

**Anbindung von Digital Libraries
in
Peer-to-Peer-Netze**

Stand: 30. März 2004

Diplomarbeit

im Studiengang Mathematik mit Studienrichtung Informatik
an der Universität Hannover

Helge Reinsch

Erstprüfer Prof. Dr. W. Nejd,
Institut für Rechnergestützte Wissensverarbeitung KBS

Zweitprüfer PD Dr. Friedrich Steimann,
Institut für Rechnergestützte Wissensverarbeitung KBS

Betreuer Wolf Siberski,
Institut für Rechnergestützte Wissensverarbeitung KBS

Hiermit erkläre ich, die vorliegende Diplomarbeit „Anbindung von Digital Libraries an Peer-to-Peer-Netze“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Helge Reinsch

Hannover, den 30. März 2004

Inhaltsverzeichnis

INHALTSVERZEICHNIS.....	II
1 EINLEITUNG.....	4
2 GRUNDLAGEN.....	5
2.1 Digitale Bibliotheken.....	5
2.2 Metadaten.....	8
2.3 Dublin Core.....	9
2.4 Datenquellen.....	11
2.4.1 Open Archive Initiative (OAI).....	11
2.4.2 DSpace.....	13
2.4.3 ILIAS.....	16
2.5 Datenformate.....	19
2.5.1 XML.....	19
2.5.2 Aufbau und Bestandteile von XML.....	21
2.5.3 RDF (Resource Description Framework).....	22
2.5.4 Beispiele.....	24
2.6 Datenverteilung.....	31
2.6.1 Peer-to-Peer-Netzwerke.....	31
2.6.2 Edutella.....	33
2.7 Datentransformation.....	42
2.7.1 XSL (eXtensible Style Language).....	42
3 KONZEPTION.....	54
3.1 OAI/DSpace-Anbindung.....	56
3.1.1 OAI Server Adapter.....	56
3.1.2 OAI Protokoll Adapter.....	59
3.2 ILIAS-Anbindung.....	60
3.2.1 XML File Adapter.....	60
4 IMPLEMENTATION.....	63
4.1 RDF Server Adapter Interface.....	63
4.2 OAI/DSpace Anbindung.....	64
4.2.1 OAI Server Adapter.....	64
4.2.2 OAI-Datenübertragungsprotokoll.....	64
4.2.3 Aufbau des OAI Protokoll Adapters.....	72
4.2.4 Übersetzung der OAI-Datensätze in RDF.....	73
4.3 ILIAS-Anbindung.....	76
4.3.1 ILIAS DTD.....	76
4.3.2 Übersetzung der ILIAS-Datensätze.....	80
5 AUSBLICK.....	84
ANHANG.....	85
A OAI-ANFRAGE-ATTRIBUTE.....	85
B XML-STYLESHEETS.....	87

B.1	OAI	87
B.2	ILIAS	88
LITERATURVERZEICHNIS.....		90
ABBILDUNGSVERZEICHNIS.....		94

1 Einleitung

Die Entwicklung digitaler Informationssysteme erreicht immer mehr auch öffentliche Einrichtungen. Bibliotheken stellen nicht mehr nur ihre Informationen in gedruckter Form zur Verfügung, sondern erweitern ihr Angebot zu immer komplexer werdenden Informationsportalen. Neben den klassischen Angeboten, wie OPACs (Open Public Access Catalogs) und internetgestützter Ausleihe, kommen elektronische Speicherung, Aufbereitung und Bereitstellung digitaler Informationsobjekte hinzu. [1] Zu den Informationsobjekten zählen dabei nicht nur vollständige Zeitschriften, sondern auch Videos, Bilder und multimediale Anwendungen.

Im Kontext von Lehre und Lernen ändert sich dadurch der Stellenwert der Bibliotheken. Umfassendere Verfügbarkeit von Informationen bezieht die Bibliotheken enger in den Arbeitsalltag von Lehrenden und Lernenden ein. Die Vorzüge der digitalen Bibliothek gehen aber noch weit über die reine Verfügbarkeit hinaus. Recherchen in Verbundkatalogen sind dabei ebenso eine Verbesserung wie medienübergreifende und individualisierbare oder individuelle Dienste zur Kommunikation und Kollaboration im Hinblick auf die Nutzung der Informationsobjekte.[2]

Im Zusammenhang mit der Verfügbarkeit stellt sich die Frage nach der Form der Distribution der bereitgestellten Informationsobjekte. Im Bereich Filesharing erfreuen sich *Tauschbörsen* bereits einer großen Beliebtheit. Dienste wie Napster[3] oder Gnutella[4] ermöglichen den Austausch von Daten basierend auf dem Konzept der Peer-to-Peer-Netzwerke. Innerhalb der Tauschbörsen fungiert der einzelne angebundene Benutzer sowohl als Anfrager wie auch als Anbieter von Daten. Die Suche nach Informationen innerhalb dieser Tauschbörsen beruht auf dem Vergleich von Zeichenketten. Genauer gesagt wird der angegebene Suchbegriff innerhalb der Dateinamen der zu Verfügung stehenden Dateien gesucht.

Durch die Anbindung von digitalen Bibliotheken an ein Peer-to-Peer-Netzwerk besteht die Möglichkeit, zeitgleich auf alle angebotenen Bibliotheken zuzugreifen und dadurch die Informationssuche wesentlich zu bereichern.

Einen modernen Ansatz von Peer-to-Peer-Netzwerken bietet das Edutella-Projekt. Es handelt sich dabei um eine „learning web infrastructure“, die im Rahmen des PADLR-Projektes entwickelt wurde.[5] Innerhalb des Edutella-Netzwerkes wird das Resource Description Framework zur Annotation der Informationsobjekte verwendet. Diese erlaubt eine wesentlich komplexere Suche innerhalb des Peer-to-Peer-Netzwerkes, als es die Suche nach Dateinamen zulässt. Eine ausführliche Beschreibung des Edutella-Projektes findet sich in Abschnitt 2.6.2.

Diese Diplomarbeit teilt sich im Wesentlichen in drei Kapitel: Grundlagen, Konzeption und Implementation. Im Kapitel Grundlagen werden digitale Bibliotheken, Peer-to-Peer-Netzwerke und die technischen Rahmenbedingungen vorgestellt. Im Kapitel Konzeption wird aufgezeigt, welche Voraussetzungen an eine Anbindung der digitalen Bibliotheken gestellt werden und in welcher Weise diese Voraussetzungen umgesetzt werden sollen. Im letzten Kapitel werden schließlich die konkreten Implementierungen vorgestellt.

2 Grundlagen

In Abschnitt 2.1 dieses Kapitels werden digitalen Bibliotheken vorgestellt. Es werden Beweggründe für ihre Entwicklung aufgezeigt und worin ihre Vorteile gegenüber der klassischen Bibliothek bestehen.

Im Abschnitt 2.2 wird erläutert, aus welchem Grunde und in welcher Weise die Informationsmaterialien der digitalen Bibliothek benutzerfreundlich zugänglich gemacht werden können. Dabei richtet sich das Augenmerk weniger auf den eigentlichen Zugriff als vielmehr auf die Suche nach Informationsmaterialien. In diesem Zusammenhang wird auf eine für Suchanfragen geeignete Repräsentationsform für Informationsmaterialien eingegangen.

