Entwicklung eines technischen Rahmenwerks für standardkompatible Lernobjekte

Universität Paderborn, Fakultät EIM, GET-LAB
Michael Bungenstock, Andreas Baudry, Bärbel Mertsching

(bungenstock|baudry|mertsching)@upb.de

Abstract: Dieses Paper soll die Lücke zwischen den Theorien für Lernobjekte und den technischen Realisierungen schließen. Auf Basis gängiger Standards, wie z.B. IMS Content Packaging und SCORM, wird die Entwicklung einer Programmierschnittstelle (API) für Lernobjekte beschrieben. Aus den XML-Bindings der Standards wird ein objektorientiertes Binding für moderne Programmiersprachen abgeleitet. Anhand einer eigenen Implementation in Java wird die Funktionalität dieses Ansatzes demonstriert.

1 Einleitung

Das *Lernobjekt* ist ein essentielles Konzept für E-Learning-Inhalte. Es definiert Größe, Struktur, Format und weitere Eigenschaften, die für einen reibungslosen Einsatz notwendig sind. Autorenwerkzeuge, Anzeigeprogramme, Repositories, etc. müssen sich auf verbindliche Definitionen für Lernobjekte verlassen können. Ansonsten setzen sich proprietäre Formate durch, die zu Inkompatibilitäten führen.

Am Anfang eines Projekts müssen sich die Entwickler/-innen entscheiden, welche Kodierungen das zukünftige System unterstützen soll. Diese Entscheidung wird durch die vielen verschiedenen Theorien und Standards nicht einfacher. Hinzu kommt noch die Diskrepanz zwischen den theoretischen Erwägungen und der tatsächlichen Praxis. Ein universelles technisches Konzept lässt sich aus den Theorien nicht ableiten. Die Standards für Lernobjekte wiederum haben meist keine theoretischen Hintergrund und zeichnen sich durch eine technische Detailfülle aus. Sie sind eine gute Grundlage für Dateiformate und Protokolle, aber Aufbau, Umgang und Darstellung der Lernobjekte müssen individuell von den Entwicklern/-innen festgelegt werden. Bei einer ungünstigen GUI — Entwickler/-innen neigen dazu, technische Aspekte in den Vordergrund zu stellen — haben die späteren Anwender/-innen das Nachsehen. Aus diesem Grund bieten wir ein technisches Rahmenwerk mit zugehörigen Schnittstellen als Java-Library an, das theoretisch fundiert ist und die wichtigen Standardformate unterstützt.

Durch diesen Ansatz kann sich die Entwicklung erheblich verbessern. Alle technischen Spezifikationen, sei es nun für Lernobjekte oder Metadaten, bieten lediglich ein *XML-Binding* an. Dieses Binding beschreibt, wie die Datenstrukturen in XML kodiert werden.

Für APIs fehlen meist solche Beschreibungen, und durch das geringe Angebot an fertigen Implementationen müssen die Spezifikationen eigentlich immer neu nachimplementiert werden. Es müssen somit Zeit und Geld für unnötige Arbeiten investiert werden. Ein guter, aber nicht perfekter Ansatz für dieses Problem ist die *Open Knowledge Initiative* (O.K.I.) [TSM02]. Hierbei handelt es sich um eine offene und erweiterbare Architektur für E-Learning-Technologien. Besonders interessant sind die detaillierten Spezifikationen der Schnittstellen, die jeweils den Zugriff auf bestimmte Dienste erlauben, wie z.B. für die Benutzerverwaltung oder Benachrichtigungen. Doch obwohl auch Personen von Organisationen mit Bezug zu Lernobjekten beteiligt sind, wie z.B. vom *Instructional Management Systems Project* (IMS), beinhaltet O.K.I. keine Schnittstelle für Lernobjekte.

In den folgenden Abschnitten wird unser Ansatz für ein technisches Lernobjekt-Rahmenwerk beschrieben. Zuerst wird ein Überblick der Lernobjekt-Theorien gegeben und eine Verbindung zu den Standards hergestellt. Anschließend wird anhand der XML-Bindings ein objektorientiertes Binding (OO-Binding) abgeleitet, das eine einfache Implementation mit modernen Programmiersprachen erlaubt. Einige Beispiele am Ende zeigen den Einsatz unser eigenen Java-Implementation. Eine Bewertung inklusive Ausblick auf zukünftige Arbeiten schließen dieses Paper ab.

Alle Ergebnisse dieser Arbeit basieren auf den Resultaten des Projekts mαth-kit[1] [UBB+04, UOM02, SU02].

2 Theorie

Es gibt eine Reihe verschiedener Theorien zu den Lernobjekten und es ist schwer, die geeignete zu finden. Hier werden nun die vier vorgestellt, die als allgemein bekannt und akzeptiert gelten. Einige allgemeinere Anforderungen an Lernobjekte werden in [AW69] und [GNR02] aufgelistet.

