`-Elements platziert werden, wie beispielsweise die im Anschluss verknüpfte Datei *common_info.html*.

Für alle HTML-Dateien gelten prinzipiell die gleichen Regeln. Die Texte des Hauptbereichs unterscheiden sich von den anderen lediglich durch eine differente Formatierung der Überschrift, was am Beispiel der *common_info.html* zu erkennen ist:

```
<body>
<textformat leading="2">
      <p class="infotitle">Hinweis</p>
      <br />
      <p>Klicken Sie bitte mit der Maus auf eine der links abgebildeten
Schaltflächen.</p>
</textformat>
</body>
```

Quellcode-Bsp. 3.48: *common_info.html*

Die Verwendung des `special`-Elements soll in der fiktiven *page_init.xml* anhand eines Downloads demonstriert werden (siehe Quellcode-Beispiel 3.46). Analog lassen sich unter diesem auch die notwendigen Daten für Bilder, Interaktionen oder Beispielanimationen definieren (siehe Abschnitt 3.3.1). Im verwendeten Beispiel wird eine zu ladende Flash-Download-Schaltfläche (*fla_download_button.swf*) mit der Datei *archiv1.zip* verknüpft. Die Darstellung erfolgt an der y-Position 580 inklusive horizontal mittiger Ausrichtung im Hauptbereich. Letzteres geschieht automatisch durch den fehlenden x-Wert.

Abbildung 3.18 veranschaulicht, was die bisher beschriebenen Schritte bewirken:

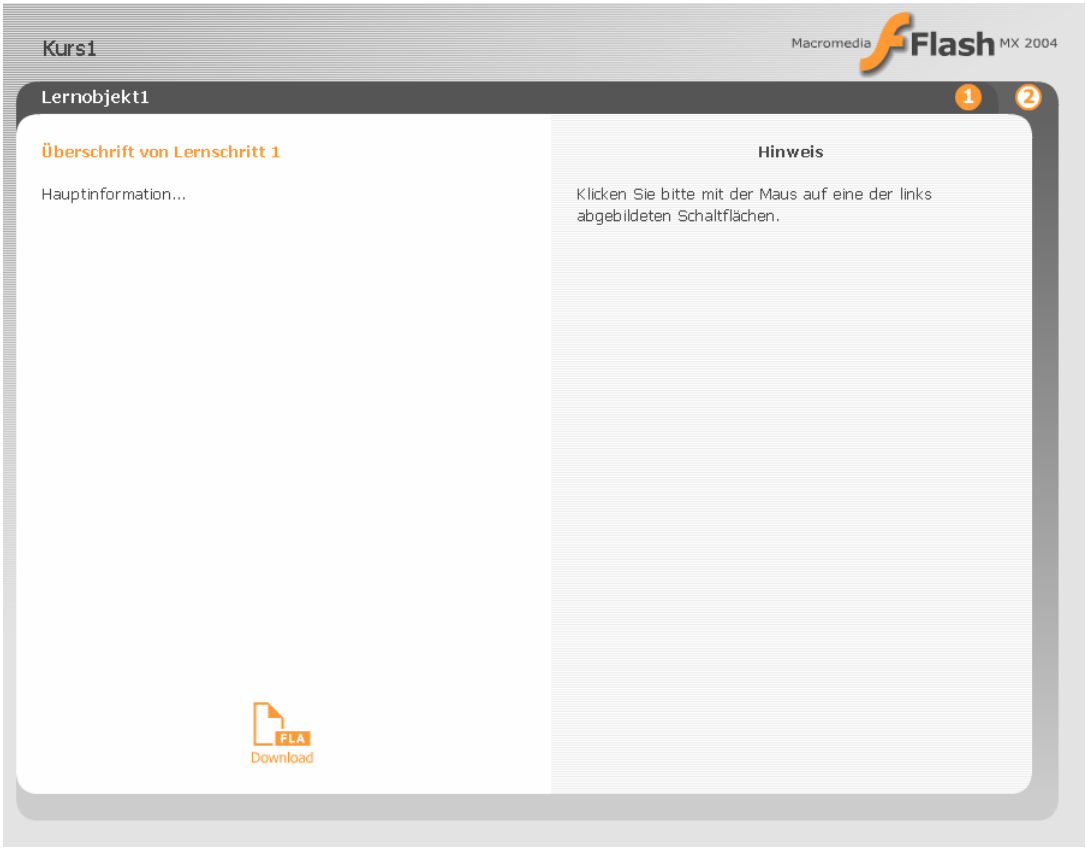

Abbildung 3.18: Die Darstellung der unter `common` spezifizierten Daten

Als nächstes sollen der Seite zwei Textschaltflächen hinzugefügt werden. Die Struktur der Datei *page_init.xml* wird dazu um folgenden Elemente erweitert:

```
<page_init>
   ...
   <labeled_buttons>
      <coordinates>
         <x>30</x>
         <y>190</y>
      </coordinates>
      <buttons>
         <button1>
            <theLabel>Beschriftung1</theLabel>
            <tooltip>Beschreibung</tooltip>
            <coordinates>
               <x>0</x>
               <y>0</y>
            </coordinates>
```

```
            <sizes>
               <w>205</w>
               <h>30</h>
            </sizes>
            <info>
               <htmlLink>buttons/textsf1_info.html</htmlLink>
               <info_hint>
                  <htmlLink>buttons/textsf1_info_tip.html</htmlLink>
               </info_hint>
            </info>
         </button1>
         <button2>
            <theLabel>Beschriftung2</theLabel>
            <tooltip>Beschreibung</tooltip>
            <coordinates>
               <x>0</x>
               <y>35</y>
            </coordinates>
            <sizes>
               <w>205</w>
               <h>30</h>
            </sizes>
            <info>
               <htmlLink>buttons/textsf2_info.html</htmlLink>
            </info>
         </button2>
      </buttons>
   </labeled_buttons>
   …
</page_init>
```

Quellcode-Bsp. 3.49: `common`-Abschnitt einer *page_init.xml*

Die erste Koordinatenangabe bezieht sich auf den Schaltflächen-Container, in dem die beiden interaktiven Elemente generiert werden. Durch die explizite Zuweisung eines x- und eines y-Werts erfolgt keine automatische Ausrichtung an den betreffenden Koordinatenachsen.

Unterhalb von `buttons` werden im Anschluss die entsprechenden Schaltflächen-bezogenen Daten angegeben. Die Bezeichnung der einzelnen Schaltflächen (in diesem Fall `button1` und `button2`) kann gemäß den XML-Richtlinien frei gewählt werden.

Für jedes Element werden zunächst zwei Eigenschaften definiert. Dabei bestimmt `theLabel` die Beschriftung einer Schaltfläche, `tooltip` hingegen den dargestellten Hinweis im inaktiven Zustand[75]. Anschließend folgen die obligatorischen Koordinaten-[76] und Größenangaben. Der Abstand zwischen zwei benachbarten Schaltflächen sollte wie bei den entwickelten Lernobjekten *fünf Bildpunkte* betragen. Da die erste Schaltfläche eine Höhe von 30 Bildpunkten besitzt, wird die zweite deshalb an der y-Position 35 platziert.

Das nachfolgende `info`-Element beinhaltet jeweils die HTML-Verknüpfung. Darüber hinaus wurde diesem bei der ersten Textschaltfläche noch das Element `info_hint` untergeordnet. Auf diese Weise wird festgelegt, dass im Zusatzbereich die Anzeige der Symbolschaltfläche für nützliche Tipps erfolgt und bei Bedarf somit weitere Informationen geladen werden können.

Die bisher durchgeführten Schritte werden in nachfolgender Abbildung veranschaulicht:

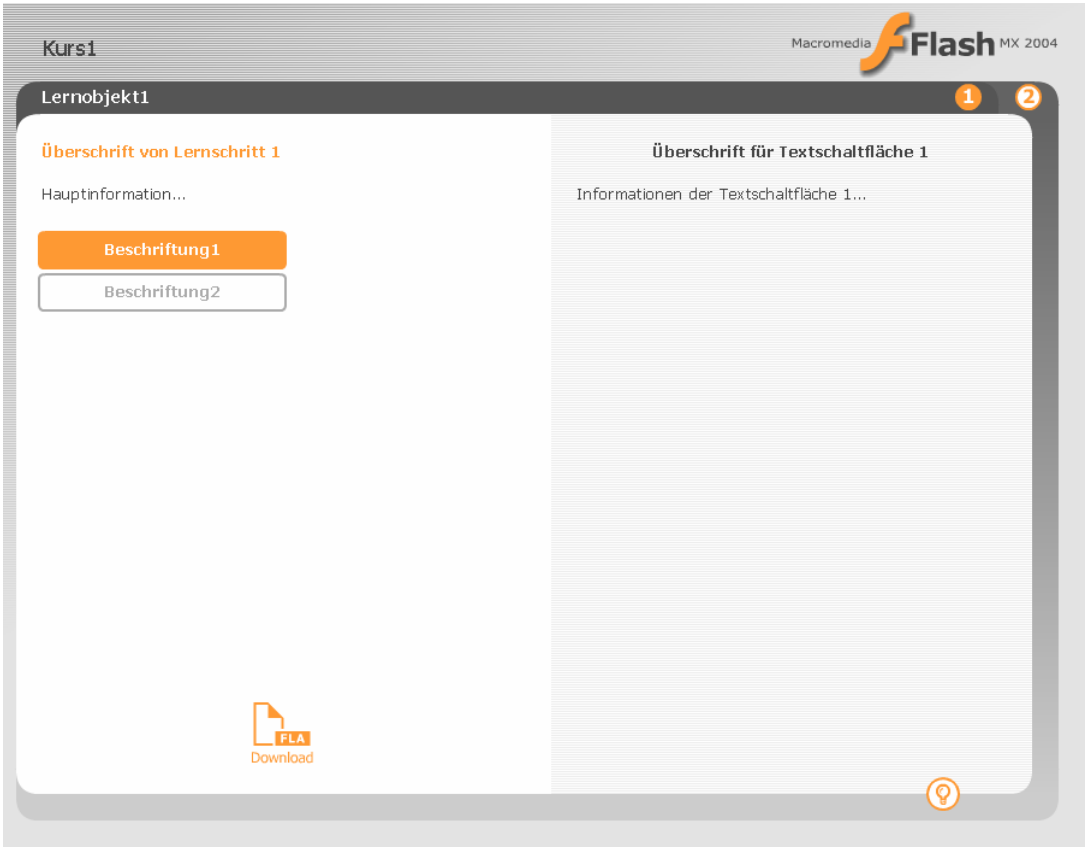

Abbildung 3.19: Textschaltflächen sowie *Nützliche Tipps*-Symbolschaltfläche

Als letztes sollen zwei Bildschaltflächen hinzugefügt werden. Bevor auf den XML-Code

[75] Der Tooltipp im aktiven Zustand ist bei allen Schaltflächen gleichen Typs identisch und wird in der Datei *module_init.xml* festgelegt

[76] Die Koordinaten werden relativ zu denen des Schaltflächen-Containers angegeben.

eingegangen wird, erfolgt zunächst eine Betrachtung der grafischen Ressource. Die in der folgenden Abbildung orange markierten Bereiche stellen die Bildfläche der späteren Schaltflächen dar. Die Werte im Textfeld geben die Koordinaten der linken oberen Ecke sowie Breite und Höhe an.

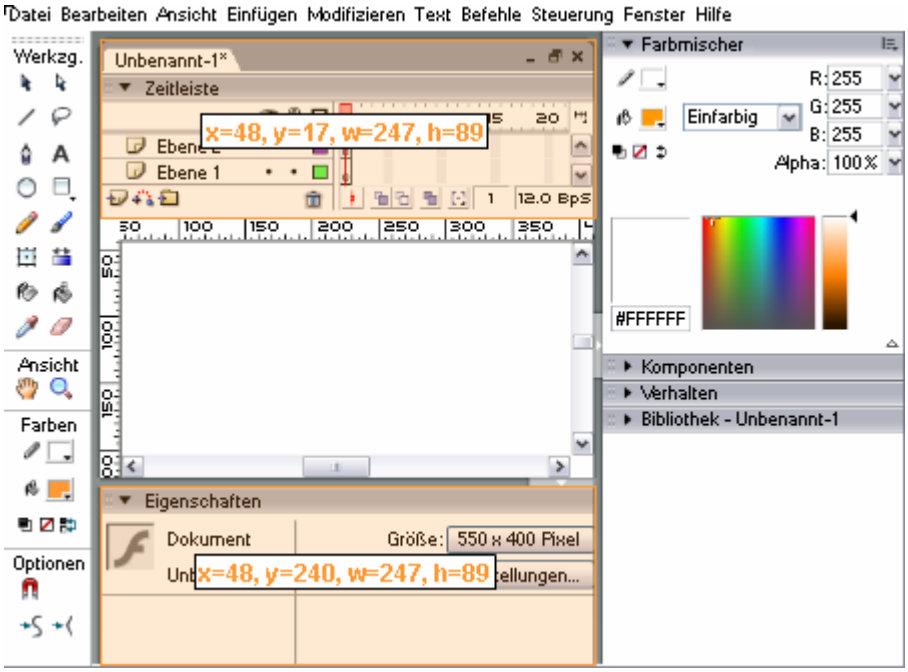

Abbildung 3.20: Bildbereiche der späteren Schaltflächen (orange markiert)

Die Grafik wurde im PNG-Format importiert und enthält transparente Bereiche. Sie wurde in einem Movieclip-Symbol platziert, das als Verknüpfungsbezeichner den Wert *screenshot* zugewiesen bekam. Das Flash-Dokument wurde unter dem Namen *flash_mx_2004.swf* im Verzeichnis *ressources/screenshots/* veröffentlicht. Der Pfad ist vorgegeben, lässt sich jedoch in der Datei *module_init.xml* ändern[77].

Der folgende XML-Code stellt die Ergänzungen der *page_init.xml* dar und wird im Anschluss erläutert:

```
<page_init>
    ...
    <screenshot_buttons>
        <link>flash_mx_neu.swf</link>
        <id>screenshot</id>
        <coordinates>
```

[77] Eine Änderung wirkt sich auf alle Bildschaltflächen-Hintergrundgrafiken aus, so dass bereits vorhandene unter Umständen in das neue Verzeichnis verschoben werden müssen.

```
      <y>308</y>
   </coordinates>
   <buttons>
      <button1>
         <tooltip>Beschreibung</tooltip>
         <coordinates>
            <xm>48</xm>
            <ym>17</ym>
            <x>0</x>
            <y>0</y>
         </coordinates>
         <sizes>
            <w>247</w>
            <h>89</h>
         </sizes>
         <info>
            <htmlLink>buttons/bildsf1_info.html</htmlLink>
         </info>
      </button1>
      <button2>
         <tooltip>Beschreibung</tooltip>
         <coordinates>
            <xm>48</xm>
            <ym>240</ym>
            <x>0</x>
            <y>94</y>
         </coordinates>
         <sizes>
            <w>247</w>
            <h>89</h>
         </sizes>
         <info>
            <htmlLink>buttons/bildsf2_info.html</htmlLink>
            <info_key>
               <htmlLink>buttons/bildsf2_info_key.html</htmlLink>
            </info_key>
         </info>
      </button2>
```

```
      </buttons>
   </screenshot_buttons>
</page_init>
```

Quellcode-Bsp. 3.50: common-Abschnitt einer *page_init.xml*

Die Struktur von screenshot_buttons unterscheidet sich nur geringfügig von der für labeled_buttons festgelegten. Mit den Elementen link und id werden sowohl die SWF-Datei als auch der Verknüpfungsbezeichner angegeben. Das Weglassen der x-Koordinate bewirkt wie schon zuvor, dass der Schaltflächencontainer mittig ausgerichtet wird. Höhe und Breite ergeben sich aus dem Begrenzungsrahmen aller enthaltenen Schaltflächen-Elemente.

Die zusätzlichen Koordinaten xm sowie ym, die für die einzelnen Schaltflächen definiert werden müssen, bestimmen die Position der linken oberen Ecke des Bildbereichs innerhalb des Movieclips, in dem die Grafik platziert wurde (vgl. auch Abbildung 3.20).