Einer Auswahl von Datenquellen widmet sich der Abschnitt 2.4. Als Datenquellen sollen in diesem Fall technische Plattformen bezeichnet werden, die bibliografische Inhalte zur Verfügung stellen und die im Rahmen dieser Diplomarbeit an ein Peer-to-Peer-Netzwerk angebunden werden.

Da sich ein wesentlicher Teil dieser Arbeit mit der Verarbeitung von Informationen digitaler Bibliotheken beschäftigt, werden im Abschnitt 2.5 konkrete Repräsentationsformen für Informationen vorgestellt und somit die Grundlage für das Verständnis der späteren Informationsverarbeitung geschaffen.

Das Prinzip von Peer-to-Peer-Netzwerken wird in Abschnitt 2.6 behandelt. Es wird aufgezeigt, in welcher Weise sich Peer-to-Peer-Netzwerke von üblichen Client/Server-Netzwerken unterscheiden und welche Vorteile man aus ihnen ziehen kann. Im Verlaufe dieses Kapitels wird am Beispiel des Edutella Netzwerkes – das für die spätere Verteilung der Informationen verwendet werden soll – eine mögliche Infrastruktur eines solchen Netzwerkes vorgestellt.

Der letzte Abschnitt beschäftigt sich mit der Frage, wie verschiedene Repräsentationsformen von Daten ineinander überführt werden können. In diesem Rahmen werden die Grundzüge von XSL dargestellt.

2.1 Digitale Bibliotheken

Rein formal definiert sich eine digitale Bibliothek als überwachte Sammlung von Informationen mit dazugehörigen Dienstleistungen, wobei die Informationen in digitalen Formaten gespeichert werden und über ein Netzwerk abrufbar sind.[6] Die digitale Bibliothek bedient sich dabei einheitlicher Architektur und Interoperabilitäts-Standards, um die unterschiedlich strukturierten Informationen bei Recherche-, Navigations- und anderen Vorgängen in die angebotenen Dienstleistungen zu integrieren. Sie stellt Funktionen zur Selektion, Organisation und Archivierung von Informationen zur Verfügung und erlaubt den Benutzern, über diese Funktionen auf die Informationen zuzugreifen.

Ein Grund für die Entwicklung digitaler Bibliotheken ist eine verbesserte Distributionsmöglichkeit von Informationen im Vergleich zur klassischen Form der Bibliothek. Der Schritt zur digitalen Bibliothek erlaubt es, die Informationen von nahezu jedem Personal Computer über ein Netzwerk abzurufen, indem man beispielsweise die Homepage einer Bibliothek besucht. So ist es auf einfache Weise möglich, Informationen und Dokumente auch von solchen Bibliotheken zu beziehen, die sich nicht in der Nähe des Benutzers befinden, sondern rund um

den Globus verteilt sein können. Demgegenüber kann bei klassischen Bibliotheken ein Exemplar eines Dokumentes – aufgrund geographischer Gegebenheiten – für den Einzelnen nahezu unerreichbar sein.

Ein weiterer Vorteil der digitalen Bibliothek gegenüber der klassischen Variante ist die effektivere Suche nach Informationen. Papierdokumente sind zwar angenehm zu lesen, doch gestaltet sich die Recherche nach Informationen aufwendig, wenn es in den Bereich der Volltextsuche geht. Demgegenüber können in digitalen Bibliotheken z.B. stichwortgestützte Suchvorgänge über den gesamten Datenbestand in wenigen Augenblicken durchgeführt werden. Es sollte allerdings erwähnt werden, dass eine manuelle Suche in einer klassischen Bibliothek den positiven Nebeneffekt hat, den Suchenden gelegentlich durch einen glücklichen Zufall auf weitere ihm nützliche Informationen stoßen zu lassen.

Einer der größten Nachteile der klassischen Bibliothek besteht darin, dass Informationen über Dokumente und die Dokumente an sich nicht permanent verfügbar sind. So sind Benutzer an die Öffnungszeiten gebunden, wenn sie z.B. ein Buch ausleihen wollen. Der Zugriff auf Informationen und Dokumente wird aber nicht nur durch Öffnungszeiten beschränkt. Dokumente stehen oft nur in begrenzter Anzahl zur Verfügung, so dass ein Benutzer möglicherweise warten muss, bis ein ausgeliehenes Buch zurückgegeben und somit für ihn zugänglich wird. Demgegenüber ist eine digitale Bibliothek prinzipiell rund um die Uhr erreichbar und nicht an Öffnungszeiten gebunden. Auf diese Weise sind die gespeicherten Informationen und Dokumente jederzeit für jedermann erreichbar. Dabei beschränkt sich „jederzeit“ nicht allein auf die Öffnungszeiten. Die Informationen sind auch für eine beliebige Anzahl von Benutzern verfügbar, unabhängig davon, ob mehrere Benutzer gleichzeitig auf die Informationen zugreifen.

Ein zusätzlicher Vorteil der digitalen Bibliothek ist die zentrale Verwaltung der Informationen und Dokumente. Sie liegen als digitale Informationsobjekte – mit eventuell einigen wenigen Kopien auf anderen Servern – zentral vor. Dies ist eine erhebliche Verbesserung gegenüber der teuren Vervielfältigung z.B. wenig benutzter Dokumente bei klassischen Bibliotheken. Die gedruckten Informationsobjekte sind aufwendig aktuell zu halten, zumal Dokumente erneut gedruckt und alle älteren Versionen ersetzt werden müssen, was eine nicht unerhebliche Kostenbelastung darstellt. In einer digitalen Bibliothek werden die Informationsobjekte an einer zentralen Speicherstelle einmalig durch eine aktuelle Version ersetzt und sind fortan für alle Benutzer in der aktuellen Version verfügbar, ohne dass weitere Kosten entstehen.

Informationen, die innerhalb einer digitalen Bibliothek vorgehalten werden, lassen sich in die zwei Kategorien *Daten* und *Metadaten* unterteilen. In beiden Fällen handelt es sich um Daten, die sich in digitaler Form darstellen lassen. Die eigentlichen Informationsobjekte – z.B. Bücher, Zeitschriften oder Videos – werden als Daten bezeichnet, wohingegen Metadaten „Daten über Daten“ darstellen. Sie lassen sich in *beschreibende Metadaten* (z.B. bibliografische Informationen) und *administrative Metadaten* (z.B. Informationen über Rechte an der Information und Zugriffsbefugnisse auf die Information) unterteilen.

Die Unterscheidung zwischen Daten und Metadaten hängt von dem Kontext ab, in dem die Information betrachtet wird. Im bibliografischen Kontext werden Katalogeinträge und Zusammenfassungen üblicherweise als Metadaten angesehen, da sie andere Informationsobjekte beschreiben. Im Kontext einer Datenbank von Zusammenfassungen handelt es sich bei den einzelnen Zusammenfassungen hingegen um die *Daten* der Datenbank.[6] Digitale Bibliotheken

ken stützen sich auf standardisierte Metadaten, die es ermöglichen, die Informationen über und die Beziehungen zwischen digitalen und nicht-digitalen Objekten unterschiedlicher Formate und Informationsträger zusammenzuführen. Abschnitt 2.2 geht detaillierter auf Metadaten ein.