Die Firma Cisco Systems definiert in ihrem Strategie-Paper [Cis99] eine Definition für Lernobjekte, die auf den *Reusable Information Objects* (RIO) basiert. Ein RIO ist eine kleine wiederverwendbare Informationseinheit, die in verschiedenen Kontexten eingesetzt werden kann. Das eingesetzte Medium und dergleichen spielen hierbei keine Rolle. Jedes RIO kann beliebig mit anderen kombiniert werden und zu höheren Strukturen, den *Reusable Learning Objects* (RLO), zusammengesetzt werden.

Cisco unterscheidet hierbei die Vorteile der RIOs für Autoren/-innen und Studierende. Die Autoren/-innen profitieren von der Wiederverwendung und den flexiblen Kombinationsmöglichkeiten. Hingegen erhalten die Studierenden ein konsistentes Erscheinungsbild und neue technische Möglichkeiten, wie z.B. *Personalized Learning* [Ma00], bei dem individuelle Lernpfade angeboten werden.

Wayne Hodgins stellt in seinem White-Paper [Ho00] Lernobjekte als kleine und wiederverwendbare Informationseinheiten vor. Hierbei handelt es sich um das Wissen, das aus den Köpfen von Experten/-innen „extrahiert“ wurde. Diese Umwandlung von Wissen und

[1] Gefördert vom Bundesministerium für Bildung und Forschung (BMBF), Projekt Math-Kit (08NM084)

Gedanken zu Lernobjekten, muss sehr sorgfältig geschehen, um sie in Konversation, Abbildungen, Texten, Modellen und Simulationen mit den Lernenden teilen zu können. Die resultierenden Lernobjekte sollten aber nicht zu groß geraten:

> „Size matters: Smaller is better." [Ho00, p. 27]

Um sie später leichter aufzufinden, sollten Lernobjekte mit Metadaten versehen werden. Hodgins schlägt hierfür den Einsatz anerkannter Standards vor. Gleiches gilt auch für die Verbindungsmechanismen zwischen den Lernobjekten. Letztendlich unterscheidet sich Hodgins Definition nicht sehr von der Cisco RLO/RIO Struktur und es verwundert nicht, wenn er sie zur Umsetzung empfiehlt.

Eine profunde Idee von Hodgins ist die Einführung der LEGO™ Metapher. LEGO Steine sind sehr flexibel und können zu fast allen Formen, Größen und Funktionen zusammengesetzt werden [Ho02]. Alle Lernobjekte haben die gleichen Eigenschaften und durch die Analogie ist es für die Anwender/-innen ein leichtes, sich in die Thematik Lernobjekte einzuarbeiten.

David A. Wiley ist überzeugt, dass das Internet die Kommunikation zwischen den Menschen beeinflusst und zwangsläufig auch auf das Lernen einwirkt [Wi02]. Aus diesem Grund müssen heutige Lernmaterialien auf die kommenden Anforderungen angepasst werden. Die führende Technik sind hierbei die Lernobjekte, weil sie wiederverwendbar, generisch, anpassbar und skalierbar sind. Zum endgültigen Durchbruch kann es aber nur dann kommen, wenn sich bindende Standards durchsetzen. Ansonsten können Universitäten und Unternehmen keine adäquate Investitionssicherheit erlangen. Der Ursprung von Wileys Definition für Lernobjekte beruht auf dem IEEE LOM Standard:

> „For this standard, a learning object is defined as any entity — digital or non-digital — that may be used for learning, education or training." [IEE02, p. 5]

Er kritisiert aber das „non-digital", weil es sich nicht mit den Anforderungen des Internets deckt. Das „may be used" sei ihm zu allgemeingültig und würde auch nicht wiederverwendbare Materialien einschließen. Sein neue Formulierung lautet nun:

> „... will define a learning object as any digital resource that can be used to support learning." [Wi02, p. 7]

Die LEGO Metapher von Wiley lehnt er kategorisch ab, da sie zu falschen Assoziationen verleite. Stattdessen schlägt er das Atom als präzisere Alternative vor [Wi99]. Das Hauptproblem sei die einfache Beschaffenheit der Bausteine. Sie könnten zu beliebigen Strukturen zusammengesetzt werden, ob diese nun Sinn machen oder nicht. Zudem seien LEGO Bausteine so einfach zu handhaben, dass jedes Kind mit ihnen spielen kann. All diese Eigenschaften übertragen auf Lernobjekte, könnten nicht zu pädagogisch sinnvollen Lerneinheiten führen. Hingegen fordere die Atom Metapher einen genaueren Umgang mit Lernmaterialien. Atome könnten nur mit bestimmten Atomen kombiniert werden, die Strukturen seien eingeschränkt und es bedürfe schon einiger Übung, um Atome zu verbinden. Nur durch solch restriktive Eigenschaften lassen sich hochqualitative Lernobjekte

garantieren. Die genauen Details zur Granularität und dem Sequenzierung finden sich in [Wi00, WRG00].

