

Preview

This document is a preview version and not necessarily identical with the original.

Technische Universität Chemnitz
Fakultät für Informatik
Professur Betriebssysteme und Systemprogrammierung

Diplomarbeit

Entwicklung eines WYSIWYG Editors für das Erstellen von Lehrmaterial im
XML Format



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Bearbeitet von: Thomas Weise
geboren am: 19.06.1981
in: Karl-Marx-Stadt

Betreuender Hochschullehrer: Prof. Dr. Ing. habil. Winfried Kalfa
Betreuer: Frau Dr. Wällnitz

Aufgabenstellung für die Diplomarbeit

Bearbeiter: Thomas Weise, Matrikelnummer 25630
Beginn: 01.02.2005
Ende: 30.08.2005
Betreuender Hochschullehrer: Prof. Dr. Ing. habil. Winfried Kalfa
Betreuer: Frau Dr. Wällnitz

Thema: Entwicklung eines WYSIWYG Editors für das Erstellen von Lehrmaterial im XML Format

Zielstellung:

Es ist von folgenden Szenarien auszugehen:

- Schaffung eines zentralen Datenpools für informatische Lehrfächer auf Basis von <ML3>
- Dezentrale Erstellung und Bearbeitung des Datenpools durch nichtinformatisch vorbelastete Lehrkräfte
- Individuelle Zusammenstellung von Online-Modulen zur Ergänzung der Präsenzlehre in Schule und Studium sowie zum Selbststudium

Der Editor basiert auf der Sprache <ML3> des Projekts WWR und soll folgende Anforderungen realisieren:

- Erfassen von Inhalten (Text und Grafik)
- Zusammenstellen von Kursmodulen nach didaktischen Gesichtspunkten (Drag und Drop)
- Ausgabe der Kursmodule als Online-Fassung (HTML-Format) und als Textfassung (PDF-Format)

Dabei sind zu unterstützen:

- verschiedene Layouts für die Darstellung (Beschreibung – einfacher Text, Definition, Satz, Algorithmus, Beispiel, Hinweis, Aufgabe, Anmerkung)
- Einbindung von Listen und Tabellen

Im Rahmen der Arbeit ist zu untersuchen, auf welcher Grundlage der WYSIWYG-Editor zu entwickeln ist. Die Entwurfsentscheidungen sind zu begründen.

Im praktischen Teil soll eine Erprobung mit schulspezifischen Inhalten zur Testung der Handhabbarkeit und Akzeptanz des Editors erfolgen.

Technische Universität Chemnitz
Fakultät für Informatik
Professur Betriebssysteme und Systemprogrammierung

Bibliographische Beschreibung

Entwicklung eines WYSIWYG Editors für das Erstellen von Lehrmaterial im XML Format

Thomas Weise, 2006. 67 S., 24 Abb., 1 Tab., 26 Lit.

Chemnitz, Technische Universität Chemnitz

Fakultät für Informatik

Professur Betriebssysteme und Systemprogrammierung

Diplomarbeit

Kurzreferat

Die vorliegende Arbeit beschreibt den Entwurf, die Entwicklung und die Anpassung eines XML-WYSIWYG-Editors für die Verwendung in der Lehre. Der praktische Teil der Arbeit hat einen Editor zum Ergebnis, welcher den XML-Dialekt KML implementiert, mit dessen Hilfe $\langle ML \rangle^3$ -formatiertes Lehrmaterial erstellt werden kann. Der Entwurf des Editors ist jedoch besonders auf Erweiterbarkeit und Wartbarkeit gewichtet. Dadurch wird der erstellte Editor für beliebige XML-Dialekte mit geringem Aufwand anpassbar.

Hinweis:

Bezeichnungen von Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht als solche kenntlich gemacht. Aus dem Fehlen der entsprechenden Markierungen kann nicht geschlossen werden, dass die Bezeichnung ein freier Warenname ist. Eben so wenig wird auf Patenten oder Gebrauchsmusterschutz hingewiesen.

Abkürzungsverzeichnis

| | |
|-------------------|---|
| API | <u>A</u> pplication <u>P</u> rogrammer's <u>I</u> nterface |
| CSS | <u>C</u> ascading <u>S</u> tylesheet |
| DOM | <u>D</u> ocument <u>O</u> bject <u>M</u> odel |
| GCC | <u>G</u> NU <u>C</u> ompiler <u>C</u> ollection |
| GPL | GNU <u>G</u> eneral <u>P</u> ublic <u>L</u> icense |
| GNU | <u>G</u> NU is <u>N</u> ot <u>U</u> nix |
| GUI | <u>G</u> raphical <u>U</u> ser <u>I</u> nterface |
| HTML | <u>H</u> ypertext <u>M</u> arkup <u>L</u> anguage |
| JAXP | <u>J</u> ava <u>A</u> PIs for <u>X</u> ML <u>P</u> arsing |
| MathML | <u>M</u> athematical <u>M</u> arkup <u>L</u> anguage |
| <ML> ³ | <u>M</u> ultidimensional <u>L</u> earning <u>O</u> bjects and <u>M</u> odular <u>L</u> ectures <u>M</u> arkup <u>L</u> anguage |
| PDF | <u>P</u> ortable <u>D</u> ocument <u>F</u> ormat |
| SAX | <u>S</u> imple <u>A</u> PI for <u>X</u> ML |
| URI | <u>U</u> niversal <u>R</u> esource <u>I</u> dentificator |
| URL | <u>U</u> niversal <u>R</u> esource <u>L</u> ocator |
| WWR | <u>W</u> issenswerkstatt <u>R</u> echensysteme |
| WYSIWYG | <u>W</u> hat <u>Y</u> ou <u>S</u> ee <u>I</u> s <u>W</u> hat <u>Y</u> ou <u>G</u> et |
| XML | <u>E</u> xtensible <u>M</u> arkup <u>L</u> anguage |
| XSD | <u>X</u> ML <u>S</u> chema <u>D</u> efinition |
| XSL(T) | <u>E</u> xtensible <u>S</u> tylesheet <u>L</u> anguage (<u>T</u> ransformations) |

Formatierungsregeln

Um die Verständlichkeit der Arbeit zu erhöhen, werden folgende Formatierungen eingeführt:

Java-Kode oder Pseudocode ist in der Monospace-Schriftart Courier New verfasst.

Beispiel: `public static void print();`

Design-Patterns [dp] werden kursiv dargestellt:

Beispiel: *Factory, Model-View-Controller, Singleton*

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | EINLEITUNG | 13 |
| 2 | ENTWURFSSTUDIE | 14 |
| 2.1 | Analyse des XML-Dialekts der WWR..... | 14 |
| 2.2 | KML..... | 16 |
| 2.2.1 | Inhaltsdialekt..... | 16 |
| 2.2.2 | Didaktikdialekt..... | 16 |
| 2.3 | Definition der Anwendungsfälle..... | 17 |
| 2.4 | Benutzeroberfläche..... | 18 |
| 2.5 | Spezifikation der grundlegenden Ziele des Entwurfs | 19 |
| 2.5.1 | WYSIWYG-Fähigkeit | 19 |
| 2.5.2 | Ausrichtung auf die Lehre | 19 |
| 2.5.3 | Speicherung der Daten im XML-Format | 19 |
| 2.5.4 | Umwandlung der Daten in webfähige, visuelle Formate | 20 |
| 2.5.5 | Kosteneffizienz..... | 20 |
| 2.5.6 | Einfachheit, Verfügbarkeit..... | 20 |
| 2.5.7 | Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit | 20 |
| 2.6 | Auswahl der Entwicklungswerkzeuge | 21 |
| 2.6.1 | Reengineering des bestehenden Editors X2X | 22 |
| 2.6.2 | Erweitern von Microsoft Word durch Makros | 24 |
| 2.6.3 | Programmierung in Java..... | 26 |
| 2.6.4 | Entscheidung für ein Entwicklungswerkzeug | 27 |
| 3 | ENTWURF | 28 |
| 3.1 | Generelle Festlegungen..... | 28 |
| 3.2 | Schichtenarchitektur..... | 30 |
| 4 | SCHICHT 0: JAVA-BIBLIOTHEKEN | 31 |
| 4.1 | Modell der graphischen Oberfläche | 31 |
| 4.2 | Laden, Speichern und Transformieren von XML-Daten | 32 |

| | | |
|-------|--|----|
| 5 | SCHICHT 1: XML-UNTERSTÜTZUNG | 33 |
| 5.1 | XMLTag und XMLTagSet..... | 34 |
| 5.2 | XMLAttribute und XMLAttributeSet | 35 |
| 5.3 | Internationalisierung..... | 36 |
| 5.4 | Plugin-API..... | 37 |
| 6 | SCHICHT 2: DOKUMENTUNTERSTÜTZUNG | 38 |
| 6.1 | Editor-Kern..... | 38 |
| 6.1.1 | <i>Model</i> | 39 |
| 6.1.2 | <i>View</i> | 40 |
| 6.1.3 | <i>Controller</i> | 40 |
| 6.1.4 | Zusammenfassung zur <i>Model-View-Controller</i> Architektur | 41 |
| 6.2 | Transformationsunterstützung | 42 |
| 6.3 | Kompositions- und Inhaltsunterstützung | 43 |
| 6.4 | Hilfesystem | 44 |
| 7 | SCHICHT 3: ANPASSUNGSSCHICHT, XML-DIALEKT | 45 |
| 7.1 | Dialekt-Plugin..... | 46 |
| 7.1.1 | Erzeugung der XMLTags..... | 46 |
| 7.1.2 | Erzeugung des XMLTagSets..... | 47 |
| 7.1.3 | Erzeugung der XMLViewFactory | 49 |
| 7.1.4 | Erzeugung der Dialektaktionen..... | 50 |
| 7.1.5 | Erzeugung der Dialektdefinition | 51 |
| 7.1.6 | Erzeugung des Didaktikdialekts | 52 |
| 7.1.7 | Erzeugung der Hilfeunterstützung des Plugins | 53 |
| 7.1.8 | Erzeugung des Plugins | 54 |
| 7.2 | Transformations-Plugin | 55 |
| 8 | AUSGABE | 56 |
| 8.1 | Webpräsentation | 56 |
| 8.2 | Skripten..... | 57 |
| 8.3 | Unterrichtsfolien | 58 |

| | | |
|------|---|-----------|
| 8.4 | Weitere Formate | 59 |
| 9 | ERPROBUNG MIT SCHULSPEZIFISCHEN INHALTEN | 60 |
| 10 | AUSBLICK..... | 61 |
| 10.1 | Editor für Lehrmaterial..... | 61 |
| 10.2 | allgemeiner XML-Editor..... | 61 |
| 10.3 | Internet-Portal | 62 |
| 11 | LITERATURVERZEICHNIS | 63 |
| 12 | ABBILDUNGSVERZEICHNIS..... | 65 |
| 13 | TABELLENVERZEICHNIS | 66 |
| 14 | QUELLKODEVERZEICHNIS | 66 |

1 Einleitung

Im Februar 2001 startete das Projekt „Wissenswerkstatt Rechensysteme“, kurz WWR. Es handelt sich um ein Verbundprojekt des Bundesministeriums für Bildung und Forschung, an dem sich zwölf deutsche Hochschulen und Partner aus der Industrie beteiligten. Im Rahmen dieses Projekts wurde ein internetbasiertes System von multimedialen Lehr- und Lernmodulen zur Unterstützung der Aus- und Weiterbildung im Wissensgebiet Technische Informatik aufgebaut. [wwr]

Um die Verständlichkeit und Anschaulichkeit der neuen Lehrmaterialien zu erhöhen, lag der Schwerpunkt auf dem Einsatz von Simulationen und Animationen.

Da bei dem Projekt viele Partner kooperieren, waren Richtlinien für ein einheitliches Layout der Lehrmaterialien erforderlich. Außerdem wurde ein Datenformat benötigt, das es erlaubt, neben Grafiken auch Simulationen und Multimediaobjekte anzubinden.

Bereits mit HTML¹ ist es möglich, nahezu beliebige Multimediakomponenten mit Text zu kombinieren. HTML besitzt jedoch einige entscheidende Nachteile.

Der schwerwiegendste Nachteil ist, dass HTML Inhalt und Design in sich vereint. Um einen Abschnitt hervorzuheben wird beispielsweise dessen Schrift rot eingefärbt. Es fehlt jedoch eine explizite Markierung für „wichtig“. Weiterhin muss das zu verwendende Design zu Beginn der Entwicklung von HTML-Dokumenten festgelegt werden. Spätere Änderungen sind aufwändig. Ebenso aufwändig ist es, HTML Dokumente in andere Formate wie zum Beispiel in PDF² umzuwandeln.

Aus diesen Gründen erfolgt die Speicherung des Inhalts der WWR-Module in XML³-Dokumenten. XML ist ein Datenformat, welches erst durch die Umwandlung in andere Formate wie HTML oder PDF dargestellt wird.

Durch die Trennung von Daten und Darstellung kann jedoch auch kein allgemeiner XML Editor in der Art eines WYSIWYG⁴-Textverarbeitungssystems existieren.

Bereits für im Umgang mit XML geschultes Personal ist es mit den bestehenden allgemeinen Hilfsmitteln (zum Beispiel XMLSpy⁵) zeitraubend, längere XML-Dokumente zu erfassen. Nur mit diesen Werkzeugen ist der Einsatz von XML für die Lehre in Gebieten außerhalb der universitären Informatik praktisch nicht realisierbar.

Damit ein XML-Dialekt breitflächig zum Einsatz kommen kann, muss er in Verbindung eines speziell zugeschnittenen WYSIWYG-Editors angeboten werden. Der Entwurf und die Implementation eines solchen Editors ist der Gegenstand dieser Arbeit.

¹ Hypertext Markup Language – Die meisten Webseiten sind in diesem Format abgelegt.

² Portable Document Format – Dieses Format wird häufig für Skripte oder E-Books genutzt.

³ Extensible Markup Language – Ein weit verbreitetes, universelles, modernes Datenformat.

⁴ What You See Is What You Get – Bearbeiten von Dokumenten auf der Ebene der visuellen Darstellung.

⁵ XMLSpy – Mit diesem Programm werden oftmals XML-Dateien bearbeitet.

2 Entwurfstudie

2.1 Analyse des XML-Dialekts der WWR

Die Lehrmaterialien der Wissenswerkstatt Rechensysteme werden in dem speziellen XML-Dialekt $\langle ML \rangle^3$ ¹ gespeichert. Zu Beginn des Entwurfs des Editors soll dieser Dialekt analysiert werden.

Durch die Verwendung von XML wird die graphische Darstellung von dem eigentlichen Inhalt getrennt. $\langle ML \rangle^3$ bietet darüber hinaus noch mehr Flexibilität.

- **Semantische Gliederung**
Es ist möglich, Teile des Inhalts der $\langle ML \rangle^3$ -Dokumente semantisch zu Klassifizieren. Markierungen wie „Beschreibung“, „Definition“, „Anmerkung“, „Satz“ und Ähnliches können für einzelne Abschnitte gesetzt werden.
- **Logische Gliederung**
Wissensgebiete werden in Module eingeteilt. Diese Module können in einer Baumhierarchie in weitere Untereinheiten gegliedert werden.
- **Didaktische Gliederung**
Aus der eingegebenen inhaltlichen Materialbasis wird das individuelle Lehrmaterial zusammengestellt. Dies erfordert oftmals eine didaktische Umordnung. $\langle ML \rangle^3$ ermöglicht dies. Damit die ursprüngliche Zusammenstellung des Lehrmaterials erhalten bleibt, wird die didaktische Gliederung getrennt von der der logischen Gliederung gespeichert.

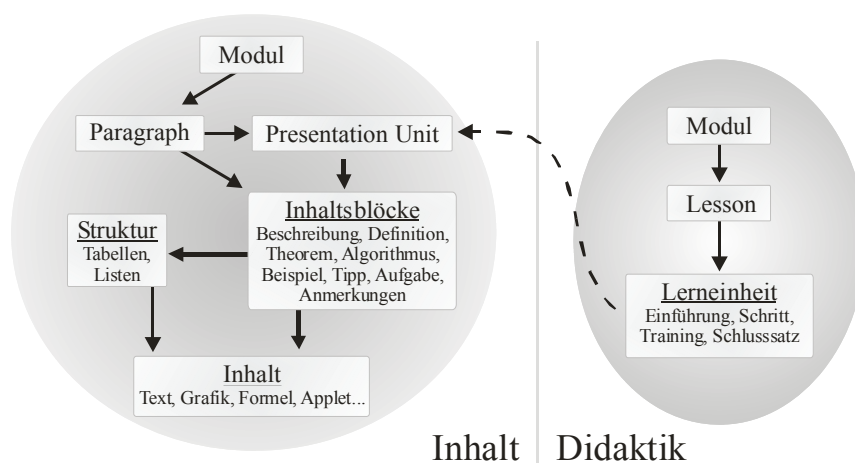


Abbildung 1: logische und didaktische Struktur von $\langle ML \rangle^3$ [kml]

¹ Multidimensional Learning Objects and Modular Lectures Markup Language

- **Markierung der Intensität**

An Lehrinrichtungen existieren Kurse oftmals in verschiedenen Intensitätsebenen. Ein Kurs „Grundlagen der Informatik“ könnte für Informatik-, Mathematik- und Mediengestaltungstheorie-Studenten angeboten werden. Die drei Kurse haben das gleiche Thema, jedoch wird unterschiedlich tief in die Materie vorgedrungen. Der Lehrinhalt kann mit $\langle ML \rangle^3$ mit den Niveaus „Expert“, „Advanced“ und „Basic“ markiert werden.

- **Unterscheidung von Material für Lehrer und Schüler**

Mit $\langle ML \rangle^3$ können Lehrer und Schüler verschiedenes Unterrichtsmaterial nutzen. Dadurch ist es möglich, spezielle Anmerkungen oder Lösungen von Aufgaben nur für Lehrer sichtbar zu machen.

- **Verschiedene Ausgabeformate**

$\langle ML \rangle^3$ -gemäßes XML kann in verschiedene Ausgabeformate konvertiert werden. Folien/Projektionen, Webseiten und Papierskripten stehen als Medien zur Verfügung.