Die digitale Bibliothek an sich besteht aus mehreren Servern, die durch ein Netzwerk verbunden sind und im Wesentlichen drei Hauptfunktionen bereitstellen:

1. Speichern von Inhalten
2. Interaktion mit der digitalen Bibliothek
3. Bereitstellung von Diensten für Benutzer

Typische Dienste für Benutzer sind zum einen der *Suchdienst*, zu dem z.B. Kataloge und Indexe gehören, die bei der Suche nach Informationen behilflich sind, und zum anderen der *Lokationsdienst*, der dazu verwendet wird, über die Verweise in einem ermittelten Suchergebnis auf die tatsächlichen Informationsobjekte zuzugreifen.[6]

Die Inhalte einer digitalen Bibliothek werden in sog. *Repositories* abgelegt. Dabei handelt es sich um Server, die Informationssammlungen speichern und den Zugriff auf einzelne Informationen ermöglichen. Eine besondere Form des Repository ist das *Archiv*, das für die Langzeitaufbewahrung von Informationen bestimmt ist.

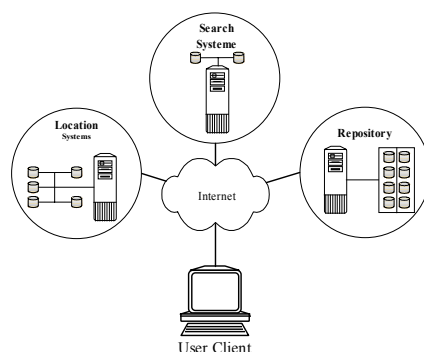


Abbildung 1: Schematischer Netzwerkaufbau einer digitalen Bibliothek

Die Persistenzform von Informationsobjekten in Repositories (d.h. das Datenformat, in dem sie gespeichert sind) kann sich sehr von der späteren Repräsentationsform (Darstellung beim Benutzer) unterscheiden. Als Beispiel soll ein Simulator zum Training von Piloten dienen. Ein solcher Simulator könnte aus einem Computerprogramm, einer Datenstruktur, digitalisierten Bildern zur Darstellung der Umgebung sowie anderen Datenobjekten bestehen. Bei diesen Objekten – Programm, Bilder, Daten, usw. – handelt es sich um die Persistenzform des Informationsobjektes. Der Benutzer wiederum sieht nicht die Daten selbst, sondern er erkennt eine Serie von Bildern, synchronisiert mit Klang und Kontrollsequenzen.[6]

Daraus resultiert, dass während der Interaktion zwischen Benutzer und Repository die Informationsobjekte aus der bibliotheksinternen Persistenzform in die vom Benutzer gewünschte Repräsentationsform konvertiert werden müssen. Diese Konvertierung wird als *Rendern* bezeichnet.

Der Zugriff der Benutzer einer digitalen Bibliothek auf die gespeicherten Informationsobjekte (Ressourcen) wird meist über Internetanbindungen und in diesem Zusammenhang speziell über Internetbrowser realisiert. Um dies zu ermöglichen, bedarf es z.B. eines Web-User-Interfaces, das die Verbindung zwischen dem Repository und dem World Wide Web herstellt.

Nach einer Recherche und der Anzeige der entsprechenden Treffermenge soll der Benutzer auf eine bestimmte Ressource geführt werden. Um eine eindeutige Identifizierung der Ressourcen vornehmen zu können, ist es notwendig ihnen einen eindeutigen Namen zu geben. Die Internet Engineering Task Force (IETF) entwickelt Standards zur eindeutigen Benennung von Ressourcen, die unter anderem über das Internet erreichbar sind. Die von der IETF entwickelten Benennungsschemata werden unter dem Begriff *Uniform Resource Identifier* (URI) zusammengefasst. Zurzeit sind zwei Benennungsschemata identifiziert: *Uniform Resource Locator* (URL) und *Uniform Resource Name* (URN).[6]

2.2 Metadaten

Wird eine Suchanfrage an eine der vielen Suchmaschinen gestellt, bekommt man in der Regel eine sehr große Anzahl von Treffern geliefert, die mit dem gewünschten Suchergebnis häufig nicht viel gemeinsam haben. Das Problem liegt in der Art und Weise begründet, wie die Ergebnisse der Suchmaschinen bestimmt werden. Diese suchen in ihrer Datenbank lediglich nach den eingegebenen Wörtern bzw. Sätzen und nicht nach der Bedeutung oder dem Sinn, der hinter den Begriffen steht.

Während textliche Dokumente zumindest nach Begriffen durchsucht werden können, ist eine solche Suche in nicht-textlichen Dokumenten wie Bildern, Audio- oder Videodateien, aussichtslos. Für die notwendige Transparenz sind inhaltsbeschreibende Daten (was, wo, wie, wer, wohin etc.) erforderlich.[7] Tim Berners-Lee, der Erfinder des World Wide Web, definiert diese „Metadaten“ als

„[...] maschinenlesbare Informationen über elektronische Ressourcen oder andere Dinge“ [8].

Metadaten sind demnach „Daten über Daten“. Etwas präziser handelt es sich um beschreibende, erklärende oder ergänzende Daten zu einer Ressource. Sie werden benötigt, um einerseits die vielfältigen Eigenschaften digitaler Ressourcen ausreichend granular modellieren zu können, zum anderen um die Vielfalt digitaler Medien zu klassifizieren.

Bibliografische Daten sind ein Spezialfall von Metadaten, nämlich deskriptive Parameter zur Sacherschließung von Publikationen, um diese zu archivieren und einen langfristigen Zugang zu ihnen zu gewährleisten, insbesondere auch in Bezug auf ihre Auffindbarkeit innerhalb eines größeren Dokumentenbestandes. Dies geschieht durch die Verwaltung bibliografischer Kataloge, in die nach jeweils vereinbarten und eindeutigen Kriterien die verfügbaren Publikationen über Autor, Titel, Sachgebiet und ähnliches eingetragen werden. [2]

Prinzipiell existieren zwei Möglichkeiten, um diese Verwaltung zu organisieren. Zum einen anhand *verbaler Nachweismethoden*, zum anderen anhand *klassifikatorischer Nachweismethoden*.

