Courseman: Eine Webapplikation zur Erstellung und Verwaltung von komplexen XML Konfigurationsdateien für eine Lernumgebung

Semesterarbeit im Fach Informatik

vorgelegt von

Tibor Dekany,
Illnau−Effretikon, Schweiz
Matrikelnummer: 96−908447

angefertigt am
Institut für Informatik
der Universität Zürich

Prof. Dr. K. R. Dittrich

Betreuung: Anca Dobre

Abgabe der Arbeit: 18.6.2002

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Ausgangslage

Lernanwendungen im Internet gewinnen heute immer mehr an Bedeutung, sowohl in der Industrie, als auch im universitären Bereich. Im Jahr 2001 ist an der Universität Zürich durch die Diplomarbeit von Robert Milic [MIL01] ein Prototyp einer Lernumgebung entwickelt worden. In dieser Lernumgebung können Studenten virtuelle Kurse belegen, indem sie über das Internet zur Verfügung gestellte Lerneinheiten bearbeiten können. Zu dieser Anwendung soll eine weitere Hilfsapplikation erstellt werden, mit welchem solche Lernkurse aus einzelnen, vorhandenen Kurselementen zusammengesetzt werden können.

1.2 Aufgabenstellung

In der vorhandenen Lernumgebung werden komplexe Lernkurse in Form von grossen, monolithischen XML[1] Konfigurationsdateien eingegeben, in welcher die elementaren Komponenten (HTML[2]− und Bilddateien) angegeben sind. Diese Konfigurationsdatei ist aber nicht sehr handlich, und es stellt sich als äusserst unpraktisch heraus, Kurse manuell in dieser Datei zusammenzustellen. Daher soll Courseman diese Lücke füllen, damit die auf dem Internet bereits vorhandenen HTML− und Bilddateien zu einfachen XML Objekten erfasst werden können, zu sogenannten simplen Lernobjekten. Courseman soll diese XML Objekte in einer Datenbank erfassen, ändern und löschen können. Diese Objekte sollen mit Courseman verwaltet werden können.

Mit Hilfe dieser Datenbank soll es nun möglich sein, komplette Kursen aus den vorhandenen Elementen zusammenzusetzen. Als Endergebnis liefert diese Applikation eine vollständige XML Konfigurationsdatei, welche als Eingabe für die Lernumgebung dienen kann.

1.3 Voraussetzungen

Vorgegeben ist, dass in dieser Applikation die selben Technologien angewendet werden sollen, wie dies auch im Prototyp der Lernumgebung der Fall ist. Die Software ist somit in der Programmiersprache Java zu entwickeln und soll als Webapplikation auf einem entsprechendem Applikationserver lauffähig sein. Damit ist gegeben, dass Courseman plattformunabhängig ist. Als persistenten Speicher soll die objektorientierte Datenbank FastObjects von POET eingesetzt werden.

Obwohl diese Software von Milics Prototyp unabhängig ist, wird hiermit eine eventuell spätere Integration beider Prototypen stark erleichtert, da die selbe Infrastruktur benötigt wird.

Für diese Arbeit wird somit folgendes Wissen vorausgesetzt:

- Erfahrung mit der Programmiersprache Java
- Grundlegende Kenntnis über Datenbank− und XML Technologien
- Verständnis der Systemarchitektur einer Webapplikation.

1 eXtensible Markup Language
2 Hyper Text Markup Language

1.4 Persönliche Ziele

Informatik ist ein Gebiet, das mich sehr stark interessiert. Im Selbststudium habe ich bereits viel Erfahrung sammeln können, unter anderem im Bereich Betriebssysteme, Netzwerke und Computerarchitektur.

Die Entwicklung komplexer Webapplikationen und der Umgang mit Datenbanken blieben bis jetzt aber aussen vor, da Experimente in diesem Bereich sehr aufwändig sind bzw. es liessen sich für den Privatgebrauch keine sinnvollen Anwendungen finden, welche den Aufwand für solche Systeme rechtfertigen würden. Auch meine bisherigen Erfahrungen mit der Programmiersprache Java gehen nicht viel über das während dem Studium der Wirtschaftsinformatik Erlernte hinaus. Mein persönliches Ziel war es nun, mich nicht nur mit einzelnen mir neuen Technologien auseinander zu setzen (wie Datenbanken, Applikationsserver, Java), sondern diese auch miteinander zu einem Gesamtsystem verbinden zu können. Daher kommen in dieser Arbeit verschiedene neue Technologien wie Java Servlets, JSP, objektorientierte Datenbanken und XML zum Einsatz.

1.5 Gliederung der Dokumentation

Diese Arbeit ist hauptsächlich in drei Abschnitte eingeteilt: Abschnitt 2 behandelt die Funktionalität von *Courseman*, indem es zunächst in den Gesamtkontext eingeordnet wird, wonach die genauen Anforderungen an *Courseman* beschrieben wird. Im darauf folgendem Abschnitt wird die zugrunde liegende Datenstruktur der XML Konfigurationsdatei erklärt. Zum Schluss wird gezeigt, wie *Courseman* von der Benutzersicht her aussieht und wie es angewendet wird.

Abschnitt 3 behandelt die angewendeten Technologien. Zunächst wird die verwendete Programmiersprache Java besprochen und ein Überblick über die Technologien Applet, Servlet, und JSP gegeben. Danach wird die hier eingesetzte, objektorientierte Datenbank diskutiert, wie diese funktioniert und in wie diese bei der Programmierung angewendet wird. Zum Schluss wird XML und der Umgang damit behandelt, insbesondere, wie es mit *XML Data Binding* zusammen mit der Programmiersprache angewendet werden kann.

Im Abschnitt 4 werden schliesslich die Architektur von *Courseman* diskutiert und einige Details der Programmierung gezeigt.

Der 5. Abschnitt stellt nicht nur die erreichten Ziele dieser Arbeit vor, sondern deckt auch Probleme und Fehler auf und gibt Ausblicke, wie der Prototyp *Courseman* zu einem vollständig System weiterentwickelt werden könnte.

Abschnitt 6 gibt eine kurze Zusammenfassung der gesamten Arbeit.

Im Anhang A ist die verwendete DTD Datei und im Anhang B der ganze Verzeichnisbaum des Projektverzeichnisses von *Courseman* angegeben.

2 Funktionalität von Courseman

In diesem Abschnitt soll dargestellt werden, in welchem Kontext dieses Projekt Namens *Courseman* steht, welchen Anforderungen das Programm genügen muss, und wie der Endbenutzer diese Plattform gebrauchen kann. *Courseman* steht für "Kurs Manager", da mit der Applikation Kurse verwaltet werden sollen.

2.1 Kontext

Wie schon in der Einleitung erwähnt, existiert ein Prototyp eines computergestützten Lernverwaltungssystems Namens *Calmecac*, welcher 2001 im Rahmen der Diplomarbeit von Robert Milic [MIL01] am Institut für Informatik der Universität Zürich erstellt wurde. Ein Lernverwaltungssystem ist grundsätzlich eine System, welches benutzt wird, um Lernprozesse zu planen, zu organisieren, zu implementieren und zu steuern. Ein vollständiges Lernverwaltungssystem bietet somit eine Plattform, mit welcher Lerninhalte erstellt und wiedergeben werden können. Des weiteren werden damit die Benutzer und deren Rollen (Student, Assistent, Tutor, etc.) nicht nur verwaltet, sondern auch die Leistung der Studenten erfasst, z.B. welche Lerninhalte bereits bearbeitet wurden oder wie viele Punkte in Übungen erzielt wurden. Ausserdem kann so ein System weitere Möglichkeiten zur Interaktion zwischen den Benutzern bieten, sowohl die Kommunikation zwischen den Studenten, als auch zwischen Studenten und Tutoren, z.B. mit thematisch gegliederte Diskussionsforen, Chaträumen, Videokonferenzen oder Mailinglisten.

Calmecac kann zwar mit vordefinierten Lerninhalten umgehen, aber es ist nicht möglich, solche zu erstellen, zu ändern, gescheige denn zu verwalten. Diese Lücke soll die Applikation *Courseman* füllen.

Was versteht man nun unter einem Lerninhalt? Ein Lerninhalt ist das, was gelernt werden soll, um ein Lernziel zu erreichen. Das kann ein vollständiger Kurs sein, was z.B. einem zu belegendem Fach in einem Semester entspricht, aber auch nur ein Teil davon sein kann. Denn ein Kurs kann aufgeteilt werden in mehrere Module, welche wieder in sich abgeschlossene Lerninhalte mit eigenen Lernzielen darstellen. Module entsprechen somit den einzelnen Lektionen eines Semesters. Module werden weiter unterteilt auf die einzelnen darzustellenden Seiten in der Lernumgebung, den einzelnen Lerneinheiten, sogenannte zusammengesetzte Lernobjekte (CLO). Schlussendlich besteht so eine Lerneinheit aus den simplen Lernobjekten (SLO), d.h. den einzelnen Bildern, Texten, Videos, Übungsfragen. Diese Lernobjekte sind somit das effektiv vorhandene Lernmaterial. Diese alleine sind aber nutzlos, um diese zu einem Kurs zusammenzufügen, müssen diese Daten mit Hilfe von Metadaten beschrieben werden. Metadaten sind Daten über Daten, somit können die vorhandenen, unstrukturierten Lernobjekte einheitlich beschrieben werden und dadurch effizient verwaltet werden. Lerninhalte sind demnach Metadaten über Lernobjekte.

2.2 Anforderungen an Courseman

Courseman soll nun ein Prototyp einer eigenständigen Hilfsapplikation zu *Calmecac* sein, welche zunächst unabhängig funktionieren soll. Mit Hilfe von *Courseman* sollen zu *Calmecac* kompatible Kurse erstellt und verwaltet werden können. Dabei ist davon auszugehen, dass die Lernobjekte (Bilder, Texte, etc.) bereits vorhanden sind und die Dateien mit Hilfe

einer URL[3] über das Internet erreichbar sind. Mit Hilfe von *Courseman* sollen nun die Metadaten, die simplen Lernobjekte, erfasst werden können. Diese Lernobjekte sollen in einer internen Datenbank gespeichert, aufgelistet, geändert und gelöscht werden können. Die vorhandenen Lernobjekte selber werden hierbei nicht in die Datenbank kopiert, sie werden im Feld *lolocation* per URL referenziert.

Auch die weiteren Bestandteile eines Kurses sollen ebenfalls erfasst, aufgelistet, geändert und gelöscht werden können. Bei den zusammengesetzten Lernobjekten muss die Möglichkeit gegeben sein, die bereits erfassten Lernobjekte in die Liste einzufügen bzw. wieder löschen zu können. Auf die selbe Weise müssen ebenfalls die Module und die Kurse bearbeitet werden können. Schlussendlich sollen in *Courseman* mehrere Kurse erfasst werden können.

Ein ausgewählter Kurs muss dann in einer für *Calmecac* kompatiblen Datei ausgegeben werden können, der XML Konfigurationsdatei, welche einen kompletten Kurs beschreibt.

Um eine spätere Integration in *Calmecac* nicht schon von Anfang an zu verhindern, sollen die gleichen Technologien angewendet werden, d.h. *Courseman* soll eine Java Servlet basierte Applikation sein, welche auf die Datenbank FastObjects von POET basiert.

2.3 Datenmodell

Wie ein ganzer Kurs genau aufgebaut ist, wird im Datenmodell spezifiziert, welches im folgenden Abschnitt beschrieben wird. *Calmecac* liest hierzu eine komplette Konfigurationsdatei eines Kurses ein. Diese Konfigurationsdatei ist eine XML Datei, die Spezifikation wurde daher in einer XML DTD Datei definiert, welche im Anhang A zu finden ist.

Wir gehen davon aus, dass einzelne Lernobjekte in Form von Dateien vorhanden sind. In einer Onlineumgebung sind diese über das Netzwerk mit einer URL (z.B. http://www.online−learning.ch/course/picture.jpg) erreichbar und können Text−, Bild−, oder Videodateien sein.

3 Uniform Resource Locator

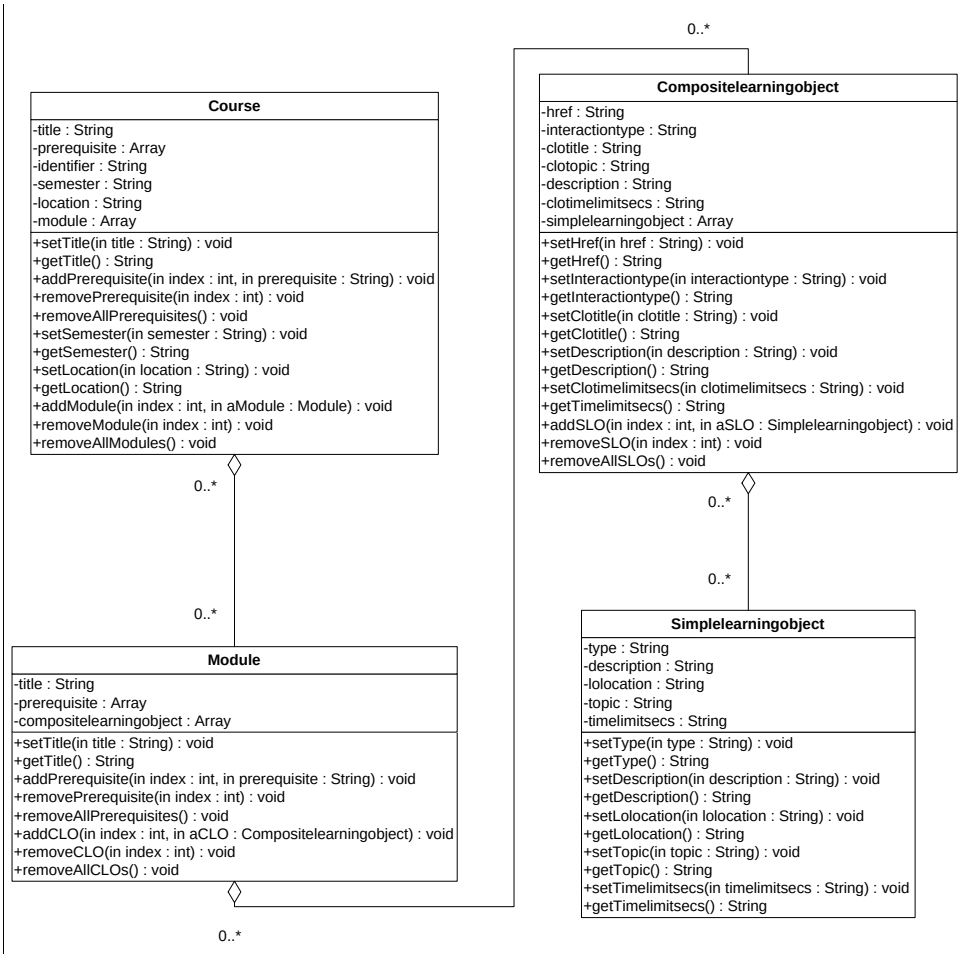

Abbildung 2.1 UML Diagramm des Datenschemas

Abbildung 2.1 stellt das in Courseman verwendete Datenschema in UML[4] dar, welches in den folgenden Abschnitten erklärt wird. Im Diagramm wurden bewusst nur die wichtigsten Attribute und Methoden dargestellt.