Die zweite Schaltfläche besitzt darüber hinaus ein info_key-Element, anhand dessen die Symbolschaltfläche für Tastaturkürzel dargestellt wird. Die mit selbiger verknüpfte HTML-Datei soll im nachfolgenden Beispiel veranschaulicht werden.

```
<body><textformat leading="2" tabstops="20,113,125">
      <p class="infotitle">Tastenkürzel - Bildschaltfläche 2</p>
      <br />
      <!-- Tabstops vor 'STRG', ':' und 'Bedeutung…' -->
      <p><span class="listpoint">◊</span> STRG/<span class="symbols">
&#x7a;</span> + F3      :     Bedeutung…</p>
      <!-- drei Tabstops eingefügt -->
      <p>               Text mit Tabulator-Taste einrücken…</p>


</textformat>
</body>
```

Quellcode-Bsp. 3.51: common-Abschnitt einer *page_init.xml*

Das tabstops-Attribut wird an den Stellen verwendet, wo sich der Text nicht durch Stilklassen einrücken lässt bzw. wo deren Verwendung keinen nennenswerten Vorteil brächte. In diesem Beispiel ist der gewünschte Abstand sehr speziell und wird nur für diese HTML-Datei festgelegt. Deshalb erscheint ein Einrücken des dritten p-Absatzes[78] mit Hilfe einer

[78] Es lässt sich nur der Beginn einer Zeile einrücken über das Stil-Element margin-left. Deshalb würde eine

Stilklasse nicht notwendig.

Zum Abschluss wird noch einmal der Lernschritt dargestellt, wie er sich anhand der soeben eingefügten Ergänzungen dem Anwender präsentiert. Gleichzeitig wird das Einrücken des Textes veranschaulicht.

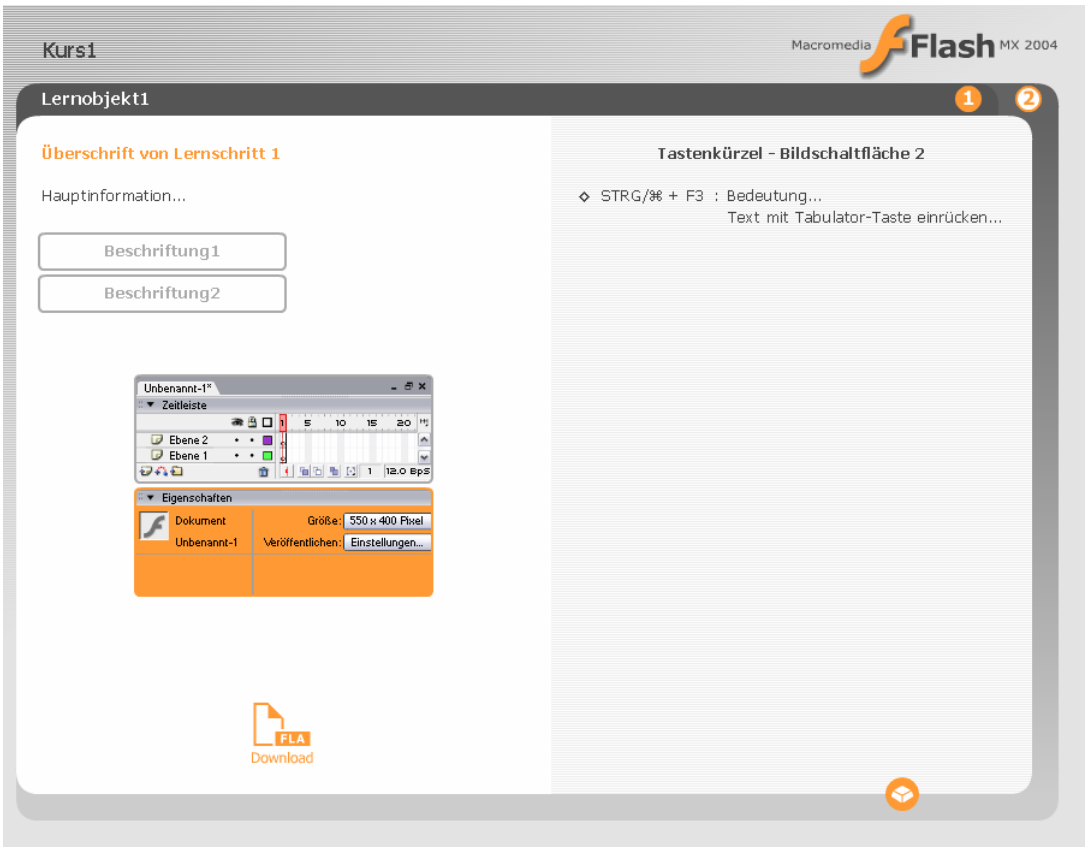

Abbildung 3.21: Bildschaltflächen, Tastenkürzel-Symbolschaltfläche und eingerückter Text

Stilklasse, eingebettet in ein `span`-Element, vor „STRG“ nichts bewirken, denn der Listenpunkt stellt den Anfang der Zeile dar.

4 Einsatz des Lernmoduls „Die interaktive Arbeit mit Flash MX 2004“

4.1 Die Schnittstelle zum Lernmanagementsystem

Im letzten Entwicklungsschritt galt es, die Kompatiblität des Lernmoduls zur SCORM 1.2-Spezifikation herzustellen. Die folgenden Abschnitte beschreiben Umsetzung mit Hilfe von JavaScript-Funktionen, das Erstellen von Metadaten- sowie das Content Packaging.

4.1.1 Die Implementation der SCORM 1.2-Spezifikation

In der SCORM-Spezifikation wird festgelegt, dass die Kommunikation zwischen einem SCO und dem Lernmanagementsystem über ECMAScript bzw. das darauf basierende JavaScript zu erfolgen hat [SCORM 2001c, S.16]. ADL stellt zu diesem Zweck zwei lizenzfreie JavaScript-Dateien zur Verfügung. Dabei handelt es sich um die Dateien *APIWrapper.js* sowie *SCOFunctions.js*, die auch für das Lernmodul verwendet wurden, letztere von beiden in modifizierter Form (vgl. auch Abschnitt 3.2.1). Auf sie wird im Folgenden näher eingegangen.

APIWrapper.js

Diese Datei bildet die Vorraussetzung für eine erfolgreiche Einbindung eines SCOs in ein LMS. Sie verfügt über folgende Funktionalitäten:

- Auffinden der API in der DOM-Struktur
- Starten und Beenden der Kommunikation
- Datenaustausch
- Behandlung von Fehlern, die während der Kommunikation auftreten

Bereitgestellt werden diese in Form von mehreren JavaScript-Funktionen. Das Auffinden der API geschieht dabei mittels `getAPIHandle()`. Die Funktion wird an mehreren Stellen aufgerufen, unter anderem beim Initialisieren der Kommunikation.

```
function getAPIHandle()
{
   if (apiHandle == null)
   {
      apiHandle = getAPI();
   }
   return apiHandle;
}
```

```
function getAPI()
{
   var theAPI = findAPI(window);
   if ((theAPI == null) && (window.opener != null) &&
       (typeof(window.opener) != "undefined"))
   {
      theAPI = findAPI(window.opener);
   }
   if (theAPI == null)
   {
      alert("Unable to find an API adapter");
   }
   return theAPI
}
function findAPI(win)
{
   while((win.API == null)&&(win.parent != null) && (win.parent != win))
   {
      findAPITries++;
      // 7 is an arbitrary number, but should be more than sufficient
      if (findAPITries > 7)
      {
         alert("Error finding API -- too deeply nested.");
         return null;
      }
      win = win.parent;
   }
   return win.API;
}
```

Quellcode-Bsp. 4.1: Auffinden der API in der DOM-Struktur

Das Initialisieren bzw. das Beenden der Kommunikation zwischen SCO und LMS erfolgt mit Hilfe der Funktionen `doLMSInitialize()` bzw. `doLMSFinish()`. Dies geschieht anhand der korrespondierenden API-Methoden `LMSInitialize()` bzw. `LMSFinish()`, deren Aufruf für SCOs obligatorisch ist und durch die SCORM 1.2-Spezifikation vorgeschrieben wird (vgl. Abschnitt 1.2.3).

```
function doLMSInitialize()
{
   var api = getAPIHandle();
   if (api == null){…}

   var result = api.LMSInitialize("");
   …
   return result.toString();
}
function doLMSFinish()
{
   var api = getAPIHandle();
   if (api == null)
   {…}
   else
   {
      //call the LMSFinish function that should be implemented by the API
      var result = api.LMSFinish("");
      …
   }
   return result.toString();
}
```

Quellcode-Bsp. 4.2: Starten und Beenden der Kommunikation

Der Austausch von Daten erfolgt auf Grundlage des AICC CMI Data Model, dessen Datenelemente jedoch nur in reduzierter Form in SCORM 1.2 Einzug hielten (vgl. Abschnitt 1.2.4). Für den Transfer beinhaltet die Datei *APIWrapper.js* die Funktionen `doLMSSetValue()`, `doLMSGetValue()` sowie `doLMSCommit()`, mit denen die betreffenden API-Methoden gekapselt werden. Für das Auslesen bzw. Setzen eines Datenelements wird dessen Name als Zeichenkette (z.B. `"cmi.core.lesson_status"`) der jeweiligen Funktion übergeben. Mittels `doLMSCommit()` werden zwischengespeicherte Daten an das LMS übermittelt.

```
function doLMSSetValue(name, value)
{
   var api = getAPIHandle();
   if (api == null)
   {…}
```

```
   else
   {
      var result = api.LMSSetValue(name, value);
      …
   }
   return;
}
function doLMSGetValue(name)
{
   var api = getAPIHandle();
   if (api == null)
   {…}
   else
   {
      var value = api.LMSGetValue(name);
      …
      return value.toString();
      …
   }
}
```

Quellcode-Bsp. 4.3: Funktionen zum Datentransfer in beide Richtungen

Für die Fehlerbehandlung stehen außerdem die API-Methoden `LMSGetErrorString()`, `LMSGetLastError()` sowie `LMSGetDiagnostic()` in Form entsprechender `do`-Funktionen zur Verfügung.

SCOFunctions.js

Bei den in der Datei *SCOFunctions.js* enthaltenen Funktionen handelt es sich um jene, die direkt durch das SCO aufgerufen werden. Sie nutzen die durch APIWrapper.js bereitgestellte Funktionalität, um die Verbindung zwischen LMS und Lernobjekt herzustellen und bestimmte Daten auszutauschen. Der Original-Quellcode wurde den Bedürfnissen entsprechend angepasst bzw. um zusätzliche Funktionsdefinitionen erweitert.

Die Funktion `initLMS()` wird beim Laden des SCOs automatisch aufgerufen (mehr dazu später in diesem Abschnitt). Sie stellt die Kommunikation zum LMS her und fragt den Status (cmi.core.lesson_status) des Lernobjekts ab, der bedingt durch den Aufbau des Lernmoduls maximal folgende (String)Werte annehmen kann:

- `"not attempted"`: Dies ist der Standardwert und bedeutet, dass ein SCO noch nicht gestartet wurde. Es obliegt dem LMS, den Status mit diesem Wert zu initialisieren.
- `"incomplete"`: Dieser Wert bedeutet, dass der Lerninhalt noch nicht komplett durchgearbeitet wurde.
- `"complete"`: Dieser Status gibt an, dass ein SCO vollständig durchgearbeitet wurde.

Die Werte `"failed"` (nicht bestanden) bzw. `"passed"` (bestanden) werden durch ein SCO des Lernmoduls nicht gesetzt, da diese nur im Zusammenhang mit Lernerfolgskontrollen von Bedeutung wären. Auf derartige wurde jedoch vor allem aus zeitlichen Gründen verzichtet (vgl. Abschnitt 2.3.5). Ebenso wurde Wert `"browsed"` nicht berücksichtigt. Dieser gibt an, dass ein im Modus *Browse*[79] laufendes SCO gestartet wurde.

Nach dem ersten Aufruf eines Lernobjekts wird der Status automatisch auf `"incomplete"` gesetzt. Als letztes wird in der Funktion `initLMS()` ein Timer gestartet, um beim Beenden die Bearbeitungszeit ermitteln zu können.

```
function initLMS()
{
  var result = doLMSInitialize();

  if(result == "true")
  {
      var status = doLMSGetValue( "cmi.core.lesson_status" );
      if (status == "not attempted"){
            // the student is now attempting the lesson
            doLMSSetValue( "cmi.core.lesson_status", "incomplete" );
      }
      startTimer();
  }
}
```

Quellcode-Bsp. 4.4: Funktion, die beim Laden eines SCO aufgerufen wird

Beim Beenden eines Lernobjekts wird automatisch die Funktion `doQuit()` aufgerufen. Darin wird die benötigte Bearbeitungszeit über `computeTime()` ermittelt und im Element `cmi.core.session_time` gespeichert. Anschließend erfolgt die Übertragung eventuell

[79] Vorschaumodus auf ein Lernobjekt; lässt sich über das Datenelement `cmi.core.lesson_mode` abfragen

temporär gespeicherter Daten an das LMS, bevor die Kommunikation beendet wird.

```
function doQuit()
{
  if(checkForLMS)
  {
      //zusätzliche Abfrage wegen onbeforeunload() + onunload()
      if (!LMSIsInitialized()) return;

      computeTime();
      doLMSCommit(); //temporäre Daten übertragen
      doLMSFinish(); //Kommunikation beenden
  }
}
```

Quellcode-Bsp. 4.5: Beenden eines SCOs

Die im Folgenden genannten Funktionen wurden für den Austausch von Daten hinzugefügt. Ein Lernobjekt gilt als abgeschlossen, wenn alle Lernschritte desselbigen mindestens einmal angezeigt wurden. In diesem Fall wird aus dem Hauptfilm (*main.swf*) heraus die Funktion `setCompletedStatus()` aufgerufen.

```
function setCompletedStatus()
{
      var status = doLMSGetValue("cmi.core.lesson_status");
      if (status != "completed")
      {
            doLMSSetValue("cmi.core.lesson_status", "completed")
      }
}
```

Quellcode-Bsp. 4.6: SCO-Status wird auf *completed* gesetzt

Einmal durchgearbeitete Lernschritte sollen bei einem wiederholten Aufruf eines SCOs nicht erneut geladen werden müssen, bevor selbiges als abgeschlossen gilt. Daher werden beim Verlassen des Lernobjekts die betreffenden Seitenzahlen sowie die Nummer des aktuellen Lernschritts gespeichert, um bei einem nochmaligen Start wieder eingelesen zu werden. Dies geschieht über die Funktion `setChapterStatus()` sowie unter Verwendung des Datenelements `cmi.core.lesson_location`. Letzteres eignet sich insbesondere deshalb, da es als Wert eine beliebige Zeichenkette zugewiesen bekommen kann.

```
function setChapterStatus( status )
{
      doLMSSetValue( "cmi.core.lesson_location", status );
}
```

Quellcode-Bsp. 4.7: Bearbeitungsstatus speichern

Die Daten werden in folgender Form gespeichert:

Aktueller_Lernschritt&Bearbeiteter_Lernschritt1, Bearbeiteter_Lernschritt2,…

So würde beispielsweise bei einem Lernobjekt, dessen erster und dritter Lernschritt durchgearbeitet wurden und der vierte den aktuellen im Moment des Schließens darstellt, der Wert `"4&1,3,4"` lauten. Da ein Lernschritt als bearbeitet gilt, sobald er vollständig geladen und angezeigt wurde, zählt auch der vierte als solcher und wird daher mit gespeichert.

Wird ein SCO gestartet, erfolgt der Aufruf der Funktion `getChapterStatus()`, in der die Daten wieder ausgelesen und zurückgegeben werden. Für den Fall, das mit dem Lernobjekt erstmalig gearbeitet wird, ist der Rückgabewert `"1&1"`.

```
function getChapterStatus()
{
      var status = doLMSGetValue( "cmi.core.lesson_location");
      if ( status == "" ) status = "1&1";
      return status;
}
```

Quellcode-Bsp. 4.8: Bearbeitungsstatus auslesen

Obwohl die Datei *ModuleFunctions.js* bereits im Abschnitt 3.2.2 teilweise vorgestellt wurde, soll nunmehr eine Betrachtung selbiger unter dem Gesichtspunkt der SCORM-Implementation erfolgen.