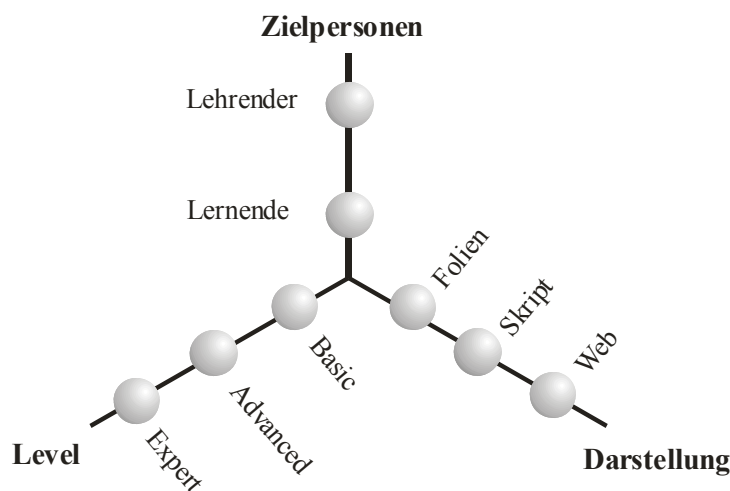


Abbildung 2: das dreidimensionale Modell von $\langle ML \rangle^3$

Durch seinen großen Umfang mit über 150 Sprachelementen ([ml3xsd]) ist die Entwicklung eines WYSIWYG-Editors für $\langle ML \rangle^3$ im Rahmen einer Diplomarbeit nicht möglich.

Stattdessen wird ein XML-Dialekt entwickelt, der ein sinnvolles Subset der $\langle ML \rangle^3$ -Funktionalität entsprechend der Aufgabenstellung implementiert und in fortführenden Arbeiten erweitert werden kann.

2.2 KML

Dieser XML-Dialekt wird mit KML (Knowledge Markup Language) bezeichnet. KML besteht aus zwei Teilen, dem Inhaltsdialekt (englisch *Content*) und dem Didaktikdialekt (englisch *Didactic*). Beide Dialekte sind durch XML Schemas¹ definiert, welche unter <http://www.kml-edu.org/kml/schema/1.0/content.xsd> bzw. <http://www.kml-edu.org/kml/schema/1.0/didactic.xsd> abgelegt sind.

2.2.1 Inhaltsdialekt

Die Lehrinhalte werden im Inhaltsdialekt abgelegt. Dafür stehen die aus der herkömmlichen Textverarbeitung bekannten Strukturen wie Absätze, Listen und Tabellen zur Verfügung. Wissensgebiete gliedern sich in Lehreinheiten, die sich weiter in Untereinheiten einteilen lassen. In ihnen können Objekte wie Grafiken und Java-Applets (Simulationen und Animationen) angebunden werden.

Auch die semantische Gliederung wird mit Hilfe des Inhaltsdialekts vorgenommen. Texte lassen sich als „Beschreibungen“, „Algorithmen“, „Beispiele“, „Definitionen“, „Anmerkungen“, „Sätze“ und „Tipps“ klassifizieren.

Für jedes Element des Inhalts existieren optionale Markierungen für die Intensität. Es wird unterschieden, welche Elemente nur für Lehrer und welche auch für Schüler sichtbar sind und welche Inhalte in den verschiedenen Ausgabeformaten erscheinen: Animationen gehören nicht ins Skript, welches wiederum ausführlichere Texte als die Präsentation für den Unterricht enthält. Damit finden sich die in Abbildung 2 (Seite 15) dargestellten drei Dimensionen von $\langle ML^3 \rangle$ hier in KML wieder.

2.2.2 Didaktikdialekt

Nachdem der Lernstoff des Themenkomplexes gespeichert wurde, können die Pädagogen ihren Unterricht zusammenstellen. Dazu müssen lediglich die gewünschten Gebiete ausgewählt und sortiert werden.

Um eine solche Ordnung vorzunehmen, erfolgt die Bindung der eindeutig bezeichneten Elemente des Inhaltes in didaktische Lehreinheiten. Die Elemente werden in Ordner sortiert, welche wiederum geschachtelt werden können.

¹ XML Schema Definition – ein XML-Dokument, das einen XML-Dialekt definiert.

2.3 Definition der Anwendungsfälle

Anwendungsfälle sind grundlegend für den Entwurf der Software. Sie beschreiben, wie und wofür der Editor genutzt wird. Beim KML-Editor sind drei Anwendungsfälle zu unterscheiden.

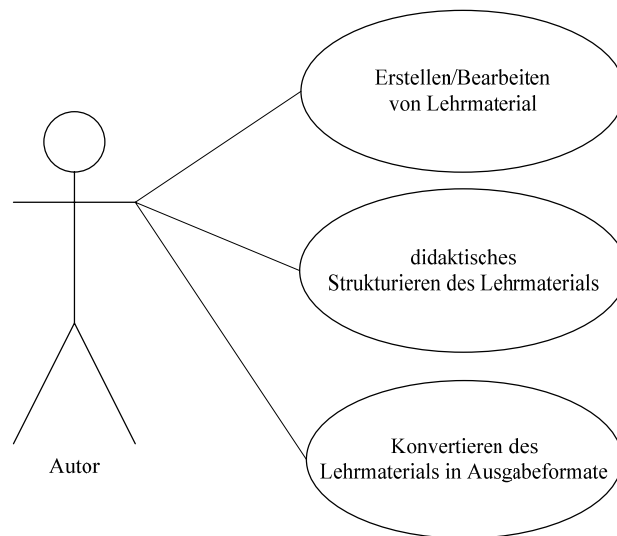


Abbildung 3: UML-Anwendungsfalldiagramm

a) Erstellen/Bearbeiten von Lehrmaterial

Der Autor gibt die Lehrtexte in den Editor ein und bindet externe Medien an. Er nutzt die Möglichkeiten der hierarchischen Dokumentstrukturierung und der semantischen Textauszeichnung.

b) Didaktisches Strukturieren des Lehrmaterials

Der Autor strukturiert vorhandenes Lehrmaterial nach didaktischen und pädagogischen Gesichtspunkten. Er ordnet Lehreinheiten und fügt Makro- und Mikromodule in eine Reihenfolge die den Lernzielen entspricht.

c) Konvertierung des Lehrmaterials in Ausgabeformate

Das Lehrmaterial liegt im XML-Datenformat vor. Um es für den Präsenzunterricht oder als Webangebot zu nutzen, muss es in Formate konvertiert werden, die diesen Anforderungen genügen. Der Autor kann dies mit dafür im Editor vorgesehen Schaltelementen durchführen.

2.4 Benutzeroberfläche

Alle bisher geforderten Eigenschaften des KML-Editors müssen in eine einfache und intuitive Benutzeroberfläche integriert werden. Das Design dieses GUI¹ spielt eine wichtige Rolle. Es hat großen Einfluss darauf, ob Lehrende das neue Werkzeug annehmen oder nicht. In der Regel sind Autoren von Lehrmaterial bereits mit vorhandenen Textverarbeitungssystemen vertraut. Der Umstieg auf den zu erstellenden Editor darf sich nicht als umständlich erweisen, andernfalls würde die Akzeptanz des neuen Werkzeugs verringert. Deshalb darf sich die Benutzeroberfläche des Editors von der einer Standardtextverarbeitung nur geringfügig unterscheiden. Sie muss möglichst einfach und verständlich entworfen werden.

Aufgrund der Trennung zwischen didaktischer Sortierung und Inhalt des Lehrmaterials besteht der KML-Editor aus zwei Fenstern. Aus dem ersten Fenster zur Bearbeitung von Lerninhalten lassen sich diese mit Hilfe des Mauszeigers in das zweite Fenster ziehen und zu didaktischen Unterrichtseinheiten gruppieren.



Abbildung 4: Fensterskizze

¹ GUI – Graphical User Interface, die Benutzeroberfläche einer Anwendung.

2.5 Spezifikation der grundlegenden Ziele des Entwurfs

Der KML-Editor ist mit folgenden Merkmalen zu entwerfen und zu implementieren:

1. WYSIWYG-Fähigkeit
2. Ausrichtung auf die Lehre
3. Speicherung der Daten im XML-Format
4. Umwandlung der Daten in webfähige, visuelle Formate
5. Kosteneffizienz
6. Einfachheit, Verfügbarkeit
7. Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

2.5.1 WYSIWYG-Fähigkeit

Um Lehrmaterial effektiv erstellen zu können, benötigt der Autor ein direktes visuelles Feedback zu seiner Eingabe. Wird beispielsweise eine Grafik angebunden, so sollte diese auch angezeigt werden. Texte, die als „wichtig“ markiert sind, müssen sich von solchen, die es nicht sind, visuell unterscheiden.

Weiterhin fließt hier die im Punkt 4. Benutzeroberfläche genannte Forderung nach Einfachheit ein.

2.5.2 Ausrichtung auf die Lehre

Der Editor muss alle in Punkt 2.2 KML genannten Features des XML Dialekts unterstützen.

2.5.3 Speicherung der Daten im XML-Format

Die mit dem Editor erstellten Dokumente sind im XML-Format abzulegen. Es muss ein XML-Schema¹ ([xsd]) zur Definition und zur Verifikation dieser Dokumente existieren.

¹ Ein XML Schema ist ein Dokument, welches Definitionen von XML-Tags enthält.

2.5.4 Umwandlung der Daten in webfähige, visuelle Formate

XML-Daten besitzen keine standardmäßige graphische Darstellung, sie lassen sich jedoch in nahezu beliebige andere Datenformate umwandeln. Besonders wünschenswert sind web-basierte Formate. Es entsteht so die Möglichkeit, Lehrmaterialien online für das Selbststudium und E-Learning zur Verfügung zu stellen.

2.5.5 Kosteneffizienz

Autoren von Lehrmaterial nutzen wahrscheinlich bereits ein Textverarbeitungssystem, das sie oder ihr Institut erworben haben. Ein Umstieg auf den zu entwerfenden Editor wird also nur in Frage kommen, wenn die damit verbundenen Kosten minimal sind. Der Nutzen des Editors muss sie um ein Vielfaches übersteigen.

2.5.6 Einfachheit, Verfügbarkeit

Die Benutzer des Editors können aus einem beliebigen Lehrumfeld stammen. Auch an Schulen soll das Programm eingesetzt werden. Der Einsatz soll nicht auf den Informatikunterricht beschränkt sein.

Es ist anzunehmen, dass die Benutzer des Editors nicht mit dem XML-Format vertraut sind oder sich damit näher befassen wollen. Die WYSIWYG-Oberfläche sollte darum die informationstechnischen Interna des Systems verbergen. Weiterhin soll das Programm für einen denkbar großen Anwenderkreis verfügbar sein, es müssen dafür möglichst viele Plattformen unterstützt werden.

2.5.7 Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

Einen WYSIWYG-Editor im vollen Umfang zu entwickeln übersteigt den Rahmen einer Diplomarbeit. Es ist anzunehmen, dass zumindest an einigen Stellen später Erweiterungen, wie das Anbinden zusätzlicher und neuer Medien, vorgenommen werden müssen.

Jede Software enthält Fehler. Diese zeigen sich oft erst im späteren Gebrauch. Zu diesem Zeitpunkt wird die Diplomarbeit bereits abgeschlossen sein. Um spätere Korrektur von Fehlern und Erweiterungen zu ermöglichen, muss das gesamte System logisch und klar strukturiert sein. Zusätzlich ist eine vollständige und exakte Dokumentation notwendig.

Dadurch kann ebenfalls ein hoher Grad an Wiederverwendbarkeit der Komponenten des Editors erreicht werden. Es ist zu prüfen, inwiefern sich der Editor auf andere XML-Dialekte portieren lässt.

Der Entwurf des Editors sollte eine Internationalisierung vorsehen, die eine Anpassung für andere Sprachgebiete ermöglicht.

2.6 Auswahl der Entwicklungswerkzeuge

Um den Editor zu entwickeln werden folgende Möglichkeiten erwogen:

1. Reengineering des bestehenden Editors X2X
2. Erweitern von Microsoft Word durch Makros
3. Programmierung in Java

Diese werden anhand der im vorigen Kapitel vorgestellten Punkte bewertet. Zusätzlich fließen folgende Punkte aus Sicht der Entwicklung des Editors in die Bewertung ein:

- (1) WYSIWYG-Fähigkeit
- (2) Ausrichtung auf die Lehre
- (3) Speicherung der Daten im XML-Format
- (4) Umwandlung der Daten in webfähige, visuelle Formate
- (5) Kosteneffizienz
- (6) Einfachheit, Verfügbarkeit
- (7) Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit
- (8) Einarbeitungsaufwand
- (9) Entwicklungsumgebung

Unter Einarbeitungsaufwand ist der Aufwand für den Entwickler (und gegebenenfalls der Nacharbeiter) zu verstehen, den ein effektiver Umgang mit der Programmierumgebung, der Programmiersprache sowie anderer notwendiger Hilfsmittel erfordert. Dabei spielen Vorkenntnisse eine große Rolle.

Die Entwicklungsumgebung bestimmt wesentlich die Geschwindigkeit des Programmierprozesses. Es lässt sich allerdings nicht vermeiden, dass bei ihrer Bewertung auch subjektive Präferenzen einfließen.

2.6.1 Reengineering des bestehenden Editors X2X

An der TU Chemnitz wurde durch Dr. Anders ein WYSIWYG-Editor für den XML-Dialekt LConML¹ entwickelt. Dieses Projekt würde zwei Ansätze zur Entwicklung eines WYSIWYG-Editors für einen eigenen XML-Dialekt bieten:

1. Erstellen eines neuen Editors durch Nachprogrammieren
2. Erstellen eines neuen Editors durch Anpassungsprogrammierung des X2X-Editors

Vorraussetzungen für die Entwicklung des X2X-Editors sind:

- libgtk-2.x.y²
- libxml-2.x.y³
- X11⁴ oder MS-Windows
- C++ Compiler

(1) WYSIWYG-Fähigkeit

Der X2X Editor ist ein vollständiger WYSIWYG-Editor.

(2) Ausrichtung auf die Lehre

Der Editor produziert LConML-Dokumente. Damit ist er für die Lehre an Hochschulen ausgelegt. Er würde also eine gute Vorlage bieten.

(3) Speicherung der Daten im XML-Format

Das Speicherziel von X2X ist XML.

(4) Umwandlung der Daten in webfähige, visuelle Formate

Für LConML ist bereits eine Umwandlung in HTML und Postscript implementiert, die jedoch leider nicht in die Oberfläche integriert ist. Eine Neuimplementierung oder Nutzung anderer vorhandener Komponenten wäre für eine Unterstützung von KML notwendig.

(5) Kosteneffizienz

X2X nutzt ausschließlich OpenSource⁵-Bibliotheken unter der GPL-Lizenz⁶ und wird mit C++ (für das z.B. mit der GCC⁷ freie Compiler zur Verfügung stehen) entwickelt. Er ist daher kostenfrei.

¹ LConML – ein proprietärer XML-Dialekt zum Erstellen von Lehrmaterial

² Bei libgtk-2.x.y – eine OpenSource-Bibliothek, für graphische Oberflächen

³ Bei libxml-2.x.y – eine OpenSource-Bibliothek für die Unterstützung von XML

⁴ X11 – ein Dienst zur Darstellung von graphischen Oberflächen

⁵ OpenSource – Softwareprojekte, für die der Quellcode allgemein zugänglich veröffentlicht wird

⁶ GPL – die meistgenutzte OpenSource-Lizenz, GNU General Public License

⁷ GCC – GNU Compiler Collection, eine kostenlose Sammlung von Compilern für Linux/Unix

(6) Einfachheit, Verfügbarkeit

X2X ist einfach zu verwenden. Es kann für Windows und Linux kompiliert werden und ist damit für die gebräuchlichsten Plattformen verfügbar. Das eventuelle Neukompilieren erfordert jedoch zusätzliche Kenntnisse beim Benutzer.

(7) Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

X2X ist gut dokumentiert. Bei Fragen wäre direkter Kontakt mit dem Entwickler Dr. Anders möglich. Der X2X-Editor nutzt zwei gut dokumentierte OpenSource-Bibliotheken. Sie haben allerdings auch prinzipielle Nachteile. Es gibt häufig neue Versionen. Der Funktionsumfang kann sich von Version zu Version ändern. Abhängigkeiten von mehreren OpenSource-Bibliotheken sind also aus softwaretechnologischer Sicht für ein dauerhaftes Projekt nicht wünschenswert.

(8) Einarbeitungsaufwand

Es liegen bereits Erfahrungen in der Programmierung mit C und C++ vor. Für die eigentliche Programmierung müsste die GCC unter Linux genutzt werden. Dort wäre der Einarbeitungsaufwand groß.

(9) Entwicklungsumgebung

Als Entwicklungsumgebung wird die GCC und Standard-Editoren unter Linux genutzt. Aus subjektiver Sicht ist dies weniger komfortabel. Die Fehlersuche in den entwickelten Programmen nimmt viel Zeit in Anspruch. Für die Dokumentation kann Doxygen¹ genutzt werden.

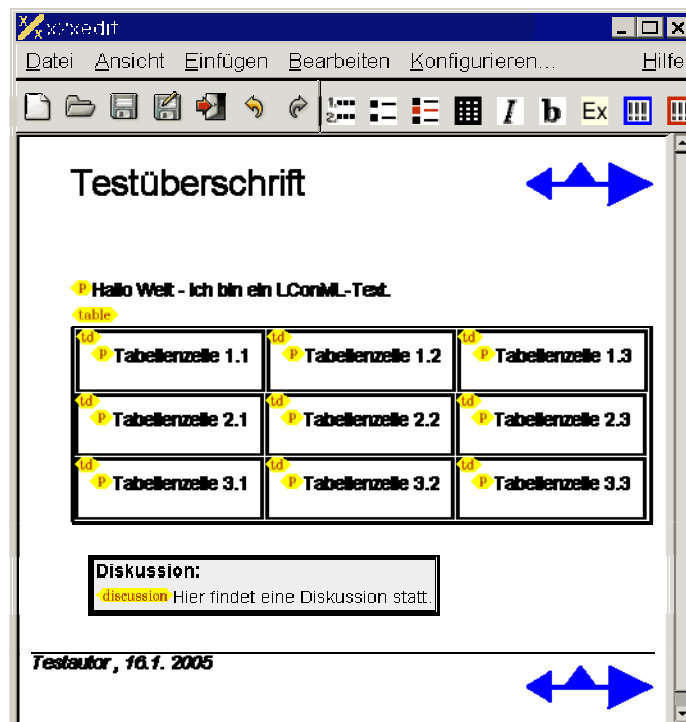


Abbildung 5: Screenshot des X2X-Editors in der Windows-Version

¹ Doxygen – ein Werkzeug zum Erzeugen von Dokumentation aus Kommentaren im Quelltext

2.6.2 Erweitern von Microsoft Word durch Makros

Microsofts Word ist ein sehr weit verbreitetes, kommerzielles Textverarbeitungssystem. Es ist durch das Einbinden von kleinen Skripten, so genannten Makros, erweiterbar.