Stephen Downes geht über eine funktionale Definition an das Thema Lernobjekte heran [Do02]. Er ist weder gegen die LEGO noch die Atom Metapher, aber hegt Zweifel an deren Erfolgsaussichten [Do00b]. Anstatt zu definieren, was Lernobjekte sind, zieht es Downes vor, die zu lösenden Probleme auszumachen.

Lernmaterialien sollten nur einmal entwickelt werden und dann in möglichst vielen Kursen zum Einsatz kommen. Lernobjekte sollen also helfen, die Entwicklungskosten durch Wiederverwendung zu minimieren [Do00a]. Kleine Module in digitaler Form können zu höheren Strukturen zusammengesetzt und wieder in ihre Bestandteile zerlegt werden. Volle Flexibilität kann nur erreicht werden, wenn Lernobjekte interoperabel sind, die Autoren/-innen also nicht auf Formate und Layouts achten müssen. Zuletzt müssen sie schnell auffindbar sein. Für jede Person mit durchschnittlichen Computer-Wissen muss es möglich sein, die gewünschten Inhalte in akzeptabler Zeit zu besorgen. Downes fasst seine funktionale Definition wie folgt zusammen:

> „In conclusion, learning objects are digital materials used to create online courses where these materials are sharable, modular, interoperable and discoverable.“ [Do02]

Doch was ist nun die ideale Definition für Lernobjekte? Wir haben uns aus mehreren Gründen für die LEGO Metapher von Hodgins entschieden [BBM02b, BBM04b]. Die Begriffe Baustein und Modell (auch Kurs genannt) spielen eine zentrale Rolle bei unseren Konzepten. Hodgins Konzept ist gut zu implementieren und sehr dicht an den Standards, die im nächsten Abschnitt beschrieben werden. Eine allgemeine API für Lernobjekte sollte nicht zu viele konzeptionelle Einschränkungen machen, weshalb Wileys Definition nicht unsere erste Wahl ist. Zudem sind die Definition von Hodgins, Cisco und Downes sehr ähnlich, sodass ein Umsteigen zwischen den Konzepten ohne Aufwand möglich ist.

3 Standards

Der Abschnitt Theorie hat deutlich gezeigt, dass der Einsatz von Standards ein Muss ist. Aus diesem Grund mussten wir uns für einen oder zwei Standards entscheiden, die unsere API und Library unterstützten soll. Bei unseren Untersuchungen haben sich *IMS Content Packaging* (CP) und *ADL Sharable Content Object Reference Model* (SCORM) als die wichtigsten herausgestellt. Beide Spezifikationen bieten ein *XML-Binding* an, wodurch der Entwurf einfacher ist. Entsprechend dem XML-Binding haben wir ein objektorientiertes Modell, das OO-Binding, entwickelt. Hierdurch bleibt die Nähe zum Standard gewahrt und die Umwandlung zwischen den Bindings wird vereinfacht.

SCORM definiert die kleinste Einheit als *Asset* und IMS CP als *Physical File*. Es handelt sich um atomare Einheiten, wie z.B. Bilder, Animationen, Sounds und Texte. Zusammenhängende Assets werden zu einem *Sharable Content Object* (SCO) zusammengesetzt und bei IMS CP zu einem *Package (Interchange File)*. Da beide Spezifikation sehr ähnlich

sind, SCORM basiert auf IMS CP, werden hier beide Formate als Paket bezeichnet. Physikalisch handelt es sich um *zip*, *jar* oder *cab* Dateien. Als Beispiel soll eine HTML-Datei dienen, die vielleicht zwei Bilder und eine Macromedia Flash Animation referenziert. Diese Dateien gehören zusammen und bilden ein Paket.

Komplexere Pakete brauchen zusätzliche Strukturinformationen. Beispielsweise besteht ein Kurs aus mehreren Lernobjekten, die zu einem Paket mit einer festgelegten Ordnung zusammengesetzt wurden. Diese Strukturdaten werden in einem *Manifest* gespeichert. Normalerweise ist es eine extra XML-Datei innerhalb des Paketes, die *Ressourcen* (Resources), *Strukturen* (organizations) und Metadaten (meta-data) deklariert. Hauptaufgabe einer Library für Lernobjekte ist die Manipulation von Manifesten. Abbildung 1 illustriert daher für die weiteren Ausführungen den schematischen Aufbau eines Manifests.