ModuleFunctions.js

Beim Start eines Lernobjekts wird automatisch `loadFlashContent()` aufgerufen. Läuft das Lernmodul in einer SCORM 1.2-konformen Lernumgebung, so werden die Komunikation initialisiert sowie der Bearbeitungsstatus abgefragt. Beim Rückgabewert der Funktion `getChapterStatus()` handelt es sich um ein Objekt vom Typ `String`, so dass die enthaltenen Daten mit Hilfe des Methodenaufrufs `split('&')` in ein Array mit zwei

Teilstrings zerlegt werden können. Diese werden in den Variablen `LESSON_LOCATION` [80] und `VISITED_PAGES`[81] gespeichert und zusammen mit anderen Eigenschaften an *init.swf* übergeben.

```
function loadFlashContent()
{
      …
      //falls das Lernmodul in ein LMS integriert wurde, werden
      //zusätzliche Variablen gesetzt und die SCORM-Schnittstelle
      //initialisiert
      if(checkForLMS)
      {
            //Funktionen sind in der Datei SCOFunctions definiert
            initLMS();
            //Bearbeitungsstatus abfragen
            var ll_vp   = getChapterStatus().split('&');

            flashVars += "&LESSON_LOCATION=" + ll_vp[0] +
                         "&VISITED_PAGES=" + ll_vp[1];
      }
      …
}
```

Quellcode-Bsp. 4.9: Aufruf der Funktionen von *SCOFunctions.js*

Im body-Element des dynamisch generierten HTML-Codes wurde den Ereignisroutinen `onunload` und `onbeforeunload`[82] die Funktion `doQuit()` (aus *SCOFunctions.js*) zugewiesen. Dadurch wird diese beim Entladen des Dokuments automatisch aufgerufen.

[80] Lernschritt, auf dem das SCO verlassen wurde

[81] durchgearbeitete Lernschritte

[82] `onunload` wird für die korrekte Abarbeitung im Microsoft Internet Explorer benötigt.

```
function loadFlashContent()
{      …
       document.write('<body bgcolor="#e3e3e3" onbeforeunload="doQuit();"
                          onunload="doQuit();" … >');
       …
}
```

Quellcode-Bsp. 4.10: `doQuit()`-Aufruf im `body`-Element

Die an den Initialisierungsfilm (*init.swf*) übergebenen JavaScript-Variablen werden im Hauptfilm verarbeitet. Besteht ein Lernobjekt aus mehreren Lernschritten, so werden beim Start die in `VISITED_PAGES` sowie `LESSON_LOCATION` gespeicherten Daten in der Funktion `loadFirstPage()` ausgewertet. Für bearbeitete Lernschritte wird dabei die `visited`-Eigenschaft (besichtigt) der betreffenden Navigationsschaltfläche auf `true` (wahr) gesetzt. Der in `LESSON_LOCATION` gespeicherte Wert bestimmt anschließend, welche Seite zu Beginn geladen wird.

```
function loadFirstPage()
{
       //ein Lernschritt
       if(pages<2)
       {
              loadPage("page1");
       }
       //mehrere Lernschritte
       else
       {
              initVisitedPages();
              //Position, an der das Lernobjekt zuletzt verlassen
              //wurde; bei erstmaligem Aufruf oder mangelnder SCORM-
              //Unterstützung ist dieser Wert 1
              var loc:Number = INIT_CLIP.IS_LMS == "true" ?
                                  parseInt(INIT_CLIP.LESSON_LOCATION) : 1;

              var mc:NavigationButton = navigationBar.getButton(loc);
              navigationBar.onReleaseAction(mc);
              //gespeichertes pageX-Verzeichnis an loadPage übergeben
              loadPage(mc.path);
       }
```

```
}
initVisitedPages = function()
{
	var mc:MovieClip;
	//einzelne Seitennummern in Array übertragen
	var a:Array = INIT_CLIP.VISITED_PAGES.split(',');

	for(var i=0; i<a.length; i++)
	{
		navigationBar.getButton( parseInt(a[i]) ).visited = true;
	}
	if (a.length == pages) CHAPTER_COMPLETED = true;
}
```

Quellcode-Bsp. 4.11: Auswertung von `LESSON_LOCATION` sowie `VISITED_PAGES`

Der Aufruf von JavaScript-Funktionen lässt sich in Flash MX 2004 auf zwei Wegen realisieren: über die Befehle `getURL()` sowie `fscommand()`. Prinzipiell ist die `getURL()`-Variante vorzuziehen, da sie eine breite Unterstützung durch die auf dem Markt befindlichen Browser erfährt und darüber hinaus den Vorteil bietet, JavaScript-Funktionen direkt mit einer beliebigen Anzahl von übergebenen Parametern aufzurufen. Beim technischen Test des Lernmoduls im Bildungsportal Sachsen unter SABA offenbarten sich jedoch Fehler, insbesondere bei der Verwendung des MS Internet Explorers für Windows. So erfolgte unter Umständen kein Aufruf der `doQuit`-Funktion, was dazu führte, dass zwischengespeicherte Daten nicht an das LMS übertragen wurden. Als Folge dessen galten beispielsweise Lernobjekte weiterhin als *incomplete*, obwohl sie bereits vollständig durchgearbeitet wurden. Da diese Probleme bei der Verwendung von `fscommand()` nicht auftraten, die Methode jedoch von verschieden Browsern nicht unterstützt wird, erschien eine zugangsabhängige Lösung am geeignetsten. So wird unter *Opera für Windows* sowie allen *MacIntosh-basierten Internetplattformen* weiterhin `getURL()` genutzt, bei allen anderen hingegen die zweite Variante.

Für die Verarbeitung von `fscommand()`-Aufrufen aus Flash musste der JavaScript-Code erweitert werden. Zum einen erfordert der Internet Explorer ein zusätzliches VBScript, dass der HTML-Veröffentlichungsvorlage *Flash mit FSCommand* entnommen wurde. Zum anderen müssen die `fscommand()`-Befehle über eine spezielle Funktion abgefangen werden, die sich aus der ID bzw. dem Namen des Flash-Objekts sowie der Endung `_DoFSCommand` zusammensetzt. Im folgenden Quellcode-Beispiel werden die Erweiterungen veranschaulicht:

```
//Behandlung der fscommand-Aufrufe unter IE und Firefox/Mozilla (Windows)
function flashfilm_DoFSCommand(command, args)
{
      switch(command)
      {
            case "setChapterStatus"       : setChapterStatus(args);break;
            case "setCompletedStatus"     : setCompletedStatus();break;
            case "messagebox"             : alert(args);break;
      }
}
//dynamisch erzeugter HTML-Code zum Laden des Flash-Inhalts
function loadFlashContent()
{
      var useGetURL = navigator.platform.indexOf("Win") == -1 ||
                      navigator.userAgent.indexOf("Opera") != -1;
      var movie_id = "flashfilm";
      …
      //explorer-hook für fscommand
      document.write('<script language=\"VBScript\"\>\n');
      document.write('On Error Resume Next\n');
      document.write('Sub '+movie_id+'_FSCommand(ByVal command, ByVal
args)\n');
      document.write('  Call ' + movie_id + '_DoFSCommand(command,
args)\n');
      document.write('End Sub\n');
      document.write('</script\>\n');
      …
      document.write('<object … id="'+movie_id+'…>');
      …
      document.write('<embed … id="'+movie_id+'" name="'+movie_id …');
      …
}
```

Quellcode-Bsp. 4.12: Erweiterung der *ModuleFunctions.js* zur Verarbeitung von `fscommand()`

Mit Flash 8 wurde die Klasse `ExternalInterface` eingeführt, über die sich ebenfalls mit JavaScript kommunizieren lässt. Jedoch ist diese Variante wesentlich flexibler als die zuvor genannten, weshalb sie auch von Macromedia empfohlen wird. So lassen sich beispielsweise beliebig viele Parameter unterschiedlichen Datentyps an eine JavaScript-Funktion übergeben

oder auch Rückgabewerte empfangen.

Die Kommunikation mit JavaScript erfolgt im Lernmodul nach dem Laden eines Lernschritts. Zu diesem Zweck wird aus der *page.swf*-Funktion `showPage()` die Funktion `checkCompletedStatus()` des Hauptfilms aufgerufen. Diese aktualisiert den Bearbeitungsstatus und ruft für den Fall, dass jeder Lernschritt mindestens einmal geladen wurde, die Funktion `setCompletedStatus()` der Datei *SCOFunctions.js* auf.

```
updateChapterStatus = function(button:NavigationButton)
{
	//Seitennummer ermitteln
	var pageNum:String = String(navigationBar.getButtonNumber(button));
	if(pageNum == INIT_CLIP.LESSON_LOCATION)
	{
		return;
	}
	else
	{
		INIT_CLIP.LESSON_LOCATION = pageNum;
		//ein Lernschritt wird erstmalig aufgerufen
		if(!button.visited)
		{
			button.visited = true;
			INIT_CLIP.VISITED_PAGES += ',' + pageNum;
		}
	}
	var chapterStatus:String = pageNum + '&' + INIT_CLIP.VISITED_PAGES;

	if(INIT_CLIP.USE_GETURL == "true")
	{
		getURL("javascript:setChapterStatus('"+chapterStatus+"')");
	}
	else
	{
		fscommand("setChapterStatus", chapterStatus);
	}
}
```

```
checkCompletedStatus = function():Void
{
	//Bearbeitungsstatus aktualisieren
	if(pages>1) updateChapterStatus(navigationBar.getActiveButton());
	if(CHAPTER_COMPLETED) return;

	var a:Array = INIT_CLIP.VISITED_PAGES.split(',');

	var isCompleted = a.length==pages;
	//Lernobjekt wurde vollständig durchgearbeitet
	if(isCompleted)
	{
		if(INIT_CLIP.USE_GETURL == "true")
		{
			getURL('javascript:setCompletedStatus();');
		}
		else
		{
			fscommand("setCompletedStatus", "");
		}
	}
}
```

Quellcode-Bsp. 4.13: Aktualisierung des SCO-Status im Film *main.swf*

4.1.2 Metadaten definieren

Die Beschreibung von Lernressourcen anhand XML-basierter Metadaten ist unter SCORM 1.2 keine Notwendigkeit, wird aber im Sinne der Wiederverwendung und Recherchierbarkeit empfohlen. In der Spezifikation wird zwischen drei unterschiedlichen Arten unterschieden [SCORM 2001b, S.8]:

- *Content Aggregation Meta-data*: beschreibt den SCORM-Inhalt als Ganzes
- *SCO Meta-data*: beschreibt ein wieder verwendbares Lernobjekt
- *Asset Meta-data*: beschreibt ein einzelnes Asset

Im Lernmodul kamen die ersten beiden Metadaten-Typen zum Einsatz, d.h. es wurden XML-Beschreibungen sowohl für das Lernmodul wie auch für jedes einzelne SCO erstellt.

Die Metadaten werden gemäß Spezifikation in verschiedene Kategorien unterteilt, deren untergeordnete Elemente jeweils in einem semantischen Zusammenhang stehen. So enthält beispielsweise die Kategorie `general` den Titel oder auch die Beschreibung des Inhalts,

unter `technical` hingegen sind Angaben zu den technischen Eigenschaften und Anforderungen zu finden. Ebenso wird durch ADL spezifiziert, welche Elemente – für den Fall, dass Metadaten erstellt werden – zwingend erforderlich sind (*mandatory*) und welche einer freiwilligen Angabe (*optional*) unterliegen.

Erstellt wurden die Metadaten mit Hilfe des frei erhältlichen *Reload-Editors Version 1.3* der Firma Reload (Reusable eLearning Object Authoring & Delivery). Die Beschreibung erfolgt auf einer abstrahierten Ebene mittels einer Formular- oder einer Baumansicht (vgl. Abbildungen 4.1 und 4.2).