2.3.1 Simples Lernobjekt (SLO)

Zu jedem Lernobjekt müssen nun Metadaten erfasst werden, welche diese Objekte auf eine einheitliche Weise beschreiben. Dieser Metadatensatz heisst simples Lernobjekt und enthält nebst einem Attribut "identifier", dem eindeutigen Objektschlüssel, folgende fünf Felder:

1. **type:** Dies ist der MIME[5] Typ des Objektes. Damit wird beschrieben, um was für eine Art es sich bei diesem Objekt handelt, ob es ein Bild, ein Text oder ein Video ist. Das Datei−format wird ebenfalls angegeben. Beispiele sind für eine normale Textdatei "text/plain", für ein mit JPEG komprimiertes Bild "image/jpeg", oder für ein eingebettetes Java Applet "application/x−java−applet".

4 Unified Modeling Language
5 Multipurpose Internet Mail Extension

2. **description:** Hier kann das Objekt frei beschrieben werden.

3. **lolocation:** Dieses Feld enthält die URL des Objektes, es darf sowohl eine relative ("/images/picture.jpg"), als auch eine absolute URL ("http://www.online−learning.ch/images/picture.jpg) angegeben werden.

4. **topic:** Das Themengebiet, zu welchem das Objekt gehört, wird hier angegeben.

5. **timelimitsecs:** Hier kann eine Zeitbeschränkung eingegeben werden, so dass ein Objekt nur eine gewisse Zeit lang angezeigt wird. Die Eingabe erfolgt in Sekunden. Ist keine Beschränkung der Zeit gewünscht, wird "unlimited" in diesem Feld gespeichert.

2.3.2 Zusammengesetztes Lernobjekt (CLO)

Hierbei handelt es sich nun um ein zusammengesetztes Objekt. Hier können simple Lernobjekte zu einer neuen Lerneinheit zusammengefügt werden. Dies wird dann als eine ganze HTML−Seite ausgegeben, welche sich aus den einzelnen, ausgewählten s zusammen−setzt. Auch CLOs haben einen eindeutigen Objektschlüssel, und zusätzlich folgende Felder:

1. **href:** Dieses Feld enthält die URL für die Startseite dieses CLOs.

2. **interactiontype:** Hier wird der Typ des CLOs angegeben. Der Interaktionstyp ist nötig, damit die Software das Objekt richtig darstellen kann und dem Benutzer die entspre−chenden Interaktionsmöglichkeiten bieten kann. Somit wird hier angegeben, ob es sich um eine Aufgabe mit Antwortmöglichkeit handelt, oder ob die Seite nur Lerntext enthält, ohne dass der Benutzer eine Antwort abgeben muss.

3. **clotitle** enthält den Titel des Lerninhaltes.

4. **clotopic:** Das Themengebiet, zu welchem das Objekt gehört, wird hier angegeben.

5. **description:** Hier kann das Objekt frei beschrieben werden.

6. **clotimelimitsecs:** Hier kann eine Zeitbeschränkung eingegeben werden, so dass die Seite nur eine gewisse Zeit lang angezeigt wird. Somit kann dem Benutzer, ähnlich wie bei ei−ner Prüfung, eine bestimmte Zeit auferlegt werden, in welcher er die Aufgaben zu bear−beiten hat. Die Eingabe erfolgt in Sekunden. Ist keine Beschränkung der Zeit gewünscht, wird "unlimited" in dieses Feld eingetragen.

7. **simplelearningobject:** Dies ist nun die Liste, in welcher die in diesem zusammengesetz−ten Lernobjekt enthaltenen simplen Lernobjekte angegeben werden. Die SLOs werden dann der Reihe nach ausgegeben, daher spielt es eine Rolle, in welcher Reihenfolge diese in der Liste gespeichert sind.

2.3.3 Modul

Ein Modul entspricht nun einer ganzen Lektion zu einem bestimmten Thema und sollte vom Benutzer an einem Stück bearbeitet werden. Ein Modul besteht somit aus mehreren HTML−Seiten, mit Lerntexten und dazugehörenden Aufgaben. Somit enthält ein Modul mehrere zusammengesetzte Lernobjekte, ähnlich wie ein CLO aus mehreren SLOs besteht.

Ein Modul besitzt folgende Felder:

1. **title:** enthält den Titel für das Modul

2. **prerequisite:** In dieser Liste sind diejenigen Module enthalten, welche als Voraussetzung für dieses Modul gelten. Somit können Abhängigkeiten erfasst werden, womit z.B. ein Modul, welches zur Einführung in ein Thema erstellt wurde, als Voraussetzung für ein weiterführendes Modul definiert werden kann.

3. **compositelearningobject:** Dies ist, wie schon die Liste der simplen Lernobjekte in einem zusammengesetztem Lernobjekt, die Liste der zusammengesetzten Lernobjekten, welche dieses Modul enthält. Auch hier werden dann die Lernobjekte der Reihe nach dargestellt, wie sie in diese Liste aufgenommen wurden.

2.3.4 Kurs

In einem Kurs werden schlussendlich die zusammengehörenden Module zu einem Thema zusammengefasst. Währen ein Modul in einem Durchgang bearbeitet werden sollte, ist der Kurs für einen längeren Zeitraum gedacht, zu welchem der Benutzer immer wieder zurück−kehrt, um ein nächstes Modul zu bearbeiten.

Ein Kurs enthält folgende Felder:

1. **title:** Der Titel des Kurses

2. **prerequisite:** Sollten andere Kurse für diesen Kurs vorausgesetzt sein, sind deren Identi−fikationsschlüssel in dieser Liste enthalten.

3. **identifier:** Dies ist der eindeutige Schlüssel für den Kurs.

4. **semester:** Hier steht das Semester, für welches der Kurs gedacht ist.

5. **location:** Dieses Feld enthält die URL für die Startseite dieses Kurses.

2.4 Benutzung von Courseman

In diesem Abschnitt soll nun gezeigt werden, wie *Courseman* angewendet werden kann, um Kurse zu verwalten. Die Benutzerführung wurde in englisch gehalten, da bereits *Cal−mecac* und weitere Programme, welche am Institut für Informatik entstanden sind, ebenfalls eine englische Benutzerschnittstelle haben.

Um die Applikation zu starten, muss die URL der Applikation in einem beliebigen Web−browser eingegeben werden. Die URL setzt sich zusammen aus der Adresse des Applikati−onsservers, auf welchem die Anwendung läuft und dem Wurzelverzeichnis, unter welchem die Applikation installiert wurde. Da der Prototyp im vorliegenden Fall auf der gleichen

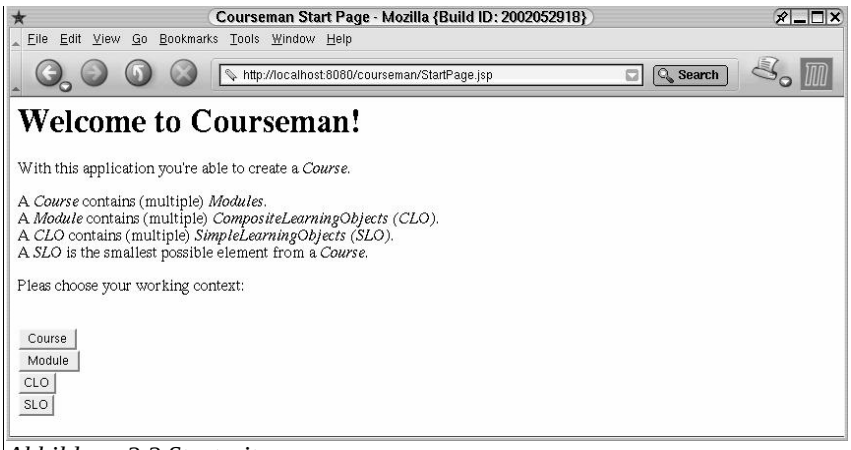

Abbildung 2.2 Startseite

Maschine läuft, ist die Serveradresse "localhost:8080". Hierbei ist "8080" die Portnummer, unter welcher der Applikationsserver selber erreichbar ist, dies ist die Standardeinstellung des Applikationsservers Tomcat. Die Applikation selber hat "courseman" als Wurzelver−zeichnis, die einzugebende Startadresse ist demnach "http://localhost:8080/courseman/". Es erscheint die Startseite der Applikation, siehe unten in der Abbildung 2.2. Hier kann nun das gewünschte Modul gewählt werden, zur Auswahl stehen die Verwaltungsmodule Kurs, Modul, CLO und SLO.

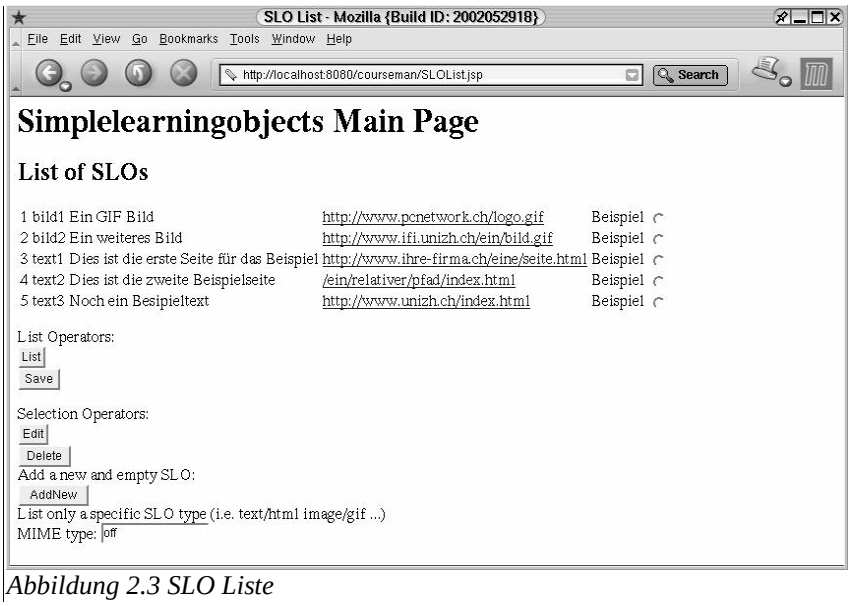

Abbildung 2.3 SLO Liste

Nach dem Druck auf einen Knopf erscheint die entsprechende Seite, mit der Liste aller vorhanden Einträge. In Abbildung 2.3 befinden wir uns im SLO Verwaltungsmodul, es werden alle in der Datenbank vorhandenen simplen Lernobjekte aufgelistet, und es werden folgende Funktionen zur Auswahl angeboten (die zur Auswahl stehenden englisch beschrif−teten Knöpfe sind fett gedruckt):

- **List** − Auflisten: Mit dieser Funktion werden wieder alle in der Datenbank vorhandenen simplen Lernobjekte ausgegeben, es wird wieder diese Seite Aufgerufen.

- **Save** − Speichern: Die s werden in eine XML Datei ausgegeben. Diese Funktion ist ins−besondere dafür gedacht, einen ganzen Kurs zu exportieren. Die Datei wird unter "/tmp/output.xml" gespeichert und kann in *Calmecac* weiterverwendet werden.

- **Edit** − Bearbeiten: Ein ausgewähltes SLO kann bearbeitet werden.

- **Delete** − Löschen: Das selektierte SLO wird aus der Datenbank gelöscht. Zur Sicherheit muss das Löschen auf einer weiteren Seite zusätzlich bestätigt werden.

- **AddNew** − Neu Hinzufügen: Hiermit wird ein neues, leeres SLO eingefügt.

- **MIME type**: MIME Typ: in diesem Feld kann ein gewünschter MIME−Typ eingegeben werden. Wird danach die Liste neu dargestellt, etwa durch drücken des "List" Knopfes, erscheinen nur die simplen Lernobjekte des angegebenen Typs.

Im vorliegenden Beispiel soll ein Bild−Lernobjekt editiert werden. Um nur Lernobjekte des Typs "image/gif" anzuzeigen, muss im Suchfeld unten "image/gif" eingegeben werden und der Knopf "List" gedrückt werden. Im Suchfeld kann auch nur der Typ ohne dem Da−teiformat eingegeben werden, dann werden alle Bilder jeglichen Dateiformats angezeigt.

Abbildung 2.4 SLO Liste − gefiltert nach image/gif

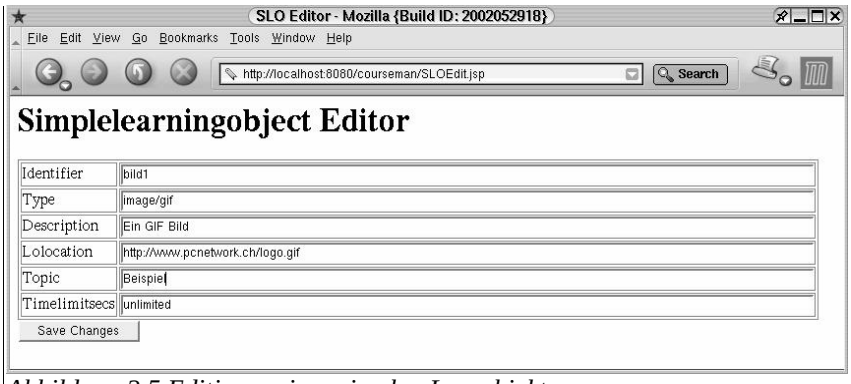

Abbildung 2.5 Editieren eines simplen Lernobjekts

In Abbildung 2.4 kann nun das zu ändernde simple Lernobjekt ausgewählt werden, danach ist die Funktion *Bearbeiten* (engl.: edit) auszuführen. Nun erscheint das Editierformular in Abbildung 2.5, in den Feldern stehen die Werte, welche nun geändert werden können.