Verbale Nachweismethoden werden auch als indexierende Methoden bezeichnet. Bei dieser Art des Nachweises werden aus einem sprachbezogenen Vokabular geeignete deskriptive

Begriffe ausgewählt, um eine Ressource semantisch zu erfassen. Man unterscheidet zwischen der Vergabe von Stichwörtern und der Verschlagwortung.[2]

Stichwörter sind frei wählbare Begriffe zur thematischen Beschreibung einer Ressource, die direkt aus ihren textlichen Komponenten entnommen werden. Sie werden zumeist aus Titel sowie Untertitel des Dokumentes ermittelt.[2]

Schlagwörter sind das Thema einer Publikation beschreibende Substantive, welche nicht notwendigerweise der zu erfassenden Publikation entnommen sein müssen. Man unterscheidet hier zwischen freien Termini, die direkt dem Dokument oder dem Wortschatz des Indexierers entnommen sind, und kontrollierten Termini, die aus einem Thesaurus oder einer Schlagwortliste stammen.[2]

Bei den *klassifikatorischen Nachweismethoden* werden die Publikationen bezüglich definierter Eigenschaften klassifiziert und in eine hierarchische Struktur eingeordnet. Die Klassifikationen teilen dabei die Objekte in jeweils disjunkte Gruppen (Klassen) mit entsprechend unterscheidbaren Merkmalen oder Eigenschaften ein. Ein Klassifikationssystem ist also die strukturierte Darstellung von Klassen und der zwischen ihnen bestehenden Begriffsbeziehung. Es werden jeweils die Objekte in einer Klasse zusammengefasst, die mindestens ein gemeinsames Klassenmerkmal haben. Die Klassen werden dabei durch entsprechende Namen bezeichnet.[2]

Bei den genannten Nachweismethoden handelt es sich um einander ergänzende Verfahren, da zum einen die Menge der Publikationen, die ein Stichwort gemeinsam haben, in unterschiedlichen Klassen eingeordnet sein können, zum anderen die in einer Klasse zusammengefassten Medien disjunkte Stichwortmengen aufweisen können. Entsprechend eignet sich Verschlagwortung nicht zur begrifflichen Klassifikation, jedoch zur thematischen Eingrenzung.[2]

Bibliografische Klassifikationen sind normierte Untergliederungen – eines oder mehrerer Fachgebiete – deren Zweck es ist, Publikationen anhand ihrer Eigenschaften gezielt klassifizieren zu können, um für den potenziellen Benutzer das Finden der zu einem gegebenen Thema passenden Literatur zu erleichtern. Eine typische klassifikatorische Nachweismethode ist die Klassifikation nach *ACM* (Association of Computing Machinery) aus dem Fachgebiet der Informatik.[2]

Neben den bibliografischen Daten benötigt man für digitale Medien noch weitere anwendungsspezifische Daten. Beispielsweise – im Kontext einer Lehr- und Lernumgebung – die notwendigen technischen Rahmenbedingungen zur Mediennutzung, wie etwa das verwendete Dateiformat, pädagogische und psychologische Hinweise über die anvisierte Zielgruppe oder den Schwierigkeitsgrad sowie etwa die logische, zeitliche und räumliche Einordnung bezüglich anderer Medien. *Learning Objects Metadata* ist ein Metadaten-Schema für pädagogische Ressourcen. Der Standard ist abwärtskompatibel zu Dublin Core, geht jedoch verstärkt auf die im didaktischen Umfeld auftretenden Erfordernisse ein.[2]

2.3 Dublin Core

Die Dublin Core Metadata Initiative ist ein offenes Forum, das sich in der Entwicklung kompatibler Online-Metadaten-Standards engagiert. Die Mitglieder dieses Forums die verschie-

densten Disziplinen und stammen aus Organisationen und Instituten weltweit. Einige der Disziplinen sind:[9]

- Archiv- und Museums-Informationssysteme
- Digitales Informationsmanagement
- Digital Libraries (digitale Bibliotheken)
- Bibliotheks-Technologie-Service
- Regierungsbehörden – lokale, regionale und staatliche
- Angewandte Informatik
- Netzwerktechnologien
- Produktforschung und -entwicklung

Entstanden ist die Dublin Core Metadata Initiative auf der zweiten Internationalen World Wide Web Konferenz 1994 aus einer Diskussion über Semantik und die Schwierigkeiten beim Auffinden von Ressourcen im Web. Diese Diskussion führte zu einer Konferenz im März 1995 in Dublin. Bei diesem Event, dem „OCLC/NCSA Metadata Workshop“, diskutierten 50 Teilnehmer darüber, wie nützlich eine Kernmenge von Semantik für webbasierte Ressourcen wäre. Sie vereinigten ihre Ergebnisse als „Dublin Core Metadata Schema“.[10]

Die Aktivitäten der Dublin Core Metadata Initiative umfassen beschlussorientierte Arbeitsgruppen, globale Workshops, Konferenzen und pädagogische Bemühungen, um eine breite Akzeptanz von Metadata Standards zu erreichen. Ausführliche Informationen zur Dublin Core Metadata Initiative findet man unter [11].

In Rahmen dieser Diplomarbeit sollen die Elemente der Dublin Core Metadata Initiative als Grundlage zur Annotierung von bibliografischen Daten verwendet werden. Das Dublin Core Metadaten Schema dient als allgemeiner Kern für bibliografische Attribute. Es ist eine Anwendungsgebiet unabhängige Beschreibung gemeinsamer Metadaten und soll zum einen den Nachweis, die Erfassung, die Indexierung und die Nutzung digitaler Publikationen erleichtern, zum anderen sollen Autoren und Herausgeber dazu ermutigt werden, die zum Nachweis erforderlichen Daten zusammen mit den jeweiligen Publikationen zu veröffentlichen.

Den Kern der Metadaten bilden 15 Felder. Sie sind in Tabelle 1 zusammengefasst sind lassen sich in drei Gruppen einteilen[2]:

1. solche, die sich auf den Inhalt einer Ressource beziehen: TITLE, SUBJECT, DESCRIPTION, SOURCE, LANGUAGE, RELATION, COVERAGE
2. solche, die sich auf den intellektuellen Besitz beziehen: CREATOR, PUBLISHER, CONTRIBUTOR, RIGHTS; diese Attribute beziehen sich auf Personen oder Institutionen, die Rechte an einer Ressource besitzen oder diese verwalten
3. solche, die sich auf eine konkrete Instanz einer Ressource beziehen, DATE, TYPE, FORMAT, IDENTIFIER

Dublin Core Element	Beschreibung
TITLE	Titel des Dokumentes oder der Ressource
CREATOR	Autor/Verfasser der Ressource, der verantwortlich für deren Inhalt ist
SUBJECT	Thema, Fachgebiet, Schlag- und Stichwörter
DESCRIPTION	Beschreibung, kurze Inhaltsangabe, Zusammenfassung
PUBLISHER	Verlag, veröffentlichende Instanz oder Institution
CONTRIBUTOR	weiter an der Veröffentlichung beteiligte Autoren oder Mitarbeiter
DATE	Zeitpunkt der Veröffentlichung/Herausgabe
TYPE	Art der Publikation
FORMAT	technisches Format, beispielsweise MIME
IDENTIFIER	weltweit eindeutige Kennung, beispielsweise ISBN
SOURCE	Quelle abgeleiteter Publikationen
LANGUAGE	Sprache, in der die Publikation verfasst wurde
RELATION	logische Einordnung zu anderen Ressourcen
COVERAGE	zeitliche und räumliche Bezugnahme zu anderen Ressourcen
RIGHTS	Verweis auf die Nutzungsbedingungen eines Mediums

Tabelle 1: Dublin Core Elemente

2.4 Datenquellen

2.4.1 Open Archive Initiative (OAI)

Die Open Archive Initiative (OAI) entstand im Jahr 1999. Sie entwickelt und unterstützt interoperable Lösungen, deren Ziel es ist, eine effiziente Verteilung von Inhalten bereitzustellen. Die Wurzeln des OAI liegen in der E-Print-Gemeinschaft. In den letzten Jahren verschob sich der Fokus allerdings auf allgemeinere Bereiche.[12]