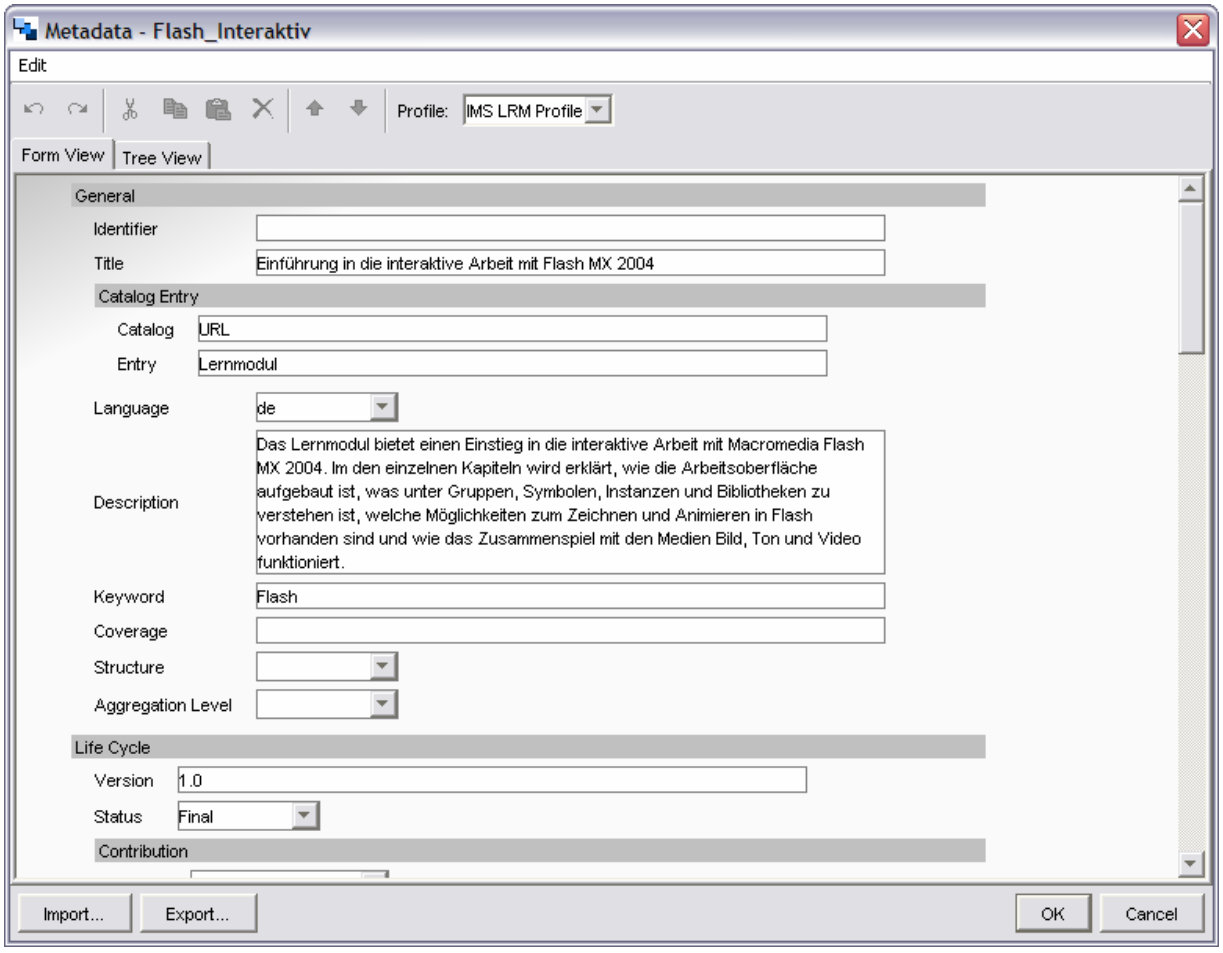

Abbildung 4.1: Formularansicht im Reload-Metadateneditor

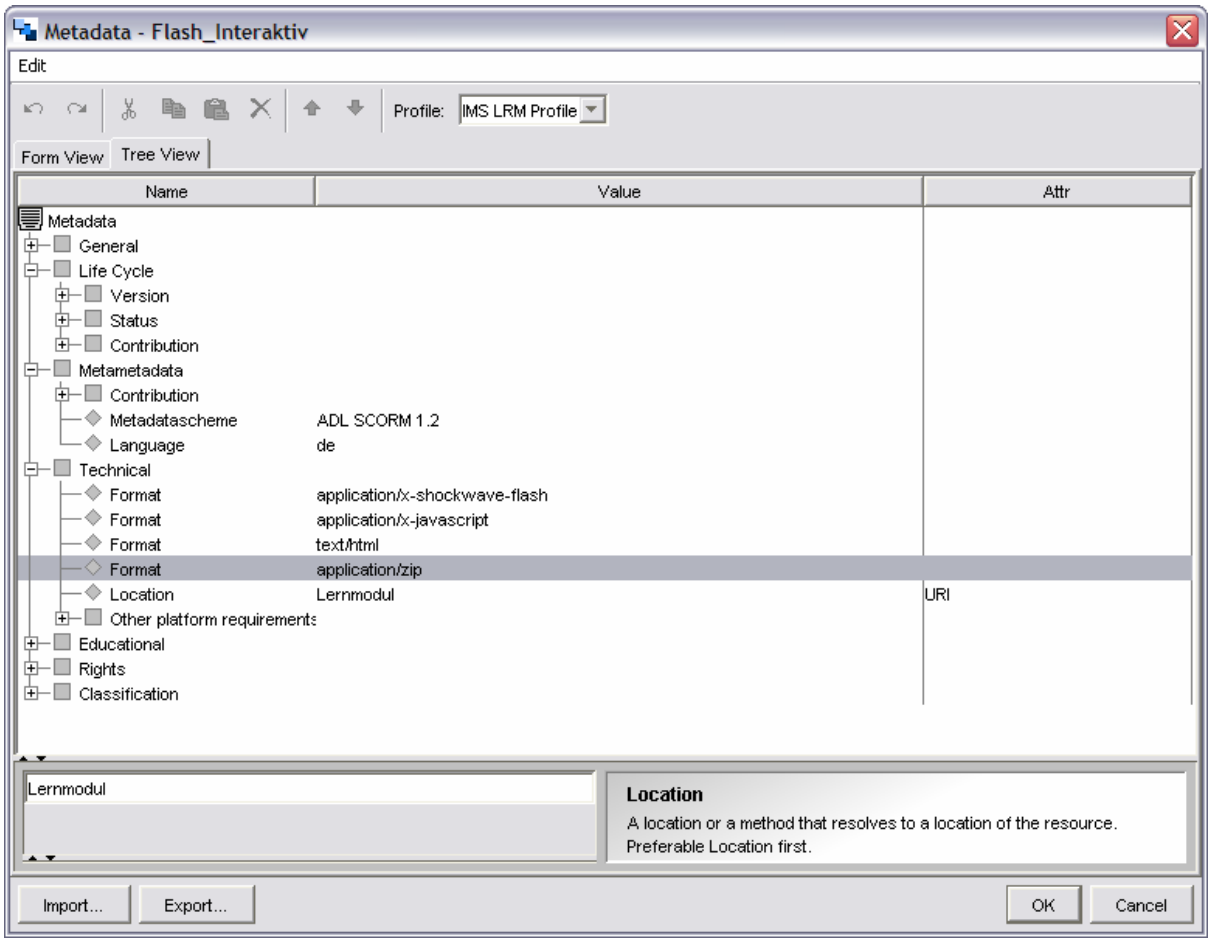

Abbildung 4.2: Baumansicht im Reload-Metadateneditor

Verschiedene Beschreibungselemente, beispielsweise `keyword` unter `general` oder auch `format` in der Kategorie `technical` (vgl. Abbildung 4.2), sind in multipler Ausführung erlaubt. Im Reload-Editor lassen sich diese ausschließlich über die Baumansicht einfügen.

Das folgende Quellcode-Beispiel stellt einen Auszug aus der Metadaten-Datei dar, wie sie für das Lernobjekt Die Arbeitsoberfläche erstellt wurde:

```
<?xml version="1.0" encoding="UTF-8"?>
<lom xmlns="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1
                         imsmd_rootv1p2p1.xsd">
  <general>
    <title>
      <langstring xml:lang="de">Die Entwicklungsumgebung - Die
                                Arbeitsoberfläche</langstring>
    </title>
```

```
    <catalogentry>
      <catalog>URL</catalog>
      <entry>
        <langstring xml:lang="de">lernmodul_flash_interaktiv/chapter_1_1
        </langstring>
      </entry>
    </catalogentry>
    <language>de</language>
    <description>
      <langstring xml:lang="de">Der Lernende wird mit der
                                Arbeitsoberfläche von Flash MX 2004
                                vertraut gemacht.</langstring>
    </description>
    <keyword>
      <langstring xml:lang="de">Flash</langstring>
    </keyword>
    …
  </general>
 …
</lom>
```

Quellcode-Bsp. 4.14: Auszug aus einer SCO-Metadaten-Datei

4.1.3 Content Packaging

Der letzte Schritt bei der Entwicklung des Lernmoduls bestand in der Beschreibung der Inhaltsstruktur sowie der Zusammenfassung aller Daten zu einem SCORM-Paket (Content Package). Realisiert wurde dies ebenfalls mit Hilfe des bereits genannten Reload-Editors.

Die inhaltliche Struktur des SCORM-Pakets sowie Informationen zu den physischen Ressourcen werden in einer so genannten *Manifestdatei* (*imsmanifest.xml*, vgl. Abschnitt 1.2.3) abgebildet. Deren Gliederung durch die Elemente `manifest`, `metadata`, `organizations` sowie `ressources` wird in nachstehender Abbildung deutlich:

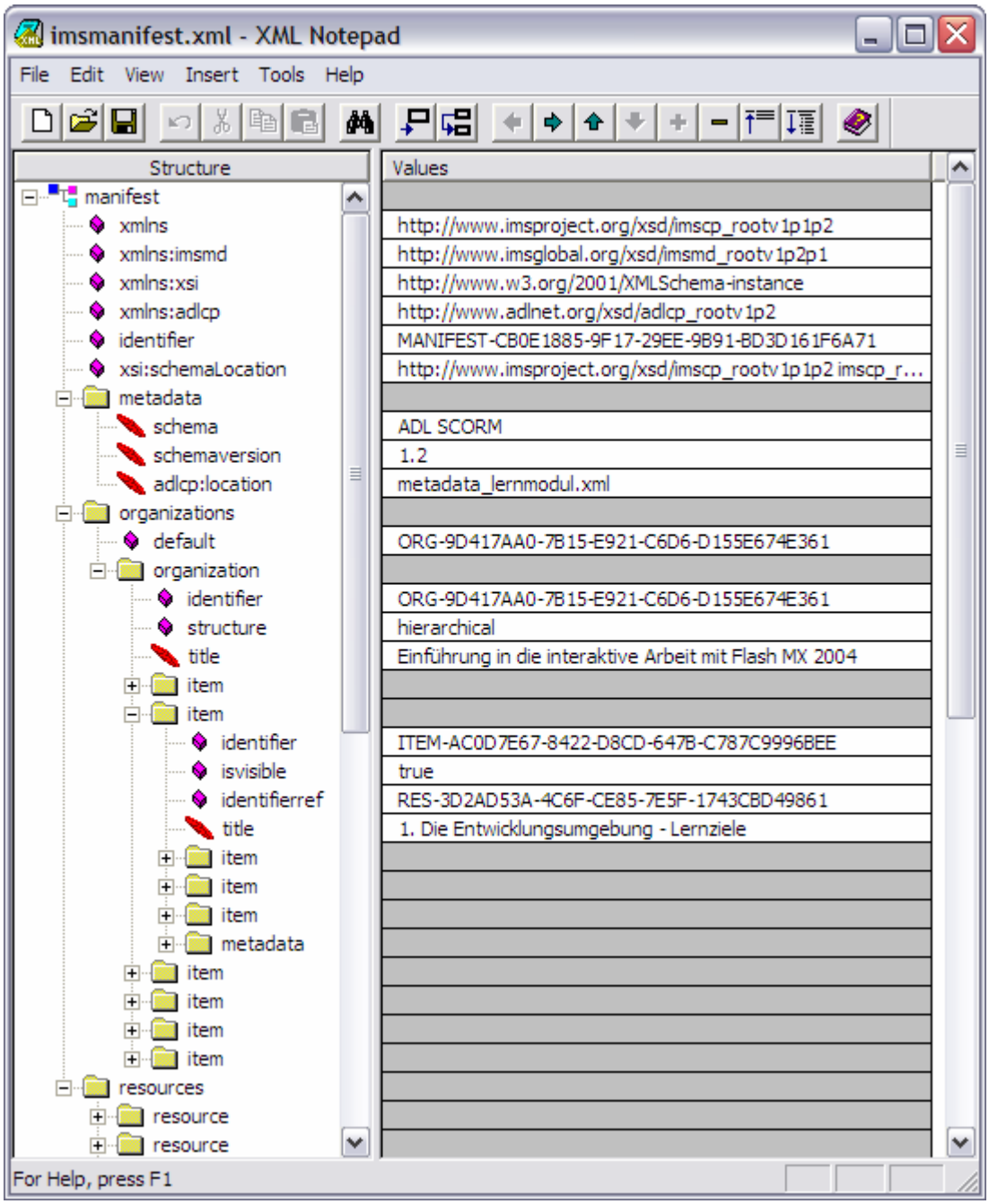

Abbildung 4.3: Die Gliederung der *imsmanifest.xml*

Metadaten lassen sich entweder direkt in der Struktur definieren, oder aber wie im Fall des Lernmoduls in externe Dateien auslagern. Zu diesen wird anschließend über das Element `adlpc:location` lediglich eine Verknüpfung angelegt.

Unterhalb von `organizations` wird die inhaltliche Struktur des Lernmoduls beschrieben. Mit Hilfe verschiedener `organization`-Elemente kann dabei eine erste Gliederung vorgenommen werden – beispielsweise lassen sich auf diese Weise unterschiedliche Lernwegstrukturen organisieren. Jede Organisation ist wiederum in eine beliebige Anzahl von `item`-Elementen unterteilt. Diese lassen sich entweder als

Blockelemente zur Gliederung einsetzen, oder repräsentieren ausführbare Lerneinheiten (Assets, SCOs). In letzterem Fall verweisen sie über ein Attribut namens `identifierref` auf ein korrespondierendes `ressource`-Element, das im Bereich `ressources` definiert wurde. Selbiges enthält Angaben zu den physischen Dateien einer einzelnen Lerneinheit. So wird darin beispielsweise festgelegt, welche Datei gestartet und ob diese als SCO oder Asset eingestuft werden soll. Über optionale `file`-Elemente lässt sich außerdem angeben, von welchen Dateien eine Lerneinheit abhängt, d.h welche Ressourcen notwenig sind für eine problemlose Ausführung.

Die Editierung der Manifestdatei mit Hilfe des Reload-Editors erfolgt in übersichtlicher und vereinfachter Form. Dazu wird die Dokumentansicht in drei Bereiche unterteilt: ein Ressourcen- (links), ein Manifest- (rechts oben) sowie ein Attributsfenster (rechts unten). Mittels Drag&Drop lassen sich Ressourcen hinzufügen oder Elemente innerhalb einer Lernorganisation verschieben. Viele Bearbeitungsschritte lassen sich außerdem über Kontextmenüs erledigen. Abbildung 4.4 stellt die Bearbeitungsoberfläche dar:

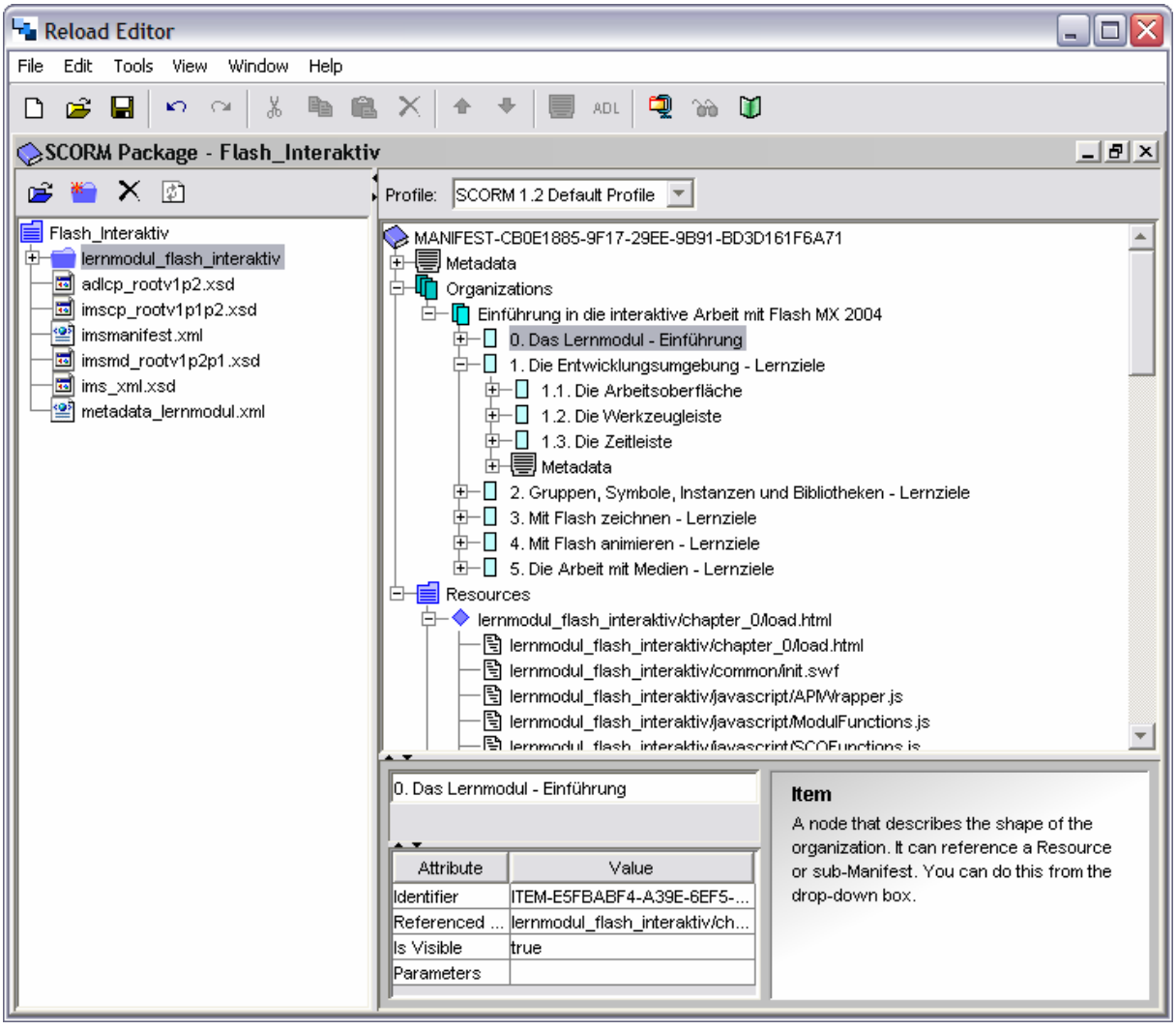

Abbildung 4.4: Bearbeitungsfenster beim Erstellen der Manifestdatei

Basierend auf den eingegebenen Daten erstellt bzw. aktualisiert der Reload Editor die Datei *imsmanifest.xml*. Zu Testzwecken lässt sich auch eine Vorschau der inhaltlichen Struktur generieren und in einem Internetbrowser darstellen.

Die erstellte Manifestdatei lässt sich inklusive der physischen Ressourcen über den Reload Editor zu einem ZIP-Archiv zusammenfassen. Dieses wird im SCORM-Sprachgebrauch auch als *Package Interchange File* (*PIF*) bezeichnet und dient dem einfachen Austausch des Content Package zwischen verschiedenen Systemen (vgl. Abschnitt 1.2.3).

4.1.4 Das Lernmodul im Bildungsportal Sachsen

In diesem Abschnitt soll beschrieben werden, wie der Zugriff auf das Lernmodul im Bildungsportal Sachsen erfolgt. Auf eine explizite Erläuterung der Integration des Lernmoduls in die OLAT-Umgebung wird jedoch verzichtet. Es handelt sich bei dieser lediglich um *ein* mögliches Lernmanagementsystem, unter dem der Kurs lauffähig ist, und sie stellt inklusive des Bildungsportals Sachsen kein zentrales Thema dieser Arbeit dar.

Der Aufruf des BPS-Internetauftritts erfolgt über die HTTP-Adresse (Hypertext Transfer Protocol) `http://www.bildungsportal-sachsen.de`. Auf der Startseite (vgl. Abbildung 4.5) präsentieren sich dem Nutzer verschiedene Schaltflächen mit Verknüpfungen zu vier Hauptbereichen [Wagner 2005, S.68]:

- *Über uns*: In diesem Bereich werden allgemeine Informationen zum Bildungsportal Sachsen angezeigt, wie beispielsweise die Zielsetzungen des Projekts, aktuelle Meldungen oder Ausschreibungen.
- *Weiterbildung*: Der Link führt zum Weiterbildungsangebot mit meist kostenplichtigen Inhalten.
- *Hochschulen*: In diesem Bereich werden alle am BPS beteiligten Hochschulen augelistet inklusive einer Verknüpfung zu ihrer Internetpräsenz.
- *Ausbildung*: Über diese Verknüpfung gelangt man zur Lernplattform OLAT und damit zu den Bildungsangeboten für registrierte Benutzer, Studenten und Angehörige der Hochschulen.