Abbildung 2.6 drei simple Lernobjekte als Inhalt eines zusammengesetzten Lernobjekts

Die weiteren Module, CLO, Modul und Kurs, sehen im grossen und ganzen ähnlich aus, in der Listenansicht existiert aber eine Funktion mehr. Denn einem zusammengesetztem Lernobjekt müssen mehrere SLOs zugewiesen werden können, Module enthalten verschiedene CLOs und Kurse bestehen aus mehreren Modulen. Hierfür ist die Funktion *Inhalt* (engl.: content) gedacht. Zunächst muss in der Listenansicht das gewünschte zusammengesetzte Lernobjekt ausgewählt werden, nach dem Drücken auf die "Content" Taste erscheint Abbildung 2.6, hier werden die simplen Lernobjekte, welche das ausgewählte zusammengesetzte Lernobjekt bereits enthält, angezeigt.

Mit der Funktion *Element hinzufügen* (engl.: add item) erscheint die bereits bekannte SLO Listenansicht aus Abbildung 2.3. Hier kann das gewünschte Lernobjekt ausgewählt werden. Mit *Element löschen* (engl.: delete item) wird das markierte Lernobjekt entfernt.

3 Verwendete Technologien

Der vorliegende Abschnitt behandelt die bei *Courseman* eingesetzten Technologien. *Courseman* wurde mit Hilfe der Programmiersprache Java entwickelt. Das Programm wird aber nicht auf dem Computer des Anwenders ausgeführt, sondern läuft auf einem Applikationsserver, hier auf dem frei erhältlichen Server Tomcat, welches im Rahmen des Jakarta Projektes[6] bei Apache entwickelt wurde. Dadurch benötigt der Anwender nur einen Computer mit einem beliebigen HTML−Browser und einem Internetanschluss. Somit handelt es sich hierbei um eine *fat server / thin Client* Architektur, da die Software auf dem Server ausgeführt wird, und der Client Computer nur für die Darstellung der Benutzerschnittstelle genutzt wird.

3.1 Java

Java ist eine Programmiersprache, welche von Sun Microsystems ursprünglich für den Einsatz im Internet entwickelt wurde. Von der Syntax her ist Java ähnlich aufgebaut wie C/C++, die grössten Unschönheiten wurden aber vermieden, indem Java streng objektorientiert ist und die Arbeitsspeicherverwaltung aus Programmierersicht wesentlich unkritischer ist. Der grösste Unterschied aber ist, dass Java plattformunabhängig ist. Dies wird erreicht, indem der Sourcecode ein einziges Mal kompiliert wird, das Ergebnis ist eine Objektcode Datei, der sogenannte Java Bytecode. Bei C/C++ entspricht diese Datei der kompilierten Objektdatei (*.o), welche aber noch nicht gelinkt wurde. Erst durch das Linken wird das Programm mit den plattformabhängigen Bibliotheken gekoppelt und somit ausführbar gemacht. Bei Java fällt nun das Linken weg, stattdessen wird der Java Bytecode erst beim Ausführen im Objektcodeinterpreter, der Java Virtual Machine (JVM), lauffähig. Diese JVM kann man sich vereinfacht als eine Art Emulator vorstellen, eine *virtuelle Maschine*, ein Computer im Computer. Diese JVM ist in [LIN99] offen spezifiziert, und es existieren davon Implementationen für eine Vielzahl von Computersystemen, vom einfachen PC bis hin zu Grossrechnern, aber auch für Handhelds und sogar Mobiltelefone bis hin zu Chipkarten.

3.1.1 Applets

Urspünglich war Java für Anwendungen im Internet gedacht. Mit Hilfe von Java Applets können Java Programme von einem Webserver auf den Client heruntergeladen und dort mit Hilfe der JVM ausgeführt werden, wie dies unter Abbildung 3.1 dargestellt wird. Hier fordert der Webbrowser das Javaprogramm (den Java Bytecode) "javaapplet.class" an und führt sie in der eigenen JVM aus. Dank der virtuellen Maschine kann hierbei ein hohes Mass an Sicherheit gewährt werden, da der externe Code nach dem Sandkastenprinzip nur in der JVM ausgeführt werden kann und nicht auf schützenswerte Ressourcen ausserhalb zugreifen kann.

6 http://jakarta.apache.org

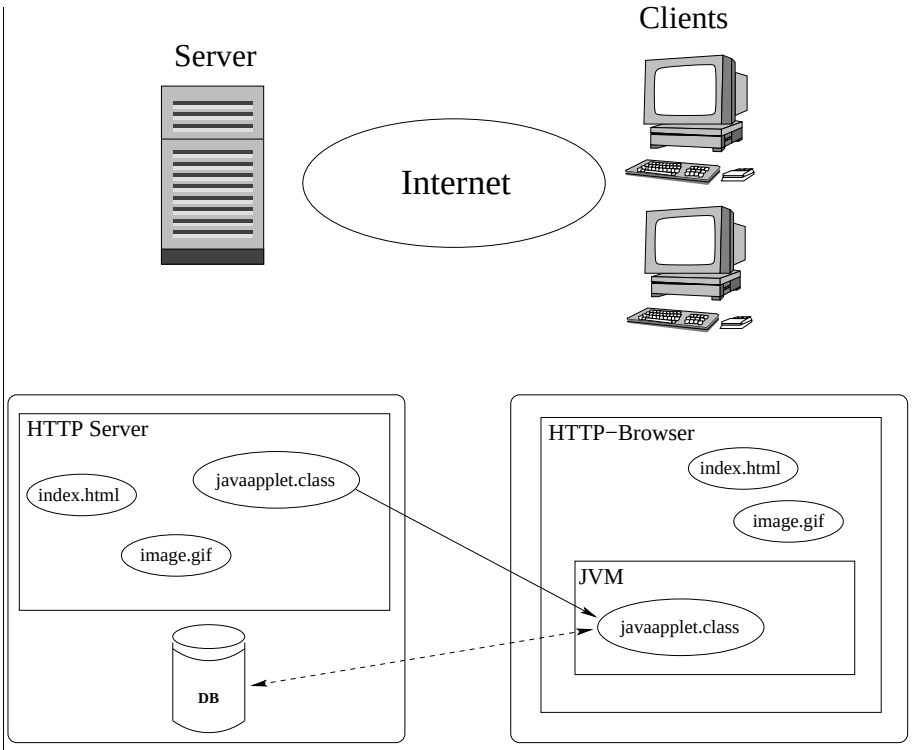

Abbildung 3.1 Ausführung eines Java Applets

Nach einigen Jahren zeichnete sich jedoch ab, dass Java nicht nur für Internetanwendungen, sondern für alle möglichen Zwecke gut geeignet ist. Im Web aber hat es nicht die Dominanz erreicht, die von Sun ursprünglich erhofft wurde. Java Applets werden heute für spezielle Anwendungen wie Onlinebank, Onlinebörse oder auch Onlinelernen eingesetzt. In anderen Bereichen haben sich andere Technologien durchgesetzt, z.B. für multimediale Inhalte Flash von Macromedia.

Heute ist Java eine universell einsetzbare Programmiersprache, welche grundsätzlich auf beinahe allem lauffähig ist, was einen Prozessor und Speicher besitzt.

3.1.2 Servlets

Vereinfacht gesagt sind Java Servlets [DAV99] für den Webserver das, was Java Applets für den Client sind. Mit Java Servlets wird es möglich, Programme zu schreiben, die eine ähnliche Verarbeitung wie CGI−Scripte (z.B. Perl, PHP) erlauben. HTML−Text kann als Ausgabe eines Programmes dynamisch erzeugt und anschliessend wie eine HTML−Seite zu einem Browser geschickt werden. Weiterhin ist es möglich, HTML−Formulare abzufragen und auszuwerten.

Damit lässt man Java Programme bereits auf der Seite des Servers laufen, und kann daher auf eine Ausführung auf dem Client verzichten. Somit werden die Anforderungen an den Client gesenkt, da keine JVM vorhanden sein muss, ein beliebiger HTML−Browser reicht aus, der Client dient nur noch als Benutzerschnittstelle zum Server.

Sofern Java Applets nicht autonome Programme sind, sondern auf bestimmte Ressourcen via Internet zugreifen müssen, handelt es sich um eine echte verteilte Applikation, was entsprechende Schwierigkeiten mit sich bringen kann, wie z.B. gleichzeitige Zugriffe auf eine Datenbank oder eine andere zentrale Schnittstelle (via CORBA oder RMI).

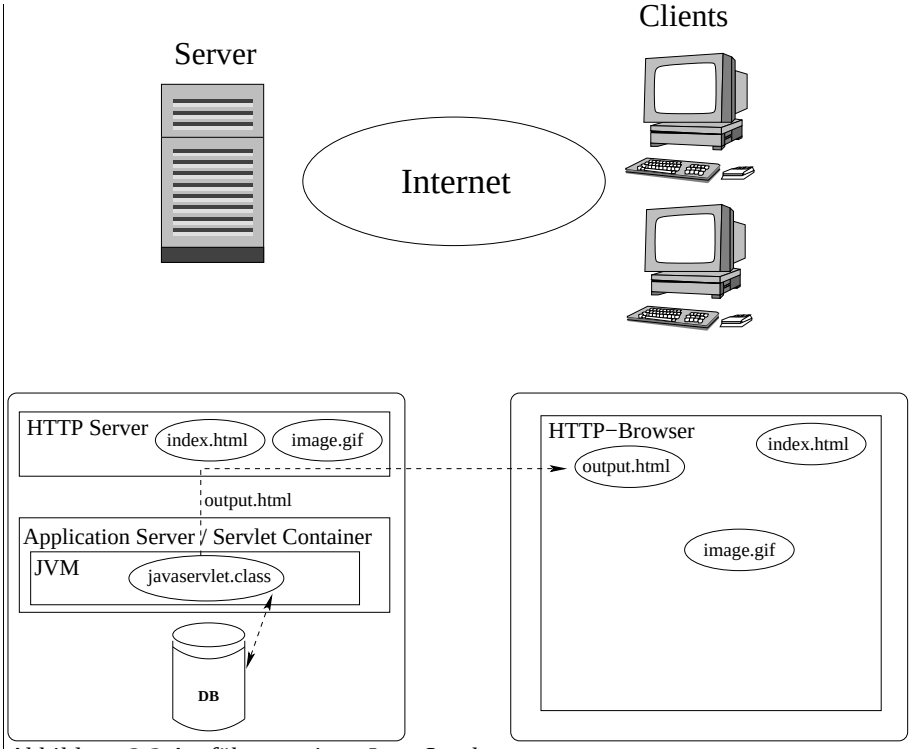

Abbildung 3.2 Ausführung eines Java Servlets

Da ein Servlet zentral in einer JVM auf einem Server ausgeführt wird, handelt es sich hierbei nicht mehr unbedingt um eine verteilte Applikation, sondern verhält sich von der Programmierersicht her wie eine normale Applikation auf einem Computersystem, wie das in Abbildung 3.2 ersichtlich ist. Selbstverständlich muss berücksichtigt werden, dass mehrere Benutzer quasi gleichzeitig bedient werden müssen. Aber der Code wird für alle Benutzer auf dem gleichen Server ausgeführt, und nicht separat auf ihren Clients.

Ein Nachteil von Servlets ist, dass die ganze, auszugebende HTML−Seite aus einem Java Programm generiert werden muss, d.h. die HTML Tags und Texte müssen relativ mühsam mit Java ausgegeben werden (z.B. mit dem Java Befehl: `out.println("<H1>Dies ist ein Header</H1>");`). Dies kann zu Schwierigkeiten führen, wenn z.B. in einer grösseren Umgebung verschiedene Leute für das Webdesign und für die Programmierung zuständig sind.

3.1.3 Java Server Pages (JSP)

Mit JSP [PEL01] soll der Nachteil von Java Servlets gelöst werden. JSP basieren ebenfalls wie die Servlets auf die Java Servlet API. Bei JSP kann nun aber, im Gegensatz zu Servlets, Java− und HTML Code gemischt werden. Eine JSP Datei sieht zunächst wie eine normale HTML Datei aus. Soll nun an einer Stelle eine Ausgabe aus einem Java Programm generiert werden, kann Javacode in den HTML Quelltext eingebettet werden, welcher von den Zeichen "<%" und "%>" umschlossen wird.

So können Webdesigner die HTML Seiten anpassen, den Javacode müssen sie höchstens verschieben, damit das Ergebnis an einer anderen Stelle erscheint.

Javaprogrammierer hingegen können sich auf den Java Code konzentrieren, ohne sich mit HTML herumschlagen zu müssen.

3.2 Datenbank

Um die Daten permanent zu speichern, wird in *Courseman* die objektorientierte Datenbank FastObjects der Firma POET[7] eingesetzt. Dieser Abschnitt gibt eine kurze Einführung in objektorientierte Datenbanken, wie FastObjects funktioniert und wie es eingesetzt werden kann.

3.2.1 Warum objektorientierte Datenbanken

Am besten lässt sich dies an folgenem Beispiel erklären. In *Courseman* sollen die Kurse und deren Unterelemente (Module, CLOs, SLOs) für die Lernumgebung *Calmecac* erstellt, geändert, gespeichert und ausgegeben werden können. In unserem Fall ist bereits das Datenmodell (siehe Abschnitt 2.3) eines Kurses objektorientiert, denn ein Kurs ist nichts anderes als ein Objekt mit gewissen Eigenschaften, welcher weitere Unterelemente bzw. Unterobjekte enthält. Daher ist es naheliegend, dass diese Objekte innerhalb des Java Programmes ebenfalls als Objekte dargestellt werden bzw. als Klassen mit den dazugehörenden Methoden.