(1) WYSIWYG-Fähigkeit

Word ist ein vollständiger WYSIWYG-Editor.

(2) Ausrichtung auf die Lehre

Word ist nicht auf die Lehre ausgerichtet. Word lässt sich auch nicht einfach um zusätzliche Fenster mit erweiterter Funktionalität erweitern.

(3) Speicherung der Daten im XML-Format

Word kann Dokumente auch im XML-Format speichern. Dieses Feature ist jedoch erst ab Word 2002 vorhanden. Dazu müsste ein Filter entwickelt werden, der den Word-eigenen XML-Dialekt in den gewünschten XML-Dialekt konvertiert.

(4) Umwandlung der Daten in webfähige, visuelle Formate

Word kann Daten auch im HTML-Format speichern, durch Nutzung spezieller Druckertreiber kann auch Postscript oder PDF erzeugt werden.

(5) Kosteneffizienz

Word ist ein kommerzielles Produkt, und daher nicht kostenlos verfügbar.

(6) Einfachheit, Verfügbarkeit

Word ist sehr einfach zu bedienen. Es ist eines der meistgenutzten Textverarbeitungssysteme, weshalb anzunehmen ist, dass eine Vielzahl der Autoren von Lehrmaterial damit vertraut ist. Word ist nur für Windows und MacOS, nicht aber für Linux/Unix verfügbar.

(7) Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

Word zu erweitern erfordert das Nutzen von Visual Basic for Applications, einer interpretierten Programmiersprache. Sie ist direkt in das Textverarbeitungssystem integriert. Aus softwaretechnologischer Sicht ist eine solche Bindung nicht wünschenswert. Es ist nicht klar, inwiefern sich die Programmiersprache bei der Weiterentwicklung von Word ändern wird. Weiterhin ist anzunehmen, dass im Abstand von ca. zwei Jahren neue Versionen von Word erscheinen, wobei sich der Funktionsumfang ändert.

(8) Einarbeitungsaufwand

Es liegen bereits Erfahrungen in der Programmierung mit Visual Basic und Visual Basic for Applications vor, die jedoch für die Anforderungen des Projekts nicht ausreichend sind.

(9) Entwicklungsumgebung

Als Entwicklungsumgebung steht ein in Word integrierter Editor zur Verfügung. Dieser verfügt über begrenzte Debugging-Funktionalität.

Für das automatische Erstellen einer Dokumentation gibt es jedoch kein Werkzeug.

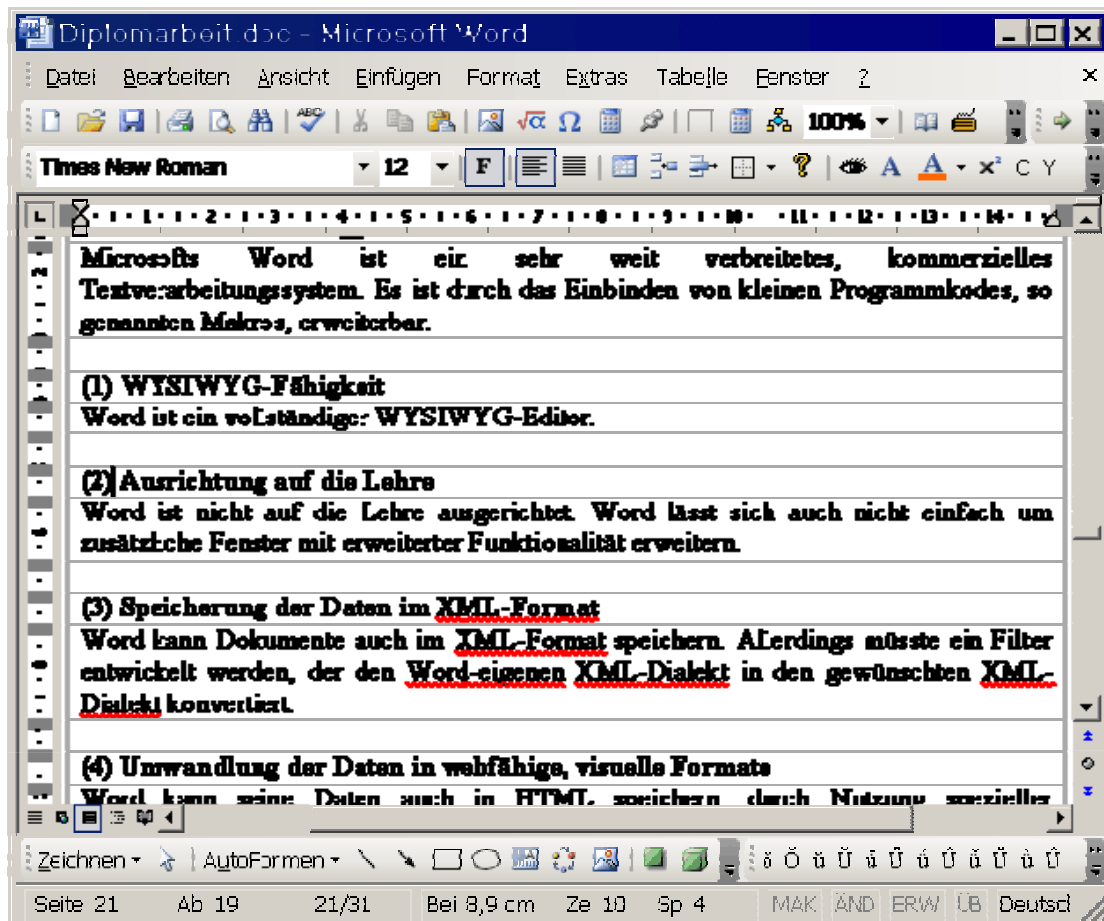


Abbildung 6: Screenshot von Microsoft Word

2.6.3 Programmierung in Java

Java ist eine weit verbreitete, moderne Programmiersprache. Sie ist speziell dafür ausgelegt, plattformunabhängige Applikationen zu entwickeln.

(1) WYSIWYG-Fähigkeit

Java verfügt über Klassen, die Design Patterns implementieren, die das Entwickeln von WYSIWYG-Editoren zulassen. Prinzipiell muss aber bei der Programmierung auf relativ niedriger Ebene angesetzt werden.

(2) Ausrichtung auf die Lehre

Der Editor müsste von Grund auf entwickelt werden. Dadurch kann dieser jedoch so entwickelt werden, dass er später auch für andere XML-Dialekte leicht anpassbar ist.

(3) Speicherung der Daten im XML-Format

Java verfügt über vordefinierte Klassen, die XML einlesen und XML ausgeben können.

(4) Umwandlung der Daten in webfähige, visuelle Formate

Dafür existieren keine vordefinierten Klassen, XML lässt sich aber mit Hilfe von XML Style Sheets in solche Formate umwandeln.

(5) Kosteneffizienz

Java ist kostenfrei.

(6) Einfachheit, Verfügbarkeit

Die Einfachheit des Editors liegt komplett in der Hand des Entwicklers. Java-Interpreter sind für fast jede Plattform verfügbar. Java steht jedoch in dem Ruf langsam zu sein, was die Nutzung des Editors mit langsamen Systemen erschweren könnte.

(7) Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit

Die Erweiterbarkeit liegt in der Hand des Entwicklers. Für die Dokumentation kann JavaDoc¹ genutzt werden.

(8) Einarbeitungsaufwand

Es ist kein Einarbeitungsaufwand notwendig.

(9) Entwicklungsumgebung

Es wird die kostenfreie Entwicklungsumgebung Eclipse genutzt. Diese verfügt über umfangreiche Werkzeuge und einen guten Debugger.

¹ JavaDoc – Ein Dokumentationswerkzeug ähnlich Doxygen.

2.6.4 Entscheidung für ein Entwicklungswerkzeug

In Tabelle 1 sind die grundlegenden Eigenschaften der einzelnen Möglichkeiten zur Lösung der Problemstellung gegenübergestellt. Ein Punkt in der Spalte eines Bewertungskriteriums steht für das Erfüllen des Kriteriums durch das Entwicklungswerkzeug.

- (1) WYSIWYG-Fähigkeit
- (2) Ausrichtung auf die Lehre
- (3) Speicherung der Daten im XML-Format
- (4) Umwandlung der Daten in webfähige, visuelle Formate
- (5) Kosteneffizienz
- (6) Einfachheit, Verfügbarkeit
- (7) Erweiterbarkeit, Wartbarkeit und Wiederverwendbarkeit
- (8) Einarbeitungsaufwand
- (9) Entwicklungsumgebung

| | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 4.1 X2X Editor | • | • | • | • | • | | • | | |
| 4.2 Microsoft Word | • | | • | | | | | | |
| 4.3 Java | | • | • | • | • | • | • | • | • |

Tabelle 1: Entwicklungswerkzeuge

Letztendlich wurde Java mit Eclipse als Entwicklungsumgebung gewählt. Das Vorwissen im Umgang mit dieser Programmiersprache gab den Ausschlag.

3 Entwurf

Bei dem Design des KML-Editors wird besonders großer Wert auf Wiederverwendbarkeit und Erweiterbarkeit gelegt, was die Anwendung einiger grundsätzlicher Paradigmen gewährleistet. Hochgradige Modularität wird erreicht durch Zerlegung und Abstraktion.

3.1 Generelle Festlegungen

Die Festlegungen für die Programmierung sollen helfen ein gutes und verständliches Design der Applikation zu sichern. Zusätzlich fördern sie eine gute Wartbarkeit und Erweiterbarkeit.

- **Vollständige Kapselung**

Java ist eine objektorientierte Sprache, d.h. die Funktionalitäten und Daten sind in Objekten gruppiert. Die Art, wie auf die Daten eines Objekts zugegriffen werden kann, ist ein wesentlicher Faktor für die Wartbarkeit eines Projekts. Sind die Daten von außerhalb eines Objekts direkt erreichbar, so sind deren Änderungen schwerer nachzuvollziehen. Bei dem Projekt wird daher das Prinzip der vollständigen Kapselung angewandt. Alle Membervariablen sind als `privat` zu deklarieren. Es darf ausschließlich durch Methoden ihrer Klasse auf sie zugegriffen werden. Es ist nur erlaubt Membervariablen als `package private` zu deklarieren, wenn von inneren Klassen auf diese zugegriffen wird.

- **Niedrigste Sichtbarkeit**

Klassen und Methoden werden stets so deklariert, dass sie eine möglichst niedrige Sichtbarkeit besitzen. In Java existieren vier Sichtbarkeitsstufen. Durch eine Deklaration als `privat` sind Methoden und Unterklassen nur für ihre Elternklasse zugänglich. Die Stufe `package private` erlaubt es auch anderen Klassen desselben Pakets eine Methode oder Klasse zu nutzen. Methoden des Typs `protected` sind zusätzlich für alle abgeleiteten Klassen sichtbar. Uneingeschränkter Zugriff ist auf als `public` deklarierte Methoden möglich.

- **Geringste Änderbarkeit**

Alle Methoden die nicht überschrieben werden müssen, alle Membervariablen deren Wert sich nicht ändern muss und alle Klassen, die nicht spezialisiert werden müssen, werden als `final` deklariert.

Der Java-Compiler kann dann statische Referenzen anstelle von dynamischen Referenzen generieren, was die Performance der Applikation erhöht.

- **Bezeichner**

Festlegungen für die Schreibweise von Bezeichnern erhöhen die Wartbarkeit und Lesbarkeit des Quelltextes.

1. **Klassen- und Schnittstellenbezeichner**

Alle Bezeichner von Klassen beginnen groß. Setzen sie sich aus mehreren Wörtern zusammen, so beginnen auch diese groß. Bezeichner von Schnittstellen werden wie Bezeichner von Klassen gebildet, allerdings beginnen ihre Namen mit dem Präfix `I`.

Beispiel:

```
public class      DefaultCompositionDriver
    public interface ICompositionDriver.
```

2. **Konstantenbezeichner**

In Java gibt es kein Konstrukt zum Erstellen von Konstanten. Man kann jedoch statische Membervariablen als `final` deklarieren, wodurch sich ihr Wert nicht mehr ändern lässt. Solche Variablen werden fortan als Konstanten bezeichnet.

Alle Bezeichner von Konstanten werden komplett groß geschrieben. Setzen sie sich aus mehreren Wörtern zusammen, so werden diese durch Unterstriche getrennt.

Konstanten sind die einzigen Datenfelder, die als `public` deklariert werden dürfen.

Beispiel:

```
public static final int ONE_PLUS_TWO = 3;.
```

3. **Variablenbezeichner**

Alle Bezeichner von Variablen werden stets klein geschrieben. Bestehen sie aus mehreren Wörtern, so werden diese durch Unterstrich getrennt und klein geschrieben.

Membervariablen beginnen mit dem Präfix `m_`.

Statische Variablen beginnen mit dem Präfix `s_`.

Lokale Variablen beginnen mit dem Präfix `l_`.

Parameter einer Methode beginnen mit dem Präfix `p_`.

Beispiel:

```
private final int m_my_int ;.
```

4. **Methodenbezeichner**

Alle Bezeichner von Methoden werden stets klein geschrieben. Bestehen sie aus mehreren Wörtern, so werden diese durch Unterstrich getrennt und klein geschrieben.

Beispiel:

```
int get_int();.
```

Der gesamte Entwurf des Editors basiert auf *Design Patterns* [dp], um eine hohe Modularität und Wiederverwendbarkeit zu gewährleisten.

3.2 Schichtenarchitektur

Der KML-Editor wird gemäß dem Hierarchischen Schichtenmodell [ka] entworfen.

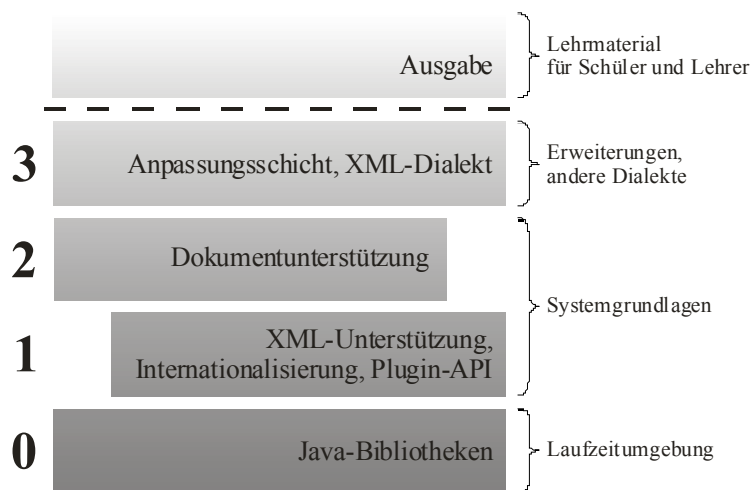


Abbildung 7: Schichtenarchitektur des KML-Editors

Schicht 0

Die Java-Laufzeitumgebung ist die Basis des Editors.

Schicht 1

Die XML-Unterstützung implementiert die interne Darstellung von XML-Tags und deren Attribute als Bausteine für allgemeine Schablonen für XML-Dialekte.

Die Internationalisierung definiert wie der Editor für die Benutzung in anderen Sprachen lokalisiert werden kann.

Die Plugin-API erlaubt ein beliebiges Erweitern des Editors und das Austauschen nahezu aller seiner Komponenten.

Schicht 2

Die Dokumentunterstützung stellt die Abstraktion für das Verhalten des Editors bereit. Hier befinden sich der innere Editor-Kern, der die Verarbeitung von Benutzereingaben regelt, sowie die Abstraktionen der übrigen Dienste des Editors.

Schicht 3

In dieser Schicht werden XML-Dialekte wie $\langle ML \rangle^3$ oder KML durch Instantiierung und Ausfüllen der Schablonen aus Schicht 1 definiert.

Ausgabe

Der Editor produziert die Ausgabe der Lehrmaterialien für Schüler und Lehrer im jeweiligen Format, z.B. HTML.

Die folgenden Kapitel beschreiben die wichtigsten Einheiten der einzelnen Schichten.

4 Schicht 0: Java-Bibliotheken

4.1 Modell der graphischen Oberfläche

Der Editor verfügt über eine graphische Oberfläche. Java besitzt zwei Modelle zur Darstellung solcher Oberflächen: AWT und Swing. [javb]

Abstract Windowing Toolkit (AWT)

Das AWT ist das ältere der beiden Modelle. Mit dem AWT lassen sich Dialogelemente erstellen und graphische Operationen durchführen. Programme basieren auf einem Nachrichtensystem und können auf Maus- und Tastaturereignisse reagieren.

Alle AWT-Dialogelemente basieren direkt auf graphischen Ressourcen des Betriebssystems, deshalb werden sie schwergewichtig (oder englisch „*heavy weight*“) genannt.

Swing

Da jeder AWT-Komponente einer Ressource des Betriebssystems zugeordnet ist, ist die Erscheinungsform von AWT-Programmen und auch das Verhalten ihrer Benutzeroberflächen stark von der verwendeten Plattform abhängig. Mit Swing wird dieses Problem umgangen, indem nur noch die „oberste“ Komponente eines Fensters eine „*heavy weight*“ Komponente darstellt. Die darin beinhalteten Komponenten sind „*light weight*“ (leichtgewichtig). Ihr komplettes Verhalten und ihr Zeichenkode sind in Java implementiert. Dadurch entfallen plattformspezifische Besonderheiten und Unterschiede in der Bedienung. Auch verfügt Swing über eine größere Menge von Dialogelemente mit neuen Funktionalitäten.