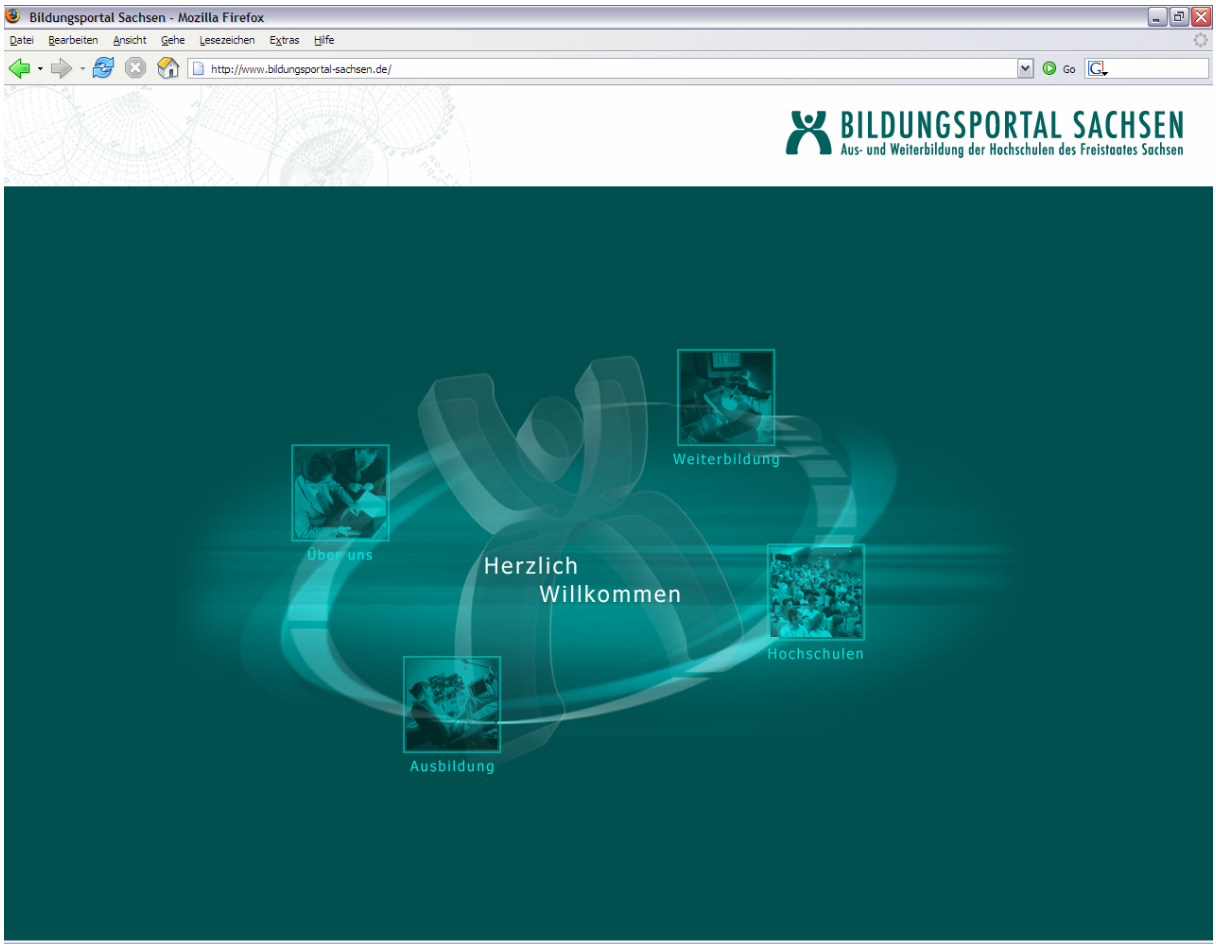

Abbildung 4.5: Die Startseite des Bildungsportals Sachsen

Über den Punkt *Ausbildung* gelangt man zum Login-Bildschirm (vgl. Abbildung 4.6) und kann sich beim System anmelden. Als Student bzw. Angehöriger einer Hochschule ist keine spezielle Registrierung erforderlich. Stattdessen werden die Zugangsdaten des jeweiligen Hochschul-Rechenzentrums zur Anmeldung verwendet. Zum Einloggen muss zunächst die betreffende Hochschule im Klappmenü ausgewählt und anschließend die Schaltfläche *Zum Login* geklickt werden. Im darauffolgenden Bildschirm lassen sich die Zugangsdaten eingeben und an das System übermitteln.

Nutzer, bei denen es sich weder um Studenten noch um Angehörige einer Hochschule handelt, können sich alternativ auch mit ihrem privaten Login[83] anmelden.

[83] Lässt sich beim BPS beantragen.

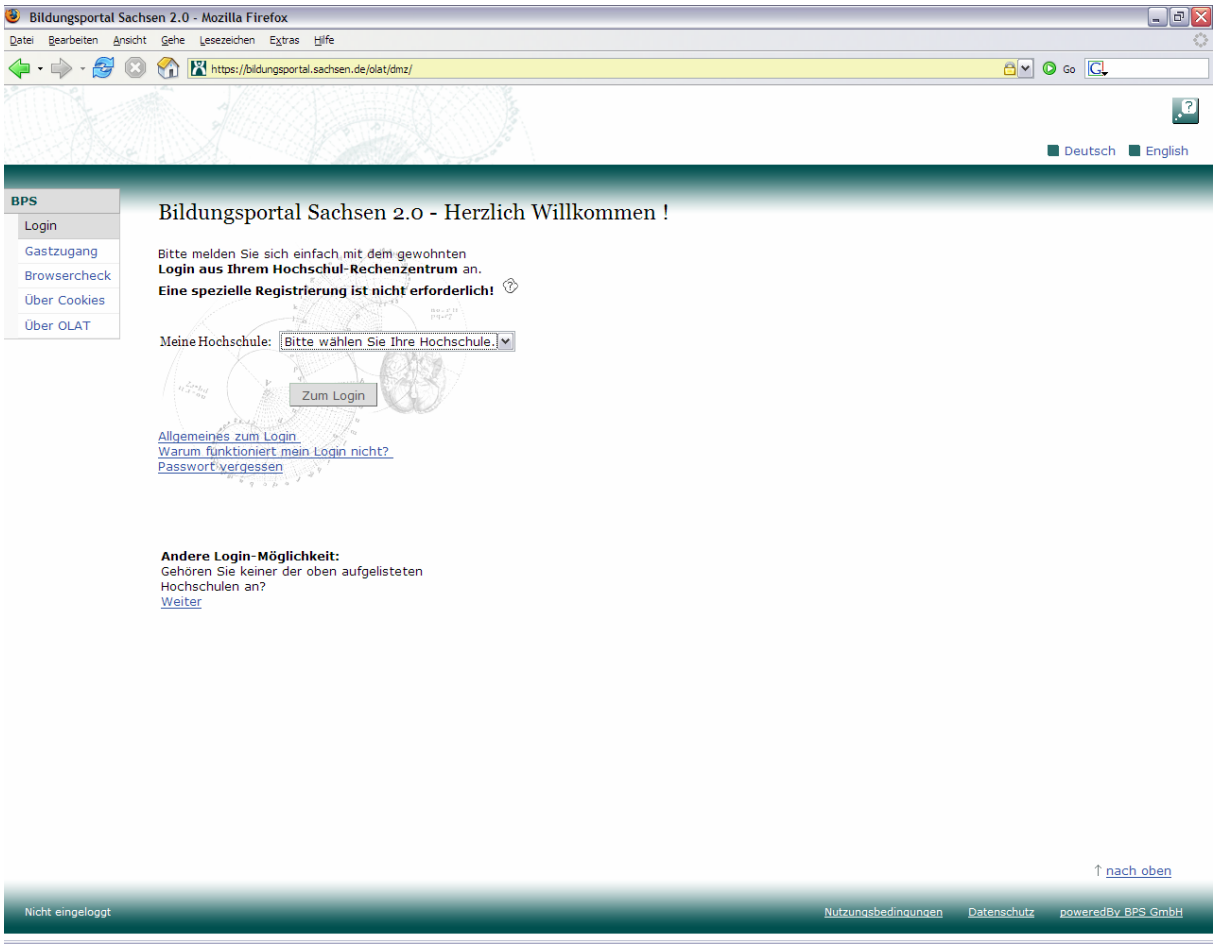

Abbildung 4.6: Der Anmeldebildschirm

Nach der Anmeldung über den Hochschulzugang werden befindet man sich auf der eigentlichen Lernplattform. Am oberen Rand der Benutzungsoberfläche werden vier Registerkarten dargestellt:

- *Home*: Dieser Bereich wird zu Beginn dargestellt und repräsentiert den privaten Bereich des Nutzers. Es werden verschiedene Funktionen wie beispielsweise Bookmark- und Notizenverwaltung zur Verfügung gestellt. Außerdem können persönliche Einstellungen vorgenommen sowie die Sprache gewählt werden.
- *Meine Gruppen*: Listet die Gruppen (Lern-, Arbeits sowie Rechtegruppen) auf und stellt kollaborative Werkzeuge zur Verfügung. Nutzer können mit anderen Nutzern des BPS beispielsweise chatten oder Dateien austauschen.
- *Lernressourcen*: In diesem Bereich lässt sich das verfügbare Bildungsangebot abrufen und nutzen. Darüber hinaus können eigene Lernressourcen erstellt und hinzugefügt werden.
- *Support*: Hilfe-Seite des BPS mit verschiedenen Tipps und Tricks, Beispielen und Foren.

Um das Lernmodul zu starten, wird also zunächst die Registerkarte *Lernressourcen* aufgerufen. Darin wählt man im Menü auf der rechten Seite den Punkt *Katalog*, um eine Übersicht der Lernangebote kategorisiert nach den einzelnen Hochschulen und Fachbereichen zu erhalten. Das Lernmodul befindet sich als Kurs „Einführung in die interaktive Arbeit mit Flash MX 2004“ im Ordner *Hochschule für Technik und Wirtschaft Dresden* > *Fachbereich Informatik/Mathematik* (siehe folgende Abbildung).

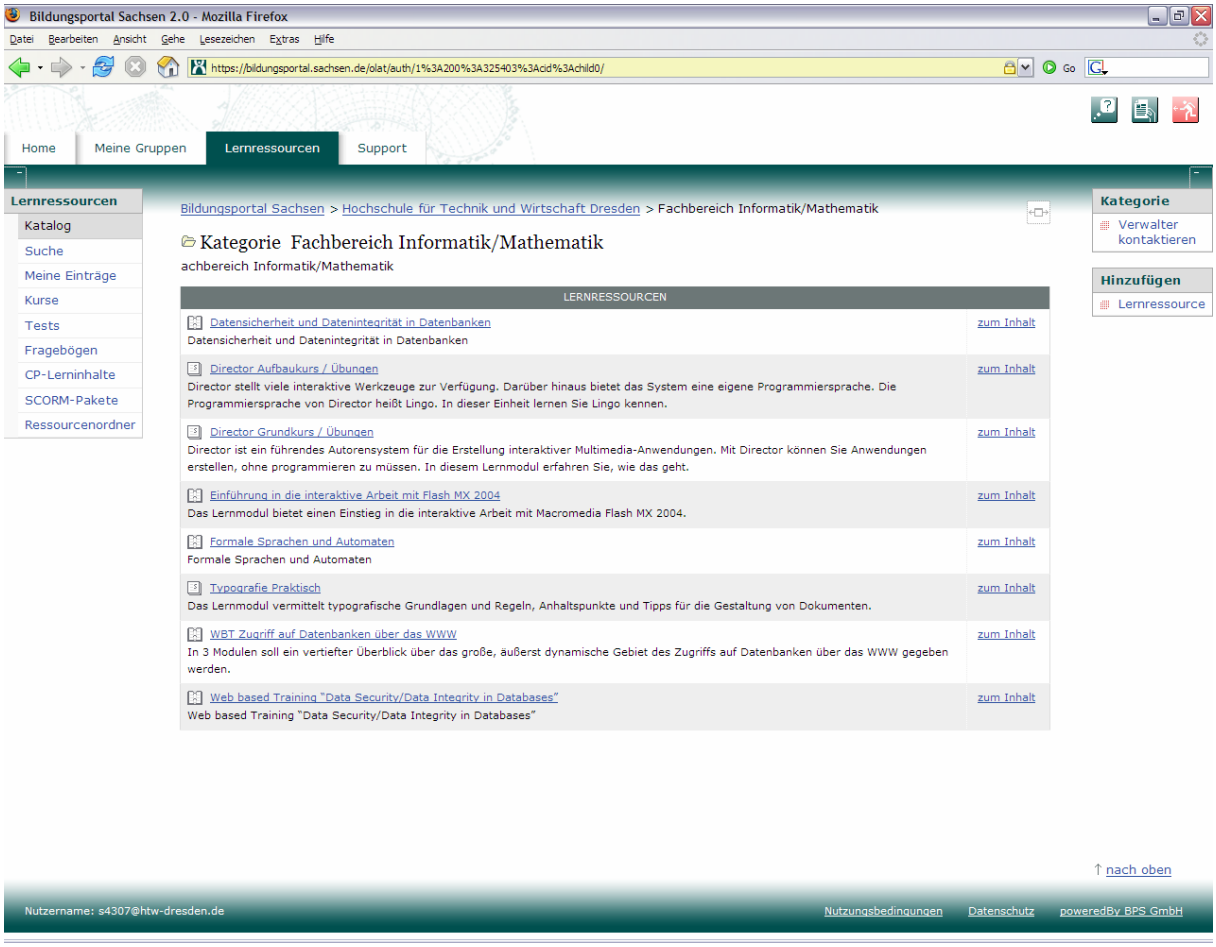

Abbildung 4.7: Die Auswahl des Kurses

Alternativ können natürlich auch die Suche oder der Menüpunkt *Kurse* verwendet werden, um das Lernangebot aufzufinden.

Nach dem Aufruf wird zunächst eine Übersicht inklusive einer kurzen Beschreibung des Inhalts angezeigt.

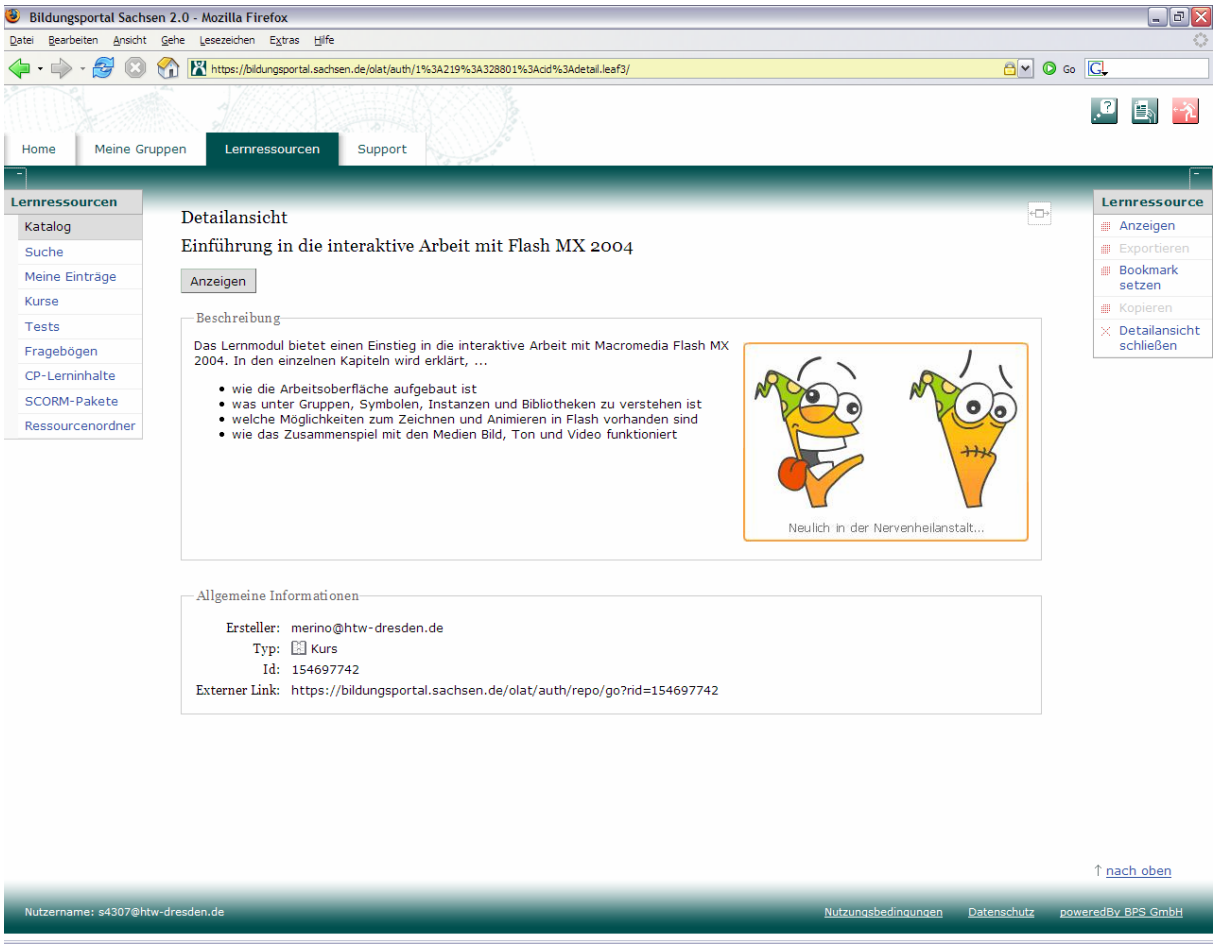

Abbildung 4.8: Übersicht zum Kursinhalt

Über die Schaltfläche Anzeigen gelangt man schließlich zum eigentlichen Kurs. Dieser repräsentiert noch *nicht* das Lernmodul selbst, sondern beschreibt ein Element von OLAT. Ein Kurs ist modular aufgebaut und kann sich aus einem oder mehreren so genannten *Kursbausteinen* zusammensetzen. Dazu zählen beispielsweise SCORM-Lerninhalte, Tests oder auch Foren. Die einzelnen Bausteine sowie der Kurs selbst lassen sich durch die Autoren individuell konfigurieren und somit an gewünschte Bedürfnisse anpassen. Außerdem verfügen alle Elemente über ein eigenes kleines Piktogramm und lassen sich auf diese Weise leichter unterscheiden.