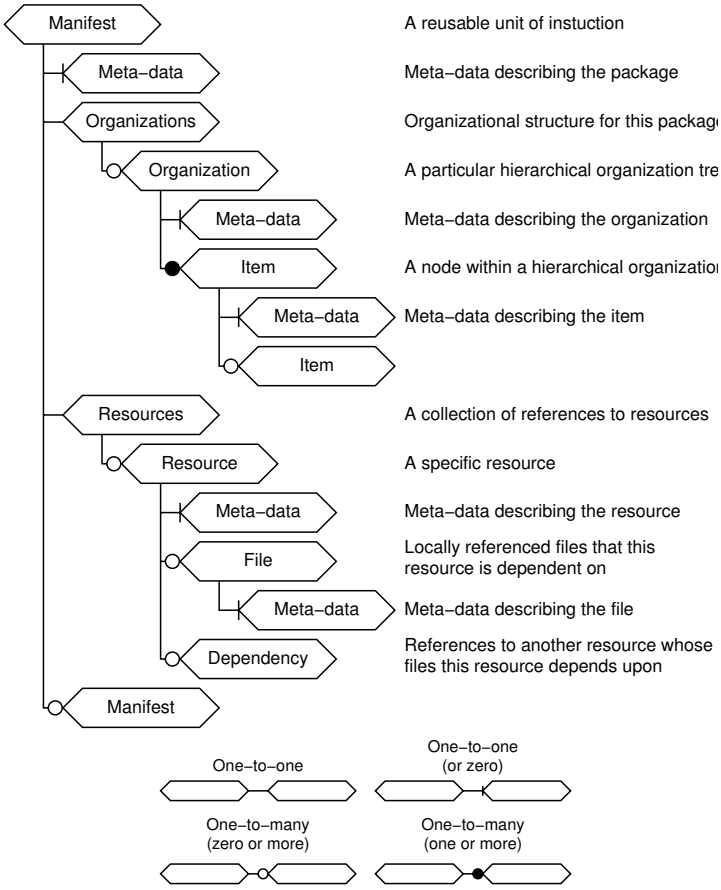


Abbildung 1: Aufbau eines Manifests aus [IM03]

Ein Manifest kann mehrere Strukturen haben, um verschiedene Lernpfade zu ermöglichen. Jede Struktur enthält verschachtelte *Einträge* (items), die einen Titel und eine Referenz auf

eine Ressource haben. Bei einer Ressource handelt es sich um eine Sammlung zusammengehöriger Dateien. Komplexere Strukturen können durch *Substrukturen* (sub-manifests) aufgebaut werden.

4 OO-Binding

Das XML-Binding definiert 10 Elemente und ist Ausgangspunkt für unser OO-Binding. Es ist ein Modell für objektorientierte Programmiersprachen, bestehend aus Klassen und Schnittstellen. Diese API erlaubt Programmen den einheitlichen Zugriff auf Lernobjekte, die als Pakete kodiert sind. Es handelt sich hierbei nicht um eine fertige Library. Schnittstelle und Implementation haben wir aus zwei Gründen bewusst getrennt. Erstens können Anwendungen verschiedene Libraries über die gleiche Schnittstelle nutzen. Fehlerbehebungen, Verbesserungen und Anpassungen an einer Library können ohne Änderungen an den Anwendungen durchgeführt werden. Zweitens gibt es keine Bindung an eine bestimmte Programmiersprache.

Das OO-Binding soll über die Eigenschaften der XML-Elemente abgeleitet werden. Hierzu werden sie in Tabelle 1 aufgelistet. Gemeinsame Eigenschaften werden dann in Klassen zusammengefasst.

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

●: ja ○: nein ◐: beides

Tabelle 1: Gemeinsame Eigenschaften der Manifest-Elemente

Zuerst sollen die unwichtigen Elemente gestrichen werde. Das Element `title` ist mehr ein Attribut als ein Element. Es kann durch eine einfache Zeichenkette dargestellt werden und wird als Bestandteil der Klasse *Item* aufgefasst. Ähnlich wird mit den Elementen `organizations` und `resources` verfahren. Sie lassen sich auf einfache Listen reduzieren, sodass sie Bestandteil der Klasse *Manifest* werden.

Die übrigen Element haben alle zwei Eigenschaften gemeinsam. Jedes kann als Kind-Element auftreten und kann Änderungen erfahren. Somit lässt sich eine gemeinsame Basisklasse für alle Klassen des Manifests festlegen. Ihr Name ist *HierarchicalElement* und verwaltet die Beziehungen zu Elternelementen sowie die Benachrichtigung bei Änderungen. Diese Benachrichtigungen sind besonders wichtig für den Einsatz von Entwurfsmustern [GHJV95], wie z.B. dem *Model/View/Controller* (MVC) [KP88]. Alle modernen GUIs nutzen dieses Muster für die Darstellung von Daten.

Eine weitere Eigenschaft sind die Metadaten, die in der Klasse *MDElement* verwaltet werden. Die Strukturen von Metadaten-Standards, wie z.B. LOM, können recht komplex sein, weshalb sie nicht durch eine einzelne Klasse repräsentiert werden können. Deswegen steuert *MDElement* vielmehr den Zugriff auf spezielle Metadaten-Klassen und bietet keine eigenen Attribute an. Sie erbt freilich direkt von *HierarchicalElement*.