Der Name spiegelt dabei die Herkunft der Open Archive Initiative aus dem Bereich des E-Prints wider, in dem der Begriff „Archive“ allgemein als Synonym für einen „Aufbewahrungsort von wissenschaftlichen Arbeiten“ steht. Die OAI verwendet den Begriff „Archive“ in einem allgemeineren Sinn: Aufbewahrungsort von gespeicherten Informationen. Der Begriff „Open“ bezieht sich auf die Architektur in Bezug auf die Definition und Verbreitung von Schnittstellen, die die Erreichbarkeit von Inhalten einer Vielzahl von Anbietern ermöglichen. Mit „Open“ ist hierbei nicht „frei“ oder „unbegrenzter Zugriff“ gemeint. [12]

„Die Open Archive Initiative entwickelt und fördert kompatible Standards, um eine effiziente Verbreitung von Inhalten zu erreichen. Die Open Archive Initiative hat ihre Wurzeln in der Anstrengung den Zugriff auf E-Print Archive zu verbessern im Sinne der Verbesserung der Erreichbarkeit der wissenschaftlichen Kommunikation. Kontinuierliche Unterstützung dieser Arbeit verbleibt als Eckpfeiler des Open Archive Programms. Das fundamentale technische Framework und die Standards, die entwi-

ckelt wurden, um diese Arbeit zu unterstützen, sind jedoch unabhängig sowohl vom angebotenen Inhalt sowie von den ökonomischen Mechanismen, die den Inhalt umgeben, und versprechen eine weit umfangreichere Relevanz in der Eröffnung von Zugriff auf ein Reihe von digitalen Materialien.“ [12]

Das technische Framework der Open Archive Initiative ist ein Ansatz, Kompatibilität bereitzustellen, der nur mit geringen Hindernissen versehen sein soll. Dabei ist das Framework nicht dazu gedacht, andere Ansätze zu ersetzen, sondern vielmehr dazu, eine einfach zu entwickelnde und zu implementierende Alternative für andere Zwecke als diejenigen bereitzustellen, die durch bereits existierende Lösungen abgedeckt werden.

Den technischen Vereinbarungen liegt eine Unterscheidung in zwei Klassen von Teilnehmern zugrunde. Zum einen die *Data Provider*, die das OAI-Framework als Mittel einsetzen, um Metadaten über ihre Daten zu veröffentlichen, und zum anderen die *Service Provider*, die die Metadaten der Data Provider abrufen und sie als eine Basis für eigene Mehrwertdienste verwenden.[12]

Service Provider und Data Provider kommunizieren über das dem OAI zugrunde liegende *OAI Protocol of Metadata Harvesting* – kurz OAI-PMH. Dieses Protokoll basiert auf dem im Internet verbreiteten HTTP-Protokoll zur Kommunikation im Client/Server-Bereich.

Prinzipiell lässt sich die Funktionalität der Kommunikation zwischen Service Provider und Data Provider wie in Abbildung 2 darstellen.

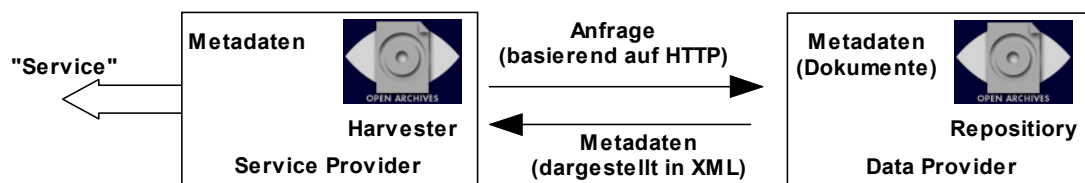


Abbildung 2: Prinzipielle Funktionsweise des OAI-PMH¹

Der Service Provider schickt eine Anfrage in einem definierten Format an den Data Provider. Der Data Provider schickt daraufhin seine Metadaten in Form eines XML-Dokumentes an den Service Provider, der mit diesen Daten wiederum weitere Services zur Verfügung stellen kann.

Das technische Framework des OAI adressiert zwei grundlegende Metadaten-Anforderungen:

- Kompatibilität
- Erweiterbarkeit

Die Anforderung der Kompatibilität ist bestimmt durch die Notwendigkeit, dass alle OAI Data Provider Metadaten in einem gemeinsamen Format bereitstellen. In diesem Fall bildet die gemeinsame Grundlage die Menge der Metadaten-Elemente der Dublin Core Metadata Initiative. Beschreibungen oder Metadatenmengen einer bestimmten Anwendergemein-

¹ entnommen aus [41]

schaft wird im Framework durch die Unterstützung paralleler Metadatenmengen Rechnung getragen. Das Framework beinhaltet keine Einschränkungen bezüglich paralleler Metadatenmengen, außer dass diese Metadaten in XML notiert werden müssen und es ein korrespondierendes XML-Schema geben muss, um eine Validierung durchführen zu können.[12]

Eine weiterführende Beschreibung des technischen Konzeptes findet sich im Abschnitt 4.2.2, der sich ausführlich mit der Datenübertragung beschäftigt.

2.4.2 DSpace

Anfang 2000 hat man in den MIT Libraries verstärkt damit begonnen, ein geeignetes institutionales Repository aufzubauen, das Publikationen auf lange Sicht speichert und bereitstellt. Ergebnis ist das in Kooperation mit Hewlett-Packard entwickelte und im November 2002 in der Version 1.0 fertig gestellte DSpace, welches inzwischen auch in weiteren (insbesondere amerikanischen) Einrichtungen eingesetzt wird.[13] Es dient als Repository für digitale Forschungsmaterialien und Materialien mit pädagogischen Inhalten, die von Mitgliedern von Universitäten und anderen Organisationen erstellt werden.[14] DSpace stellt eine Reihe von Diensten zur Verfügung, um digitale Materialien zu speichern, zu verwalten und zu verteilen, und ist in der Lage, Daten in jeglichem Format aufzunehmen – z.B. Text, Video, Audio und andere Formate.

Die Inhalte werden indiziert, so dass sie von Benutzern durchsucht, über das World Wide Web verteilt und über einen langen Zeitraum bewahrt werden können. Auf diese Weise bietet DSpace die Möglichkeit, Publikationen und Forschungsmaterialien in einem professionellen Repository zu verwalten und sie so breiter verfügbar zu machen.