In diesem Fall besteht der Kurs aus dem SCORM-Lerninhalt (das Lernmodul):

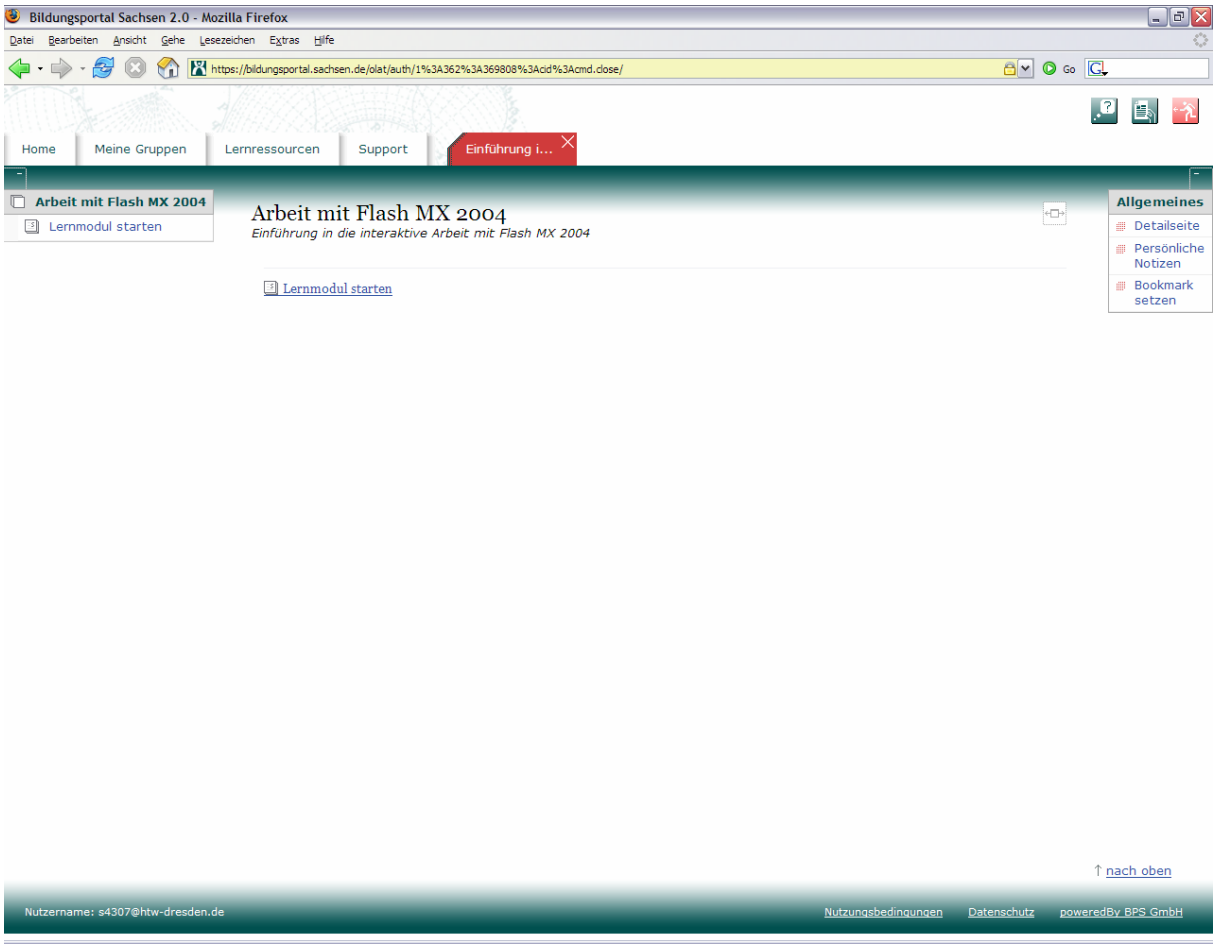

Abbildung 4.9: Kurs mit SCORM-Lerninhalt und Einschreibung

Für den Fall, dass ein SCORM-Lernmodul ohne den Umweg als Kursbaustein veröffentlicht wird, lässt sich dieses in *Lernressourcen* unter dem Menüpunkt *SCORM-Pakete* wiederfinden und problemlos starten. Jedoch werden werden die lernerspezifischen Daten nicht dauerhaft gespeichert und beim Verlassen der Lernanwendung wieder gelöscht.

Das SCORM-Lernmodul kann unter den Modi *Normal*, *Navigation*[84] sowie *Keine Punktevergabe*. Dazu klickt man zunächst auf die Verknüpfung *Lernmodul starten* und wählt im anschließenden Klappmenü den gewünschten Punkt. Die Standardeinstellung lautet *Normal* und sollte für das Lernmodul verwendet werden.

Die Darstellung der Inhalte im BPS ist zum gegenwärtigen Zeitpunkt scheinbar nur in Form eines *IFrame* möglich. Bei der Größe des Lernmoduls von 900 mal 700 Bildpunkten ist diese Variante leider nicht optimal und resultiert oftmals in Anzeige von Scrollbalken. Es lässt sich jedoch zusätzliche Breite gewinnen, indem das Navigationsmenü des LMS ausgeblendet wird. Die folgenden beiden Bildschirmfotos zeigen die Unterschiede:

[84] Das Lernmodul lässt sich ganz normal anschauen, es werden jedoch vom Lernmanagementsystem keine lernerspezifischen Daten erfasst.

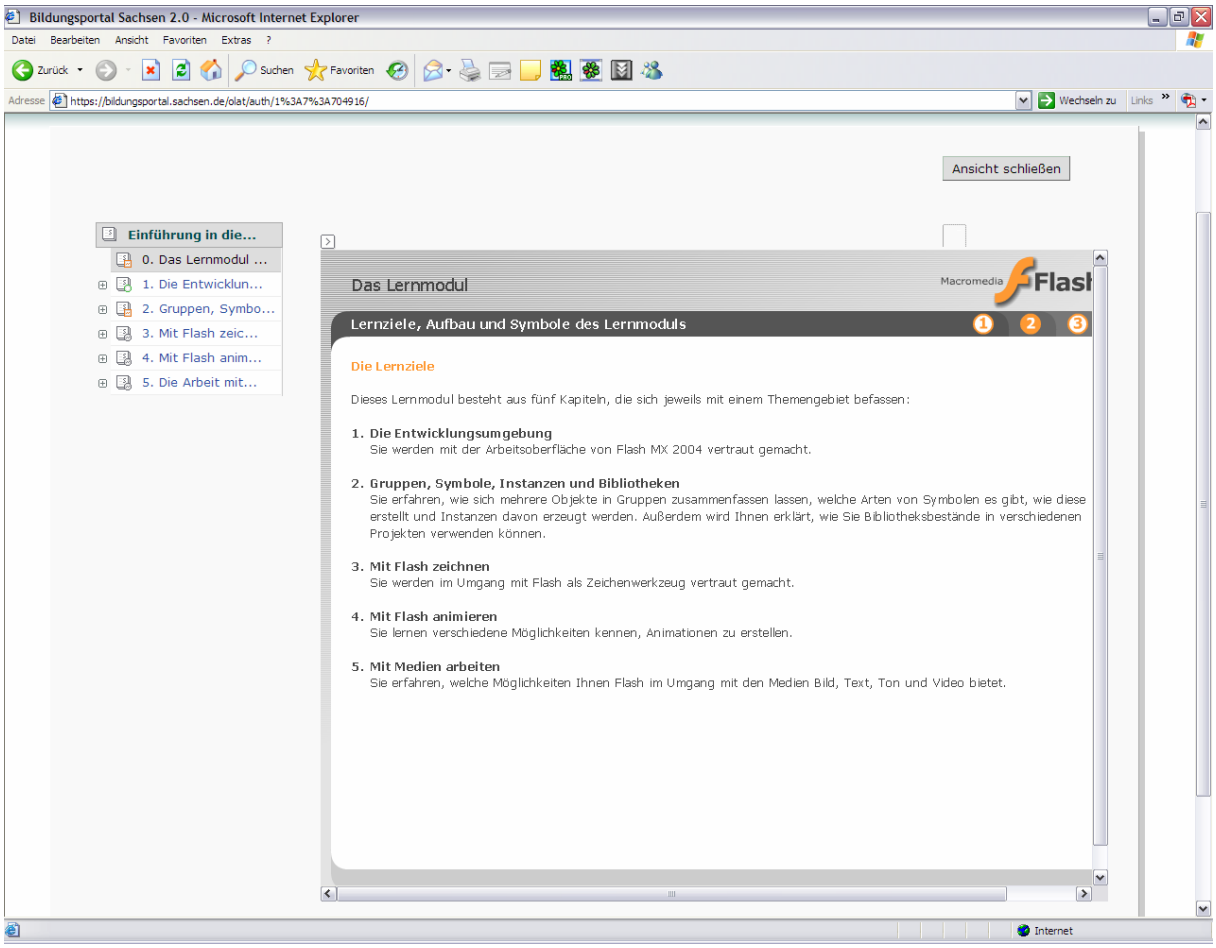

Abbildung 4.10: Die Darstellung der Lerninhalte mit eingeblendetem Navigationsmenü

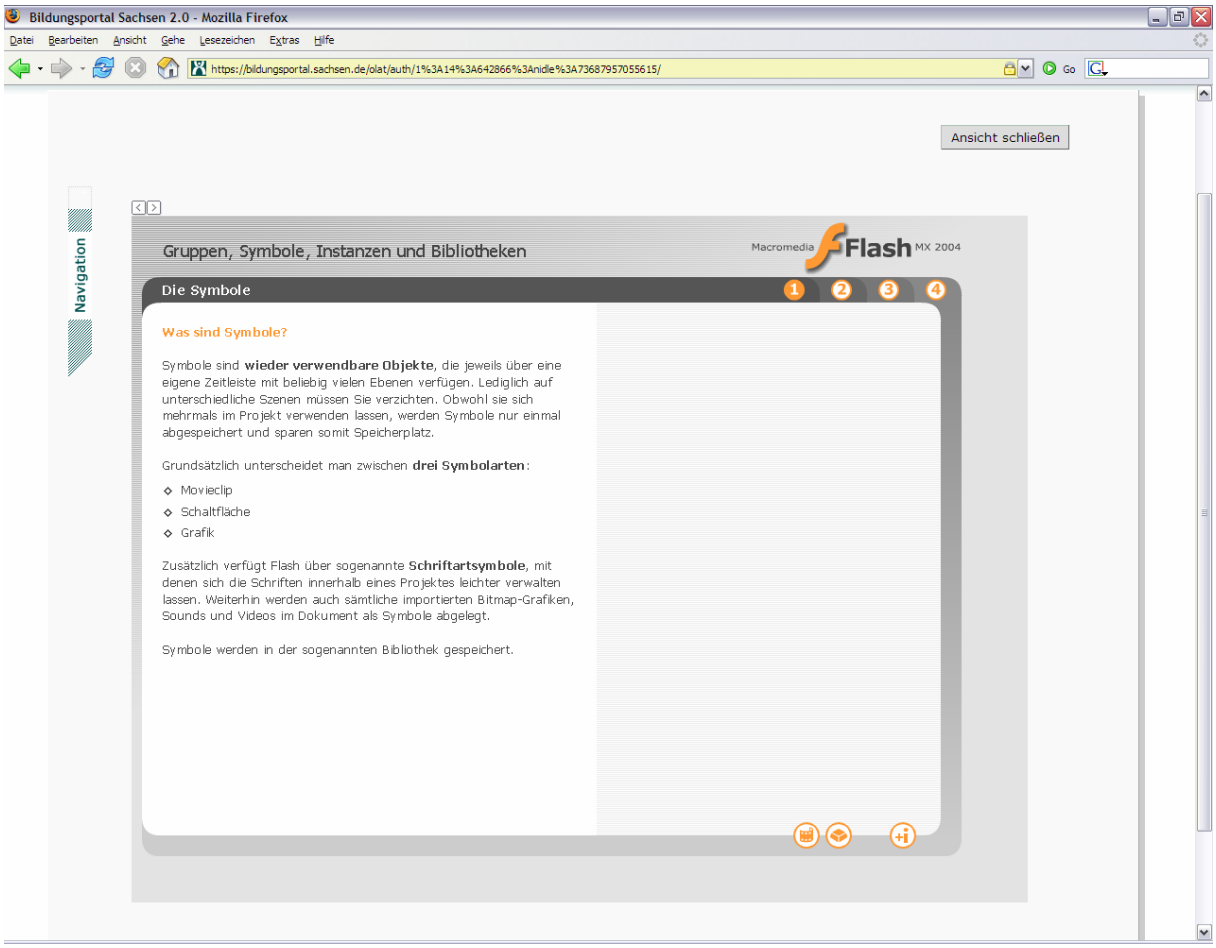

Abbildung 4.11: Die Darstellung der Lerninhalte mit ausgeblendetem Navigationsmenü

Beim Darüberfahren mit der Maus wird das Navigationsmenü zeitweilig eingeblendet und lässt sich verwenden. Zusätzlich befinden sich oberhalb des Lerninhalts zwei Schaltflächen zum Vorwärts- und Zurückbewegen.

Der Bearbeitungsstatus eines SCOs wird anhand kleiner zusätzlicher Piktogramme im Navigationsmenü angezeigt. Ein grüner Kreis bedeutet *completed*, ein kleines rotes Quadrat *visited*. Ist keinzusätzliche Symbol vorhanden, so wurde das Lernobjekt noch nicht aufgerufen.

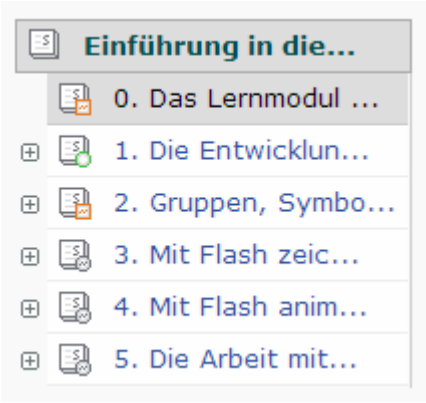

Abbildung 4.12: Darstellung des Bearbeitungsstatus

Obwohl das Lernmanagementsystem OLAT 4 grundsätzlich auch die Darstellung des Inhalts innerhalb eines PopUp-Fensters vorsieht, so wurde diese Möglichkeit scheinbar noch nicht in das BPS implementiert bzw. konnte nicht gefunden werden.