Diese Objekte müssen nun in einer Datenbank abgelegt werden. Bei einer relationalen Datenbank stünden folgende Arbeiten an (unter der Annahme, dass auf die Datenbank mit der JDBC Schnittstelle zugegriffen wird, mit welcher es möglich wird, vom Java Programm aus per SQL[8] mit der Datenbank zu kommunizieren:

- Um die Objekte in Tabellen speichern zu können, muss neben dem objektorientierten Datenmodell zusätzlich ein relationales Datenmodell erstellt werden, in welches das Datenmodell abgebildet werden kann.

- Häufig ist das objektorientierte Datenmodell viel zu komplex. Insbesondere bei vielschichtigen Objektabhängigkeiten wird entweder der Aufwand für eine saubere Abbildung unverhältnismässig hoch, oder die Abbildung ist nicht vollständig Dies kann eine Menge Fehlerquellen eröffnen, wenn das relationale Datenmodell mehr zulässt, als das ursprüngliche Datenmodell erlaubt. Als letzten Ausweg bleibt die Vereinfachung des Datenmodells.

- Der Programmierer muss eine Menge Programmroutinen entwickeln, welche jeweils die Java Objekte in Tabellen und die Tabelleninhalte in Objekte konvertiert. Je komplexer die Objekte sind, desto höher ist der zusätzliche Aufwand.

7 http://www.fastobjects.com
8 Structured Query Language

Daher ist es naheliegend, wenn ein Datenmodell bereits objektorientiert definiert wurde und das Programm mit einer objektorientierten Sprache implementiert wird, auch für die Speicherung einen objektorientierten Ansatz zu wählen.

3.2.2 Funktionsweise von FastObjects

FastObjects ist eine objektorientierte Datenbank und kann mit den Programmiersprachen C++ und Java eingesetzt werden. FastObjects nimmt dem Programmierer die aufwändige Arbeit der Abbildung zwischen dem Datenmodell der Datenbank und des Programmes ab, indem es die Objekte im Speicher des Java Programmes automatisch in Objekte der Datenbank abbildet und umgekehrt. Durch diese direkte Abbildung kann das Datenbankschema automatisch generiert werden. D.h. wenn der Programmierer eine Klasse in Java implementiert, definiert er damit auch gleichzeitig das Datenbankschema der Klasse. Dies funktioniert selbst über komplexe Klassenhierarchien hinweg, da die Datenbank die Objekte kennt und damit auch über die Beziehungen und Abhängigkeiten zwischen den Objekten bescheid weiss. Verweise und Zeiger auf andere Objekte werden einfach gespeichert und genau gleich wieder direkt aus der Datenbank zurückgeladen.

Hierdurch ergeben sich folgende Vorteile:

- Das Datenmodell kann modelliert werden, ohne auf das hinter der Applikation liegende Datenbanksystem Rücksicht nehmen zu müssen. Bei einem relationalen Datenbanksystem müssten gewisse Einschränkungen in Kauf genommen werden.

- Der Entwicklungsaufwand ist erheblich kleiner, da die Abbildung zwischen Programm und Datenbanksystem automatisch geschieht. Insbesonders, wenn während der Entwicklungsphase Änderungen am Datenmodell vorgenommen werden, müssen nicht auch noch die entsprechenden Abbildungsfunktionen angepasst werden.

FastObjects ist aber kein Ersatz für die bekannten Datenbanken wie Oracle oder DB2. FastObjects ist gar nicht geeignet für die typischen Datenbankanwendungen. Es ist vielmehr dafür gedacht, einem Programmierer die Möglichkeit zu bieten, die Objekte möglichst einfach zu speichern bzw. in sein Programm Funktionalitäten einer Datenbank einbinden zu können. Das im Programm implementierte Datenmodell steht im Fokus, die Datenbank richtet sich dann danach. FastObjects speichert dann die kompletten Objekte, nicht bloss einzelne Attribute.

3.2.3 Programmieren mit FastObjects

FastObjects muss auf der Maschine des Programmierers installiert werden. Einzig das Vorhandensein des JDK[9] wird vorausgesetzt. Die Entwicklung einer Applikation geht folgendermassen vor sich [POE01]:

1. Objektorientierte Analyse und Entwurf: Den Anforderungen an die Applikation entsprechend wird ein objektorientiertes Datenmodell entworfen, ohne an die dahinter liegende Datenbank zu denken.

9 Java Development Kit

2. Implementierung der persistenten Klassen: Das Programm kann nun erstellt werden. Es wird nun einzelne Klassen geben, von welchen Objekte gespeichert werden sollen. Diese Klassen können ganz normal entworfen werden, es sind keine datenbankspezifische Einschränkungen zu berücksichtigen. Es gibt keinen speziellen Code, welcher eingefügt werden muss. Es gibt keinen Unterschied zwischen persistenten und nicht−persistenten Klassen bei der Implementierung. Um die gewünschten Klassen persistent zu machen, müssen diese in einer speziellen Konfigurationsdatei ("ptj.opt") erwähnt werden. Ein FastObjects Werkzeug, "ptj", erstellt anhand der Informationen in der Konfigurationsdatei das Datenbankschema automatisch und generiert, falls nicht bereits vorhanden, eine leere Datenbank.

3. Speichern und Lesen von Objekten: FastObjects bietet eine Klassenbibliothek, welche den Zugriff auf die Datenbank ermöglicht. Die wichtigsten Klassen sind *Database* und *Transaction*. Mit diesen Klassen können Objekte in der Datenbank gespeichert und wieder gelesen, in der Datenbank navigiert, oder OQL[10] Befehle abgesetzt werden.

3.3 XML

XML [BRA00] wird je länger je mehr zum wichtigsten neuen Standard für Datenrepräsentation und den elektronischen Datenaustausch. Seit 1996 arbeitet das World Wide Web Consortium[11], die Instanz für Standards im Web, an dieser neuen Sprache für das Internet. Anlass für die Entwicklung gaben die Limitationen von HTML [RAG99]. HTML muss immer mit demselben Satz an Tags auskommen, um Dokumente auszuzeichnen. XML ist hier ungleich flexibler, auf sehr einfache und individuelle Weise, erlaubt es dagegen beliebige strukturierte Informationen in einem entsprechenden, textbasierten Format abzubilden. Ein Dokument, in welchem Daten dargestellt werden, besteht im weitesten Sinne aus den drei Teilen Inhalt, Struktur und Layout. Mit XML ist es möglich, diese drei Teile getrennt zu erfassen.

Dies ist auch der Grund, weshalb HTML nicht zur Speicherung bzw. für den Datenaustausch geeignet ist. Denn bei HTML liegt der Fokus auf die Darstellung der Daten, Daten und Layout sind in einer einzigen Datei gemischt gespeichert. Mit den Tags wird die Ausgabe gesteuert. Mit einem automatischen Skript wäre es zwar möglich, zur Datenübertragung eine HTML Seite zu lesen, die Struktur der Daten richtet sich aber nach dem Layout. Wird das Layout der Seite geändert, müsste auch das Skript, welches die Datei importiert, geändert werden, da es das geänderte Layout nicht interpretieren kann.

Mit XML wird dieses Problem elegant gelöst, da die reinen, semistrukturierten Daten in der XML Datei gespeichert sind, das Datenmodell kann in einer DTD[12] bzw. XML Schema Datei [FAL01] definiert werden. Schlussendlich können die Daten durch ein beliebiges Programm in ein gewünschtes Ausgabeformat umgewandelt werden, für die Umwandlung sind aber insbesondere XSL[13] Steuerdateien [ADL01] vorgesehen Diese enthalten Anwei−

10 Object Query Language
11 http://www.w3c.org
12 Document Type Definition, die Spezifikation ist Teil der XML Spezifikation
13 eXtensible Stylesheet Language

sungen für einen XSLT[14] Prozessor. Dieser Prozessor wandelt zunächst die XML Datei in eine anderes XML Dokument um, welches eine andere Datenstruktur hat, welche besser für die Ausgabe geeignet ist (z.B. interessieren in einer Listenansicht nicht alle Daten, oder die Reihenfolge der Daten ist anders, als in der ursprünglichen Datei). Schlussendlich kann dieses XML Dokument in ein beliebiges, auch nicht−XML−konformes Ausgabeformat umgewandelt werden, wie z.B. HTML, PDF oder sogar ein proprietäres Format wie eine Microsoft Excel Datei.

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<simplelearningobject identifier="1234">
    <type>text/html</type>
    <description>Dies ist eine Beispiel</description>
    <lolocation>http://ihre.firma.com/slo.html</lolocation>
    <topic>Beispiele</topic>
    <timelimitsecs>unlimited</timelimitsecs>
</simplelearningobject>
```

Abbildung 3.3 Beispiel für eine XML Datei: Simplelearningobject.xml

In Abbildung 3.3 ist ersichtlich, wie eine XML Datei aussieht. In der ersten Zeile befindet sich der Prolog, in welchem Informationen über die verwendete XML Version und dem Zeichensatz enthalten sind. Danach folgt das eigentliche XML Element "simplelearningobject". Dieses Element enthält zum einen das Attribut "identifier" mit dem Wert "1234" und zum anderen weitere Unterelemente "type", "description", "lolocation", "topic" und "timelimitsecs", die ihrerseits hier nur noch Textdaten zum Inhalt haben.

3.3.1 DTD / XML Schema

Die Datenstruktur einer XML Datei kann in einer DTD oder XML Schema Datei [FAL01] definiert werden. XML Schema ist neuer und bietet viele neue Möglichkeiten gegenüber DTD. Ausserdem ist eine XML Schema Datei selbst XML konform und entspricht daher eher der XML Philosophie. Damit wird auch ermöglicht, XML Schema Dateien mit den normalen XML Werkzeugen zu bearbeiten. Z.B. wäre es möglich, mit XSL XML Schema Dateien automatisch zu generieren. XML Schema wird DTD wahrscheinlich bald ablösen.

```
<!ELEMENT simplelearningobject
(type,description,lolocation,topic,timelimitsecs)>
<!ATTLIST simplelearningobject identifier CDATA "">
<!ELEMENT type (#PCDATA)>
<!ELEMENT lolocation (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ELEMENT timelimitsecs (#PCDATA)>
```

Abbildung 3.4 Beispiel einer DTD Datei: simplelearningobject.dtd

Eine XML Datei kann nach einer DTD Datei validiert werden. Damit kann bestimmt werden, ob eine XML Datei der definierten Datenstruktur entspricht. Erst dann können Daten sicher ausgetauscht oder sinnvoll automatisch weiterverarbeitet werden. Eine DTD, welcher die XML Datei in Abbildung 3.3 entspricht, ist in Abbildung 3.4 dargestellt.

14 XSL Transformation

Mit <!ELEMENT> wird ein XML Element definiert, die erste Anweisung definiert das Element "simplelearningobject" und besagt, dass das simple Lernobjekt je genau ein Unterelement "type", "description", "lolocation", "topic" und "timelimitsecs" haben muss, und zwar genau in der angegebenen Reihenfolge. "(A|B)" bedeutet, dass es entweder Element A oder Element B besitzen muss. "A*" bedeutet, dass das Element null bis unendlich viele Elemente A enthalten sein dürfen, "A+" heisst mindestens ein Element A muss enthalten sein etc. <!ATTLIST> definiert, dass ein simples Lernobjekt ein Attribut *identifier* haben darf, der Wertebereich ist CDATA[15], und der Standardwert ist leer. Die Unterelemente vom simplen Lernobjekt haben alle den Wertebereich PCDATA[16], das kann normaler Text, aber auch hier nicht weiter spezifizierte XML Elemente sein.

3.3.2 XSLT

Mit Hilfe eines XSLT Prozessors und den entsprechenden XSLT Steuerdateien [CLA99] können XML Datenstrukturen in andere XML Datenstrukturen umgewandelt werden. Während mit XSLT Dateien nur eine Umwandlung in ebenfalls XML konforme Dateien möglich ist (z.B. XHTML, das ist XML konformes HTML, welches zu HTML aufwärtskompatibel ist), können mit XSL auch nicht−XML konforme Dateien erzeugt werden (PDF, HTML).

```
'<xsl:template' [ 'match=' pattern_expr ] '>'
      { xsl:directive }
      { result-elements }
'</xsl:template>'
```

Abbildung 3.5 Syntax einer XSLT Datei

Die Syntax von XSLT Steuerdateien ist in Abbildung 3.5 in *BNF*[17] dargestellt.