Einige Elemente mit Metadaten haben zusätzlich einen Bezeichner, über den sie von den Elementen `item` und `dependency` referenziert werden können. Jeder Bezeichner muss innerhalb des Manifests eindeutig sein. Änderungen an einem Manifest, wie z.B. durch Hinzufügen einer neuen Ressource oder eines anderen Submanifests, können jedoch zu Konflikten bei der Bezeichnervergabe führen. Dies zu verhindern, ist Aufgabe der Klasse *IDElement*. Sie verwaltet alle Bezeichner, erzeugte neue eindeutige und löst Konflikte auf.

Nun sind alle Superklassen für die Elemente des Manifests definiert. Abbildung 2 zeigt das resultierende Klassendiagramm in UML-Notation [Ob03]. Jede Klasse hat ihre Methoden, um den Zugriff auf Attribute und Kind-Elemente zu gestatten.

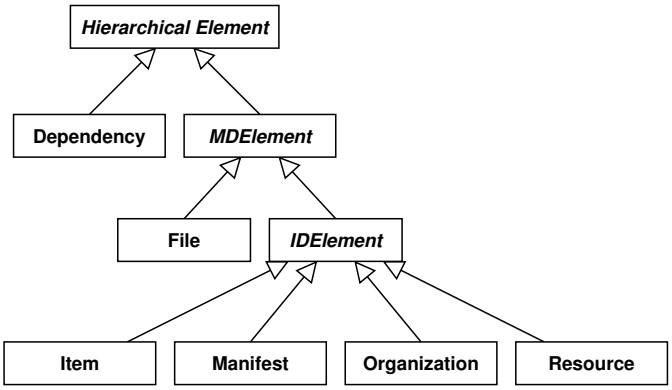


Abbildung 2: Manifest-Klassen in UML-Notation

Die resultierende API umfasst mehr als die Element-Klassen. Es gibt zwei Schnittstellen für das Schreiben und Einlesen von Manifesten. Unterschiedliche Implementationen können spezielle Formate unterstützen oder Manifeste direkt in Datenbanken und Repositories speichern. Andere Klassen unterstützen die Erstellung neuer Pakete und die Fehlerbehandlung.

5 Formale Lernobjekte

Lernobjekte bestehen in der Regel aus einer Vielzahl unterschiedlicher Dateien, den so genannten Assets, die den Inhalt der Lernobjekte darstellen. Dieses können z.B. HTML-Seiten, Powerpoint-Präsentationen oder Videos sein. Die in diesem Artikel vorgestellten Standards beschreiben zwar die Struktur der Inhalte, jedoch machen sie keine Aussage über den Aufbau der Dokumente, die in den Paketen gekapselt sind. Aus diesem Grund eignet sich das IMS Content Package und der SCORM Standard vielmehr zum Übersenden von Inhalten, als für deren Aggregation. Zwar können die Manifeste zusammengeführt und zu neuen Manifesten zusammengesetzt werden, aber deren Dokumente lassen sich mit den vorhandenen Mechanismen nicht mit einbeziehen. Aus diesem Grund bieten wir eine Erweiterung für Lernobjekte an, die ergänzend zu der Content Package Schnittstelle ein Modell zur formalen Beschreibung von Dokumenten innerhalb der Pakete anbietet. Mit diesem Modell ist es für Autoren möglich andere, ebenfalls formal beschriebene Lernobjekte mit den eigenen zu kombinieren und sie so zu umfangreicheren Kursen zusammenzuschnüren. Für die Beschreibung der Inhalte verwenden wir die Sprache XML (eXtendable Markup Language) [Ha02], weil sie die generelle Gestaltung strukturierter Dokumente unterstützt. Ähnliche Ansätze werden auch in [FSD02] und [KM04] beschrieben. Im Gegensatz zu dem in diesem Artikel beschriebenen Modell, liegt deren Schwerpunkt jedoch auf der pädagogischen Vernetzung von Lerninhalten, sowie deren Adaption für unterschiedliche Lernsituationen.