4.2 Tests

Die formative Evaluation sowie der technische Test wurden jeweils von Studenten der HTW Dresden durchgeführt. In den folgenden Abschnitten sollen die daraus gewonnenen Erkenntnisse sowie resultierenden Änderungen am Lernmodul beschrieben werden.

4.2.1 Formative Evaluation

Während der formativen Evaluation wurden bereits lauffähige Lernobjekte hinsichtlich ihrer technischen, didaktischen und gestalterischen Qualitäten getestet. Auf Basis der gewonnenen Erkenntnisse konnten anschließend Maßnahmen ergriffen werden, mit denen sich die aufgedeckten Mängel größtenteils beseitigen ließen. Die formative Evaluation wurde von den durchführenden Studenten in zwei Teile untergliedert: einen *Nutzer-* sowie einen *Expertentest*.

Nutzertest

Der Nutzertest wurde mit sechs Personen durchgeführt, von denen vier keine und jeweils einer grundlegende bzw. fortgeschrittene Kenntnisse im Umgang mit Flash MX 2004 besaßen. Ihnen wurde eine Reihe von Aufgaben gestellt, anhand deren Bearbeitung bzw. Lösung sich Rückschlüsse gewinnen ließen, wie die Nutzer mit dem Lernmodul zurechtkommen. Die Auswertung durch die Testleiter deckte folgende Mängel auf:

- *„Es war eine deutliche Tendenz zuerkennen, dass die verschiedenen symbolhaften Schaltflächen für spezielle Informationen, wie zum Beispiel ‚Zusätzliche Informationen', ‚Nützliche Hinweise' oder ‚Tastaturkürzel', von den Testpersonen*

schwer auseinander zu halten waren bzw. sich die Bedeutung der Symbolik trotz Erläuterung am Anfang des Tutorials nicht einprägte. Häufig war auch zu beobachten, dass die Bedeutung der Schaltflächen falsch interpretiert wurde.“:

Als Maßnahme dessen wurde die ursprüngliche Symbolik der Schaltflächen verworfen und durch eine neu entwickelte ersetzt (siehe Tabelle 4.1). Zudem erfolgt die Anzeige des Tooltipps nach einem deutlich kürzeren Intervall.

Altes Symbol	Neues Symbol	Semantik
		Animation
		Tastaturkürzel
		Nützliche Tipps
		Zusätzliche Informationen

Tabelle 4.1: Alte und neue Symbolik der Schaltflächen

- *„Des Weiteren wurden verschiedene Schaltflächen subjektiv als zu klein empfunden. Auffällig waren hier die Zahlen-Schaltflächen sowohl Schaltflächen, die Ausschnitte aus dem Programm ‚Macromedia Flash MX 2004’ zeigen. Letztere waren bezüglich der Lesbarkeit problematisch.“*

  An besonders kritischen Stellen wurde die Größe einzelner Bildschaltflächen etwas erhöht. Ebenso wurde der Durchmesser der Navigationsschaltflächen erweitert, jedoch mit Rücksicht auf das Gesamtlayout des Lernmoduls nur minimal. Das Problem bezüglich der Lesbarkeit bezog sich – nach Rücksprache mit den Testleitern – speziell auf die im Lernobjekt *Die Arbeitsoberfläche* dargestellten Bildschaltflächen. Bedingt durch die notwendige Skalierung der Bildschirmfotos erschienen die Schriftzeichen deutlich verwaschen und waren kaum zu erkennen. Das Problem ließ sich umgehen, indem sämtliche Schriftzeichen aus der Originalgrafik entfernt und durch Flash-Textfelder mit aktivierter Alias-Text-Option ersetzt wurden (zur Erstellung von Bildschaltflächen vgl. Abschnitt 3.3.3). In den folgenden beiden Abbildungen werden die alte und die neue Variante gegenübergestellt.

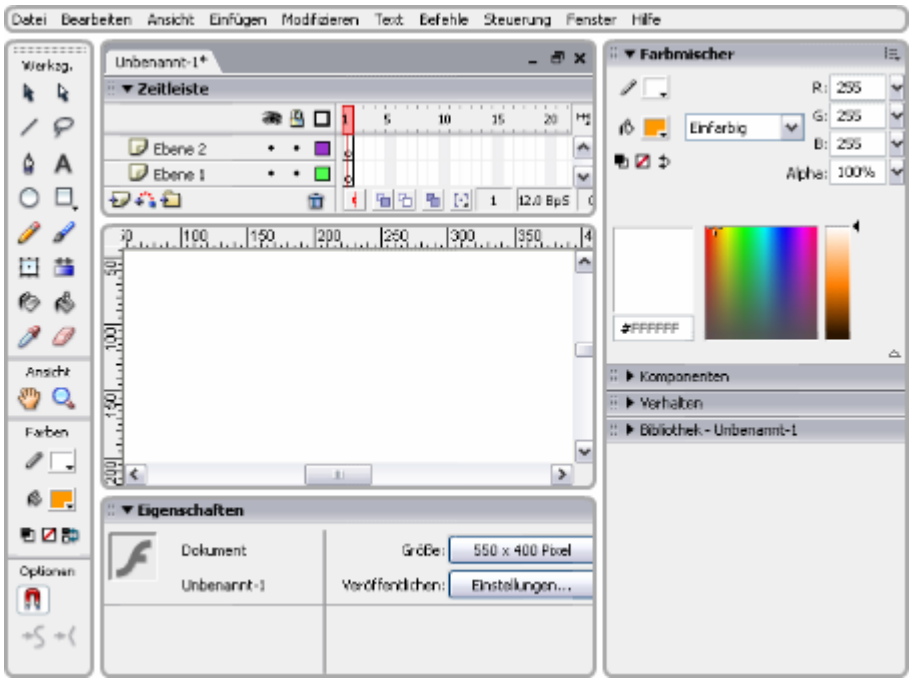

Abbildung 4.13: Alte Version der Bildschaltflächen

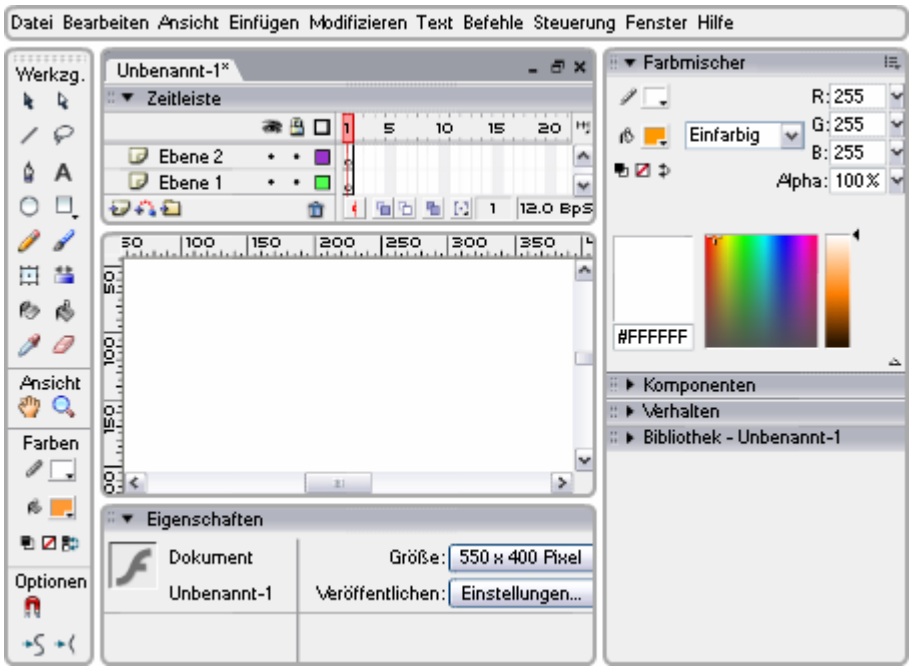

Abbildung 4.14: Neue Version der betreffenden Bildschaltflächen

Expertentest

Der Expertentest wurde in Form einer *heuristischen Evaluation* von den Studenten durchgeführt, die bereits zuvor die Leitung des Nutzertests übernommen hatten. Sie entwickelten dazu einen Kriterienkatalog, der sich mit folgenden Themen auseinandersetzt:

- *Grundlagen der "guten" Gestaltung von Anwendungen (nach Nielsen u.a.)*: Beinhaltet die Punkte *Dialoggestaltung*, *Sprachgestaltung*, *Systemrückkopplung*, *Konsistenz*,

wiedergegebene Inhalte durch die Medien sowie *ästhetischer Anspruch*

- *Benutzermotivation und Dialoggestaltung nach dem ARCS-Modell (nach Keller, 1983)*: Beinhaltet die Punkte *Aufmerksamkeit*, *Relevanz*, *Erfolgszuversicht* und *Zufriedenheit*

Obwohl das Lernmodul im Großen und Ganzen positiv beurteilt wurde, gab es auch einige Kritikpunkte seitens der Experten. Diese werden im Folgenden inklusive der ergriffenen Maßnahmen aufgelistet:

- *Sprachgestaltung*:
  "*Zum einen sind die Tooltips untereinander nicht konsistent benannt, wie zum Beispiel ‚Weitere Bibliotheksarten' im Gegensatz zu ‚Zum Bereich der Bilder'. Hierbei ist eine einheitliche Benennung vorzunehmen (entweder mit ‚Zu…' oder ohne). Zum anderen sollten die Titel der Tooltips mit dem Titel übereinstimmen, auf den sie verweisen (z.B. Tooltip ‚Zum Bereich der Ebenen' im Gegensatz zu Titel des Lernschritts ‚Der Ebenenbereich').*"

  Die Kritik bezog sich auf die Tooltipps der Navigationsschaltflächen. Sie wurden dahingehend abgeändert, dass nunmehr der Hinweis identisch ist mit der Überschrift eines Lernschritts.

- *Systemrückkopplung*:
  "*Bei der Benennung der Kapitel und Unterkapitel in den einzelnen Flashfilmen ist im Sinne einer besseren Systemrückkopplung zu überlegen, ob hier die Nummerierung aus dem Hauptmenü übernommen werden sollte.*"

  Auf die Angabe der Nummerierung wurde explizit verzichtet. Ein SCO sollte im Sinne der Wiederverwendbarkeit möglichst unabhängig vom umgebenden Kontext sein. Die Position innerhalb eines Kurses wird durch die Manifestdatei bestimmt und kann sich im Fall einer Umstrukturierung oder bei einer Verwendung des Lernobjekts in einem anderen Content Package ändern. Dies würde zu unerwünschten Anzeigen führen und hätte jedes Mal eine Überarbeitung des SCOs zur Folge.

  Genaugenommen ist in diesem Zusammenhang auch die Angabe des Kapitelnamens/des Themas bei den entwickelten Lernobjekten fraglich, wurde sie doch auf die inhaltliche Struktur des Lernmoduls abgestimmt und kann in einem anderen Kontext überflüssig oder unkorrekt sein. Die Darstellung wurde jedoch auf Wunsch der Auftraggeberin beibehalten.

- *wiedergegebene Inhalte durch die Medien*:
  „Der Inhalt, insbesondere der Text der Schaltflächen, die Ausschnitte aus dem Programm ‚Macromedia Flash MX 2004' zeigen, sind besonders bei Konfigurationen mit geringer Bildschirmauflösung schlecht erkennbar bzw. lesbar."

  Diese Erkenntnis wurde bereits im Nutzertest gewonnen, weshalb die getroffenen Maßnahmen an dieser Stelle nicht noch einmal erläutert werden sollen.

- *Aufmerksamkeit*:
  „Durch eine verstärkte Konfrontation des Lernenden mit Fragen oder zu lösenden Problemen während der Lektionen könnten jedoch Neugier und Fragehaltungen angeregt bzw. informationssuchendes Verhalten mehr stimuliert werden."

  Auf eine Implementierung von Lernerfolgskontrollen wurde aus zeitlichen Gründen und in Absprache mit der Auftraggeberin verzichtet.

- *Erfolgszuversicht*:
  „Er sollte jedoch über den Inhalt und die Lernziele am Anfang eines jeden Kapitels informiert werden. Es wäre denkbar, dass die einzelnen Kapitel im Hauptmenü auch mit einem Informationsfenster hinterlegt werden, das eine kurze allgemeine Einführung in die jeweilige Thematik und die zu erwartenden Lernziele liefert."

  Das Lernmodul wurde dahingehend erweitert, dass jeder Hauptpunkt der inhaltlichen Struktur ein separates Lernobjekt repräsentiert, in dem die Lernziele kurz vorgestellt werden. Zusätzlich wurde der Name des jeweiligen Punktes um die Endung *„- Lernziele"* ergänzt, um diese Besonderheit stärker kenntlich zu machen. Im Zusammenhang mit dieser Änderung erschien auch die Zuweisung der Kapitelnummer 0 für die Einleitung (davor Punkt 0.1) als angebracht.

4.2.2 Technischer Test

Der technische Test wurde ebenfalls von einer Studentin der HTW Dresden durchgeführt. Darin wurde die Lauffähigkeit im Bildungsportal Sachsen unter verschiedenen System- und Internetplattformen geprüft. Die Ergebnisse wurden in tabellarischer Form zusammengefasst und werden nachfolgend dargestellt. Es muss jedoch erwähnt werden, dass diese fälschlicherweise aus dem Test mit einer veralteten, jedoch noch im BPS vorhandenen Version des Lernmoduls resultieren.

lfd. Nr.	Plattform	Browser	Flash Player	Internet-verbindung	Ergebnis
1	PC / WinXP	Internet Explorer 6.0	Version 7	10,0 MBit/s (Standleitung)	läuft ohne Probleme; aber: im Inhalts-verzeichnis wird der Abschlussstatus in der Regel nicht korrekt angezeigt
2	PC / WinXP	Internet Explorer 6.0	Version 7	DSL	wie oben; Ladezeiten teilweise ein bisschen länger
3	PC / WinXP	Mozilla Firefox 1.07	Version 7	10,0 MBit/s (Standleitung)	in einigen Fällen wird der Flash-Film im Lernmodul nicht geladen ; abgesehen davon: ohne Probleme, Abschlussstatus wird korrekt angezeigt
4	MAC / OS X 10.4	Safari 2.0	Version 7	10,0 MBit/s (Standleitung)	kein Test möglich, Flash-Film wird nicht geladen (wie bei Nr.3)

Tabelle 4.2: Ergebnisse des technischen Tests

Die Probleme im Internet Explorer resultierten – auch bei der in diesem Test nicht berücksichtigten aktuelleren Version des Lernmoduls – aus der Verwendung von `getURL()` zum Aufruf von JavaScript-Funktionen aus Flash heraus. Der Fehler trat nicht mehr auf, nachdem (wie in Abschnitt 4.1.1 beschrieben) eine Browser-abhängige Verwendung von `fscommand()` implementiert wurde.

Im Fall des Nichtladens der Flash-Filme unter Firefox bzw. Safari war das Problem auf das veraltete Lernmodul zurückzuführen. Bei einem erneuten Test mit der zum damaligen Zeitpunkt aktuellen Programmversion konnten die Fehler nicht reproduziert werden.

4.2.3 Summative Evaluation

Die summative Evaluation wurde von Medieninformatik-Studenten innerhalb der Lehrveranstaltung „Entwicklungswerkzeuge für Multimediasysteme“ durchgeführt und fand zu Beginn des Sommersemesters 2006 statt. Ziel war es, den Einsatz des kompletten Lernmoduls unter Praxisbedingungen zu testen und eventuelle Schwachstellen aufzudecken.

Zu diesem Zweck erarbeiteten die Tester einen Fragenkatalog, um zu ermitteln, wie die Mitstudenten bei der Arbeit mit dem Programm zurechtkamen und wie sie die inhaltliche sowie gestalterische Qualität beurteilten.

Die meisten Studenten besaßen zu Beginn keine oder wenige Vorkenntnisse in Flash. Die Nutzung der Lernanwendung erfolgte zu nahezu gleichen Teilen begleitend zur Lehrveranstaltung sowie vom heimischen Arbeitsplatz aus.

Die Auswertung der beantworteten Fragen durch die Tester förderte eine im Durchschnitt positive Bewertung des Lernmoduls zu Tage. Jedoch wurden von den Studenten auch einige Kritikpunkte bwz. Verbesserungsvorschläge genannt, die in anschließender Tabelle zusammengefasst aufgeführt werden.