```
<vehicles>
      <vehicle>
            <make>Fiat</make>
            <model>Punto</model>
      </vehicle>
      <vehicle>
            <make>Toyota</make>
            <model>Corolla</model>
      </vehicle>
</vehicles>
```

Abbildung 3.6 Beispiel einer XML Fahrzeugdatenbank

In XSL wird eine Vorlage (engl.: template) mit dem Tag *xsl:template* eingeführt. Der *match* Teil sorgt für die Auswahl der richtigen XML Knoten im XML Baum, auf welche die Regeln angewendet werden sollen (ohne diesen Teil werden die Regeln auf das ganze XML Dokument angewendet). *pattern_expression* ist ein Ausdruck in der XPath−Sprache. *result−elements* gibt die neuen Elemente im Ergebnisbaum an, und *xsl:directive* ist eine der fol−

15 Character Data, d.h. normaler Text, welcher nicht weiter interpretiert wird
16 Parsed Character Data
17 Backus Naur Form

genden Möglichkeiten. Bei allen Direktiven wählt das *select* Attribut einen speziellen Knoten zur Bearbeitung aus, damit die Nachfolgerknoten von der Bearbeitung ausgeschlossen werden. Für alle Beispiele wird das in Abbildung 3.6 dargestellte Beispiel als Eingabedokument verwendet:

- `'<xsl:value-of' [ 'select=' pattern_exprt ] '>'`

```
<xsl:template match="vehicle">
      <tr>
            <td><xsl:value-of select="make"/></td>
            <td><xsl:value-of select="model"/></td>
      </tr>
</xsl:template>
```

Abbildung 3.7 XSLT Beispiel für <xsl:value-of>

```
<tr>
      <td>Fiat</td>
      <td>Punto</td>
</tr>
<tr>
      <td>Toyota</td>
      <td>Corolla</td>
</tr>
```

Abbildung 3.8 XML Ergebnis des <xsl:vaule-of> Beispiels

Hiermit wird der Inhalt des ausgewählten Elementes als ein Text ausgegeben. In der XSLT Datei in Abbildung 3.7 werden die beiden Unterelemente <*make*> und <*model*> als Text ausgegeben. Das Ergebnis ist in Abbildung 3.8 ersichtlich.

- `'<xsl:copy-of' 'select=' pattern_expr '>'`

```
<xsl:template match="vehicle">
      <newvehicle>
            <xsl:copy-of select="model"/></td>
            <xsl:copy-of select="make"/></td>
      </newvehicle>
</xsl:template>
```

Abbildung 3.9 XSLT Beispiel für <xsl:copy-of>

Dieser Befehl unterscheidet sich gegenüber <*xsl:value-of*> nur bei der Ausgabe: statt nur den Inhalt des ausgewählten Elementes auszugeben, wird das ganze Element zusammen mit den XML Tags ausgegeben. Diese Direktive kommt meistens dann zur Anwendung, wenn die Struktur eines XML Dokumentes geändert werden soll, und ein Teilbaum unverändert übernommen, aber an einer anderen Stelle eingefügt werden soll. In der Vorlage in Abbildung 3.9 werden die Elemente <*model*> und <*make*> vertauscht, und das Ele-

```
<newvehicle>
      <model>Punto</model>
      <make>Fiat</make>
</newvehicle>
<newvehicle>
      <model>Corolla</model>
      <make>Toyota</make>
</newvehicle>
```

Abbildung 3.10 XML Ergebnis des <xsl:copy−of> Beispiels

ment <*vehicle*> wird in <*newvehicle*> umbenannt, das Ergebnis ist in Abbildung 3.10 dargestellt.

- `'<xsl:apply−templates' [ 'select=' pattern_expr ] '>'`

```
<xsl:template match="/vehicles">
      <table>
            <xsl:apply−templates/>
      </table>
</xsl:template>

<xsl:template match="vehicle">
      <tr>
            <td><xsl:value−of select="make"/></td>
            <td><xsl:value−of select="model"/></td>
      </tr>
</xsl:template>
```

Abbildung 3.11 Beispiel für xsl:apply−templates

```
<table>
      <tr>
            <td>Fiat</td>
            <td>Punto</td>
      </tr>
      <tr>
            <td>Toyota</td>
            <td>Corolla</td>
      </tr>
</table>
```

Abbildung 3.12 XML Ergebnis für <xsl:apply−templates> Beispiel

Diese Direktive sorgt dafür, dass die weiteren Unterelemente "normal" weiterverarbeitet

werden sollen. D.h. für die Unterelemente des Teilbaumes werden, falls vorhanden, die entsprechenden xsl:templates ausgeführt, ansonsten wird der Teilbaum kopiert. Existieren z.B. Vorlagen, welche dafür sorgen, dass die Elemente die richtigen *<tr>*, *</tr>* und *<td>*, *</td>* Tags für eine Tabellendarstellung in HTML erhalten, wie dies in Abbildung 3.7 der Fall ist, wird noch ein Vorlage benötigt, welches die ganze Tabelle zwischen *<table>* und *</table>* Tags setzt. Diese Vorlage in Abbildung 3.11 sieht dann so aus, dass sie für das Wurzelelement *<vehicles>* der Tabelle im XML Baum ausgeführt wird und in die Ausgabe ein *<table>* Element erstellt. Für den Inhalt des Tabellenelements besteht aber bereits eine Vorlage (das ist die *<xsl:template match="vehicle">* Vorlage). Durch die *<xsl:apply−templates>* Direktive wird der XSLT Prozessor angewiesen, für die Weiterverarbeitung der weiteren Elemente unterhalb *<vehicles>* die entsprechenden Vorlagen zu benützen. Somit entsteht das Resultat in Abbildung 3.12.

- `'<xsl:for−each' 'select=' pattern_expr '>'`
  Hiermit kann über mehrere Knoten iteriert werden, welche das select Kriterium erfüllen. Erst mit diesem Konstrukt wird es möglich, Tabellen flexibel gestalten zu können, insbe−sondere, um z.B. Tabellen aus verschiedenen XML Teilbäumen zu erstellen, oder um Spalten und Zeilen zu vertauschen.

3.4 XML Data Binding

Obschon XML heute schon fast als Lösung für alle Probleme angeboten wird, da es sehr viele positive Eigenschaften besitzt wie Portabilität, einfache Verarbeitbarkeit, gute Lesbar−keit etc., ändert dies an einer Tatsache nichts: XML ist nach wie vor nichts anderes als ein weiteres Datenformat. Damit bleibt dem Programmierer immer noch die Aufgabe, die Daten aus den XML Dokumenten in Java Objekte zu importieren und diese für den Datenaustausch wieder nach XML zu exportieren. *Binding* ist allgemein der Begriff für die Abbildung ex−terner Applikationsdaten in ein für die Programmiersprache natives Datenformat bzw. um−gekehrt. Hierbei spielt es keine Rolle, ob es sich um Textdateien, proprietäre Binärformate, Datenbanken oder eben XML Dateien handelt. *Binding* ist zwar ein englischer Begriff, wird aber in der deutschen Literatur eins zu eins übernommen.

Es existieren zwar zwei bekannte APIs, mit welchen Applikationen auf XML Daten zu−greifen können: DOM[18] API vom World Wide Web Consortium, und SAX[19] von Sun Micro−systems. Allerdings sind diese APIs eher für grundlegende Manipulationen an den XML Daten gedacht, sie befassen sich mit der Struktur des XML Dokuments, daher sind diese APIs für die meisten Applikationen zu umständlich, da sie den Programmierer zwingen, sich mit der Struktur des XML Dokuments auseinander zu setzen.

"*Data Binding* ist der Prozess des Zuordnens von Komponenten eines Datenformats (z.B. SQL Tabellen oder XML Schema) zu spezifischen Repräsentationen einer Programmier−sprache (z.B. Java Objekte), die die Bedeutung des Datenformates kodieren. Data Binding erlaubt es dem Entwickler, wie gewohnt mit nativem Code zu arbeiten, ohne gleichzeitig die Intention der Daten zu verlieren." [MAN02]

18 Document Object Model
19 Simple API for XML

XML Data Binding ermöglicht es, ohne eine eigene API entwickeln zu müssen, auf XML Daten zugreifen zu können, ohne wissen zu müssen, wie die Daten repräsentiert sind. Dies wird auf eine im Kontext der Programmiersprache natürlich erscheinende Art und Weise geboten, in einem Format, das die beabsichtigte Bedeutung der Daten wiederspiegelt.

3.4.1 Funktionsweise von XML Data Binding

XML Data Binding nimmt dem Programmierer zwei mühsame Arbeiten ab. Zum einen müsste der Programmierer das Datenschema der XML Dokumente auf ein Schema der Programmiersprache abbilden. Konkret muss er die Struktur der XML Dokumente, welche entweder per DTD oder XML Schema definiert sind, abbilden in Java Klassen. Meistens ergeben die verschiedenen Stufen der hierarchisch aufgebauten XML Dokumente verschiedene Java Klassen mit den entsprechenden Abhängigkeiten untereinander. Zum anderen müssen die Daten selber zwischen XML und Java konvertiert werden können. Den Prozess der Umwandlung eines XML Dokuments (welches dem definierten XML Schema entspricht) in ein Java Objekt (welches eine Instanz der entsprechenden Klasse ist) nennt man *unmarshal*. Das exportieren eines Java Objektes in das entsprechende XML Dokument heisst *marshal*. Auch diese Begriffe werden in der Deutschen Literatur nicht übersetzen, sondern werden aus dem Englischen übernommen.

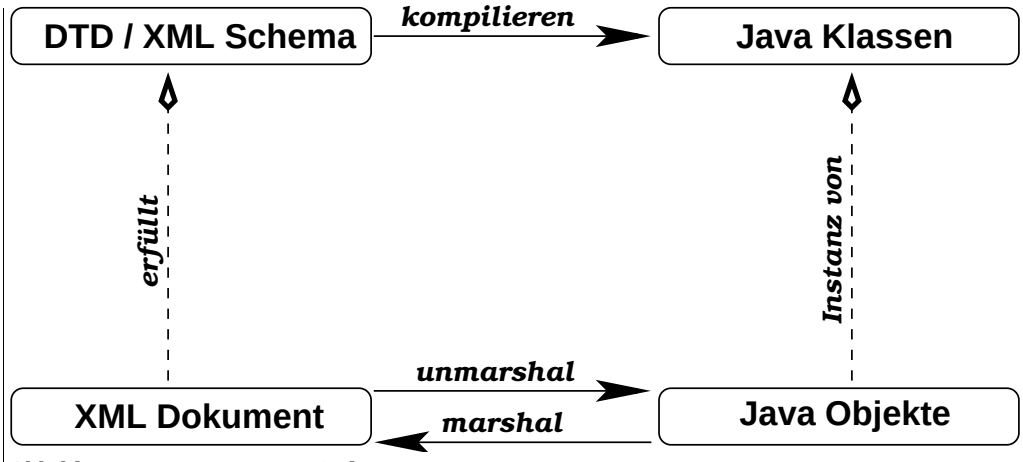

Abbildung 3.13 XML Data Binding Prozesse

Die Umwandlung des XML Schemas in Java Klassen wird einmalig ausgeführt. Je nach *XML Data Binding* Implementation muss das Schema in einer DTD oder XML Schema Datei gegeben sein. Diese Datei wird dann in den *XML Data Binding* Kompilierer eingegeben, als Resultat werden Java Klassendateien (in Java Quellcode) ausgegeben. Diese Java Klassen enthalten dann die *marshal*− und *unmarshal* Methoden, welche im Java Programm ausgeführt werden können. Diese drei Prozesse *Kompilierung*, *Marshalling* und *Unmarshalling* sind in Abbildung 3.13 dargestellt. Eine weitere, wichtige Methode ist die *Validierung*, welche ebenfalls automatisch generiert wird. Hiermit kann ein Java Objekt, insbesondere nachdem es im Java Programm verändert wurde, überprüft werden. Damit wird getestet, ob das Objekt, noch vor dem Marshalling, gültig ist und dem XML Schema entspricht. Dies ist nötig, da während der Laufzeit das Objekt nicht ständig bei jeder Änderung überprüft wird, ob alle Schranken eingehalten werden. Dadurch ist es möglich, nacheinander viele Änderungen durchzuführen, ohne eine Geschwindigkeitseinbussen durch eine ständige Kontrolle hinnehmen zu müssen.

3.4.2 XML Data Binding Implementationen

Zwei bekannte Implementationen und Spezifikationen vom *XML Data Binding* Konzept:

- JAXB[20]
  wurde von SUN Microsystems entwickelt, ist aber zur Zeit nur als Beta im "Sun Early Access" Programm erhältlich. Dies ist ein kommerzielles Produkt und ist gegen Registrierung frei erhältlich. JAXB erstellt Java Klassen aus DTD Dateien mit Hilfe von DT4DTD[21] [BUC00]. Mit Hilfe einer speziellen Konfigurationsdatei ist es möglich, XML Elemente feiner an Java zu binden, als DTD erlaubt. Damit können z.B. XML Elementen und Attributen unterschiedliche Java Typen zugewiesen werden, z.B. Integer, Arrays etc., während DTDs keine verschiedene Variablentypen kennen und daher alles als String interpretieren.

20 Java Architecture for XML Data Binding, http://java.sun.com/xml/jaxb
21 Data Types for DTD

- Castor[22]
  wurde von der Firma exolab.org entwickelt und ist eine OpenSource *Data Binding* Implementation für Java, aber nicht nur für XML, sondern auch für SQL Tabellen und LDAP[23]. Im Gegensatz zu JAXB generiert Castor Java Klassen aus XML Schema Dateien statt aus DTDs. Es stellt eine Sprache zur Verfügung, mit welcher Java Klassen in XML, Tabellen relationaler Datenbanksysteme und LDAP abgebildet werden können.
  Castor ist ausserdem ein Laufzeitprodukt, da es die Selbstüberprüfung von Java Beans unterstützt und versucht, Elemente und Attribute an Klassen und Variablen zu binden.

22 http://castor.exolab.org
23 Lightweight Directory Access Protocol

4 Aufbau und Implementation von Courseman

Im vorliegenden Abschnitt soll die Dateistruktur, der Aufbau von *Courseman* gezeigt werden und welche Bibliotheken benötigt werden, um es in Betrieb nehmen zu können. Des weiteren wird anhand eines Beispieles, der Erfassung eines simplen Lernobjekts, gezeigt, wie ein Teil des Programms funktioniert.

4.1 Der Aufbau von Courseman

Courseman wurde mit Hilfe von *Forte4Java 3.0 Community Edition* entwickelt. Dies ist eine von Sun Microsystems gegen Registrierung frei erhältliche Entwicklungsumgebung. Erfreulicherweise ist Tomcat, der freie Java Servlet Applikationserver, hier integriert und kann direkt aus Forte heraus gestartet werden, der Sourcecode wird automatisch kompiliert, der Servlet Code an die richtigen Stellen kopiert. Es muss nur noch ein Browser gestartet werden, um auf den Server zugreifen zu können.

Dieser Prozess ist bei der Entwicklung eine grosse Unterstützung, denn alleine mit dem JDK und einer separater Tomcat Installation beansprucht der Buildprozess, d.h. das Kompilieren, Kopieren, Anpassen der Datenbank, Einbinden und Starten unnötig viel Zeit.

4.1.1 Voraussetzungen zur Installation

Courseman wurde mit JDK Version 1.4 entwickelt. Die grösste Neuerung gegenüber den früheren Versionen ist, dass viele XML Werkzeuge bereits integriert sind. Sollte *Courseman* auf einer älteren JDK Version ausgeführt werden, müssen diese XML APIs extern eingebunden werden, z.B. der XSLT Prozessor Xalan. Daher empfiehlt es sich, JDK 1.4 oder höher zu verwenden.

Courseman operiert über drei Datenschemata: Kurse und deren Unterelemente müssen zum einen als XML Dokumente ausgegeben werden können, dies ist eine Anforderung an *Courseman*. Die Repräsentation der Daten im XML Format ist somit nötig, damit ein Kurs als XML Dokument für *Calmecac* exportiert werden kann.

Das zweite Datenschema ist das von *Courseman* selbst, wie es Kurse und deren Elemente intern mit Hilfe von Java Klassen verwaltet. Diese Klassen werden automatisch mit Hilfe von Castor generiert und sind im Java Paket *courseman.data.castor* enthalten. Diese Klassen benötigen aber die Castor *XML Data Binding* API, daher wird für *Courseman* die Castor XML Bibliothek benötigt. Für *Courseman* wurde die Version 0.9.3.9 verwendet.

Das dritte Datenschema wird für die Datenbank FastObjects benötigt für die persistente Speicherung der Daten. Daher muss diese Datenbanksoftware auf dem Server installiert sein, und die enthaltene Java Bibliothek muss ebenfalls in den Classpath aufgenommen werden, damit auf die Datenbank API zugegriffen werden kann. Für *Courseman* wurde die die Demoversion Fastobjects Trial Edition 8.0.1.25 verwendet. Selbstverständlich wird für den Produktionsbetrieb eine Vollversion benötigt, um nicht gegen die Lizenzen zu verstossen.

Des weiteren wird ein Applikationsserver benötigt, welcher mit Java Servlets und JSP umgehen kann. *Courseman* wurde zwar nur mit Tomcat Version 3.1 bis 3.3 getestet, sollte

aber auch mit anderer Software zusammenarbeiten, welche die Java Servlet Spezifikationen nach Version 2.2 bzw. JSP Spezifikationen Version 1.1 unterstützen.

4.1.2 Verzeichnisstruktur von Courseman

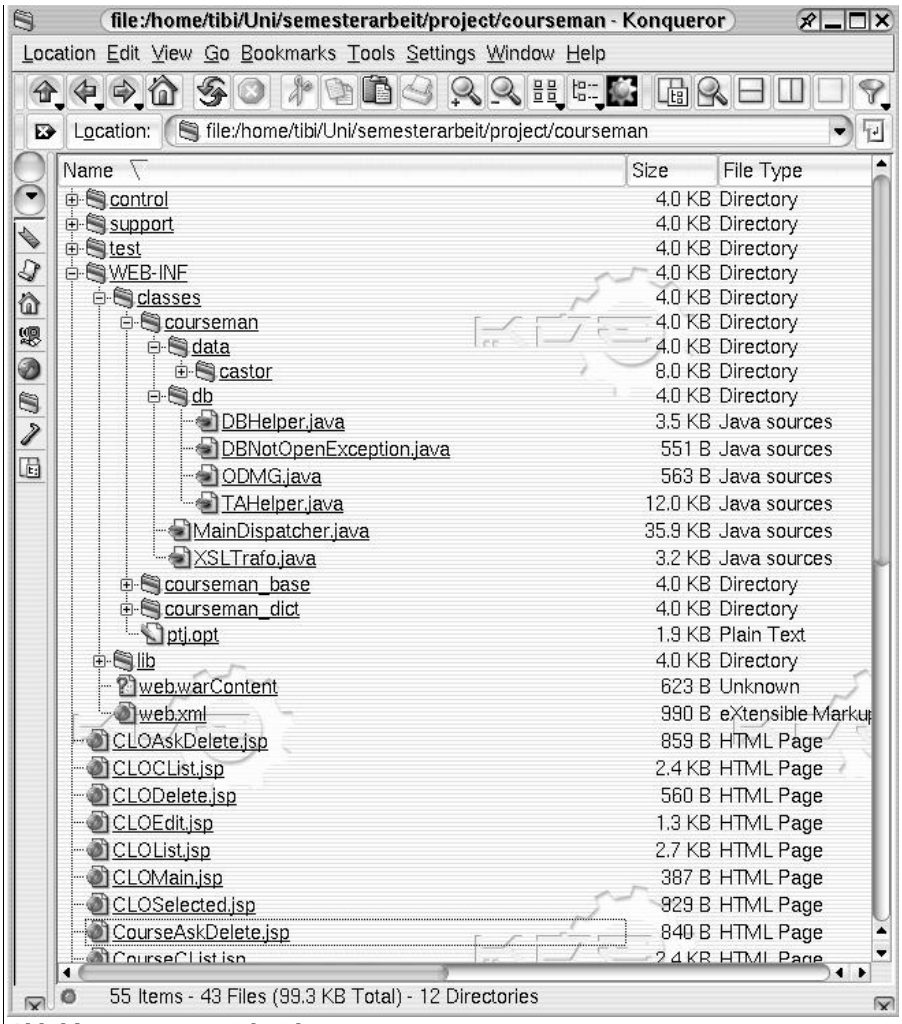

Abbildung 4.1 Verzeichnisbaum courseman

Das Verzeichnis **courseman/** in Abbildung 4.1 ist das so genannte Serverroot Verzeichnis, d.h. dies ist das oberste Verzeichnis, auf dieses kann indirekt über den Browser zugegriffen werden. Dieses Verzeichnis muss als Applikation in Tomcat installiert werden, in der Konfigurationsdatei von Tomcat (server.xml) kann dieser Applikation eine URL zugewiesen werden, über welche die Applikation vom Browser aus erreichbar sein soll. In unserem Beispiel ist das immer die URL "http://localhost:8080/courseman". In diesem Verzeichnis befinden sich die meisten JSP Dateien.

Im Verzeichnis **control/** befinden sich Hilfs JSP Dateien, diese Seiten werden aber nie im Browser angezeigt, deshalb sind diese hier getrennt von den haupt JSP Dateien gespeichert.

Das **support/** Verzeichnis enthält alle XML und XSLT Dateien.

Über den externen Browserzugriff sind nur die statischen Dateien aus diesen Verzeichnissen abrufbar. Die dynamischen JSP Seiten können nur direkt ausgeführt werden, nicht aber als Sourcecode heruntergeladen werden. Der Applikationserver weiss dies zu verhindern.

Auch auf das spezielle Verzeichnis **WEB−INF/** kann über den Webbrowser nicht direkt zugegriffen werden. Hier könnten sich weitere Java Bibliotheken im **lib/** Verzeichnis befinden, dieses ist aber im Fall von *Courseman* leer, da die externen Bibliotheken über den Classpath von Java Systemweit eingebunden sind. **web.xml** ist eine Konfigurationsdatei für die Applikation, hier kann z.B. definiert werden, welches die Standarddatei sein soll, wenn im Browser nur das Applikationsverzeichnis angegeben wird, statt direkt die auszuführende Datei. Hier ist das index.jsp (oder falls nicht gefunden index.html oder index.htm). Des weiteren wird die Variable *session−timeout* auf 30 Minuten gesetzt, sollte ein Benutzer 30 Minuten lang nichts tun, verfällt sein Sitzungsobjekt und er muss seine Sitzung wieder von vorne beginnen.

Im Verzeichnis **classes/** befinden sich schliesslich alle Java Pakete wie auch die Datenbank. In diesem Verzeichnis befindet sich die FastObjects Konfigurationsdatei **ptj.opt**, die Datenbank selbst ist auf die beiden Verzeichnisse **courseman_base/** und **courseman_dict/** verteilt. Diese zwei Verzeichnisse werden ausschliesslich durch FastObjects verwaltet.

Die Klassendateien von *Courseman* befinden sich alle im Java Paket **courseman**. Direkt in diesem Verzeichnis befindet sich die zentrale **MainDispatcher.java** Klasse, welche die meisten Browseranfragen empfängt, die Daten auswertet, benötigte Daten bereitstellt und die Anfrage an die richtige JSP Datei weiterleitet, damit diese die Daten für den Browser aufbereiten und ausgeben kann. **XSLTrafo.java** ist ein Hilfsklasse, welche für die Verarbeitung von XML Daten durch XSLT Dateien zuständig ist.

Unterhalb von **courseman/** befinden sich noch die beiden Java Pakete **courseman.db** und **courseman.data.castor**. In **db/** befinden sich Hilfsklassen, welche für den Datenbankzugriff verantwortlich sind, während sich in **data/castor/** alle durch Castor generierte Java Klassen befinden.

4.2 Diskussion des Codes

In diesem Abschnitt soll anhand eines Beispieles gezeigt werden, wie ein Programmzyklus abläuft. Iinsbesondere, wie XML Daten in Java Objekte umgewandelt werden, wie diese in der Datenbank gespeichert werden, später von dort wieder ausgelesen werden und schlussendlich im Browser angezeigt werden. Im Beispiel soll ein simples Lernobjekt erfasst werden und auch wieder ausgegeben werden können.

4.2.1 Initialisierung der Sitzung

Sitzung (engl.: session) ist eines der wichtigsten Konzepte bei Java Servlets. Denn das grösste Problem bei Webanwendung ist, dass HTTP statuslos (engl.: stateless) ist: der Server kann nicht wissen, im welchem Kontext sich der Endbenutzer befindet, welches seine vorhergehenden Schritte waren. In Java gibt es daher ein Objekt, welches eine Instanz der Java

Klasse "session" ist, das einem Client zugewiesen wird und existiert im Server so lange, bis der Benutzer nicht mehr auf den Server zugreift. Sollte sich der Client nach seiner Sitzung selber nicht abmelden, womit sein Sitzungsobjekt zerstört würde, wird es erst nach einer definierten Zeit der Inaktivität gelöscht.

Wie kann aber der Server einen Benutzer erkennen? Jedem wird ein eindeutiger Identifikationsschlüssel vergeben. Diesen muss der Client bei jedem Zugriff dem Server mitteilen. Hierfür gibt es verschiedene Möglichkeiten, z.B. über HTTP−Cookies: Cookies erlauben dem Server, Variablen im Webbrowser des Clients zu setzen und diese wieder auszulesen. Die meisten heute eingesetzten Browser unterstützen zwar Cookies, aber vorsichtige Benutzer, welche nicht wünschen, über Cookies identifiziert werden zu können, schalten diesen Mechanismus aus. Aber selbst hierfür gibt es eine Lösung: mit der Servlet Methode **`response.encodeRedirectURL()`** kann an die Ziel−URL, welche dem Browser geschickt wird, um die nächste Seite anzuzeigen, der Sessionschlüssel angehängt werden. In der nächsten Anforderung des Clients wird die ursprüngliche URL dem Server übergeben und zwar inklusive diesem Sitzungsschlüssel. Die oben genannte Methode erkennt, ob der Client Cookies unterstützt und hängt nur in diesem Fall den Identifikationsschlüssel an die URL an.

Eine Sitzung wird gestartet, indem die URL der Applikation im Browser aufgerufen wird, hier ist dies "http://localhost:8080/courseman/". Es wird automatisch die Datei index.jsp ausgeführt, welche eine eventuell vorhandenes Sitzungsobjekt löscht. Jetzt wird die MainDispatcher.java Klasse aufgerufen, welche die Initialisierungsroutinen enthält. In das Sitzungsobjekt werden einige Objekte gespeichert, das wichtigste ist der TAHelper. Dies ist eine Hilfsklasse, welche für den Datenbankzugriff benötigt wird.

Nach der Initialisierung wird das Hauptmenü angezeigt, für diese Seite ist StartPage.jsp zuständig. Hier wählen wir SLO aus. Daraufhin folgt die SLO Hauptseite, auf welcher alle vorhandenen Lernobjekte aufgelistet sind. Unter anderem existiert im Sitzungsobjekt eine Variable Kontext, in welcher hier "SLO" als Kontext gesetzt wird.

In MainDispatcher.java veranlasst folgender Javacode den Browser, die Hauptseite zu laden: **`response.sendRedirect(response.encodeRedirectURL("/courseman/StartPage.jsp"));`**

4.2.2 Einfügen eines leeren simplen Lernobjekts

Ein neues Lernobjekt wird eingegeben, indem in einem ersten Schritt ein leeres SLO in die Datenbank eingefügt wird. In einem zweiten Schritt wird dann das neu erstellte Lernobjekt bearbeitet.