Eines der herausragenden Eigenschaften von Swing ist die Umsetzung des *Model-View-Controller* Prinzips. Oftmals wird bei Swing jedoch eine vereinfachte Variante, das *Model-Delegate-Prinzip*, genutzt. Dabei werden View und Controller im so genannten *Delegate* zusammengefasst.

Der Editor wird auf Basis von Swing entwickelt.

Die Java Swing Bibliothek ist `javax.swing.*`.

4.2 Laden, Speichern und Transformieren von XML-Daten

Der KML-Editor nutzt einige der verschiedenen Möglichkeiten zum Laden und Speichern von Daten im XML-Format die Java bietet [jxml].

Laden von XML-Dokumenten

Zum Laden von XML-Dokumenten wird JAXP-SAX¹ genutzt. Diese Programmierschnittstelle basiert auf einem Call-Back-Verfahren. XML-Dokumente werden nach dem aus Unix bekannten Pipes-and-Filters Design-Pattern eingelesen. SAX-Parser sind schnell und benötigen wenig Speicher.

Die genutzte Bibliothek ist `javax.xml.parsers.*`.

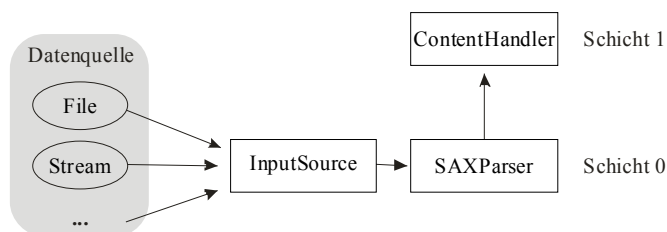


Abbildung 8: JAXP-SAX-Parser im KML-Editor

Speichern von XML-Dokumenten

Für die Speicherung eines XML-Dokuments wird sein kompletter DOM²-Baum erzeugt. DOM-Bäume sind arbeitsspeicherintensiv, die Java-Bibliotheksfunktionen unterstützen andere Möglichkeiten zur Speicherung von XML-Daten jedoch nicht.

Die genutzte Bibliothek ist `org.w3c.dom.*`.

Transformation von XML-Dokumenten

Mit JAXP werden XML-Dokumente via XSLT³ [xslt] transformiert, was z.B. bei der Konvertierung von KML-Dokumenten in eines der Ausgabeformate notwendig ist.

Die genutzte Bibliothek ist `javax.xml.transform.*`.

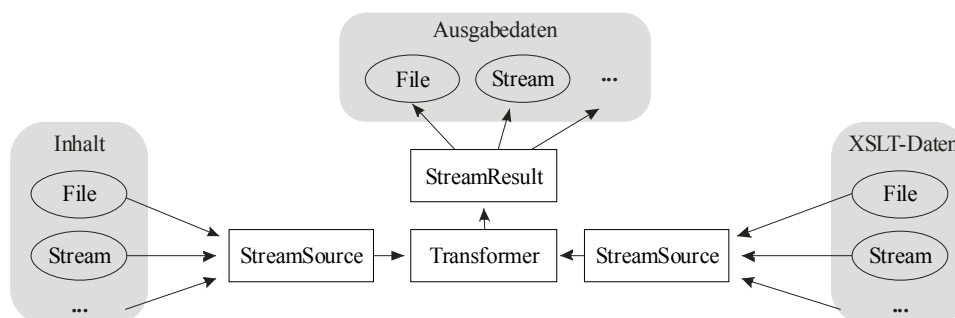


Abbildung 9: JAXP-XSLT-Transformation im KML-Editor

¹ Java APIs for XML Parsing – Standardisierte Programmierschnittstelle für Java-SAX-Parser. Simple API for XML – Eine API zum Einlesen von XML-Daten.

² Document Object Model – API für XML-Dokumente, bei der diese als Bäume abstrahiert werden.

³ Extensible Stylesheet Language (Transformations) – Sprache für XML-Transformationen.

5 Schicht 1: XML-Unterstützung

XML-Dialekte können durch XML Schemas definiert werden. Diese Methode eignet sich zur statischen Prüfung der Struktur eines Dokuments.

Ein WYSIWYG-XML-Editor stellt jedoch dynamische Anforderungen an eine Dialektdefinition:

1. Wahrung der Hierarchie

Ein XML-Dokument besteht aus Objekten verschiedener Art. Das XML-Schema definiert eine exakte Hierarchie, in der diese Objekte geschachtelt werden können. So müssen bei KML *Tabellen* stets *Zeilen* und diese wiederum *Zellen* enthalten. Beim Einfügen eines neuen Objekts in das Dokument muss diese Hierarchie gewahrt werden. Mit einem XSD-Schema kann man jedoch nur ein gesamtes XML-Dokument prüfen. Bei der Bearbeitung eines Dokuments müsste dieses also ständig gespeichert und geprüft werden. Effizienter wäre eine iterative Variante. Beim Einfügen eines neuen Objekts in ein Dokument muss prinzipiell nur geprüft werden, ob es in dem Objekt, in das es eingefügt werden soll, enthalten sein darf. Durch Induktion lässt sich beweisen, dass jede Folge solcher korrekter Operationen ein gültiges Dokument stets wieder in ein gültiges Dokument überführt.

2. Attributinformation

KML-Objekte besitzen Eigenschaften wie „Titel“ oder „ID“. Diese werden letztendlich in XML-Attribute reflektiert. Jeder XML-Tag besitzt eine wohldefinierte Menge von Attributen. Beim Erstellen eines neuen Objekts muss dieses mit der entsprechenden Menge ausgestattet werden. Aus einem XML-Schema lassen sich die benötigten Informationen zwar extrahieren, allerdings ist dieser Prozess komplex und daher langsam.

3. Fehlerresistenz

Die XML-Unterstützung vertritt bei der Erstellung von Dokumenten den Ansatz, dass alle mit dem Editor erstellten Dokumente gültig sein müssen. Wird jedoch ein Dokument mit beschädigten Daten geöffnet, so soll es nach einer best-effort Methode dargestellt und bearbeitet werden können. Eine spezielle Hervorhebung weist auf die fehlerhaften Stellen hin. Dadurch können zumindest Teile des Inhalts einer beschädigten Datei gerettet werden. Solche Funktionalität ist in XML-Schemas nicht vorgesehen.

4. Vielseitigkeit

Der Mechanismus muss sowohl für die WYSIWYG-Bearbeitung im Inhaltseditor als auch für die Zusammenstellung von Inhalt in einer Baumstruktur geeignet sein.

Die nächsten beiden Unterkapitel erklären, wie diese dynamischen Anforderungen im XML-Editor befriedigt werden.

5.1 XMLTag und XMLTagSet

Jedem Tag eines XML-Dialekts wird eine Instanz der Klasse `XMLTag` zugeordnet. Sie beinhaltet alle Informationen über den Tag und dient als Ansatzpunkt für die Darstellung sowie als Grundlage für eine Validierung von Dokumenten.

Jedes `XMLTag`-Objekt besitzt eine Liste von anderen `XMLTag`-Objekten, in denen es enthalten sein darf. Somit wird ein gerichteter Graph gebildet, der einen XML-Dialekt definiert. Der Graph besitzt mehrere Quellen. Rekursive Beziehungen sind zulässig. Der Knoten, welcher keine Quelle ist, entspricht dem Wurzeltag des XML-Dialekts. Diese Metainformation definiert, wie neue Elemente in ein Dokument eingefügt werden können. Die Dokumentunterstützung kann automatisch fehlende Tags ergänzen oder Dokumentstrukturen umordnen.

Jedem XML-Dialekt ist ein eindeutiger Namespace-URI¹ zugeordnet. Dadurch wird verhindert, dass Namenskonflikte zwischen Tags eines XML-Dialekts und gleichnamigen Tags eines eingebetteten anderen Dialekts entstehen. Es ist zum Beispiel möglich, MathML²-Tags in `<ML>`³-Dokumente einzufügen [ml3xsd].

Ein Tag wird eineindeutig durch ein Tupel aus dem Namespace-URI seines Dialekts und seinem Bezeichner identifiziert.

Der Parser muss beim SAX-Parsen gefundenen Tags entsprechende `XMLTag`-Objekte zuordnen. Um dies effizient zu gestalten, werden die `XMLTag`-Objekte in Hashes abgelegt. Genutzt wird ein zweistufiger Hash. In der ersten Stufe befinden sich die Tags des XML-Dialekts identifiziert durch ihren Bezeichner sowie die Hashes der zweiten Stufe mit den Tags der eingebetteten Dialekte, identifiziert durch ihren Namespace-URI. Somit wird der Zugriff auf die häufiger auftretenden Tags des Grunddialekts beschleunigt.

Die komplette physische Dokumentstruktur eines Dialekts wird durch einen solchen zweistufigen Hash, in der Klasse `XMLTagSet` gekapselt, beschrieben.

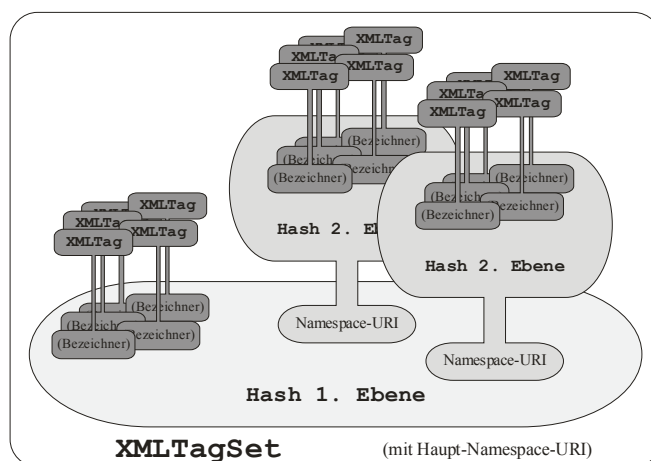


Abbildung 10: zweistufiger Hash des XMLTagSet

¹ Universal Resource Identifier – ein eindeutiger Identifikator für Ressourcen.

² Mathematical Markup Language – XML-Dialekt für mathematische Inhalte [mml].

5.2 XMLAttribute und XMLAttributeSet

XMLTag-Objekte stellen Schablonen für die Tags eines XML-Dialekts dar. Anstelle für jedes Element im Editor explizit den Identifikator des zugehörigen Tags (bestehend aus Namespace-URI und Bezeichner) zu speichern, wird lediglich eine Referenz auf die passende Instanz von XMLTag genutzt. Dadurch erhält man auch die Möglichkeit, das *Factory-Design* Pattern für XML-Attribute zu implementieren.

Während ein XMLTag-Objekt für alle Elemente eines Typs ausreicht, besitzt jedes Element eine andere Menge von Attributen. So werden zum Beispiel Tabellen in KML als Tags vom Typ „Tabelle“ gespeichert. Ihr Typ ist also gleich. Sie können jedoch unterschiedliche Titel besitzen, was sich in unterschiedlichen Attributen niederschlägt.

Jedes XMLTag-Objekt kann mit der Methode `create_xml_attributes` eine Menge von Attributen erstellen, die für seinen Tag gültig ist. Die Funktion zum Bearbeiten der Eigenschaften im WYSIWYG-Editor arbeitet mit dieser Methode, ebenso der XML-Parser in dieser Schicht.

Damit die Attribute eines Objekts im Editor leicht geändert werden können, implementiert XMLAttribute das IProperty-Interface. Mit Hilfe dessen Methoden `create_editor`, `store_to_editor` und `load_from_editor` werden Steuerelemente für die Bearbeitung der Werte des Attributs erzeugt. Die Standardimplementierung nutzt eine normale Eingabezeile für den Wert des Attributs. Um beliebige Steuerelemente einzusetzen überschreibt man diese Methoden. Es steht eine Palette mit solchen spezialisierten Attributen, z.B. mit Checkboxen für booleschen Werte oder Mengen, zur Verfügung. Ein Dialog in einer höheren Schicht kann dann die Steuerelemente beliebig anordnen.

Für XML-Attribute gelten dieselben Voraussetzungen für die Identifikation wie für XML-Tags. Daher werden auch sie in einem zweischichtigen Hash abgelegt, dem XMLAttributeSet. Gerade bei Attributen ist der schnelle Zugriff auf die Werte wichtig, da sie Einfluss auf die visuelle Darstellung im Editor haben.

Die package privaten Klassen XMLItem und XMLItemMap implementieren dieses gemeinsame Verhalten und dienen als Basisklassen.

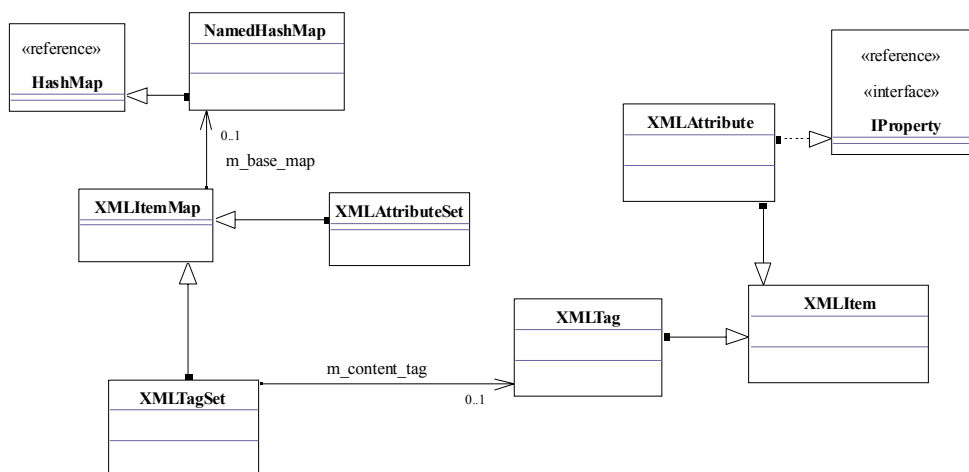


Abbildung 11: Klassendiagramm der XMLTags

5.3 Internationalisierung

Um den Einsatz des Editors nicht auf den deutschsprachigen Raum zu beschränken, wurde eine Unterstützung für dessen Lokalisierung implementiert.

Genau genommen setzt diese sogar unterhalb der eigentlichen XML-Unterstützung an, denn sie wird bereits von `XMLTag` und `XMLAttribute` genutzt, um deren Bezeichner sprachspezifisch darzustellen.

Java unterstützt die Internationalisierung durch die Klasse `Locale`, welche eine Sprache identifiziert und die Klasse `PropertyResourceBundle`, mit der auf sprachspezifische Ressourcen zugegriffen werden kann. Ziel war es, aufbauend auf diesen Hilfsmitteln, eine Sprachunterstützung zu implementieren die darüber hinaus auch Umschalten der Sprache zu jedem beliebigen Zeitpunkt ermöglicht. Dies wird durch die neue Klasse `I18N` erreicht. Instanzen von `I18N` referenzieren die Instanz von `Locale` ihrer aktuellen Sprache, sowie ein `PropertyResourceBundle` mit den entsprechenden Ressourcen. Dazu besitzt sie eine Liste mit Objekten, die bei einer Sprachänderung informiert werden müssen. Wird die Sprache einer `I18N`-Instanz geändert, so propagiert sie dies an alle solchen Objekte. Gemäß dem Design-Pattern *Singleton* existiert eine Instanz von `I18N`, bei der sich automatisch alle weiteren Instanzen anmelden. Damit genügt es, mit der statischen Methode `set_base_locale` die Sprache dieser unsichtbaren Instanz zu ändern, und alle anderen `I18N`-Objekte (auch in den Plugins) passen sich automatisch an.

Objekte, die bei einer Sprachänderung informiert werden müssen, implementieren das Interface `I18NListener`, das die Methode `locale_changed` besitzt. `I18N`-Instanzen untersuchen alle angemeldeten Objekte bei einer Sprachänderung dazu noch in die Tiefe. Implementiert eine Instanz von `Container` (der grundlegenden Klasse aller Java GUI-Bibliotheken, die Steuerelemente beliebig geschachtelt enthalten kann) das `I18NListener`-Interface, so wird der gesamte Baum ihrer Steuerelemente auf weitere `I18NListener`-Implementoren untersucht und diese automatisch informiert.

Der Editor kann für neue Sprachen lokalisiert werden, indem die Ressourcen für diese Sprache übersetzt werden. Dem Sprachmenü wird mit der Methode `register_language` ein weiterer Punkt hinzugefügt. Sie ordnet einer `Locale` eine Sprachbezeichnung und ein Icon für die Flagge der entsprechenden Nation zu.

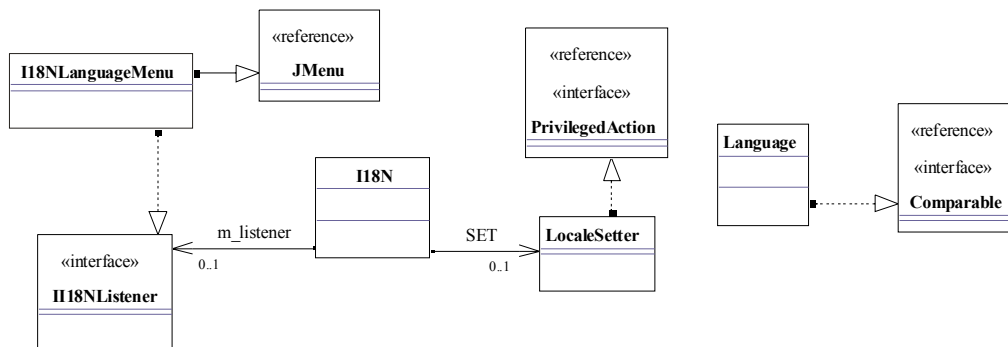


Abbildung 12: Klassendiagramm der Internationalisierungs-Klassen

5.4 Plugin-API

Um ein Maximum an Erweiterbarkeit zu erreichen, wird eine Plugin-API definiert. Die Editor-Applikation nimmt eine Doppelrolle ein. Zum einen definiert sie Abstraktionen von Diensten, die durch Plugins manifestiert werden. Zum anderen ist sie Klient der diese Dienstmanifestationen der Plugins nutzt.