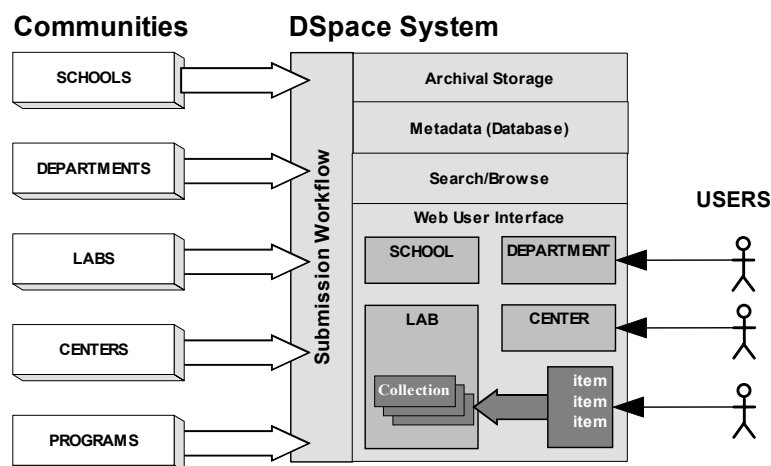


Abbildung 3: DSpace Information Model²

Das DSpace-System basiert auf der Idee von „Communities“. Communities können Fachabteilungen, Labore, Forschungseinrichtungen oder auch Schulen sein, die unterschiedliche Informationsmanagement-Bedürfnisse haben. Jede Community hat wiederum Collections, die die jeweiligen Inhalte und Dateien enthalten.

² entnommen aus [14]

2.4.2.1 DSpace-System

Zur Beschreibung der Einträge verwendet DSpace qualified Dublin Core Metadaten. Dabei sind drei Felder vorgeschrieben: `title`, `language` und `date`. Alle weiteren Felder sind optional. Darüber hinaus gibt es zusätzliche Felder für Zusammenfassungen, Schlüsselwörter, technische Metadaten und Metadaten für Nutzungsrechte. Die Metadaten werden innerhalb eines Datensatzes angezeigt, indiziert und zur Durchsicht sowie für die Suche innerhalb des Systems bereitgestellt.[14]

DSpace verwendet verschiedene Schnittstellen zur Kommunikation. Eine für Autoren und andere, die mit dem Bereitstellungsprozess verbunden sind, eine für Endnutzer, die nach Informationen suchen, und eine für Systemadministratoren. Das Enduser- oder öffentliche Interface erlaubt das Auffinden von Einträgen durch Suchen in den Daten oder durch Suche über die Metadaten. Wurde ein Eintrag innerhalb des Systems gefunden, erhält der Benutzer das Informationsobjekt durch Klicken auf einen Link, über den das archivierte Material vom Browser herunter geladen werden kann. [14]

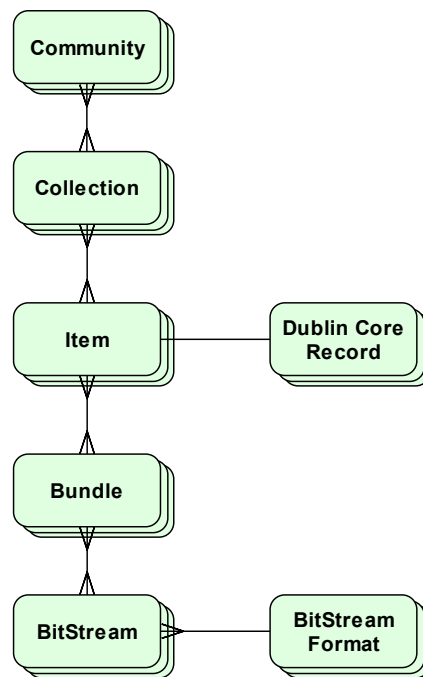
Abgesehen von einem browsergestützten Webinterface stellt DSpace die gespeicherten Metadaten über eine integrierte OAI-Schnittstelle für OAI Service Provider zur Verfügung.

Während Suchanfragen (im Normalfall) anonym möglich sind, dürfen neue Publikationen nur von autorisierten Nutzern eingestellt werden. Diese müssen sich einmalig mit Mailadresse und Passwort registrieren und erhalten daraufhin Zugang zu einem personalisierten Bereich. Mehrere Benutzer sind in Gruppen zusammenfassbar, deren Zugriffsmöglichkeiten durch die Vergabe spezifischer Rechte (READ, WRITE, ADD, REMOVE) auf bestimmte Collections, Items und sonstige Objekte beschränkt bzw. ausgeweitet werden können.[13]

Unterschiedliche Communities repräsentieren unterschiedliche Schulen, Abteilungen usw. Die verschiedenen Einrichtungen haben unterschiedliche Ansichten darüber, wer in welchem Umfang auf das System zugreifen darf. An dieser Stelle spielen Fragen eine Rolle wie: *Welche Personen dürfen Inhalte an DSpace übermitteln? Welche Personen oder Personengruppen unterliegen welchen Restriktionen? Welche Arten von Daten dürfen abgelegt werden?* All diese Themen richten sich an die Verantwortlichen der Communities, die diese Fragestellungen mit dem DSpace User Support in einem Workflow modellieren. Das System erstellt daraufhin „E-People“, die innerhalb des Workflows einer Community entsprechenden Regeln im Kontext einer bestimmten Collection unterliegen.[14]

2.4.2.2 Datenmodell

Das Datenmodell von DSpace ermöglicht die Klassifizierung und hierarchische Gliederung der Informationen. Auf unterster Ebene (und in einem Dateisystem abgelegt) befinden sich die eigentlichen Dokumente in Form von *Bitstreams* eines bestimmten *Dateiformats*. Diese sind in *Bundles* organisiert, welche wiederum als jeweils zusammengehörige *Items* inklusive *DC-Records* in einer relationalen Datenbank gespeichert werden. Gleichartige Materialien, wie z.B. multimedial aufbereitete Dissertationen, werden als *Collection* aufgefasst – und mehrere dieser Collections bilden schließlich eine *Community*. Die nachfolgende Abbildung verdeutlicht – ebenso wie das aus der DSpace-Dokumentation[15] entnommene Beispiel – das Zusammenspiel:

Abbildung 4: DSpace Datenmodell³

Beispiel 1

Community	Laboratory of Computer Science; Oceanographic Research Center
Collection	LCS Technical Reports; ORC Statistical Data Sets
Item	A technical report; a data set with description; a video recording of a lecture
Bundle	A group of HTML and image bitstreams making up an HTML document
Bitstream	A single HTML file; a single image file; a source code file
Bitstream Format	Microsoft Word version 6.0; JPEG encoded image format

2.4.2.3 Technologische Plattform

DSpace wurde für den Einsatz unter UNIX Betriebssystemen entworfen. Der originale Quelltext ist in Java programmiert. Andere Teile der Technologiesäule beinhalten relationale Datenbank Management Systeme (PostgreSQL), einen Webserver, eine Java Servlet Engine (Apache Tomcat) und weitere Bibliotheken.