```
Simplelearningobject theSLO = null;
URL url = new URL("http://localhost:8080/courseman/support/emptyslo.xml");
theSLO = new Simplelearningobject().unmarshal(new InputStreamReader(url.openStream()));
tah.addSimplelearningobject(theSLO);
response.sendRedirect(response.encodeRedirectURL("/courseman/SLOList.jsp"));
```

Abbildung 4.2 Auszug MainDispatcher.java: neues, leeres Lernobjekt einfügen

Um ein leeres, simples Lernobjekt zu erstellen, muss auf den Knopf *AddNew* gedrückt werden. Dadurch wird die MainDispatcher.java Klasse aufgerufen und unter anderem der in

Abbildung 4.2 dargestellte Code abgearbeitet. Die URL in der zweiten Zeile zeigt auf eine XML Datei, welches eine Vorlage für ein leeres SLO ist. In der dritten Zeile wird mit der unmarshal Methode diese XML Datei eingelesen und in das Java Objekt der Instanz des simplen Lernobjekts gespeichert. Diese simple Lernobjekt Klasse wurde zuvor mit Castor erstellt. Das leere SLO ist aber jetzt erst im Hauptspeicher vorhanden und muss in die Datenbank gespeichert werden. Dieser Befehl muss für ein Objekt nur ein einziges Mal ausgeführt werden, denn damit wird das Objekt an die Datenbank gebunden, es ist nun persistent. Sobald sich nun das Objekt verändert, wird die Datenbank automatisch nachgeführt. Änderungen in der Datenbank dürfen aber nur innerhalb einer Transaktion stattfinden. Dies bewerkstelligt die nächste Zeile, indem es die tah (TransAction Helper) Klasse anweist, "theSLO" in der Datenbank zu speichern. Im Kern führt diese Methode folgende Befehlszeile aus: **`db.bind(theSLO,null)`**. Hierbei ist db die Datenbank, mit der Methode *bind* wird das Objekt in der Datenbank erstellt und an das Java Objekt gebunden. Mit der letzten Zeile wird dann wieder die gleiche Seite wie vorher aufgerufen, jetzt erscheint aber das neue, leere SLO in der Liste.

4.2.3 Bearbeiten des leeren Lernobjekts

Hierfür muss das gewünschte simple Lernobjekt markiert werden und danach auf den "Edit" Knopf gedrückt werden. Dadurch wird einmal mehr die MainDispatcher.java Klasse aufgerufen. Das selektierte SLO wird im, Sitzungsobjekt gespeichert. Die Klasse ruft nun SLOEdit.jsp Datei auf, welche ein Formular darstellen soll, um das Lernobjekt bearbeiten zu können.