Die Dienstabstraktionen befinden sich in Schicht 2 des Editors, es handelt sich zum Beispiel um die „Dialektdefinition“ zur Darstellung von XML-Dialekten oder die „Transformationsunterstützung“ zur Umwandlung von Dokumenten in andere Formate. Jede solche Abstraktion besitzt einen hierarchisch geordneten, systemweit eindeutigen Identifikator, in Form einer Instanz der Klasse `Context`.

Klienten für Dienste definieren die Schnittstelle `IPluginListener` und melden sich bei einer zentralen Instanz, der *Singleton* Klasse `PluginLoader` an. Diese Instanz informiert sie dann, wenn ein Plugin geladen wird.

Die allgemeine Festlegung für Plugins ist, dass sie ein Paket `plugin` enthalten, in dem sich eine Klasse mit dem Namen `Plugin` befindet. Diese Klasse wird vom `PluginLoader` automatisch instantiiert. Sie muss die Schnittstelle `IPlugin` implementieren. Durch deren Methoden können die Klienten anfragen, ob ein von ihnen benötigter Dienst durch das Plugin manifestiert wird.

Plugins können Klienten für Dienste sein und sie werden von Änderungen der Sprache der Benutzeroberfläche automatisch unterrichtet.

Somit wird es möglich, die gesamte Definition und alle Eigenschaften des Dialekts KML aus dem Editorkern in ein Plugin zu verlagern. Der Editor ist an keinen Dialekt und an keinen fest programmierten Dienst mehr gebunden, sondern kann für beliebige Zwecke mit minimalem Aufwand angepasst werden.

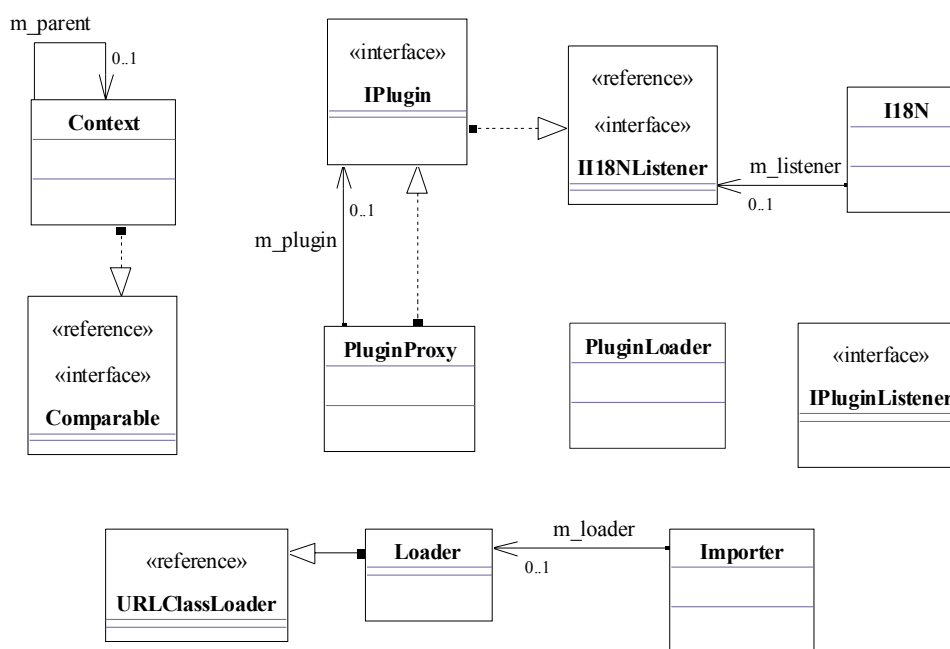


Abbildung 13: Klassendiagramm der Plugin-API

6 Schicht 2: Dokumentunterstützung

Aufgabe der Schicht Dokumentunterstützung ist es, Abstraktionen für alle Dienste eines XML-Editors zur Verfügung zu stellen. In der darüber liegenden Schicht 3 sind letztendlich nur noch Spezialisierungen der hier definierten Klassen vorzunehmen, um einen beliebigen XML-Dialekt zu implementieren.

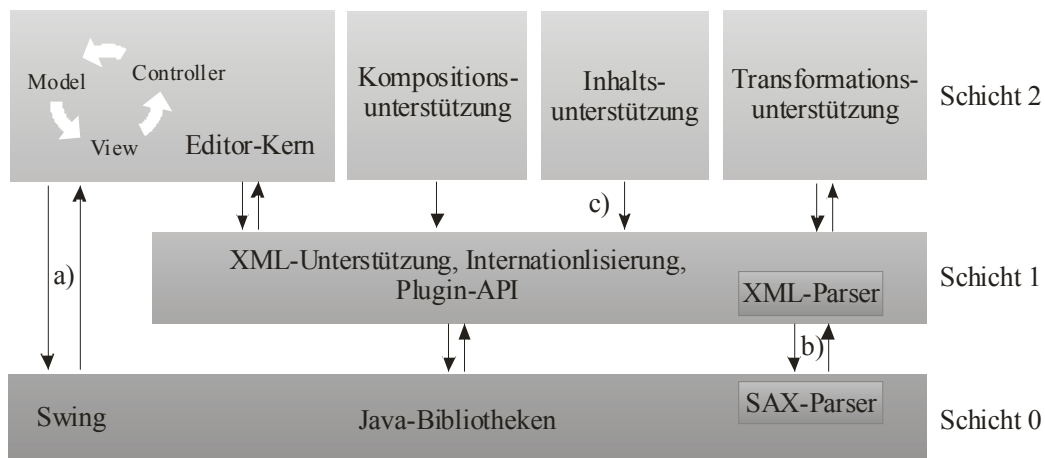


Abbildung 14: Struktur der Schicht 2 des Editors

An dieser Stelle wird beispielhaft auf einige Beziehungen zwischen den Schichten des Editors anhand des Hierarchischen Schichtenmodells eingegangen.

- Aus Swing-Klassen wird durch Betriebsmitteltransformation der Editor-Kern erzeugt. Der Swing-Event-Thread steuert das Verhalten des Editor-Kerns.
- Der SAX-Parser steuert den Parser der XML-Unterstützung. Dieser ist wiederum eine Aggregation aus Betriebsmitteln wie SAX-Parser, Listen und Puffern.
- Die Dialektunterstützungen nutzen die Internationalisierung und die XML-Unterstützung.

6.1 Editor-Kern

Der Kern des Editors ist gemäß dem Model-View-Controller Design-Pattern aufgebaut. Er setzt sich zusammen aus dem *Model*, welches die Daten enthält, dem *View*, welcher für die Darstellung verantwortlich ist und dem *Controller*, welcher Tastatur- und Mausereignisse verarbeitet und das Modell entsprechend beeinflusst.

Einer der Vorteile dieser Architektur ist es, dass jeder seiner Bestandteile prinzipiell austauschbar ist. Sinnvoll ist dies allerdings nur für den *View*, die visuelle Repräsentation.

Swing stellt einige Klassen zur Verfügung die bei der Entwicklung von WYSIWYG-Dokumenteditoren unterstützen. Diese befinden sich im Paket `javax.swing.text.*`.

6.1.1 Model

Mit den Schnittstellen `Dokument` und `Element` werden in Swing Datenmodelle für Dokumente erstellt. Die Klasse `XMLDocument` ist eine Spezialisierung der, das `Document-Interface` implementierenden, Swing-Klasse `AbstractDocument`. Sie repräsentiert ein Dokument es Editors. Die standardmäßig bereitgestellten Implementoren des `Element-Interfaces` stellten sich als zu unflexibel heraus und wurden durch die Klassen `XMLElement`, `XMLLeafElement` und `XMLBranchElement` ersetzt.

Die folgende Grafik zeigt die logische Struktur eines KML-Textblocks und deren Darstellung im Kern des XML-Editors.

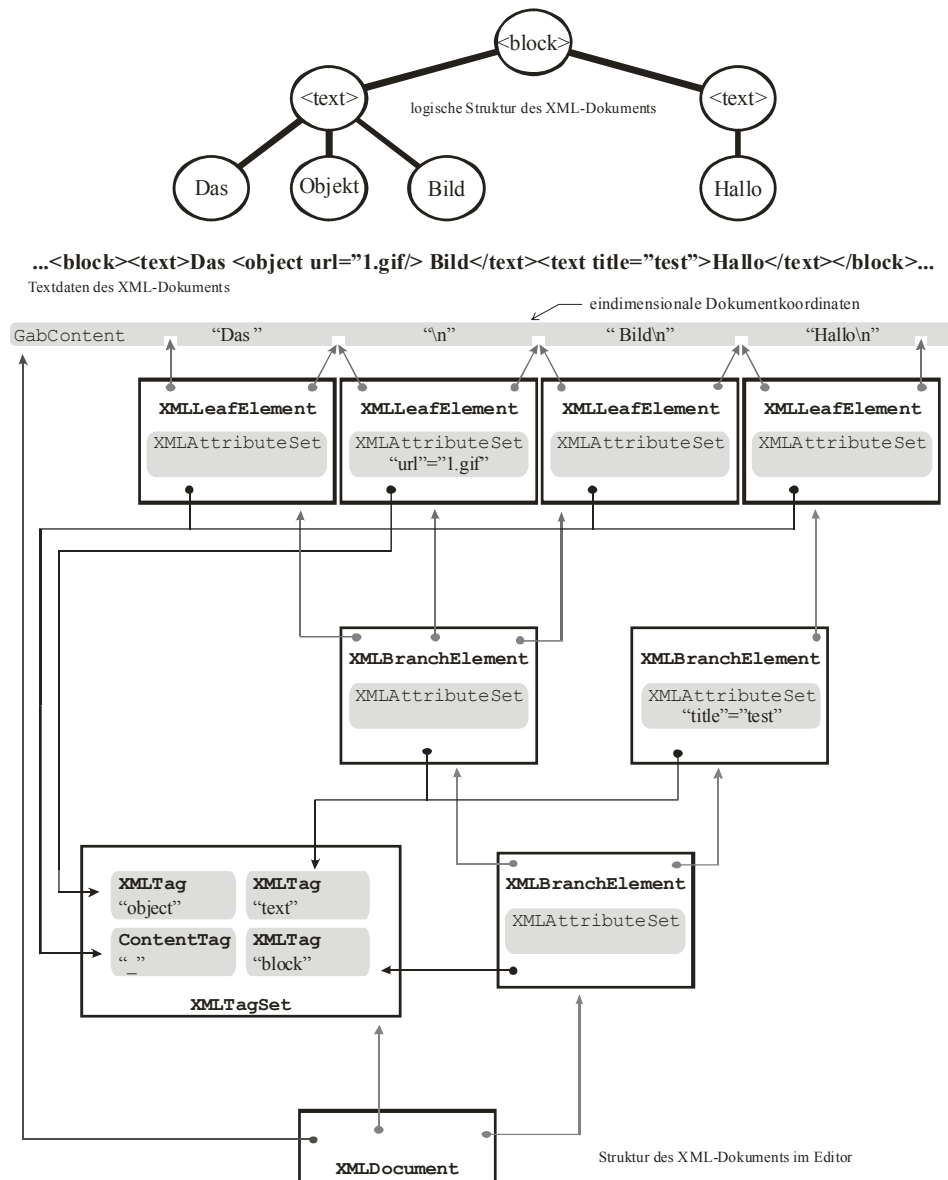


Abbildung 15: Model-Teil des Editor-Kerns

6.1.2 View

Die *View*-Komponente des Kerns stellt die im *Model* verwalteten Daten graphisch dar. Sie erlaubt das Umrechnen von eindimensionalen Dokument-Koordinaten (Position im fließenden Text) in Ansichts-Koordinaten (zweidimensionale Koordinaten des Bildschirms).

Damit stellt erlaubt sie dem Controller die Interaktion mit dem Benutzer. Das *View* ist passiv, es wird durch Ereignisse, die das *Model* erzeugt, über dessen Änderungen informiert und passt sich diesen an.

Jedem Objekt im *Model* ist mindestens ein Objekt im *View* zugeordnet. Die Views leiten sich von der Basisklasse `XMLView` ab und werden durch so genannte ViewFactories gemäß dem *Factory-Design-Pattern* erzeugt. Objekten des *Modells* werden stets Instanzen der Klasse `XMLTag` zugeordnet die diese in verschiedene Typen einteilen. Jede Instanz von `XMLTag` ist wiederum Schlüssel in einem von der Klasse `XMLViewFactory` verwalteten Hash. Der zum Schlüssel gehörende Datenteil ist eine Instanz von `ViewFactory`. Mit Hilfe dieser Übersetzung können sich die Objekte des *Views* an Änderungen im Modell anpassen, die das Erzeugen von neuen Subviews erfordern.

Erwähnt sei weiterhin, dass Objekten des *Models* die Absätze darstellen, mehrere Views zugeordnet sind. Views können „umgebrochen“ werden, um ein gewohntes Schriftbild zu erzeugen. Die so entstandenen Einzelteile werden, in Zeilen gegliedert, von logischen Views verwaltet.

6.1.3 Controller

Der einzige von Swing direkt übernommene Bestandteil des Editor-Kerns ist der *Controller*. Er wird durch die Komponente `JEditorPane` implementiert. `JEditorPane` leitet Änderungen, die der Benutzer durch Tastatur- oder Mausinteraktion durchführt, an vom *Model*-Teil implementierte Instanzen der Schnittstellen `Document` und `Element` weiter. Diese Instanzen müssen entsprechend den Änderungen reagieren und Ereignisse erzeugen, die die Veränderung im Modell beschreiben. Ein solches Ereignis enthält zum Beispiel die Information, dass ein bestimmtes Objekt des *Models* gelöscht wurde. Durch die Ereignisbehandlung des Swing-Ereignis-Threads werden sie an Instanzen der abstrakten Klasse `View` weitergeleitet. Alle Bestandteile des *View*-Teils des Editor-Kerns leiten sich von dieser ab und passen ihre Darstellung dynamisch an.

Der Editor-Kern unterstützt auch Unterstützung für Block-Operationen wie Kopieren, Ausschneiden und Einfügen, ebenso wie Drag&Drop und direkte Selektion mit Hilfe der Maus.

Die Bestandteile des *Models* implementieren Schnittstellen, durch die sie als XML-Daten serialisiert werden können.

6.1.4 Zusammenfassung zur *Model-View-Controller* Architektur

Bei der folgenden Grafik wurde das Beispiel von Abbildung 15: Model-Teil des Editor-Kerns zugrunde gelegt. Der Übersichtlichkeit halber sind die direkten Zugriffspfade des *Views* auf die Objekte des *Modells* nicht eingezeichnet.

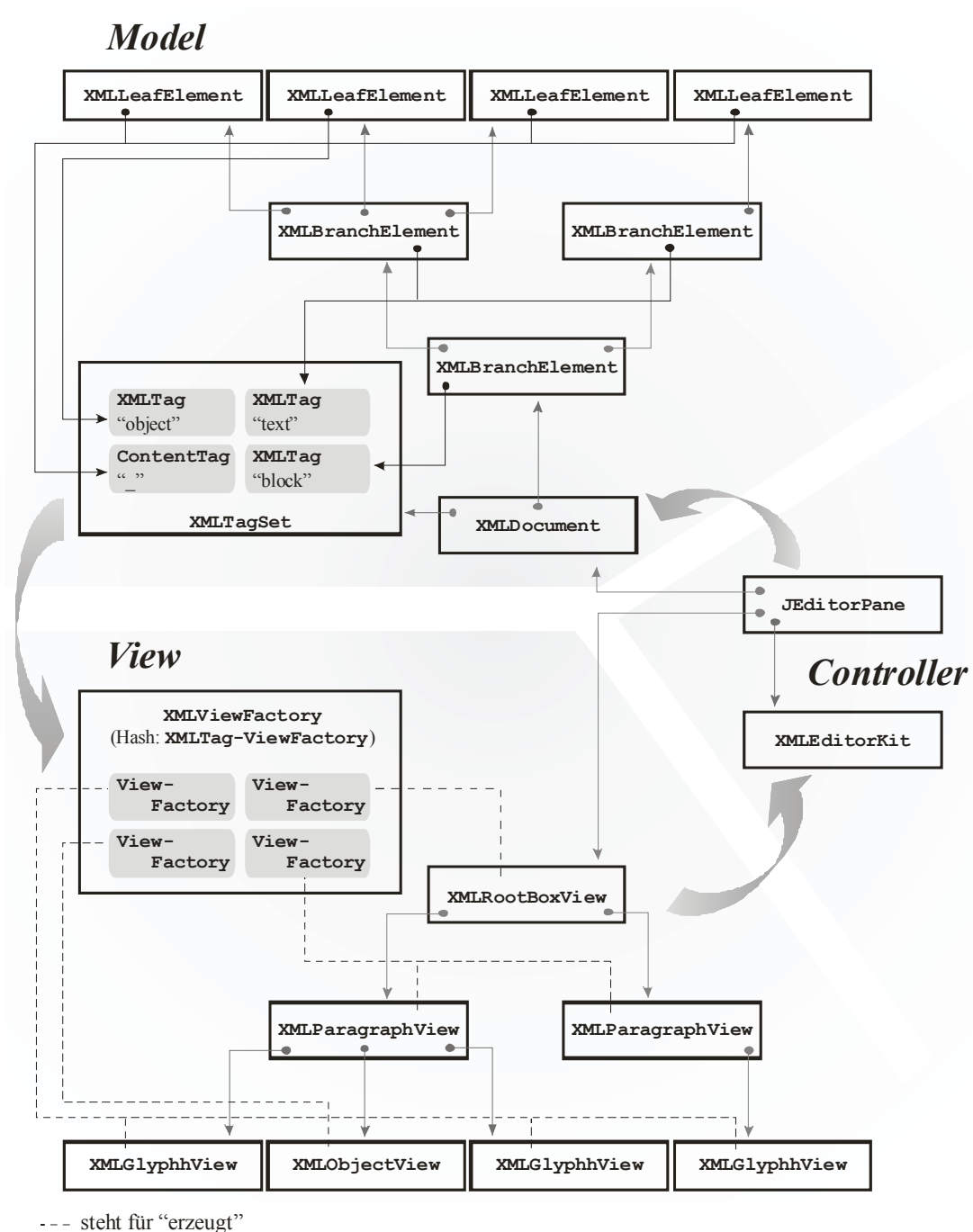


Abbildung 16: *Model-View-Controller* Architektur des Editor-Kerns

6.2 Transformationsunterstützung

Der Editor-Kern ermöglicht es, eine graphische Repräsentation von XML-Daten zu bearbeiten. Um die Projektaufgabe vollständig zu erfüllen, ist weiterhin eine Transformation dieser Daten in ein externes Ausgabeformat notwendig.