Die DSpace Architektur besteht aus drei Layern, die sich in Speicherungs- (*Storage Layer*), Betriebs- (*Business Layer*) und Anwendungslayer (*Applikation Layer*) gliedern. Der Storage Layer ist so implementiert, dass er das Dateisystem verwendet und die dort abgelegten Daten über PostgreSQL Datenbanktabellen verwaltet. Der Business Layer ist der Bereich, in dem sich die DSpace spezifischen Funktionalitäten befinden, darunter das Workflow System, Content Management, Administration sowie Such- und Browsemodule. Der Application Layer

³ entnommen aus [15]

schließlich deckt alle Schnittstellen zum System ab. Dazu gehören im Besonderen die Web UI und die OAI-Unterstützung.[14]

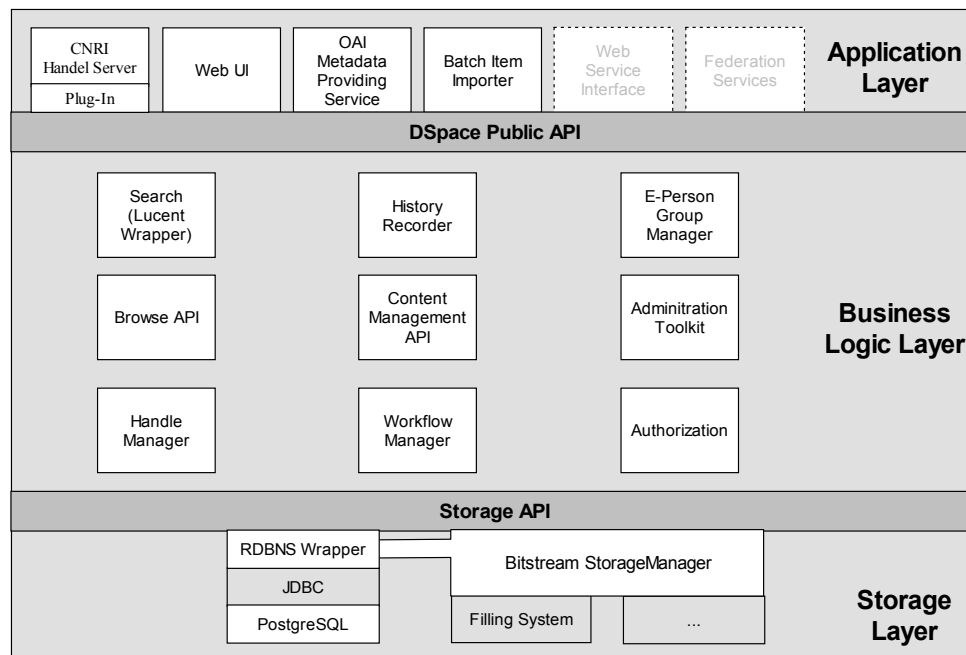


Abbildung 5: DSpace technische Architektur⁴

Ausführliche Informationen über DSpace sind unter [16], [15] und [14] verfügbar.

2.4.3 ILIAS

ILIAS ist ein webbasiertes Lern-Management-System (LMS), das ursprünglich im VIRTUS[17] Projekt an der Wirtschafts- und Sozialwissenschaftlichen Fakultät der Universität zu Köln entwickelt wurde. Die seit 1997 entwickelte Lernplattform liegt in der Version 2 und in einer technisch und funktional überarbeiteten Fassung als Version 3 vor. Mittels eines Client/Server-Systems ermöglicht ILIAS das Erstellen, Bearbeiten und Darstellen von Lehrmaterialien in einer einheitlichen Umgebung.[18] Es wurde konzipiert, um die Kosten für den Einsatz neuer Medien in der Lehre und Weiterbildung zu senken und zugleich einen höchstmöglichen Einfluss der Anwender auf die Gestaltung der Software zu sichern.[19] ILIAS erlaubt eine effiziente Erstellung von Kursmaterialien und bietet eine standardisierte Funktionspalette für den Lern- und Arbeitsprozess einschließlich einer integrierten Navigation und Verwaltung. Alle Anwender verfügen über einen personalisierten Arbeitsbereich. Je nach zugewiesener Rolle hat der Benutzer Zugriffsrechte auf die verschiedenen Funktionen und Materialien. Für die Zusammenarbeit und Kommunikation lassen sich Gruppen bilden und eine integrierte Autorenumgebung ermöglicht die Erstellung von Lern- und Arbeitsmaterialien, für die alle im Internet möglichen Formate innerhalb von ILIAS eingesetzt werden können.[20]

⁴ entnommen aus [14]



Abbildung 6: ILIAS Persönlicher Schreibtisch⁵

Der **Persönliche Schreibtisch** ist die zentrale Einheit, die Homepage des ILIAS Lernprogramms. Diese Seite erscheint, wenn der Benutzer ILIAS von außen aufruft und sich einloggt. Alle anderen Bereiche der Lernplattform können von hier über den *ILIAS masthead* erreicht werden.



Abbildung 7: ILIAS Masthead

Der persönliche Schreibtisch ist darüber hinaus auch ein Informationszentrum.[21] Von hier aus setzt der Benutzer beispielsweise die Bearbeitung eines Lernmoduls fort oder geht in seine virtuelle Arbeitsgruppe. Der Benutzer kann sich seine Lernmodule und Foren anzeigen lassen, seine persönlichen Bookmarks verwalten, seine Arbeitsgruppe ansehen und feststellen, wer zeitgleich online ist. Darüber hinaus kann er hier sein persönliches Profil (z.B. die Spracheinstellung) verwalten.

⁵ <http://www.ilias.uni-koeln.de/ios/demo.html>

Im Bereich **Magazin Alle Angebote** findet der Anwender alle für ihn zugänglichen Lernangebote und Arbeitsmaterialien, z.B. ILIAS-Lernmodule, digitale Bücher oder SCORM⁶-Module sowie Diskussionsforen und Arbeitsgruppen.

Alle Inhalte werden nach einer inhaltlichen oder organisatorischen Systematik geordnet, die bei der Einrichtung von ILIAS angelegt wird. Verschiedene Sichten (Stufenansicht, Bauman-sicht, Filteransicht) und eine interne Suchmaschine erleichtern das schnelle Auffinden der gewünschten Inhalte. Diese können dann für die weitere Nutzung abonniert bzw. belegt werden, und sind dann bis auf Widerruf direkt vom persönlichen Schreibtisch aus zugänglich.

Zurzeit können mit ILIAS drei Arten von **ILIAS-Inhaltsmodulen** erstellt und veröffentlicht werden:

- Lernmodule
- Glossare
- Digitale Bücher

Alle diese Module werden in einem integrierten Editor erstellt, liegen in XML vor und basie-ren auf einer einheitlichen DTD (Document Type Definition). Sie können in ILIAS selbst genutzt, aber auch für den Druck als PDF-Dokument aufbereitet werden. Auch andere Ausga-ben sind möglich, zum Beispiel als statische HTML-Seite, als Text-Dokument zur weiteren Nutzung in OpenOffice⁷ oder als XML-Dokument. In den ILIAS-Inhaltsmodulen sind alle webfähigen Dateiformate einsetzbar. Dateien, die nicht in einem Browserfenster dargestellt werden können, stehen in Dateilisten zum Download zur Verfügung. So sind mit ILIAS zum Beispiel hybride Lernmodule realisierbar, bei denen aktuelle und sich oft ändernde Inhalte mit solchen Lehrinhalten verbunden werden, die über längere Zeiträume unverändert bleiben und für die Offline-Nutzung als PDF-Dokument verfügbar gemacht werden.[20]

Für die **Kommunikation** in ILIAS stehen ein internes Nachrichtensystem und Diskussionsfo-ren zur Verfügung. Ein kombinierter PHP/Java-Chat wird in Kürze integriert. Das interne Nachrichtensystem steht allen Nutzern mit Verwaltung der eigenen Nachrichten und Profile zur Verfügung. Es können Nachrichten intern und/oder extern an eine E-Mail-Adresse oder an Gruppen versendet werden. Darüber hinaus gibt es Diskussionsforen mit verschiedenen Sich-ten (Wer hat zuletzt geantwortet? Wer hat auf wen geantwortet?), denen Dateien angefügt werden können. [20]