```
StringWriter aWriter = new StringWriter();
StringWriter sw = new StringWriter();
tah.beginTA();
theSLO.marshal(aWriter);


StreamSource xmlDoc = new StreamSource(new StringReader(aWriter.toString()));
StreamSource xslDoc = new
StreamSource("http://localhost:8080/courseman/support/editslo.xslt");
courseman.XSLTrafo.doXSLT(xmlDoc, xslDoc, sw);


xmlDoc = new StreamSource(new StringReader(sw.toString()));
xslDoc = new StreamSource("http://localhost:8080/courseman/support/editform.xslt");
courseman.XSLTrafo.doXSLT(xmlDoc,xslDoc,out);
tah.commitTA();
```

Abbildung 4.3 Auszug aus SLOEdit.jsp

Das Kernstück aus SLOEdit.jsp ist in Abbildung 4.3 dargestellt. Nachdem theSLO und tah aus dem Sessionobjekt ausgelesen wurden, werden zwei StringWriter initialisiert, das sind Zwischenspeicher für Texte. Mit theSLO.marshal() wird das SLO nach XML konvertiert, das Ergebnis XML Dokument wird im StringWriter zwischengespeichert. Da theSLO jetzt mit der Datenbank verbunden ist, kann dies natürlich nur innerhalb einer Transaktion geschehen, welche zuvor mit tah.beginTA() gestartet wurde, und am Schluss mit tah.commitTA() beendet werden wird.

Es folgen nun zwei XSLT Umwandlungen. Zuerst wird auf das neue XML Dokument die XSLT Steuerdatei "etitslo.xslt" angewendet. Diese wandelt das SLO spezifische XML Dokument in ein standardisiertes Dokument um, welches dann bei der zweiten Umwandlung daraus mittels "editform.xslt" ein HTTP Formular erstellt. Diese zwei Umwandlungen sind nötig, damit jedes spezifische XML Dokument (simples, zusammengesetztes Lernobjekt, Modul und Kurs) in ein standardisiertes XML Dokument umgewandelt werden, in welchem die Tags alle gleich sind. Diese zweite XSL Transformation kann dann die Elemente mit den neuen Tags in HTML Felder zur Eingabe umwandeln. Ausserdem wird das standardisierte XML Zwischenergebnis ebenfalls für die Listenansicht gebraucht, damit wieder nur ein einziges XSLT nötig ist für alle Listen.

```
<%
        TAHelper tah=(TAHelper)session.getAttribute("dbh");
        tah.beginTA();
%>
<jsp:useBean id="SLOSelection" class="courseman.data.castor.Simplelearningobject"
scope="session"  />
<jsp:setProperty name="SLOSelection" property="*" />
<%
        tah.commitTA();
%>
```

Abbildung 4.4 Auszug aus control/SLOBeanSaver.jsp

Jetzt können die Felder des Lernobjekts ausgefüllt werden. Die Daten werden beim Drücken des Speichern Knopfes an control/SLOBeansaver.jsp in Abbildung 4.4 übergeben. Wie können nun die Inhalte der Felder in ein SLO Objekt gespeichert werden? Normalerweise müssten jetzt das HTTP−Request Objekt ausgelesen werden und alle einzelnen HTTP Parameter einzeln ausgelesen und in theSLO gespeichert werden. Mit JSP geht dies aber auch viel einfacher: mit dem <jsp:useBean> Tag wird das ausgewählte simple Lernobjekt "SLOSelection" aus dem Sitzungsobjekt geholt. Hier wird die Technik von Java Beans angewendet, d.h. das Lernobjekt wurde gar nicht direkt übergeben, sondern es wird nach einem passenden Objekt im Speicher gesucht. Mit dem zweiten <jsp:setProperty> Tag werden automatisch für alle vom Formular erhaltenen Variablen entsprechende set−Methoden vom Lernobjekt aufgerufen. Hierzu müssen die Variablennamen des HTTP Formulars gleich heissen wie die des SLOs. Am Schluss wird wieder das ursprüngliche Fenster mit der Liste aller Lernobjekte angezeigt, selbstverständlich mit dem geänderten Objekt.

4.2.4 Ausgabe

Nachdem ein Kurs zusammengestellt wurde, muss es als XML Datei ausgegeben werden können. Dies geschieht, indem in der Listenansicht der gewünschte Kurs ausgewählt wird und auf den Speichern Knopf gedrückt wird. Die Datei wird unter /tmp/output.xml erstellt.

```
File file = new File("/tmp/output.xml");
Writer aWriter = new FileWriter(file);
tah.beginTA();
theCourse.marshal(aWriter);
tah.commitTA();
```

Abbildung 4.5 Auszug aus MainDispatcher.java: Speichern der XML Datei

Hierbei wird in MainDispatcher.java in Abbildung 4.5 aus dem Sessionobjekt der selektierte Kurs, *theCourse*, geladen. Statt nun einen *StringWriter* oder den *HttpWriter* für die Ausgabe zu verwenden, wird mit einem *FileWriter* eine Datei erstellt. Danach kann vom Kurs Objekt *theCourse* die marshal Methode aufgerufen werden, welche wegen der Datenbank innerhalb einer Transaktion geschehen muss.

5 Ergebnisse und Ausblicke

In diesem Abschnitt sollen die erreichten Ziele, die Probleme während der Implementierung und Ausblicke diskutiert werden.

5.1 Erreichte Ziele

Das Hauptziel war, einen Prototypen zu Implementieren, womit man Kurse erstellen, verwalten und ausgeben kann, welche zu *Calmecac* kompatibel sind. Insbesondere soll hiermit gezeigt werden, dass die eingesetzte Technologien miteinander funktionieren. Da es sich hierbei um einen Prototypen handelt, lag der Schwerpunkt nicht auf einem benutzerfreundlichen Endprodukt, was gar nie die Aufgabe war, sondern es ging darum, zu zeigen, dass *Courseman* im Prinzip funktioniert.

Daher ist das Produkt im aktuellen Zustand nicht für den produktiven Einsatz geeignet. Was hier noch fehlt, soll im Abschnitt 5.4 Ausblicke behandelt werden.

Die gestellten Aufgaben wurden jedenfalls erfüllt, *Courseman* funktioniert. Es ist möglich, damit Kurse zu erstellen, indem die einzelnen Elemente (simple und zusammengesetzte Lernobjekte, Module und Kurse) erstellt, in einer Datebank abgelegt, aus der Datenbank wieder ausgewählt und zu neuen Kursen neu verknüpft werden können. Sogar eine Suchfunktion wurde implementiert, damit eine bestimmte Auswahl von Elementen angezeigt wird.

Auch meine persönlichen Ziele wurden erfüllt: erstmals konnte ich eine wirklich komplexe Applikation entwerfen und implementieren. Dabei habe ich viel gelernt, insbesondere musste ich mich nicht nur in einige neue Produkte und Technologien einarbeiten, sondern musste diese zusammen zum Laufen bringen. Ich hatte zwar schon ein wenig Erfahrung in der Programmierung mit Java, hörte letztes Jahr im Datenbankseminar einiges über XML, hielt selber einen Vortrag über XML Anfragesprachen [DEK01]. Im Studienprojekt hatte ich die Gelegenheit, als Nebenprojekt eine ganz kleine Webapplikation mit Tomcat zu erstellen, welche Daten aus einer Access Datenbank ausgibt. Für diese Arbeit waren das alles aber höchstens Grundlagen, denn ich hatte noch nie zuvor mit XML Daten gearbeitet oder etwas mit einer objektorientierten Datenbank zu tun. Sowohl FastObjects, als auch JAXB, Castor, und die XML APIs (DOM, SAX, XSLT) waren mir völlig neu. Es war bereits eine Herausforderung, mich mit den einzelnen Produkten auseinander zu setzen, diese aber dann miteinander auch noch anzuwenden, war eine echte Knacknuss.

5.2 Probleme und Fehler

Vom Aufwand her gesehen habe ich diese Arbeit stark unterschätzt. Das erlernen der eingesetzten Technologien hat viel Zeit beansprucht, die Java Servlet Programmierung ist nicht zu vergleichen mit einer herkömmlichen Java Applikation, womit ich ja bereits immerhin ein wenig Erfahrung hatte. Das ganze dann aber auch noch mit XML APIs, *XML Data Binding* und der Datenbank zu koppeln, scheint die Komplexität exponentiell wachsen zu lassen. Zum Glück wurde ziemlich schnell klar, dass meine ursprüngliche Idee, ohne eine

IDE[24] zu programmieren, nicht praktikabel ist, da zu viele Dateien hätten verwaltet werden müssen. Diese dann von Hand zu kompilieren und herum zu schieben, um sie ausführbar zu machen, wäre zu viel unnötiger Aufwand gewesen. Glücklicherweise gibt es das freie For−te4Java, welches sogar unter meinem Linux Betriebssystem läuft.

Damit war zwar ein Problem aus dem Weg geräumt, aber es wurden zwei neue geschaf−fen: natürlich muss zuerst auch der Umgang mit diesem Werkzeug erlernt werden. Nachdem ich damit bereits einigermassen zurechtgekommen bin und mein Entscheid, es einzusetzen, schon gefallen war, tauchte unerwartet ein schweres Problem auf: ich konnte kein "]" Symbol eingeben! Es ist unglaublich, aber wahr. Alle Tasten funktionierten, ausser dieser einen. Natürlich war dies eine mühsame Beeinträchtigung, konnte ich doch dieses Problem behelfsmässig nur per kopieren/einfügen lösen. Natürlich habe ich mit allen Mitteln versucht, das Problem zu lösen, ich habe Kontakt mit anderen Forte4Java Anwendern via Newsgroups gesucht, in Foren geschrieben, bis ich herausgefunden habe, dass es sich hierbei tatsächlich um einen Fehler im JDK handelt. Das Problem tritt aber nur unter Linux und anderen Unix Derivaten auf, und das nicht nur bei Forte4Java, sondern mit allen Java Applikationen, wel−che Java Swing benutzen! Nachdem ich Sun eine Fehlermeldung geschickt habe, wurde dies, nachdem ein paar Emails zwischen einem Sun Mitarbeiter und mir gewechselt wurden, be−stätigt. Das Problem tritt nur auf mit gewissen nicht amerikanischen Tastaturen, wie auch meiner normalen Schweizerdeutschen Tastatur. Ich habe dann herausgefunden, dass all die−jenigen Tasten nicht funktionieren, welche ein sogenanntes diaeresis[25] Zeichen ergeben. In−dem ich dann das Tastaturlayout so angepasst habe, dass ein "]" erstellt wird, ohne die "Alt Gr" Taste drücken zu müssen, habe ich eine gute Lösung gefunden. Ich hoffe, dass dieser Fehler in der nächsten JDK Version behoben wird.

Ein anderer Fehler, der mich wirklich sehr viel Zeit gekostet hat, war JAXB, das *XML Data Binding Framework* von Sun. Auch hierfür musste ich natürlich den Umgang erlernen, und konnte es so nach einiger Zeit anwenden, es hat sogar ziemlich gut funktioniert. Leider habe ich erst gegen den Schluss, nachdem schon viel Code bereits vorhanden war, mich um die Datenpersistenz gekümmert und mich erst dann mit FastObjects auseinandergesetzt. Denn eigentlich sollte diese Datenbank jegliche Java Klassen speichern können. Leider konnte ich aber unter keinen Umständen die mit JAXB erstellten Klassen mit FastObjects persistent machen! Natürlich habe ich zuerst nicht an eine Inkompatibilität gedacht, sondern suchte nach dem Fehler, den ich meinte gemacht zu haben. Erst später stellte sich heraus, dass sich JAXB und FastObjects tatsächlich nicht vertragen! Daher musste ich alle JAXB Klassen und den Programmcode, welcher auf die JAXB API zugreift ersetzen und mich in ein neues, anderes *XML Data Binding Framework* einarbeiten. Da eines der Beispielprogramme in FastObjects Castor verwendet, wusste ich, dass dies meine Wahl sein muss.

Dies verursachte dann gleich das nächste Problem: Das Schema der XML Kurs Datei ist in einem DTD definiert, Castor verlang aber nach einer XML Schema Datei. Diese von Hand zu konvertieren hätte wieder einmal einen zu grossen Zeitverlust mit sich gebracht. Aber es soll ja XML Editor Programme geben, welche DTDs nach XML Schema konvertieren kön−nen sollen. Ich habe sicher drei Demoversionen kommerzieller Produkte gefunden, welche sogar 30 Tage uneingeschränkt lauffähig sind. Leider funktionierte aber aus irgend einem

24 Integrated Development Environment, integrierte Entwicklungsumgebung
25 Tastendruck, welcher den Cursor nicht weiterbewegen, z.B. "´", erst nach einem "e" erhhält man ein "é"

Grund die konvertierte Datei nie richtig mit Castor. Schlussendlich hätte ich ein brauchbares Werkzeug gefunden, aber das konvertierte strikt nach der neueste XML Schema Spezifikation, Castor verlange aber offenbar die Datei im älteren Format, dieses konnte aber keines der Produkte erstellen (alte Dateien in neue konvertieren selbstverständlich schon). Schlussendlich stellte sich heraus, dass die Castor Version, welche bei FastObjects mitgeliefert wurde, älteren Datums war. Ein Update brachte dann den gewünschten Erfolg.

Bei der Implementation musste ich oft mit ähnlichen Problemen kämpfen, welche sich schlussendlich eigentlich als trivial herausstellten. Aber da es sich um eine recht komplexe Materie handelt, gleicht das Aufspüren des an sich kleinen Problems der sprichwörtlichen Suche nach der Stecknadel im Heuhaufen. Aha−Erlebnisse kamen daher recht häufig vor.

Ein weiteres grosses Problem wurde mit der Zeit der Speicherbedarf der ganzen Entwicklungsumgebung. Solange nur ein paar wenige Dateien vorhanden waren, schien alles noch brauchbar zu sein. Sobald dann aber für das Testen der integrierte Tomcat Server gestartet werden musste, das Programm von den XML Transformatoren, Castor und der Datenbank Gebrauch machte, und um das ganze testen zu können ja auch noch ein Browser gestartet werden musste, reichten 256 MB Hauptspeicher (!!) einfach nicht mehr aus. Die Auslagerungsdatei wurde stark frequentiert, und alles schien ewig zu brauchen. Es blieb nichts anderes übrig, als den Computer aufzurüsten, was dann das Problem löste. Von der Geschwindigkeit her ist somit ein Pentium II Computer mit rund 400 MB Hauptspeicher und 500MHz Prozessor für eine solche Entwicklungsumgebung das absolute Minimum. Um die Applikation vor Ort demonstrieren zu können, habe ich alles auf einem noch nicht zu altem Notebook installiert. Mit 196MB Hauptspeicher und einem 300MHz Prozessor war die Leistung klar beeinträchtigt, an eine Entwicklung mit so einer Maschine ist nicht zu denken.

5.3 Ausblicke

Wie bereits erwähnt handelt es sich hier um einen Prototyp, welches noch weit von einem benutzerfreundlichen Endprodukt entfernt ist. Es müssten Exceptions abgefangen werden und die Eingaben der Benutzer zunächst validiert werden.

Von der Sitzungsverwaltung her ist *Courseman* zwar so ausgelegt, dass mehrere Benutzer gleichzeitig das System benutzen könnten, aber noch spielt die Datenbank nicht mit. Sobald beide Benutzer gleichzeitig je eine Datenbanktransaktion auslösen, erhält ein Benutzer eine Fehlermeldung, dass bereits schon eine Transaktion in Gang sei. Um dies zu lösen, müssten die Transkationen in eigenen Java Threads ausgeführt werden. Ein paar wenige Befehle greifen aber auf die gerade laufende Transaktion zu, bei zwei parallel laufenden Transaktionen würde hier wiederum ein Fehler entstehen. Aber mit etwas Aufwand liesse sich das ohne weiteres implementieren.

Das Datenmodell des Kurses ist zwar zu *Calmecac* kompatibel, aber es gibt ein paar wenige Einschränkungen, welche gemäss dem DTD erlaubt wären. Z.B. kann ein zusammengesetztes Lernobjekt nicht nur simple, sondern auch zusammengesetzte enthalten. Ein Modul wiederum kann nicht nur zusammengesetzte, sondern auch simple Lernobjekte enthalten. Das Datenmodell trägt dem zwar Rechnung, wurde doch das Java Objektmodell direkt aus dem DTD erstellt, aber *Courseman* selbst ist nicht flexibel genug. Würde *Courseman* daher er−

weitert, um alle Möglichkeiten gemäss DTD zu bieten, müssten weiter beachtet werden, dass z.B. keine Rekursionen entstehen dürfen, indem ein zusammengesetztes Lernobjekt sich selbst als CLO enthalten würde.

Des Weiteren müsste natürlich die Benutzeroberfläche zweckmässiger und wenn möglich ansprechender gestaltet werden. Dies könnte aber dank der JSP Technik einem Webdesigner überlassen werden, der sich mit dem Java Code gar nicht auseinander zu setzen braucht.

6 Zusammenfassung

In dieser Arbeit ging es darum, für eine existierende Lernumgebung, *Calmecac*, einen Prototypen einer Zusatzapplikation, *Courseman*, zu erstellen, mit welchem Lernkurse aus Elementen erstellt und verwaltet werden können. Im Mittelpunkt stand der Einsatz neuer Technologien. Als eine Webanwendung wurde *Courseman* mit Java Servlets und JSPs implementiert, und wird auf dem freien Applikationsserver Tomcat ausgeführt. Für den Datenaustausch zwischen *Courseman* und *Calmecac* wird eine XML Datei verwendet, das XML Datenmodell wurde in einer DTD Datei spezifiziert. Das XML Datenschema wird mit dem *XML Data Binding Framework* von Castor an das Java Objektmodell gebunden. Für die Darstellung und Umwandlung der XML Daten wird XSLT verwendet, ein grosser Teil der API ist bereits im neuen JDK 1.4 integriert. Für die Datenpersistenz ist die Datenbank FastObjects von POET zuständig.

Glossar

API
Application Programming Interface. Dies ist eine Programmierschnittstelle, eine Bibliothek von z.B. Java Klassen, welche von einem Hersteller als Sourcecode und als ausführbaren Code geliefert wird, welcher auch verändert unter der gleichen Lizenz weitergegeben werden darf.

BNF (Backus Naur Form)
Bei BNF werden Terminale mit Hochkommas (’) Umschlossen, geschweifte Klammern ({}) bezeichnen keine, eine oder mehrere, durch Kommas getrennte Elemente, eckige Klammern ([]) bedeuten optionales Auftreten, und ein senkrechter Strich (|) steht für genau eine Auswahl der dadurch getrennten Elemente.

HTML
HyperText Markup Language ist eine designorientierte Beschreibungssprache.

Lerninhalt
Lerninhalte sind Daten, die mit rechnergestütztem Lernen im Zusammenhang stehen. Insbesondere Metadaten über Lernobjekte sind Lerninhalte, wie Kurse, Module, CLOs.

Lernobjekt
Ein Lernobjekt ist eine Einheit, die während dem Lernen verwendet bzw. wiederverwendet wird, oder auf die verwiesen werden kann.

Lernverwaltungssystem
Ein Lernverwaltungssystem ist ein beliebiges Technologiesystem zum Planen, Organisieren, Implementieren und Steuern verschiedener Aspekte eines Lernprozesses.

XML
Die eXtensible Markup Language ist eine standardisierte Metasprache, mit welchem sich eine Auszeichnungssprache anwendungsspezifisch definieren lässt.

Literaturverzeichnis

[ADL01] Adler, Sharon und Berglund, Anders, Extensible Stylesheet Language (XSL), 2001, W3C, www.w3.org/Style/XSL

[BRA00] Bray, Jean, Extensible Markup Language (XML) 1.0 (Second Edition), 2000, W3C, www.w3.org/TR/REC−xml

[BUC00] Buck, Lee und Goldfarb, Charles, Datatypes for DTDs (DT4DTD) 1.0, 2000, W3C, www.w3.org/TR/dt4dtd

[CLA99] Clark, James, XSL Transformations (XSLT), 1999, W3C, www.w3.org/TR/xslt

[DAV99] Davidson, James Duncan und Coward, Danny, Java Servlet Specification, v2.2, 1999, Sun Microsystems

[DEK01] Dekany, Tibor, XML Anfragesprachen, 2001, Institut für Informatik, Universität Zürich

[FAL01] Fallside, David, XML Schema, 2001, W3C, www.w3.org/XML/Schema

[LIN99] Lindholm, Tim und Yellin, Frank, The Java Virtual Machine Specification 2nd Edition, 1999, Sun Microsystems

[MAN02] Mangner, Torsten, Verarbeitung, Austausch und Speicherung von Dokumenten mit Hilfe von XML Data Binding, 2002, Studienjahresarbeit, Technische Universität Ilmenau, Deutschland

[MIL01] Milic, Robert, Realisierung eines Systems für die Verwaltung von Lerninhalten, 2001, Institut für Informatik, Universität Zürich

[PEL01] Pelegrí−Llopart, Eduardo, JavaServer Pages Specification, Version 1.2, 2001, Sun Microsystems

[POE01] POET, FastObjects Collected Technical Documentation − Java Platform, 2001, POET

[RAG99] Ragett, Dave, HTML 4.01 Specification, 1999, W3C, www.w3.org/TR/html4

Anhang A − Course.dtd

Dies ist das DTD, welches das Schema einer XML Kurs Datei spezifiziert.