Basis dieses Moduls ist die Klasse `IXMLTransformationManager`. Der Transformationsmanager kann für jeden XML-Dialekt instantiiert werden. Er verwaltet eine Liste von Instanzen von `IXMLTransformator`, welche jeweils für die Umwandlung in ein anderes Format stehen. Die Standardimplementation des Transformationsmanagers ist zudem ein Klient für Plugins. Jedem XML-Dialekt wird automatisch eine Instanz der Klasse `Context`, des bereits erwähnten hierarchischen Identifikators, für den Dienst „Umwandlung einer Datei in dieses XML-Dialekts in ein anderes Format“ zugeordnet. Darüber kann jeder Dialekt um beliebige Plugins mit jeweils beliebig vielen Transformatoren erweitert werden.

Standardmäßig können XML-Daten sehr günstig über XSLT-Stylesheets transformiert werden. Das `IXMLTransformator`-Interface erlaubt es jedoch prinzipiell beliebige Werkzeuge, auch externe Programme, zu nutzen.

Oft macht es die Transformation von einem XML-Dialekt in einen anderen notwendig, zuerst ein Dokument aus bestimmten Elementen verschiedener anderer Dokumenten zu erstellen. Darum wird zusätzlich ein effizienter Mechanismus zum Suchen in Dokumenten zur Verfügung gestellt. Jede solche Suche wird durch eine Instanz von `Search` beschrieben, welche einen `Matcher` nutzt, um zum Beispiel Verweise auf externe Bilder oder andere Dokumente zu finden.

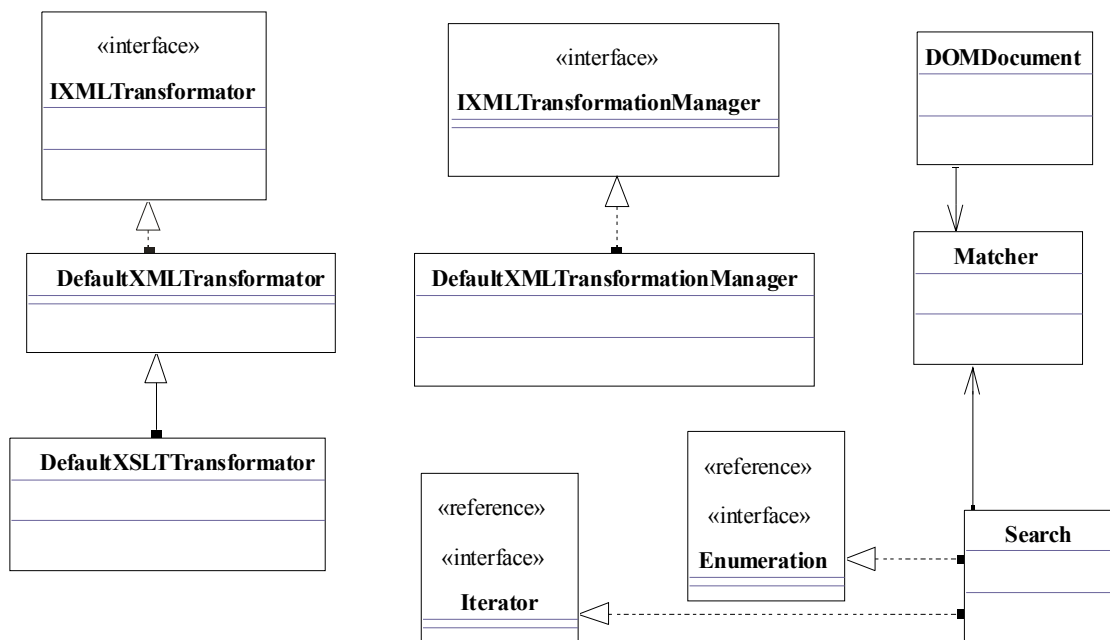


Abbildung 17: Klassendiagramm der Transformationsunterstützung

6.3 Kompositions- und Inhaltsunterstützung

Um den universellen Einsatz des Editors zu ermöglichen, ist die Definition der unterstützten XML-Dialekte von der Implementation des Editorkerns getrennt.

Das Interface `IXMLLanguageDefinition` beinhaltet alle grundlegenden Informationen einer solchen Dialektdefinition. Mit Hilfe seiner Methoden erhält der Editor eine konkrete Instanz des `XMLTagSet` aus Schicht 1, die für die Verifikation, den Tag-Router und das Parsen verwendet wird.

Als weiteres Betriebsmittel aus Schicht 1 stellt jede Dialektdefinition eine Instanz von `I18NActionManager` bereit. Hierbei handelt es sich um eine Liste von dialekt-spezifischen Aktionen. Eine Aktion ist ein Befehl, der vom Benutzer gerufen wird. Bei KML wäre dies beispielsweise das Kommando „Tabelle einfügen“ oder „Absatz einfügen“. Damit wird der Tatsache Rechnung getragen, dass verschiedene XML-Dialekte verschiedene Elementtypen besitzen. Das Konzept des `I18NActionManager` verbindet jeden Befehl mit seinem Icon, seinem (internationalisierten) Bezeichner und Tooltip und seiner Position in der Hierarchie des Menüs, der Werkzeugleiste und des Popup-Menüs. Er sorgt auch dafür, dass diese Metainformation bei einer Sprachumschaltung automatisch angepasst wird.

Einem Dialekt kann eine spezifische Dateiendung zugeordnet sein, zu der wiederum ein internationalisierter Bezeichner gehört. Beim KML-Inhaltsdialekt wären dies z.B. „kmlc“ und „KML Inhaltsdatei“.

Um weitere internationalisierte Bezeichner des Dialekts erfragen zu können, wird eine `I18N`-Instanz mit der Dialektdefinition assoziiert. Jeder XML-Dialekt kann eine zusätzlich Instanz der Schnittstelle `IXMLTransformationManager` besitzen.

Die beiden Schnittstellen `ICompositionLanguageDefinition` und `IContentLanguageDefinition` sind Spezialisierungen des Interfaces `IXMLLanguageDefinition` und erweitern es um Methoden speziell für die Inhaltskomposition in Bäumen per Drag&Drop bzw. für die Bearbeitung von Inhalten im Editor-Kern.

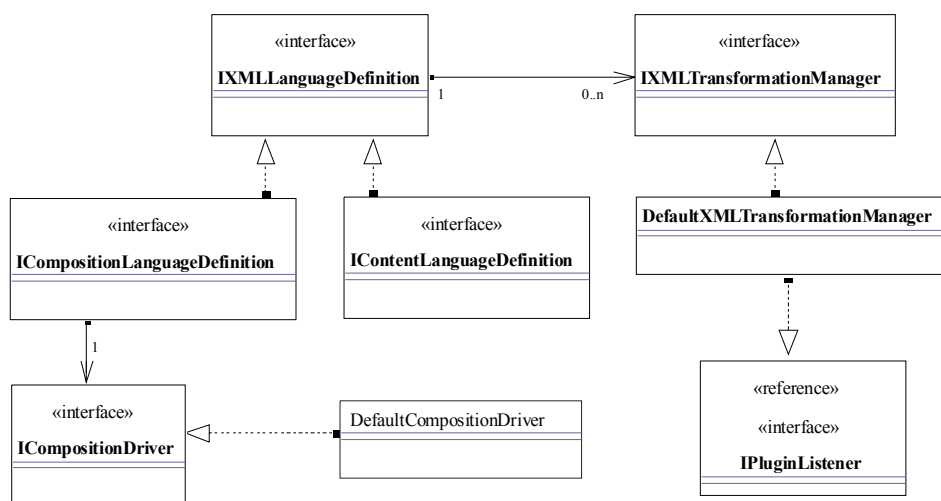


Abbildung 18: Klassendiagramm der Sprachdefinition

7 Schicht 3: Anpassungsschicht, XML-Dialekt

In Schicht 2 befinden sich die abstrakten Schablonen für alle notwendigen Betriebsmittel des Editors. Um den Editor für einen speziellen Dialekt nutzbar zu machen, müssen diese in der Anpassungsschicht instantiiert werden. Dies soll am Beispiel des XML-Dialekts KML, welcher die Anforderungen für die Nutzung im Bereich der Lehre erfüllt und der Aufgabenstellung entspricht, beschrieben werden.

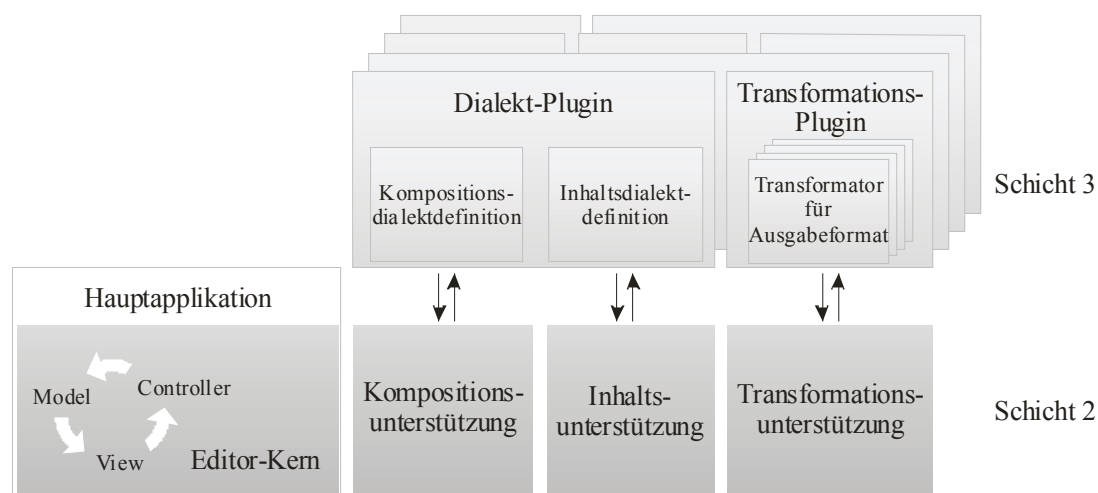


Abbildung 20: Struktur der Schicht 3 des Editors

Wie

Abbildung 20 zeigt, befindet sich die eigentliche Hauptapplikation in Schicht 2. Vergleichbar mit einem Betriebssystem lädt sie die einzelnen Komponenten (Plugins) nach, die die Arbeitsumgebung des Benutzers definieren.

Die KML-Unterstützung eignet sich als Beispiel für die Anpassung des Editors besonders gut, da sie dessen Möglichkeiten komplett ausschöpft. KML gliedert sich in einen Inhaltsdialekt und einen Kompositions-dialekt. Der Inhaltsdialekt wird verwendet, um die Dokumente mit den Lehrinhalten mit einem WYSIWYG-Editor in der fachlogischen Struktur zu erzeugen. Diese Lehrinhalte werden dann mit Hilfe des Kompositions-dialekts in einem Baum nach der didaktischen Struktur gruppiert. Das Ergebnis dieser Gruppierung soll in andere Datenformate, z.B. für das Web oder für das Drucken von Skripten, umgewandelt werden.

Im Weiteren wird die Menge der Module, die den XML-Dialekt KML auf Schicht 3 mit Hilfe des allgemeinen XML-Editors unterstützen, als KML-Editor bezeichnet, der aus Gründen der Modularität und der Übersichtlichkeit in zwei Plugins gegliedert ist.

Das Dialekt-Plugin soll zur Eingabe und didaktischen Strukturierung der Lerninhalte dienen. Diese können dann mit dem Transformations-Plugin in andere Formate umgewandelt werden. Die Plugin-API des XML-Editors ließe jedoch eine beliebige Gruppierung dieser Funktionalitäten, so z.B. ein Aufteilen in drei oder das Zusammenfassen zu einem einzigen Plugin zu.

7.1 Dialekt-Plugin

Um einen Dialekt in den XML-Editor zu integrieren, muss die Schnittstelle `IXMLLanguageDefinition` implementiert werden. Das KML-Plugin nutzt zwei Spezialisierungen dieser Schnittstelle, `ICompositionLanguageDefinition` und `IContentLanguageDefinition`.

Der Inhaltsdialekt von KML (<http://www.kml-edu.org/kml/schema/1.0/content.xsd>) wird durch eine Instanz von `IContentLanguageDefinition` beschrieben. Diese Schnittstelle wird mit Hilfe WYSIWYG-Editors dargestellt und erweitert die normale Sprachdefinition um die Möglichkeit, Instanzen von `XMLViewFactory` an den Editor-Kern zu übergeben.

Exemplarisch am KML-Inhaltsdialekt soll die Implementation der Schnittstelle `IContentLanguageDefinition` beschrieben werden.

7.1.1 Erzeugung der XMLTags

Da alle XML-Tags und -Attribute sprachspezifische Beschreibungen besitzen, ist der erste Schritt beim Erstellen eines Plugins das Erzeugen einer Instanz von `I18N`, die mit Ressourcendateien in den verschiedenen Sprachen verbunden wird. Sie wird später lokalisierte Bezeichner für alle XML-Tags und -Attribute enthalten.

```
...
public final class KMLI18N
{
/** The internationalized information for kml items */
public static final I18N I18N =
    new I18N(Design.I18N,
            "kml.resources.kml",
            KMLI18N.class);
...

```

Listing 1: Instantiierung von `I18N`

Über die exakte Bedeutung der einzelnen Methoden und deren Parameter sowie verwendeter Klassen gibt die Dokumentation Aufschluss.

7.1.2 Erzeugung des XMLTagSets

Als nächstes ist die Übertragung des XML-Schemas des Dialekts in eine Instanz von XMLTagSet notwendig. Dabei sind zunächst die Attribute der einzelnen Inhaltselemente zu untersuchen. Die Klasse XMLTag erzeugt für jede Instanz ihres Elementtyps eine leere Attributmenge (XMLAttributeSet). Besitzt ein Elementtyp eines XML-Dialekts eine nichtleere Attributmenge, so muss für diesen Elementtyp eine spezialisierte Klasse von XMLTag abgeleitet werden, bei der die Methode create_xml_attributes() entsprechend anzupassen ist. Da alle KML-Tags mindestens die Attribute „id“ und „title“ besitzen, enthält das Plugin eine solche spezialisierte Klasse. Das folgende Listing stellt beispielhaft einen Ausschnitt aus der create_xml_attributes()-Methode dar.

```
class KMLTag
{
    ...
/**
 * Creates a new attribute set for an kml element.
 * @return A default xml attribute set for kml tags.
 */
    public XMLAttributeSet create_xml_attributes ()
    {
        XMLAttributeSet l_xas;
        String          l_nsu;
        l_xas = super.create_xml_attributes();
        l_nsu = get_namespace_uri();
        l_xas.local_put( new IDAttribute(l_nsu,
                                       KMLI18N.ID_ATTRIBUTE,
                                       KMLI18N.I18N) );

        l_xas.local_put( new StringAttribute(l_nsu,
                                             KMLI18N.TITLE_ATTRIBUTE,
                                             null, KMLI18N.I18N) );

        ...
        return l_xas;
    }
    ...
}
```

Listing 2: Erzeugung eines Elementtyp-spezifischen XMLAttributeSet

Tabellenzellen, Listen und andere Elemente von KML besitzen noch zusätzliche Attribute, und werden darum durch neue spezialisierte Klassen abgeleitet von KMLTag definiert.

Sind alle notwendigen Klassen erzeugt, so können diese im XMLTagSet des Dialekts gruppiert werden. Hierbei findet das Design-Pattern *Singleton* Einsatz. Jeder Elementtyp wird durch genau eine (unveränderbare) Instanz von XMLTag und der Dialekt selbst durch genau eine Instanz von XMLTagSet beschrieben. Aus diesem Grund können alle diese Metadaten statisch angelegt werden.

Beim Erzeugen des XMLTagSets ist auch die Hierarchie der Tags zu definieren. Ein Ruf von `A.add_includer(B)` bestimmt, dass Elemente vom Typ A in Elementen vom Typ B enthalten sein dürfen. (A und B sind Instanzen von XMLTag).

Die Methode `local_put` fügt die einzelnen Elementtypen zum primären Namespace des XMLTagSets hinzu.

Ein weiter Ausschnitt aus dem Quelltext verdeutlicht diesen Vorgang.

```
public final class KMLContentTagSet extends XMLTagSet
{
    public static final XMLTag UNIT ;
    public static final XMLTag BLOCK ;
    public static final XMLTag TEXT ;
    ...
        static final KMLContentTagSet TAG_SET ;
    ...
    static
    {
        UNIT = new KMLRootTag(KML_CONTENT_URI, "unit");
        BLOCK = new KMLTag (KML_CONTENT_URI, "block");
        TEXT = new KMLTag (KML_CONTENT_URI, "text");
    ...
        BLOCK.add_includer(UNIT);

        TEXT.add_includer(BLOCK);
        TEXT.add_includer(DEFINITION);
        TEXT.add_includer(ALGORITHM);
    ...
        TAG_SET = new KMLContentTagSet();
    }

    public KMLContentTagSet ()
    {
        super(KML_CONTENT_URI);
        XMLTag l_content;
        local_put(BLOCK);
        local_put(UNIT);
        local_put(TEXT);
    ...
    }
    ...
}
```

Listing 3: Erzeugung eines XMLTagSets

Mit der Information des XMLTagSets ist der Editor-Kern bereits in der Lage, Dokumente des Dialekts zu laden, zu speichern und zu verifizieren.

7.1.3 Erzeugung der XMLViewFactory

Für die visuelle Darstellung wird mindestens eine `XMLViewFactory` benötigt.