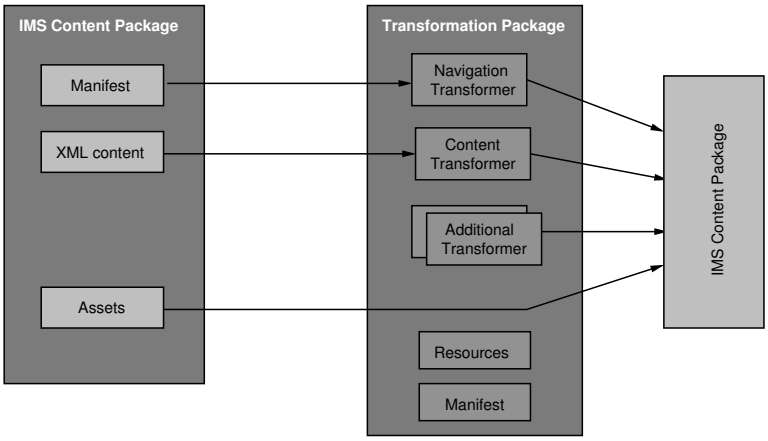


Abbildung 3: Aufbau eines Transformationspakets

Für den endgültigen Einsatz der formal beschriebenen Lernobjekte fehlt allerdings noch eine Abbildungsvorschrift auf ein von Lernenden lesbares Ausgabeformat. Zu diesem Zweck wird das in Abbildung 3 illustrierte *Transformation Package* [BBM03a] eingesetzt. Es enthält alle notwendigen Regeln und Informationen, die für eine Überführung der Lernobjekte in ein durch das Transformationspaket bestimmtes Format benötigt werden. Hierbei existieren Abbildungsregeln für den Inhalt der Lernobjekte sowie für die Struktur des Kurses, die durch das Manifest des Kurspakets festgelegt ist. Die Abbildung der Kur-

sstruktur auf eine Navigation ist natürlich nur dann erforderlich, wenn der Kurs nicht in ein Learning Management System geladen wird, welches die IMS und SCORM Standards unterstützt. Zudem können Transformationspakete weitere Transformatoren enthalten, die Aufgaben, wie z.B. die Erzeugung eines Framesets für eine Webpräsentation übernehmen.

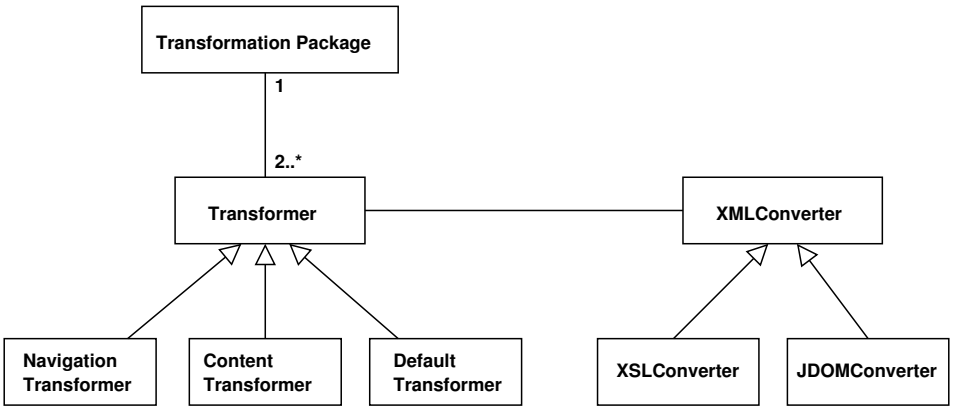


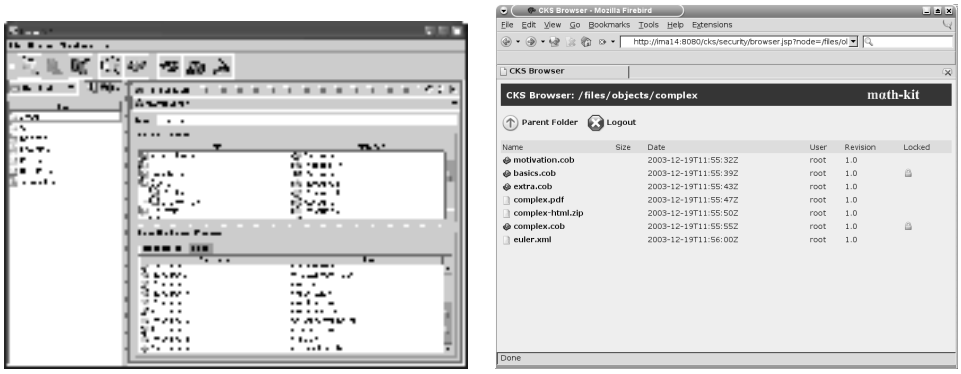
Abbildung 4: UML Diagramm des Transformation Packages

Zusätzlich zu dem in Abschnitt 4 beschriebenen OO-Binding existiert eine API, mit der Transformationspakete bearbeitet und verwaltet werden können. Wie in Abbildung 4 zu sehen ist, verwendet jedes Exemplar der `TransformationPackage` Klasse in Abhängigkeit vom Ausgabeformat eine variable Anzahl von Transformatoren. Das Exemplar der `Content Transformer` Klasse ist dabei für jeden Übersetzungsprozess notwendig, da es für die Abbildung der XML Dokumente verantwortlich ist. Jeder Transformator ist in der Lage, ein XML File zu übersetzen und das Ergebnis in eine Datei zu schreiben bzw. als DOM-Objekt anzubieten. Für die einzelnen Transformatoren unterstützt die API entweder die skriptorientierte Konvertierung mit XSL [Co99] oder eine programmgesteuerte Konvertierung auf Basis von JDOM. Beide Klassen implementieren die Schnittstelle `XMLConverter`.