```
<!ELEMENT
  course(title,prerequisite*,identifier,semester,location,
  usedsupports,content)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT identifier (#PCDATA)>
<!ELEMENT semester (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT usedsupports (support)*>
<!ELEMENT content (module)*>
<!ELEMENT support (#PCDATA)>
<!ELEMENT prerequisite (pidentifier)>
<!ELEMENT pidentifier (#PCDATA)>
<!ATTLIST pidentifier ptype CDATA "">
<!ELEMENT groupname (#PCDATA)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT coursetitle (#PCDATA)>
<!ELEMENT modulename (#PCDATA)>
<!ELEMENT semesterno (#PCDATA)>
<!ELEMENT
  module (title,prerequisite*,
  (compositelearningobject | simplelearningobject)*)>
<!ATTLIST module identifier CDATA "">
<!ELEMENT
  compositelearningobject(href,interactiontype,clotitle,
  clotopic,description,clotimelimitsecs,
  (compositelearningobject | simplelearningobject)*)>
<!ATTLIST compositelearningobject identifier CDATA "">
<!ELEMENT href (#PCDATA)>
<!ELEMENT interactiontype (#PCDATA)>
<!ELEMENT clotitle (#PCDATA)>
<!ELEMENT clotopic (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT clotimelimitsecs (#PCDATA)>
<!ELEMENT
  simplelearningobject(type,description,lolocation,
  topic,timelimitsecs)>
<!ATTLIST simplelearningobject identifier CDATA "">
<!ELEMENT type (#PCDATA)>
<!ELEMENT lolocation (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ELEMENT timelimitsecs (#PCDATA)>
```

Anhang B − Verzeichnisbaum von Courseman

```
courseman/
|-- CLOAskDelete.jsp
|-- CLOCList.jsp
|-- CLODelete.jsp
|-- CLOEdit.jsp
|-- CLOList.jsp
|-- CLOMain.jsp
|-- CLOSelected.jsp
|-- CourseAskDelete.jsp
|-- CourseCList.jsp
|-- CourseDelete.jsp
|-- CourseEdit.jsp
|-- CourseList.jsp
|-- CourseMain.jsp
|-- Header.jsp.jsp
|-- ModuleAskDelete.jsp
|-- ModuleCList.jsp
|-- ModuleDelete.jsp
|-- ModuleEdit.jsp
|-- ModuleList.jsp
|-- ModuleMain.jsp
|-- ModuleSelected.jsp
|-- SLOAskDelete.jsp
|-- SLOChooser.jsp
|-- SLODelete.jsp
|-- SLOEdit.jsp
|-- SLOList.jsp
|-- SLOMain.jsp
|-- SLOSelected.jsp
|-- SLOShow.jsp
|-- SLOThis.jsp
|-- SLOThis_1.jsp
|-- SLOpreList.jsp
|-- StartPage.jsp
|-- WEB-INF
|   |-- classes
|   |   |-- courseman
|   |   |   |-- MainDispatcher.java
|   |   |   |-- XSLTrafo.java
|   |   |   |-- data
|   |   |   |   `-- castor
|   |   |   |       |-- Compositelearningobject.java
|   |   |   |       |-- CompositelearningobjectChoice.java
|   |   |   |       |-- CompositelearningobjectChoiceDescriptor.java
|   |   |   |       |-- CompositelearningobjectChoiceItem.java
|   |   |   |       |-- CompositelearningobjectChoiceItemDescriptor.java
|   |   |   |       |-- CompositelearningobjectDescriptor.java
|   |   |   |       |-- Content.java
|   |   |   |       |-- ContentDescriptor.java
|   |   |   |       |-- ContentItem.java
|   |   |   |       |-- ContentItemDescriptor.java
|   |   |   |       |-- Course.java
|   |   |   |       |-- CourseDescriptor.java
|   |   |   |       |-- Courseman.java
|   |   |   |       |-- CoursemanDescriptor.java
|   |   |   |       |-- CoursemanItem.java
|   |   |   |       |-- CoursemanItemDescriptor.java
|   |   |   |       |-- Module.java
|   |   |   |       |-- ModuleChoice.java
|   |   |   |       |-- ModuleChoiceDescriptor.java
|   |   |   |       |-- ModuleChoiceItem.java
|   |   |   |       |-- ModuleChoiceItemDescriptor.java
|   |   |   |       |-- ModuleDescriptor.java
|   |   |   |       |-- Pidentifier.java
```

```
|   |   |   |       |-- PidentifierDescriptor.java
|   |   |   |       |-- Prerequisite.java
|   |   |   |       |-- PrerequisiteDescriptor.java
|   |   |   |       |-- Simplelearningobject.java
|   |   |   |       |-- SimplelearningobjectDescriptor.java
|   |   |   |       |-- Usedsupports.java
|   |   |   |       |-- UsedsupportsDescriptor.java
|   |   |   |       |-- UsedsupportsItem.java
|   |   |   |       `-- UsedsupportsItemDescriptor.java
|   |   |   `-- db
|   |   |       |-- DBHelper.java
|   |   |       |-- DBNotOpenException.java
|   |   |       |-- ODMG.java
|   |   |       `-- TAHelper.java
|   |   |-- courseman_base
|   |   |   |-- eventlog
|   |   |   |   `-- f0000000.ptd
|   |   |   |-- objects.dat
|   |   |   `-- objects.idx
|   |   |-- courseman_dict
|   |   |   |-- _objects.dat
|   |   |   |-- _objects.idx
|   |   |   `-- eventlog.dic
|   |   |       `-- f0000000.ptd
|   |   `-- ptj.opt
|   |-- lib
|   `-- web.warContent
|   `-- web.xml
|-- control
|   |-- CLOBeanSaver.jsp
|   |-- CourseBeanSaver.jsp
|   |-- ModuleBeanSaver.jsp
|   |-- SLOBeanSaver.jsp
|   `-- beanSaver.jsp
|-- index.jsp
`-- support
    |-- SLOShow.xsl
    |-- Test.xml
    |-- clolist.xslt
    |-- .dtd
    |-- courselist.xslt
    |-- editclo.xslt
    |-- editcourse.xslt
    |-- editform.xslt
    |-- editmodule.xslt
    |-- editslo.xslt
    |-- emptyclo.xml
    |-- emptycourse.xml
    |-- emptymodule.xml
    |-- emptyslo.xml
    |-- modulelist.xslt
    |-- simplelearningobject.dtd
    |-- simplelearningobject.xml
    |-- sloinput.xml
    |-- slolist.xslt
    |-- test.xml
    |-- test.xslt
    `-- xslt.jsp

13 directories, 118 files
```