Eine `XMLViewFactory` ordnet jedem Elementtyp einen *View*-Typ für dessen visuelle Repräsentation zu. Dabei nutzt sie Instanzen von `ViewFactory`, die jeweils eine einzelne solche Bindung realisieren. Aus einer vordefinierten Menge von Views, die in der Dokumentation ausführlich beschrieben ist, muss jeweils der passende Typ ausgewählt werden. Es ist einzusehen, dass Tabellen sich in der graphischen Darstellung von Bildern unterscheiden.

Eine `XMLViewFactory` ähnelt im Aufbau sehr stark dem `XMLTagSet`, auch hier wird auf das *Singleton* Design-Pattern zurückgegriffen und es kann durch statisches Anlegen der Metadaten Speicher gespart werden.

```
final class KMLContentViewFactory extends XMLViewFactory
{
...
private static final ViewFactory CONTENT_VIEWFACTORY =
new ViewFactory() {
public View create(Element p_elem)
{
return new XMLGlyphView(p_elem, XMLGlyphView.WORD_WRAP);
}
};

private static final ViewFactory PARAGRAPH_VIEWFACTORY =
new ViewFactory() {
public View create(Element p_elem)
{
return new KMLParagraphView(p_elem, PARAGRAPH_BOX,
PARAGRAPH_FONT, PARAGRAPH_FOREGROUND);
}
};

...
static final XMLViewFactory VIEW_FACTORY =
new KMLContentViewFactory();

public KMLContentViewFactory ()
{
super(UNKNOWN_VIEWFACTORY);
add_context(DEFAULT_CONTEXT);
add_context(SCREEN_CONTEXT);
put(KMLContentTagSet.TAG_SET.get_content_tag(),
CONTENT_VIEWFACTORY);
put(KMLContentTagSet.TEXT, PARAGRAPH_VIEWFACTORY);
put(KMLContentTagSet.BLOCK, BLOCK_VIEWFACTORY);
put(KMLContentTagSet.UNIT, UNIT_VIEWFACTORY);
...
}
...
}
```

Listing 4: Erzeugung einer `XMLViewFactory`

7.1.4 Erzeugung der Dialektaktionen

Nach der Bereitstellung der `XMLViewFactory`-Instanz können die Elemente des Dialekts im WYSIWYG-Editor graphisch dargestellt werden.

Jetzt muss man dem Benutzer noch ermöglichen, neue Elemente des XML-Dialekts in das Dokument einzufügen. Dies wird durch eine Instanz vom `I18NActionManager` erreicht, die die Menge unterstützter Befehle für den Dialekt in Form von `Action`-Instanzen beinhaltet. Ein Befehl des KML-Inhaltsdialekts ist zum Beispiel „Tabelle einfügen“. Der `I18NActionManager` entnimmt der `I18N`-Instanz des Plugins die notwendigen Informationen wie die lokalisierten Bezeichnungen der Befehle, ihre Icons und Hotkeys. Die Hauptapplikation gruppiert die Befehle automatisch im Menü, dem Popup-Menü und der Werkzeuggeste.

Auch die Information der `I18NActionManagers` sind statisch, dem *Singleton*-Design-Pattern folgend, abgelegt.

```
final class KMLContentActions
{
    static final I18NActionManager ACTIONS ;

    static
    {
        ACTIONS = new I18NActionManager(KMLI18N.I18N);
        ACTIONS.add actions(
            new Action[] {
                new InsertBlockAction(), new InsertObjectAction(),
                new InsertStructureAction(KMLI18N.INSERT_DEFINITION,
                    KMLContentTagSet.DEFINITION,
                    null),
                ...
            }, null);
    }

    private static final class InsertObjectAction extends XMLAction
    {
        InsertObjectAction()
        {
            super(KMLI18N.INSERT_OBJECT);
        }

        protected final void perform (XMLEditorPane p_pane,
                                       XMLDocument p_doc,
                                       int p_pos,
                                       int p_len)
        {
            ...
        }
    }
    ...
}
```

Listing 5: Erzeugung eines `XMLActionManagers`

7.1.5 Erzeugung der Dialektdefinition

Letztendlich müssen die Informationen noch zu einer statischen Instanz von `IContentLanguageDefinition` zusammengefasst werden. Zu beachten ist die Instanz von `DefaultTransformationManager`, die später automatisch Plugins mit Transformatoren für den XML-Dialekt nachlädt.

```
public final class KMLContentLanguageDefinition
    implements IContentLanguageDefinition
{
    private static final XMLViewFactorySet FACTORIES ;
    public static final String KML_BASE_URI =
        "http://www.kml-edu.org/kml/schema/1.0/";

    static
    {
        FACTORIES = new XMLViewFactorySet();
        FACTORIES.add_factory(KMLContentViewFactory.VIEW_FACTORY);
    }

    public static final IXMLLanguageDefinition DEFINITION =
        new KMLContentLanguageDefinition();

    private static final IXMLTransformationManager TRAFO =
        new DefaultXMLTransformationManager(DEFINITION);

    public XMLViewFactorySet get_view_factories ()
    {
        return FACTORIES;
    }

    public XMLTagSet get_tag_set ()
    {
        return KMLContentTagSet.TAG_SET;
    }

    public I18NActionManager get_language_actions ()
    {
        return KMLContentActions.ACTIONS;
    }

    ...

    public I18N get_i18n ()
    {
        return KMLI18N.I18N;
    }

    public IXMLTransformationManager get_transformations ()
    {
        return TRAFO;
    }

    ...
}
```

Listing 6: Erzeugung einer `IXMLLanguageDefinition`-Instanz

7.1.6 Erzeugung des Didaktikdialekts

Die Beschreibung eines Kompositionsdiakts, wie er für die didaktische Strukturierung der Lehrinhalte genutzt wird, unterscheidet sich nur wenig von der eines Inhaltsdiakts. Da Kompositionen immer als Baum dargestellt werden, ist keine Instanz von XMLViewFactory notwendig. Stattdessen genügt eine (weniger aufwändige) Implementierung von ICompositionDriver. Sie legt das Verhalten des Baums fest und bestimmt, wie die Umwandlung von Elementen des Inhaltsdiakts in Verknüpfungen des Kompositionsdiakts erfolgt. Auch von dieser Klasse genügt eine einzige Instanz.

Das Schema <http://www.kml-edu.org/kml/schema/1.0/didactic.xsd> definiert den KML-Didaktikdiakt.

```
final class KMLCompositionDriver extends DefaultCompositionDriver
{
    final static ICompositionDriver DRIVER =
        new KMLCompositionDriver();

    public final XMLCompositionNode create_root ()
    {
        return create_node(KMLCompositionTagSet.MODULE);
    }

    public final XMLElement get_insert_element(
        XMLElement p_element,
        XMLComposition p_composition)
    {
        ...
    }

    public final XMLCompositionNode create_leaf_node(
        XMLElement p_element,
        XMLComposition p_tree)
    {
        ...
    }

    public final XMLCompositionNode create_folder ()
    {
        return new XMLCompositionNode(KMLCompositionTagSet.PU, null);
    }

    public final XMLCompositionNode create_leaf ()
    {
        return new XMLCompositionNode(KMLCompositionTagSet.LINK, null);
    }
}
```

Listing 7: Erzeugung des Kompositionstreibers

7.1.7 Erzeugung der Hilfeunterstützung des Plugins

Zu jedem XML-Dialekt, für den eine Anpassung in den Editor integriert wird, sollte auch eine separate Hilfe existieren. Dafür sind zwei Schritte notwendig: das Schreiben der Hilfedateien im HTML-Format (in mehreren Sprachen) und das Bekanntmachen des Hilfesystems mit diesen Dateien.

Das Quellcode-Listing zeigt einen Teil dieser Bekanntmachung:

```
public final class HelpSupport
{
    public static final DefaultHelpSupport SUPPORT ;

    static
    {
        SUPPORT = new DefaultHelpSupport("help",
                                         "help.resources.help",
                                         HelpSupport.class);
        SUPPORT.add_topic("intro_title",    "intro_link");
        SUPPORT.add_topic("kml_title",     "kml_link");
        SUPPORT.add_topic("information_title", "information_link");
        ...
    }
}
```

Listing 8: Erzeugung einer Instanz von DefaultHelpSupport

Damit die Hilfe in der richtigen Sprache angezeigt wird, nutzt DefaultHelpSupport eine interne Instanz von I18N. Der Autor der Hilfe fügt internationalisierte Ressourcen für diese Instanz hinzu, die jeweils auf die richtige Sprachversion der Dateien verweisen:

```
# Deutsche Version
...
# links
intro_link      = help/resources/html/intro_de.htm
intro_title    = Was ist der KML-Editor?
...
```

Listing 9: Deutsche Version einer Hilfe-Ressource (help_de.properties)

```
# English Version
...
# links
intro_link      = help/resources/html/intro_en.htm
intro_title    = What is the KML-Editor?
...
```

Listing 10: Englische Version einer Hilfe-Ressource (help_en.properties)

Bei einer Sprachänderung wird die interne I18N-Instanz informiert und somit kann immer die korrekte Hilfe angezeigt werden.

7.1.8 Erzeugung des Plugins

Um die so erstellten Sprachdefinitionen dem Editor-Kern zur Verfügung zu stellen, sind sie in ein Plugin zu kapseln. Dafür wird eine Klasse `Plugin` im Package `plugin` angelegt, die das `IPlugin`-Interface implementiert und die der `PluginLoader` automatisch instantiiert.

```
package plugin;
...
public final class Plugin implements IPlugin
{
    private static final List LANGUAGES ;

    static
    {
        ArrayList l_l;
        l_l = new ArrayList();
        l_l.add(KMLContentLanguageDefinition.DEFINITION);
        l_l.add(KMLCompositionLanguageDefinition.DEFINITION);
        LANGUAGES = Collections.unmodifiableList(l_l);
    }

    ...

    public Object get_support (Context p_context)
    {

        if(p_context == IXMLLanguageDefinition.XML_LANGUAGE_DEFINITION)
        {
            return LANGUAGES;
        }

        if(p_context == IHelpSupport.HELP_SUPPORT)
        {
            return HelpSupport.SUPPORT;
        }

        return null;
    }
}
```

Listing 11: Erzeugung des Plugins

Der erzeugte Code ist als jar-Archiv im Unterordner „plugins“ des Editors abzulegen. Der Editor wurde somit um einen neuen XML-Dialekt erweitert.

Natürlich können in einem Plugin beliebig viele Dialekte enthalten sein, auch könnte man einen Dialekt auf mehrere Plugins verteilen. Aus Gründen der Wartbarkeit ist dies jedoch nicht zu empfehlen.

7.2 Transformations-Plugin

Für die Transformationen der Dokumente eines XML-Dialekts in andere Datenformate wird ein zusätzliches Plugin entwickelt. Dieses ist leichter zu erweitern oder auszutauschen, ohne mit der Sprachdefinition an sich in Kontakt zu kommen.

Die `get_support`-Methode des Plugins liefert eine Liste von Instanzen von `IXMLTransformator`. Jede solche Instanz ist in der Lage, ein Dokument des Dialekts in ein Dokument eines anderen Datenformats zu transformieren.

```
public final class ML3Transformator extends DefaultXMLTransformator
    implements IXMLTransformator
{
    private static final String XSLT = "ml3/xslt/kml2ml3.xslt";

    public I18N get_i18n ()
    {
        return ML3I18N.I18N;
    }
    ...
    public final boolean transform (URL p_source,
                                    File p_dest,
                                    Component p_comp)
    {
        IFileContext l_f;
        String l_n;
        if((p_source != null) && (p_dest != null))
        {
            l_f = FileContext.create(p_dest);
            l_n = Files.get_file_name(p_source);
            if(l_n != null) l_n += DocumentMerger.DIRECTORY_SUFFIX;
            try
            {
                ...
            }
            finally
            {
                l_f.close();
            }
        }
        return false;
    }
}
```

Listing 12: Erzeugung des Transformers

Das Plugin folgt in seiner Struktur den Richtlinien des vorigen Kapitels. Wie im Listing zu erkennen, besitzt es auch eine `I18N`-Instanz, welche dazu dient, die Steuerelemente zur Konfiguration der Transformatoren zu internationalisieren.

8 Ausgabe

Mit den durch die Dialektdefinition zur Verfügung gestellten Transformatoren produziert der Editor Ausgaben in verschiedenen Formaten.

Das KML-Transformations-Plugin steuert das WWR-Buildtool [wvr] an, das Dokumente des KML-Didaktik-Dialekts in Webpräsentationen, Skripten und Unterrichtsfolien umgewandelt. Im Folgenden werden diese Formate kurz mit Hilfe von Screenshots eines WWR-Moduls vorgestellt.

8.1 Webpräsentation

Die Webpräsentation von Lehrinhalten erfolgt im zukunftsträchtigen Format XHTML [xhtml]. Zurzeit unterstützen es leider noch nicht alle Browser, darunter z.B. der Microsoft Internet Explorer. Es empfiehlt sich zur Anzeige ein Browser der Mozilla-Familie.

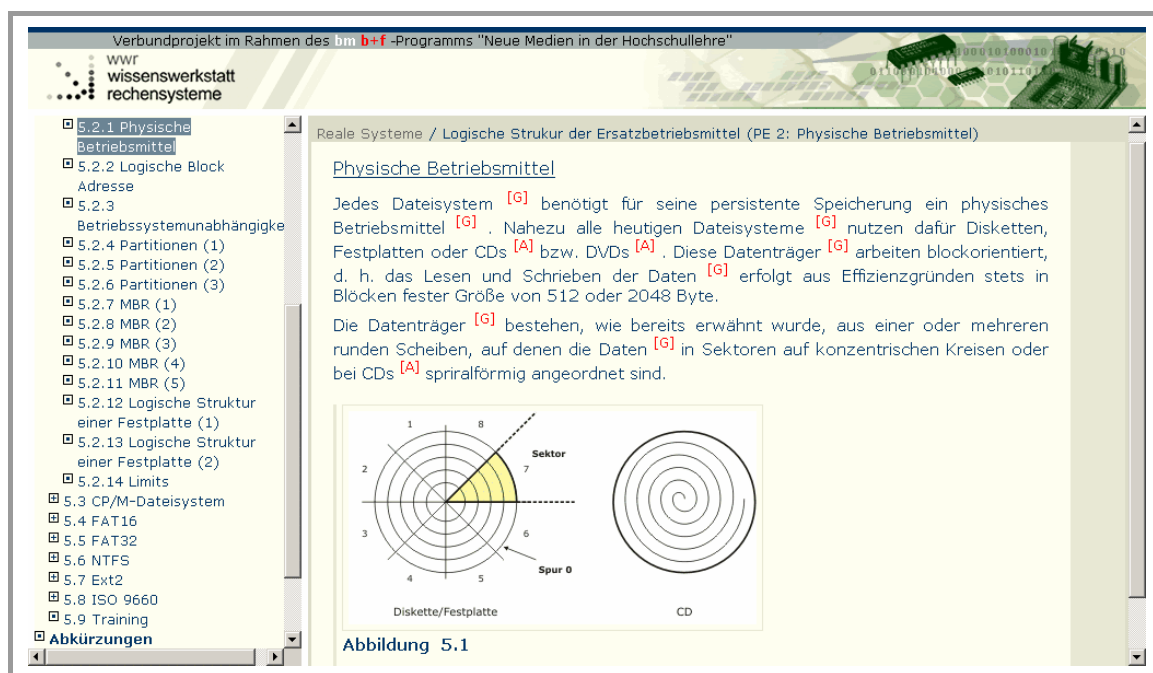


Abbildung 21: Screenshot einer Webpräsentation

8.2 Skripten

Die Skripten zum Lehrmaterial werden im Portable Document Format (PDF) von Adobe erstellt. Dieses vektorbasierte Format ist sehr verbreitet und wird für die meisten universitären Skripte und wissenschaftlichen Arbeiten genutzt.

Lernschritt 5.1:

Logische Struktur der Ersatzbetriebsmittel

Physische Betriebsmittel

Jedes Dateisystem benötigt für seine persistente Speicherung ein physisches Betriebsmittel. Nahezu alle heutigen Dateisysteme nutzen dafür Disketten, Festplatten oder CDs bzw. DVDs. Diese Datenträger arbeiten blockorientiert, d. h. das Lesen und Schreiben der Daten erfolgt aus Effizienzgründen stets in Blöcken fester Größe von 512 oder 2048 Byte.

Die Datenträger bestehen, wie bereits erwähnt wurde, aus einer oder mehreren runden Scheiben, auf denen die Daten in Sektoren auf konzentrischen Kreisen oder bei CDs spiralförmig angeordnet sind.

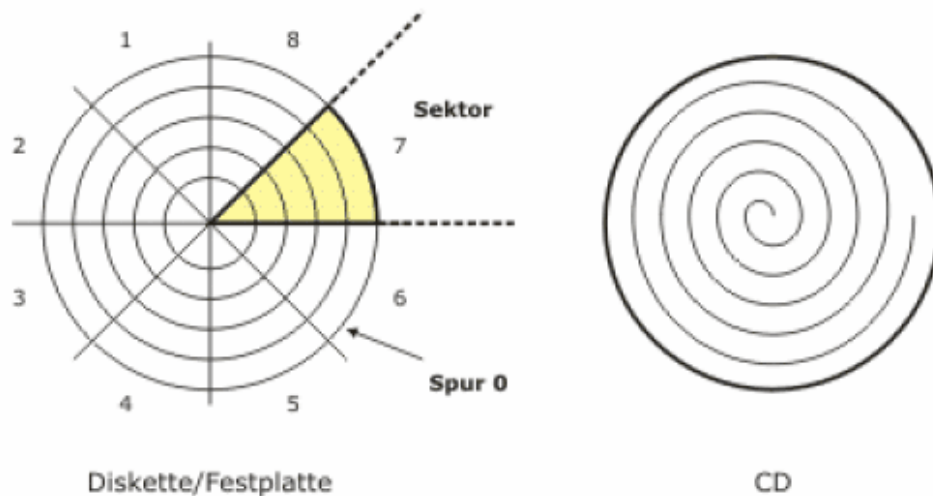


Abbildung 1.1.0

Anmerkung:

Prinzipieller Aufbau einer Disk und einer CD

Abbildung 22: Screenshot eines Skripts

8.3 Unterrichtsfolien

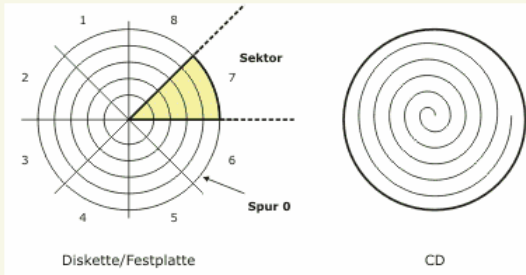
Um den Präsenzunterricht zu unterstützen, wird ein Format benötigt, das sich sowohl für die Präsentation mit einem Overhead-Projektor als auch einem Beamer eignet. Ziel des WWR-Projekts ist es, dafür Microsoft Powerpoint-Präsentationen zu nutzen, was zurzeit noch nicht möglich ist. Das WWR-Buildtool nutzt stattdessen XML, das über XSLT-Stylesheets vom Browser nach HTML transformiert wird. Damit kann ein normaler Webbrowser zur Unterrichtsunterstützung genutzt werden.