6 Beispiele

Die vorgestellte API wurde in zwei Projekten eingesetzt. Das Autorenwerkzeug *Lyssa* basiert auf der Baukastenmetapher [BBM03a, BBM03b] und setzt die API beim Umgang mit Paketen ein. Hierzu bietet es eine graphische Benutzerschnitstelle, mit der Lernobjekte unterschiedlicher Granularität erzeugt werden können. Die GUI bietet unterschiedliche Abstraktionsebenen, Drag'n'Drop und eine nahtlose Integration der erzeugten Daten in ein LMS [BBM03c]. Lyssa enthält weder einen eingebauten Editor noch Werkzeuge zum Erzeugen von Animationen, unterstützt jedoch die Verknüpfung beliebiger Werkzeuge mit den Dateien in den Paketen. Abbildung 5(a) zeigt einen Screenshot von Lyssa.

Ebenso wie beim Autorenwerkzeug Lyssa wird die API im Construction Kit Server (CKS)



(a) Lyssa (b) Construction Kit Server

Abbildung 5: Zwei Beispiele

[BBM04a, BBM02a] eingesetzt. Der CKS ist eine das Autorensystem ergänzende Plattform, welche die Teamarbeit unterstützt. Sie bietet unterschiedliche Mechanismen zur Koordination und Synchronisation des Arbeitsprozesses an. Dazu gehören das *Locken* sowie die *Versionierung* von Paketen. Wenn ein Paket bearbeitet werden soll, garantiert ein Datei-Lock einen exklusiven Zugriff. Ein solcher gegenseitiger Ausschluss verhindert die parallele Modifikation von Paketen. Wird hingegen eine Version für ein Paket angelegt, kann diese zu einem späteren Zeitpunkt wieder hergestellt werden, was freilich die Entwicklung von Paketen vereinfacht. Abbildung 5(b) zeigt exemplarisch einen Screenshot des CKS.

7 Bewertung und Ausblick

In diesem Paper wurde der Entwurf eines allgemeinen Rahmenwerks für Lernobjekte vorgestellt. Die resultierende Implementierung ermöglicht den Entwicklern/-innen, sich auf die wesentlichen Teile ihrer Anwendung zu konzentrieren. Technische Details werden verdeckt und viele hilfreiche Funktionen gestatten die Entwicklung anspruchsvoller Systeme. Optimal wäre es freilich, wenn ein wie hier vorgestelltes OO-Binding in die Standards mit aufgenommen wird. Hierdurch könnten sich die Entwickler/-innen auf eine Schnittstelle verlassen und bei Bedarf die Implementation austauschen oder anpassen.

Einige Aspekte konnten hier nicht genauer erläutert werden. Eine exakte Schnittstellendefinition mit all ihren Methoden hätte den Rahmen dieses Textes gesprengt. Stattdessen wurde die logische Struktur aufgezeigt, um die Idee zu vermitteln. Gleiches gilt auch für die Metadaten API, die mit über 50 Klassen umfangreicher ausfällt.

Bestimmte Aufgaben sind auch noch zu erledigen. Die vollständige Dokumentation der API ist noch nicht abgeschlossen und aussagekräftige Beispiele könnten die Einarbei-

tungszeit reduzieren. Trotzdem haben API und Java-Library einen Stand erreicht, der einen Einsatz in E-Learning-Anwendungen erlaubt.

Literatur

[AW69] Atkinson, R. C. und Wilson, H. A. (Hrsg.): *Computer-assisted instruction: a book of readings*. Academic Press. 1969.

[BBM02a] Baudry, A., Bungenstock, M., und Mertsching, B.: Architecture of an e-learning system with embedded authoring support. In: *E-LEARN 2002–World Conference on Educational in Corporate, Government, Healthcare, & Higher Education*. S. 110–116. 2002.

[BBM02b] Bungenstock, M., Baudry, A., und Mertsching, B.: The construction kit metaphor for a software engineering design of an e-learning system. In: Barker, P. und Rebelsky, S. (Hrsg.), *Proc. ED-MEDIA 2002–World Conference on Educational Multimedia, Hypermedia & Telecommunications*. S. 216–217. 2002.

[BBM03a] Baudry, A., Bungenstock, M., und Mertsching, B.: Nyx — a tool for generating standard compatible e-learning courses with consistent and adaptable presentation. In: *The IASTED International Conference on Computers and Advanced Technology in Education (CATE 2003)*. 2003.