Kritikpunkt/Verbesserungsvorschlag	**Anmerkung**
Es sollte weniger rein theoretisches Wissen bzw. dieses mehr auf praktischerem Wege, z.B. anhand weiterer Animationen oder Workshops, vermittelt werden.	Dieser Kritikpunkt lässt sich nachvollziehen, jedoch spielten bei der Entwicklung zeitliche sowie finanzielle Faktoren eine nicht unwesentliche Rolle. Gerade das Erstellen der Animationen erfordert einen ungleich höheren Aufwand, als das entsprechende Wissen in Textform zu präsentieren.
Umfangreichere Workshops, die sich mehrere Theorieeinheiten ziehen.	Größere Workshops zu erstellen und diese anhand semantisch miteinander verknüpfter Lernobjekte zu präsentieren, erscheint im Sinne der SCORM-Konformität nicht ratsam. SCOs sollten immer einen vom restlichen Kontext unabhängigen Lerninhalt darstellen.
Die Komplexität der Workshops sollte erhöht werden.	Von einer höheren Komplexität, deren Fehlen einige Studenten als negativ bewerteten, wurde bewusst abgesehen. Das Lernmodul soll einen Einstieg in Flash darstellen und richtet sich in erster Linie an unerfahrene Benutzer. Die Workshops dienen demnach zur Vermittlung grundlegender Prinzipien, ohne den Anwender mit zu komplexen Details zu überfordern.

Interaktive Tests sollten eingebaut werden.	Dies wurde schon im Expertentest der formativen Evaluation als negativ bewertet. Wie bereits erwähnt, spielten in diesem Fall ebenso zeitliche wie finanzielle Faktoren eine Rolle.
Das Lernmodul sollte über einen Index sowie eine Verlinkung der einzelnen Lernobjekte untereinander verfügen, um zuvor vermitteltes Wissen auffrischen bzw. nachschlagen zu können.	Diese Möglichkeit war von Anfang an im Sinne der SCORM-Konformität nicht vorgesehen. An dieser Stelle macht sich ein Nachteil von SCORM bemerkbar, denn gerade bei umfangreichen Lernkomplexen wäre eine solche Erweiterung durchaus sinnvoll.
In den theoretischen Lerneinheiten wären Literaturhinweise wünschenswert.	Dieser Kritikpunkt sollte bei zukünftigen Projekten berücksichtigt werden.

Anhang A – Inhaltliche Gliederung des Lernmoduls

SCO 0 Das Lernmodul – Einführung

Lernschritt 1 Willkommen

Lernschritt 2 Die Lernziele

Lernschritt 3 Der Aufbau des Lernmoduls

Lernschritt 4 Die verwendeten Symbole

SCO 1 Die Entwicklungsumgebung – Lernziele

Lernschritt 1 Lernziele

SCO 1.1 Die Arbeitsoberfläche

Lernschritt 1 Überblick

SCO 1.2 Die Werkzeugleiste

Lernschritt 1 Überblick

Lernschritt 2 Die Werkzeuge

SCO 1.3 Die Zeitleiste

Lernschritt 1 Überblick

Lernschritt 2 Der Ebenenbereich

Lernschritt 3 Der Bildbereich

Lernschritt 4 Die Szenen

SCO 2 Gruppen, Symbole, Instanzen und Bibliotheken - Lernziele

Lernschritt 1 Lernziele

SCO 2.1 Die Gruppen

Lernschritt 1 Was sind Gruppen?

SCO 2.2 Die Symbole

Lernschritt 1 Was sind Symbole?

Lernschritt 2 Das Movieclip-Symbol

Lernschritt 3 Das Schaltflächen-Symbol

Lernschritt 4 Das Grafik-Symbol

SCO 2.3 Die Instanzen

Lernschritt 1 Was sind Instanzen?

Lernschritt 2 Die Instanzeigenschaften

SCO 2.4 Die Bibliotheken

Lernschritt 1 Das Bedienfeld »Bibliothek«

Lernschritt 2 Weitere Bibliotheksarten

SCO 3 Mit Flash zeichnen - Lernziele

Lernschritt 1 Lernziele

SCO 3.1 Das Mischen der richtigen Farbe

Lernschritt 1 Das Bedienfeld »Farbmischer«

SCO 3.2 Workshop: Eine einfache 3D-Schaltfläche zeichnen

Lernschritt 1 Überblick

Lernschritt 2 Ein leeres Schaltflächen-Symbol anlegen

Lernschritt 3 Kreisformen zeichnen

Lernschritt 4 Bearbeiten der Kreisformen

Lernschritt 5 Beschriftung der Schaltfläche

Lernschritt 6 Transformation des Textes

Lernschritt 7 »Beleuchtung« des Textes

Lernschritt 8 Fertigstellung der Schaltfläche

SCO 3.3 Workshop: Ein Stillleben zeichnen

Lernschritt 1 Überblick

Lernschritt 2 Von Birnen und anderen Früchten

Lernschritt 3 Grundformen zeichnen

Lernschritt 4 Bearbeiten der Grundformen

Lernschritt 5 Fertigstellung des Stilllebens

SCO 4 Mit Flash animieren - Lernziele

Lernschritt 1 Lernziele

SCO 4.1 Animationen in Flash

Lernschritt 1 Überblick

Lernschritt 2 Zeitleisteneffekte

SCO 4.2 Workshop: Einen Bewegungs-Tween erstellen

Lernschritt 1 Überblick

Lernschritt 2 Zeichnen oder Herunterladen – das ist hier die Frage…

Lernschritt 3 Eine Bewegung erstellen

Lernschritt 4 Verfügbare Optionen

Lernschritt 5 Skalierung und Rotation

Lernschritt 6 Download

SCO 4.3 Workshop: Eine Pfad-Animation erstellen

Lernschritt 1 Überblick

Lernschritt 2 Eine Pfadebene anlegen

Lernschritt 3 Einen Pfad zeichnen

Lernschritt 4 Ausrichten des Objekts

Lernschritt 5 Die fertige Pfad-Animation

SCO 4.4 Workshop: Einen Form-Tween erstellen

Lernschritt 1 Überblick

Lernschritt 2 Zahlen erstellen

Lernschritt 3 *Ein einfacher Form-Tween*
Lernschritt 4 *Verfügbare Optionen*
Lernschritt 5 *Steuerung der Animation*
Lernschritt 6 *Formmarken hinzufügen*
Lernschritt 7 *Der fertige Form-Tween*

SCO 5 Die Arbeit mit Medien - Lernziele

Lernschritt 1 *Lernziele*

SCO 5.1 Medien in Flash

Lernschritt 1 *Überblick*
Lernschritt 2 *Zeitleisteneffekte*

SCO 5.2 Vektor- und Pixelgrafiken

Lernschritt 1 *Import von Bildern*
Lernschritt 2 *Konvertierung von Pixel- in Vektorgrafiken*
Lernschritt 3 *Bitmaps als Textur*
Lernschritt 4 *Export des Flash-Films*

SCO 5.3 Textfelder

Lernschritt 1 *Überblick*
Lernschritt 2 *Darstellung von Schriften*
Lernschritt 3 *HTML-Unterstützung*
Lernschritt 4 *Textfelder*

SCO 5.4 Sounds

Lernschritt 1 *Import von Sounds*
Lernschritt 2 *Verwendung von Sounds*

SCO 5.5 Workshop: Ein Schaltfläche mit Sound

Lernschritt 1 *Überblick*
Lernschritt 2 *Sound einfügen*
Lernschritt 3 *Soundeinstellungen anpassen*
Lernschritt 4 *Ruhe bitte…*

SCO 5.6 Videos

Lernschritt 1 *Import von Videos*
Lernschritt 2 *Der Videoimport-Assistent: Das Schnittfenster*
Lernschritt 3 *Der Videoimport-Assistent: Die Kodierung*
Lernschritt 4 *Der Videoimport-Assistent: Die Verschlüsselung*
Lernschritt 5 *Verwendung von Videos*

Anhang B – Der Styleguide für die HTML-Textinhalte

Textelement	Schriftart	Größe	Farbe	Besonderheiten
Kapitelüberschrift/Thema	Tahoma	18	#444444	Fett
Zweitüberschrift/ Name des Lernobjekts	Tahoma	13	#FFFFFF	Fett
Überschrift im Haupt-bereich	Tahoma	12	#FF9933	Fett
Normaler Text	Tahoma	12	#444444	Normal
Hervorgehobener Text	Tahoma	12	#444444	Fett
Überschrift im Zusatzbereich	Tahoma	12	#444444	Fett, Zentriert
Auflistung 1.Ebene	Tahoma	12	#444444	Listenpunkt ist ein Karo-Zeichen (‚◇', 0x25CA); Text wird um 20 Punkte eingerückt
Auflistung 2.Ebene - Listenpunkt	Wingdings	13	#444444	Listenpunkt ist ein kleines Karo-Zeichen (‚◆', 0x77)
Auflistung 2.Ebene - Text	Tahoma	12	#444444	Text wird um weitere 20 Punkte eingerückt (insgesamt 40)
Auflistung von Beschreibungen, Anzeige mehrerer Tipps gleichzeitig	Tahoma	12	#444444	Text auf der Zeile des Listenpunkts ist fettgedruckt Beispiel 1: ◇ **Das Beschriebene** Beschreibung… ◇ **Das Beschriebene** Beschreibung… Beispiel 2:

				◇ **Tipp 1** Der erste Tipp… ◇ **Tipp 2** Der nächste Tipp…
Flash-Menüs und -Menüpunkte, Optionen, Tastenkürzel	Tahoma	12	#444444	Großbuchstaben Beispiel: Wählen Sie im Menü EINFÜGEN den Punkt NEUES SYMBOL oder drücken Sie STRG+F8.
Verzweigte Menü-Aufrufe	Tahoma	12	#444444	Großbuchstaben, Trennung durch ‚»' Beispiel: ANSICHT»LINEALE
Schriftart, Titel eines Flash-Fensters oder Bedienfeldes	Tahoma	12	#444444	Eingefasst durch ‚»' und ‚«' Beispiel: »Arial« »Neues Symbol erstellen«
ActionScript-Befehle	Courier	12	#444444	Normal
Schlüsselwörter, Bezeichner und Strings in Quellcode-Beispielen	Courier	12	#FF9933	Normal; Fett nur im Fall einer speziellen Hervorhebung

Abkürzungsverzeichnis

ADL	Advanced Distributed Learning
AGR	AICC Guidelines and Recommendations
AICC	Aviation Industry Computer-Based Training Committee
ANSI	American National Standards Institute
API	Application Programming Interface
ARIADNE	Alliance of Remote Instructional Authoring and Distribution Networks for Europe
AS	ActionScript
BPS	Bildungsportal Sachsen
CAM	Content Aggregation Model
CBT	Computer Based Training
CD-ROM	Compact Disc Read-Only Memory
CEN	Comité Européen de Normalisation
CMI	Computer Managed Instructions
CSS	Cascading Style Sheet
DIN	Deutsches Institut für Normung
DOM	Document Object Model
DSL	Digital Subscriber Line
DTD	Document Type Definition
DVD	Digital Versatile Disc
ECMA	European Computer Manufacturers Association
EU	Europäische Union
FH	Fachhochschule
HTML	Hypertext Markup Language

HTTP	Hypertext Transfer Protocol
HTW	Hochschule für Technik und Wirtschaft
IEEE	Institute of Electric and Electronic Engineers
IMS	Instructional Management Systems
ISO	International Organization for Standardization
JAR	Java Archive
JPEG	Joint Photographic Experts Group
LCMS	Learning Content Management System
LMS	Learning Management System
LO	Learning Object
LOM	IEEE Standard for Learning Objective Metadata
LTSC	Learning Technology Standards Committee
NISO	National Institute of Standards and Technology
OOP	Object-Oriented Programming
OSTP	Office of Science and Technology Policy
PIF	Package Interchange File
PNG	Portable Network Graphic
RLO	Reusable Learning Object
RTE	Run-Time Environment
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
SGML	Standard Generalized Markup Language
SWF	Shockwave Flash Format
URL	Uniform Ressource Locator
W3C	World Wide Web Consortium
WBT	Web Based Training

WYSIWYG	What You See Is What You Get
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Quellcode-Beispiele

Literaturverzeichnis

[AICC] *Aviation Industry CBT Committee.* http://www.aicc.org [Stand 28.06.2006]

[ARIADNE] *ARIADNE Foundation for the European Knowledge Pool.* http://www.ariadne-eu.org [Stand 28.06.2006]

[Baumgartner&Häfele 2002] Peter Baumgartner, Hartmut Häfele und Kornelia Maier-Häfele: *E-Learning Praxishandbuch. Auswahl von Lernplattformen. Marktübersicht – Funktionen – Fachbegriffe.* StudienVerlag, Innsbruck 2002.

[IEEE] *IEEE LTSC.* http://www.ieee.org [Stand 28.06.2006]

[IMS] *Global Learning Consortium, Inc.* http://www.imsproject.org [Stand 28.06.2006]

[Kerres 2001] Prof.Dr.Michael Kerres, Ruhr-Universität Bochum: *Multimediale und telemediale Lernumgebungen. Konzeption und Entwicklung.* Oldenbourg Wissenschaftsverlag GmbH, 2001.

[Montandon 2004] Corinne Montandon: *Standardisierung im e-Learning. Eine empirische Untersuchung an Schweizer Hochschulen.* Arbeitsbericht 161, Institut für Wirtschaftsinformatik der Universität Bern, August 2004.

[Naumann u.a 2005] Steffen Naumann, Christian Schulz, Katrin Helth, Jessica Thiersch: *Lesbarkeit am Bildschirm. Projektbericht zur Untersuchung.* Projektbericht an der Hochschule für Technik und Wirtschaft Dresden, 2005.

[Niegemann u.a. 2004] Prof.Dr.Helmut M. Niegemann, Silvia Hessel, Dirk Hochscheid-Mauel, Kristina Aslanski, Markus Deimann und Gunther Kreuzberger: *Kompendium E-Learning.* Springer-Verlag Berlin Heidelberg 2004.

[OLAT] *Open Source LMS OLAT - eLearning made easy.* http://www.olat.org/public/index.html [Stand 28.06.2006]

[Schreiber 1998] Prof.Dr.Alfred Schreiber: *CBT-Anwendungen professionell entwickeln.* Springer-Verlag Berlin Heidelberg, 1998.

[Schulmeister 2003] Prof.Dr.Rolf Schulmeister, Universität Hamburg: *Lernplattformen für das virtuelle Lernen. Evaluation und Didaktik.* Oldenbourg Wissenschaftsverlag GmbH, 2003.

[SCORM 2001a] *Sharable Object Reference Model. Version 1.2. The Overview.* Advanced Distributed Learning (ADL), 2001. http://www.adlnet.org/SCORM/History/12/documents.cfm [Stand 28.06.2006]

[SCORM 2001b] *Sharable Object Reference Model. Version 1.2. The SCORM Content Aggregation Level.* Advanced Distributed Learning (ADL), 2001. http://www.adlnet.org/SCORM/History/12/documents.cfm [Stand 28.06.2006]

[SCORM 2001c] *Sharable Object Reference Model. Version 1.2. The SCORM Run-Time Environment.* Advanced Distributed Learning (ADL), 2001. http://www.adlnet.org/SCORM/History/12/documents.cfm [Stand 28.06.2006]

[SCORM 2004a] *Sharable Content Object Reference Model. SCORM Run-Time Environment. Version 1.3.1.*
Advanced Distributed Learning (ADL), 2004.
http://www.adlnet.org/downloads/70.cfm
[Stand 28.06.2006]

[SCORM 2004b] *Sharable Content Object Reference Model. SCORM Sequencing and Navigation. Version 1.3.1.*
Advanced Distributed Learning (ADL), 2004.
http://www.adlnet.org/downloads/70.cfm
[Stand 28.06.2006]

[Wagner 2005] Kerstin Wagner: *Konzeption und Entwicklung eines Lernmoduls für das Bildungsportal Sachsen gemäß des E-Learning Standards SCORM 1.2*
Diplomarbeit an der Hochschule für Technik und Wirtschaft Dresden, 2005.

[Wendt 2003] Matthias Wendt: *Praxishandbuch CBT und WBT – Konzipieren, entwickeln, gestalten.*
Carl Hanser Verlag München Wien 2003.

[Wikipedia 2006] *Wikipedia – Die freie Enzyklopädie.*
http://de.wikipedia.org/wiki/Standard [Stand 28.06.2006]

Selbstständigkeitserklärung

Ich versichere, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Dresden, den 28.06.2006

Torsten Rülke