Verbundprojekt im Rahmen des **WWR**-Programms „Neue Medien in der Hochschullehre“

Physische Betriebsmittel

Physische Betriebsmittel

Jedes Dateisystem benötigt für seine persistente Speicherung ein physisches Betriebsmittel. Nahezu alle heutigen Dateisysteme nutzen dafür Disketten, Festplatten oder CDs bzw. DVDs. Diese Datenträger arbeiten blockorientiert, d. h. das Lesen und Schreiben der Daten erfolgt aus Effizienzgründen stets in Blöcken fester Größe von 512 oder 2048 Byte.



Prinzipieller Aufbau einer Disk und einer CD

Abbildung 23: Screenshot der Präsentation für den Präsenzunterricht

8.4 Weitere Formate

Um das <ML>³-Buildtool anzu steuern zu können, ist es notwendig, KML-Dokumente nach <ML>³ zu konvertieren. Diese Funktionalität stellt der <ML>³-Transformator dem Benutzer zur Verfügung.

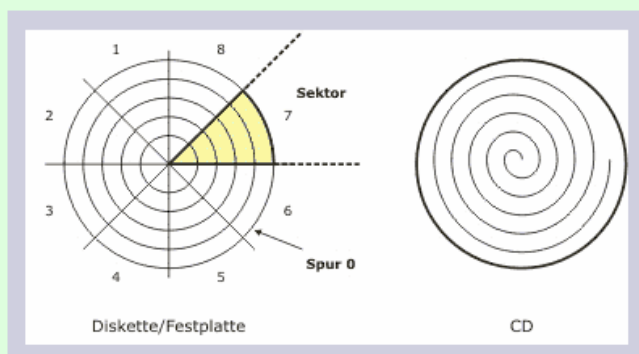
Weiterhin erlaubt es das Transformationsplugin KML-Dokumente in HTML umzuwandeln, welche durch ein simples CSS¹-Stylesheet formatiert sind. Im Gegensatz zu den bisherigen Ausgabeformaten lassen sich hier Änderungen im Layout noch mit begrenzten Kenntnissen durchführen. Als Hilfestellung wird [shtml] empfohlen.

Um die Benutzung des Editors besonders für Lehrkräfte interessanter zu machen, die Erfahrungen mit dem Produkt Microsoft Office besitzen, wurde eine Transformation nach WordML implementiert. Diese ist zwar nicht voll funktionsfähig (Tabellen und Grafiken können noch nicht übertragen werden), demonstriert jedoch die Portierbarkeit der KML-Dokumente.

1.1 Reale Systeme

1.1.1 Physische Betriebsmittel

Jedes Dateisystem benötigt für seine persistente Speicherung ein physisches Betriebsmittel. Nahezu alle heutigen Dateisysteme nutzen dafür Disketten, Festplatten oder CDs bzw. DVDs. Diese Datenträger arbeiten blockorientiert, d.h. das Lesen und Schreiben der Daten erfolgt aus Effizienzgründen stets in Blöcken fester Größe von 512 oder 2048 Byte. Die Datenträger bestehen, wie bereits erwähnt wurde, aus einer oder mehreren runden Scheiben, auf denen die Daten in Sektoren auf konzentrischen Kreisen oder bei CDs spiralförmig angeordnet sind.



Grafik 1: Prinzipieller Aufbau einer Disk und einer CD

Abbildung 24: Screenshot der HTML-Darstellung

¹ Cascading Stylesheet – Mechanismus, um HTML-Dokumente mit zentralem Layout zu versehen.

9 Erprobung mit schulspezifischen Inhalten

Entwurf und Entwicklung des Editors nahmen einen Großteil der zur Verfügung stehenden Zeit des Projekts ein, da der Editor zuerst den Grad der Verwendbarkeit erreichen musste, bevor er an Testbenutzer herausgegeben werden konnte. Dies war zwingend notwendig, denn eine vorzeitige Veröffentlichung hätte durch Fehler in der Applikation zu Enttäuschung unter den Testbenutzern geführt und somit die Durchsetzungsfähigkeit des Werkzeugs verringert.

Dennoch erfolgte eine Erprobung an lehrspezifischen Inhalten an der TU Chemnitz. Diese Versuche in zunächst geringem Umfang zeigen, dass der Editor ein sinnvolles Hilfsmittel für die Erstellung von Lehrmaterial ist. Auch ist bereits jetzt ersichtlich, dass nur eine fortgesetzte Wartung und stetige Erweiterung durch Anpassung an die Wünsche der Benutzer Nachhaltigkeit der Arbeit gewähren können.

In einer Konferenz im Sächsischen Ministerium für Kultus in Dresden zeigten Gymnasiallehrer des Lehrgebiets Informatik aus ganz Sachsen hohes Interesse an der Verwendung und Weiterentwicklung des Werkzeugs. In einer Folgeveranstaltung erfolgt eine Einweisung der Lehrkräfte in den Gebrauch des Editors, der dann allen kostenfrei zur Verfügung steht. Die Lehrer der Fachgebiete Informatiksysteme, Technik und Informationsverarbeitung sächsischer beruflicher Gymnasien mit den Fachrichtungen Informations- und Kommunikationstechnologie (iGy) und Technikwissenschaften/Datenverarbeitungstechnik werden den Editor erproben. Bei entsprechender Resonanz ist eine Ausweitung seines Einsatzes auf die allgemeine Lehre vorgesehen.

Es laufen Gespräche mit dem „Dr. Wilhelm André Gymnasium“ Chemnitz über den Einsatz des Editors in der allgemeinen Gymnasiallehre.

10 Ausblick

Die erstellte Applikation übertrifft die Anforderungen der Aufgabenstellung. Anstelle eines WYSIWYG-Editor für einen speziellen XML-Dialekt wurde ein allgemeiner Editor erstellt, von dem eine speziell für die Erstellung von Lehrmaterial angepasste Instanz produziert wurde. Somit entstand ein Werkzeug mit breiten Einsatzmöglichkeiten, die sich nicht nur auf die Lehre oder das Content-Management beschränken.

Da die Applikation sehr umfangreich ist (über 45.000 Zeilen Code mit LOCC [locc] gemessen und über 2000 Seiten Dokumentation), steht der Autor für deren Wartung weiterhin zur Verfügung.

10.1 Editor für Lehrmaterial

Die Akzeptanz des Editors für Lehrmaterial soll erhöht werden, indem er gewartet und durch fortführende Arbeiten erweitert wird:

- Ein Inline-Formeleditor integriert eine MathML-Unterstützung [mml] in den Editor.
- Das WWR-Literaturdatenbank soll auf einem Server der TU Chemnitz zur Verfügung gestellt und in den Editor eingebunden werden.
- Ein Ergebnis der Erprobung wird eine Zusammenstellung von Forderungen nach zusätzlichen Features und Anpassungen sein, die dann in den KML-Dialekt einfließt und die Verwendbarkeit des Editors erhöht.

10.2 allgemeiner XML-Editor

Der Quellcode des allgemeinen XML-Editors wird frei verfügbar sein. Unter einer speziellen Lizenzform, ähnlich der GPL, ist eine gesteuerte Weiterentwicklung vorgesehen. Weiterhin ermöglicht die Internationalisierung des Editors und aller seiner Komponenten seinen Einsatz als allgemeines Werkzeug auch über die Grenzen des deutschsprachigen Gebiets hinaus. Bei entsprechender Akzeptanz könnten beliebig viele XML-Dialekte in Form von Plugins integriert werden. Vorteilhaft für diese Entwicklung ist die Kostenfreiheit und Verfügbarkeit aller Komponenten, die umfangreiche Dokumentation und die saubere Programmierung gemäß der in Kapitel 3.1 genannten generellen Festlegungen des Entwurfs.

10.3 Internet-Portal

Um diese beiden Ansätze effektiv verfolgen zu können, entwickelt der Autor in Zusammenarbeit mit der Betreuerin der Arbeit ein Internetportal für das Projekt.

Dieses Portal soll sowohl als Forum für die Nutzer der KML-Anpassung des Editors als auch für interessierte Entwickler weltweit dienen. Verschiedene Nutzergruppen können den Editor und die Plugins herunterladen sowie Einsicht in FAQs und Buglists erhalten.

Die gesteuerte Weiterentwicklung des Editors soll unter einer Lizenz erfolgen, die sich von der GPL in dem Punkt unterscheidet, dass neue oder geänderte Module inklusive ihres Quellcodes und ihrer Dokumentation im Internetportal zur Verfügung zu stellen sind.

Eine exakte Beschreibung des geplanten Internet-Portals kann dem Dokument [ip] entnommen werden.

11 Literaturverzeichnis

- [dp] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides – Design Patterns, Elements of Reusable Object-Oriented Software
22nd Printing, July 2001, Addison-Wesley, © 1995
ISBN 0201633612
- [ip] Thomas Weise, Internetportal des XML/KML-Editors
Planungsskizze, 2005
- [javb] Guido Krüger: Handbuch der Java-Programmierung, 3. Auflage
HTML-Ausgabe 3.0.1 · © 1998-2003 Guido Krüger
Addison-Wesley, 2002, ISBN 3-8273-1949-8
www.javabuch.de, www.gkrueger.com, hjp3@gkrueger.com
- [jsrc] Java Swing Source 1.4.2.05
Teil von Java 2 SDK, Standard Edition 1.4.2.05
<http://developers.sun.com/>
- [jxml] Stefan M. Dellbrügge, Martin Kremser: Seminararbeit – Document Object Model (DOM) und Simple API for XML (SAX), 2002
Sommersemester 2002, Verteilte und Parallele Systeme 1, Prof. Dr. Rudolf Berrendorf, Fachhochschule Bonn-Rhein-Sieg
- [kal] Prof. Dr. Ing. habil. Winfried Kalfa: Betriebssysteme
2. Auflage, Berlin: Akademie Verlag, 1990
- [kml] Dr. Elke Wällnitz: Der plattformunabhängige KML-Editor als Werkzeug zur Entwicklung von E-Learning-Modulen auf der Basis von XML
- [locc] Philip Johnson, University of Hawaii, LOCC developer distribution
<http://csdl.ics.hawaii.edu/Tools/LOCC/>
- [mml] Mathematical Markup Language (MathML) Version 2.0
W3C Recommendation 21 February 2001
<http://www.w3.org/TR/2001/REC-MathML2-20010221/>
- [ml3xsd] XSD-Schemata von <ML>³
<http://www.ml-3.org/>
<http://www.ml-3.org/ML3/1.2/Didactic>,
<http://www.ml-3.org/ML3/1.2/Content>

- [wwr] Prof. Dr.-Ing. habil. Tavangarian, Djamshid: Wissenswerkstatt
Rechensysteme
Ein bundesweites Vorhaben zur Erstellung eines Baukastensystems von
multimedialen Lehr- und Lern-Modulen im Bereich Technische Informatik
<http://www.wwr-project.de>
- [shtml] Stefan Münz: Selfhtml Version 8.0 vom 27.10.2001
<http://selfaktuell.teamone.de/>
- [xhtml] XHTML™ 1.0 The Extensible HyperText Markup Language (Second
Edition) - A Reformulation of HTML 4 in XML 1.0
W3C Recommendation 26 January 2000, revised 1 August 2002
<http://www.w3.org/TR/xhtml1/>
- [xml] Extensible Markup Language (XML) 1.0 (Second Edition)
W3C Recommendation 6 October 2000
<http://www.w3.org/TR/2000/REC-xml-20001006>
- [xsd] XML Schema Part 1: Structures
W3C Recommendation 2 May 2001
<http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
- [xslt] Extensible Style Language (XSL)
<http://www.w3.org/Style/XSL/>

12 Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 1: logische und didaktische Struktur von $\langle ML \rangle^3$ [kml]..... | 14 |
| Abbildung 2: das dreidimensionale Modell von $\langle ML \rangle^3$ | 15 |
| Abbildung 3: UML-Anwendungsfalldiagramm | 17 |
| Abbildung 4: Fensterskizze | 18 |
| Abbildung 5: Screenshot des X2X-Editors in der Windows-Version..... | 23 |
| Abbildung 6: Screenshot von Microsoft Word | 25 |
| Abbildung 7: Schichtenarchitektur des KML-Editors..... | 30 |
| Abbildung 8: JAXP-SAX-Parser im KML-Editor | 32 |
| Abbildung 9: JAXP-XSLT-Transformation im KML-Editor | 32 |
| Abbildung 10: zweistufiger Hash des XMLTagSet | 34 |
| Abbildung 11: Klassendiagramm der XMLTags | 35 |
| Abbildung 12: Klassendiagramm der Internationalisierungs-Klassen | 36 |
| Abbildung 13: Klassendiagramm der Plugin-API..... | 37 |
| Abbildung 14: Struktur der Schicht 2 des Editors..... | 38 |
| Abbildung 15: <i>Model</i> -Teil des Editor-Kerns..... | 39 |
| Abbildung 16: <i>Model-View-Controller</i> Architektur des Editor-Kerns | 41 |
| Abbildung 17: Klassendiagramm der Transformationsunterstützung..... | 42 |
| Abbildung 18: Klassendiagramm der Sprachdefinition | 43 |
| Abbildung 19: Klassendiagramm des Hilfesystems..... | 44 |
| Abbildung 20: Struktur der Schicht 3 des Editors..... | 45 |
| Abbildung 21: Screenshot einer Webpräsentation | 56 |
| Abbildung 22: Screenshot eines Skripts..... | 57 |
| Abbildung 23: Screenshot der Präsentation für den Präsenzunterricht..... | 58 |
| Abbildung 24: Screenshot der HTML-Darstellung..... | 59 |

13 Tabellenverzeichnis

| | |
|---------------------------------------|----|
| Tabelle 1: Entwicklungswerkzeuge..... | 27 |
|---------------------------------------|----|

14 Quellcodeverzeichnis

| | |
|---|----|
| Listing 1: Instantiierung von I18N..... | 46 |
| Listing 2: Erzeugung eines Elementtyp-spezifischen XMLAttributeSet..... | 47 |
| Listing 3: Erzeugung eines XMLTagSets | 48 |
| Listing 4: Erzeugung einer XMLViewFactory..... | 49 |
| Listing 5: Erzeugung eines XMLActionManagers | 50 |
| Listing 6: Erzeugung einer IXMLLanguageDefinition-Instanz | 51 |
| Listing 7: Erzeugung des Kompositionstreibers..... | 52 |
| Listing 8: Erzeugung einer Instanz von DefaultHelpSupport..... | 53 |
| Listing 9: Deutsche Version einer Hilfe-Ressource (help_de.properties)..... | 53 |
| Listing 10: Englische Version einer Hilfe-Ressource (help_en.properties)..... | 53 |
| Listing 11: Erzeugung des Plugins | 54 |
| Listing 12: Erzeugung des Transformers..... | 55 |

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Fremde Hilfe verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Als Hilfsmittel ist besonders der Java-Quelltext des Paketes `javax.swing.text` zu nennen [jsrc], der mir bei der Entwicklung des Editor-Kerns eine große Orientierungshilfe war und aus dem für diesen teilweise kleinere Passagen übernommen werden konnten.

Chemnitz, den 05.04.2005

Thomas Weise

```

@mastersthesiss{W2005MT,
author      = {Thomas Weise},
title       = {Entwicklung eines WYSIWYG Editors für das Erstellen von
               Lehrmaterial im XML Format},
editor      = {Prof. Dr. Ing. habil. Winfried Kalf and Dr. Elke Wehlnitz},
year        = {2005},
month       = {Apr},
location    = {Chemnitz University of Technology},
organization = {Chemnitz University of Technology},
school      = {Chemnitz University of Technology},
note        = {My diploma thesis (equivalent to master thesis) is about creating a
               WYSIWYS editor for teaching material which stores its stuff in
               XML-format and can convert it to XHTML and pdf automatically. The
               thesis was rated with 1.0 (=A).\\
               KML-Editor Home\\
               The work is online available at
               http://www.it-weise.de/documents/index.html#W2005MT.\\
               The diploma thesis can be downloaded at
               http://www.it-weise.de/documents/files/W2005MT.pdf.\\
               The presentations can be downloaded at
               http://www.it-weise.de/documents/files/W2005MT\_slides.pdf.\\
               The software can be downloaded at
               http://www.it-weise.de/documents/files/W2005MT\_software.zip.\\
               Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/.},
copyright   = {unrestricted},
abstract     = {Die vorliegende Arbeit beschreibt den Entwurf, die Entwicklung und die
               Anpassung eines XML-WYSIWYG-Editors für die Verwendung in der
               Lehre. Der praktische Teil der Arbeit hat einen Editor zum Ergebnis,
               welcher den XML-Dialekt KML implementiert, mit dessen Hilfe
               &gt;3-formattiertes Lehrmaterial erstellt werden kann.
               Der Entwurf des Editors ist jedoch besonders auf Erweiterbarkeit und
               Wartbarkeit gewichtet. Dadurch wird der erstellte Editor für
               beliebige XML-Dialekte mit geringem Aufwand anpassbar.},
keywords     = {KML, Lehrmaterial, WYSIWYG Editor, Java, ML3, XML, Teaching Material,
               WWW, Knowledge Markup Language},
language     = {de},
url          = {http://www.it-weise.de/documents/index.html#W2005MT}
}

```