[BBM03b] Bungenstock, M., Baudry, A., und Mertsching, B.: Data exchange between lyssa and learning management systems. In: *E-Learn 2003–World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education*. S. 31–34. 2003.

[BBM03c] Bungenstock, M., Baudry, A., und Mertsching, B.: Lyssa — an authoring system complying with e-learning standards. In: *ED-MEDIA 2003–World Conference on Educational Multimedia, Hypermedia & Telecommunications*. S. 502–508. 2003.

[BBM04a] Baudry, A., Bungenstock, M., und Mertsching, B.: Administration and development of modular learning units with the construction kit server. In: *Accepted for: ED-MEDIA 2004–World Conference on Educational Multimedia, Hypermedia & Telecommunications*. 2004.

[BBM04b] Baudry, A., Bungenstock, M., und Mertsching, B.: Reusing document formats for modular course development. In: *IASTED International Conference on WEB-BASED EDUCATION (WBE 2004)*. S. 535–537. 2004.

[Cis99] Cisco Systems: *Reusable Information Object Strategy*. June 1999. Version 3.0.

[Co99] Consortium, W. W. W. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt. viewed 22.11.2002 1999.

[Do00a] Downes, S. Learning objects. Presented at Leaders in Learning 2000. May 2000.

[Do00b] Downes, S.: Nine rules for good technology. *The Technology Source*. March 2000.

[Do02] Downes, S. The learning object economy. Published by Contact North. October 2002.

[FSD02] Freitag, B., Süß, C., und Dziarstek, C.: LMML-Eine XML-Sprachfamilie für eLearning Content. In: Schubert, S. E., Reusch, B., und Jesse, N. (Hrsg.), *Informatik bewegt: Informatik 2002 - 32. Jahrestagung der GI*. 2002. LNI 19 GI.

[GHJV95] Gamma, E., Helm, R., Johnson, R., und Vlissides, J.: *Design Patterns: elements of reusable object-oriented software*. Addison Wesley. 1995.

[GNR02] Gibbons, A. S., Nelson, J., und Richards, R.: *The Instructional Use of Learning Objects*. chapter The Nature and Origin of Instructional Objects. AIT/AECT. 2002.

[Ha02] Harold, E. R.: *Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX*. Number 1. Addison-Wesley Pub Co. 2002. ISBN: 0201771861.

[Ho00] Hodgins, W. Into the future - a vision paper. erhältlich bei Commission on Technology & Adult Learning of the American Society for Training & Development (ASTD) und National Governors' Association (NGA). February 2000.

[Ho02] Hodgins, W.: The future of learning objects. In: *Proc. of the 2002 eTEE Conference*. S. 76–82. August 2002.

[IEE02] IEEE P1484.12: *Draft Standard for Learning Object Metadata*. 2002.

[IM03] IMS Global Learning Consortium, Inc. Ims content packaging information model. 2003.

[KM04] Koper, R. und Manderveld, J.: Educational modelling language: modelling reusable, interoperable, rich and personalised units of learning. *British Journal of Educational Technology*. 2004.

[KP88] Krasner, G. E. und Pope, S. T.: A cookbook for using the model view controller user infertface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*. 1(3):26–49. 1988.

[Ma00] Martinez, M.: *The Instructional Use of Learning Objects*. chapter Designing Learning Objects to Personalize Learning. AIT/AECT. 2000.

[Ob03] Object Management Group. OMG unified modeling language specification. 2003. Version 1.3.

[SU02] Schiller, S. und Unger, L.: Math-kit: a multimedia project for learning and teaching mathematics. In: *Proceedings 10th Meeting of European Women in Mathematics*. S. 383–386. 2002.

[TSM02] Thorne, S., Shubert, C., und Merriman, J.: OKI Architecture Overview. Draft document. March 2002.

[UBB+04] Unger, L., Bauch, M., Baudry, A., Bungenstock, M., Mertsching, B., Oevel, G., Padberg, K., und Thiere, B.: math-kit — ein multimedialer baukasten für die mathematikausbildung im grundstudium. *Softwaretechnik-Trends*. 24(1):62–71. 2004.

[UOM02] Unger, L., Oevel, G., und Mertsching, B.: Web-based teaching and learning with math-kit. In: *Proc. 2th International Conference on the Teaching of Mathematics*. 2002.

[Wi99] Wiley, D. A. The post-lego learning object. Homepage. 1999.

[Wi00] Wiley, D. A.: *Learning object design and sequencing theory*. PhD thesis. Department of Instructional Psychology and Technology Brigham Young University. 2000.

[Wi02] Wiley, D. A.: *The Instructional Use of Learning Objects*. chapter Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy. AIT/AECT. 2002.

[WRG00] Wiley, D. A., Recker, M., und Gibbons, A. A reformulation of the issue of learning object granularity and its implications for the design of learning objects. Homepage. 2000.