Das **Gruppensystem** unterstützt das kooperative Lernen und Arbeiten in ILIAS. Es können Lerngruppen, Arbeitsgruppen oder Gruppen für bestimmte Interessengebiete gegründet werden. Alle notwendigen Funktionen zur Verwaltung und zur Gruppenarbeit stehen je nach zu-gewiesenen Rechten zur Verfügung. Die Gruppen können mit verschiedenen Zugangsrechten (offene Gruppe ohne Anmeldung, geschlossene Gruppe mit Aufnahmeantrag, private Gruppe mit Einladung) versehen werden. Es besteht die Möglichkeit, die Gruppenressourcen (Lern-

⁶ SCORM (Shareable Content Object Reference Model) XML-basiertes Framework zum Austausch von Lern-material zwischen verschiedenen *Learning Management Systems*. Nähere Informationen siehe unter <http://www.adlnet.org>.

⁷ nähere Informationen finden sich unter <http://www.openoffice.org>

module, Foren, Dateien) und die Mitglieder der Gruppe zu Verwalten, die Aufnahme neuer Mitglieder in die Gruppe zu steuert und die Zugriffsrechte auf Ressourcen und Funktionen festzulegen.[20]

Nach dem Einrichten einer ILIAS-Installation mit der Setup-Routine können alle weiteren Tätigkeiten der **Systemadministration** in ILIAS selbst erledigt werden. Dies beinhaltet das Anlegen von Rollen für bestimmte Nutzerkreise oder die Verwaltung der Mandanten ebenso wie das Erstellen einer Wissens- oder Organisationssystematik zur Kategorisierung der einzustellenden Inhalte.

Eine ausführliche Dokumentation findet sich unter [22].

2.5 Datenformate

2.5.1 XML

XML ist eine textbasierte und daher plattformunabhängige Auszeichnungssprache für strukturierte Dokumente und Daten. Um ganz allgemein strukturierte Daten beschreiben zu können, ist es notwendig, die Auszeichnungssprache, die dieser Beschreibung zugrunde liegt, an die Anforderungen der Daten anpassen zu können. So sehen Datenmodelle für Adressen anders aus als solche, die Baugrundstücke in einem Industriegebiet beschreiben.

Auszeichnungssprachen – oder Markup-Sprachen – bestehen aus *Tags* (im Fall von XML durch in '<' und '>' geklammerte Wörter), die auch *Elemente* genannt werden, und Attributen der Form "Name=Wert".

```
<Tagname Attribut1="Wert" > Inhalt </Tagname>
```

Bei XML – als sogenannter Metasprache – besteht daher die Möglichkeit, die Menge der Tags und die Art, wie sie miteinander in Beziehung stehen, frei festzulegen. Die Namen der Tags und Attribute können frei gewählt und damit die Verständlichkeit der Dokumente vergrößert werden. Erschließt sich dem Leser des Dokumentes aus dem Tagnamen der Inhalt des Tags, so spricht man von *deskriptivem Markup*. Mit XML, genauer gesagt mit XML-Schema⁸, können auf diese Weise Auszeichnungssprachen, wie auch z.B. HTML, definiert und mit diesen Dokumente erzeugt werden. Im Sinne objektorientierter Programmierung entspricht das XML-Dokument einer Instanz, während das XML-Schema das zugrunde liegende Modell definiert.

⁸ nähere Informationen finden sich unter <http://www.w3c.org/XML/Schema>

Eine strukturierte Beschreibung von Büchern als XML-Dokument könnte wie folgt aussehen:

Beispiel 2

```
<book>
  <author>
    J.R.R. Tolkien
  </author>
  <title lang="de">
    Der Herr der Ringe
  </title>
  <title lang="en">
    Master of the ring
  </title>
  <publisher>
    Klett-Cotta
  </publisher>
  <isbn>
    3608935444
  </isbn>
</book>
<!-- Ende des ersten Buches -->

<book>
  <author>
    J.R.R. Tolkien
  </author>
  <title lang="de">
    Der Hobbit
  </title>
  <title lang="en">
    The Hobbit
  </title>
  <publisher>
    Klett-Cotta
  </publisher>
  <isbn>
    3608938052
  </isbn>
</book>
<!--Ende-->
```

Mit XML soll die logische Struktur (Hierarchie) eines Dokuments festgelegt werden. Die jeweilige Darstellung der Information (z.B. auf einem Monitor oder Drucker) bleibt einer Stylesprache, etwa XSL⁹, überlassen.

„Man trennt das Dokument von seiner Präsentation. Das heißt nicht, dass das Entwerfen von guten Dokumenten oder guten Präsentationen jetzt leichter wird, aber es bedeutet, dass man die Probleme einzeln angehen kann, was ein großer Schritt nach vorn ist.“ (Tim Bray)

Dieser Fortschritt bietet große Vorteile bei dynamischen Informationssystemen, wie z.B. Portalsystemen im Internet. Nachdem das Layout des Portals festgelegt ist, wird dieses Layout bei der Verarbeitung einer Benutzeranfrage mit in XML codierten Daten (z.B. Artikel, Preislisten, Veranstaltungstermine usw.) ausgefüllt und an den Benutzer als fertige Seite weitergegeben. Bei dieser Art der Implementierung wird aus den XML-Daten und vorher festgelegten Formatierungsvorschriften durch den Server eine HTML-Datei generiert, die an den Client weitergegeben wird.

⁹ siehe auch Abschnitt 2.7.1

2.5.2 Aufbau und Bestandteile von XML

Ein XML-Dokument besteht aus Elementen, Zeichen, Namen, Zeichendaten, Kommentaren und Verarbeitungsanweisungen (Processing Instructions).

2.5.2.1 Wohlgeformte XML-Dokumente

Der Aufbau eines XML-Dokuments unterliegt strengen syntaktischen Regeln. Es wird als *wohlgeformt* verstanden, wenn es den Anforderungen an die Struktur von XML-Dokumenten entsprechend der XML-Spezifikation¹⁰ genügt. Zu den Anforderungen gehört, dass Attribute innerhalb eines Tags nur einmal auftreten dürfen, dass die Tags des XML-Dokumentes streng hierarchisch angeordnet sein müssen und dass zu jedem Start-Tag auch ein End-Tag existieren muss. Hierarchische Anordnung bedeutet, dass ein Tag vollständig innerhalb des umgebenden Tags annotiert werden muss, wie im Beispiel 3.

Beispiel 3

richtig

```
<book>
  <author>
    J.R.R. Tolkien
  </author>
</book>
```

falsch

```
<book>
  <author>
    J.R.R. Tolkien
  </book>
</author>
```

2.5.2.2 Tags

Bei Tags in einer XML-Datei wird zwischen zwei Typen unterschieden: Tags mit und ohne Inhalt. Im Prinzip wurde der Aufbau von Tags mit Inhalt bereits oben beschrieben. Sie bestehen aus einem Start- und einem End-Tag, zwischen denen der Inhalt eingeschlossen ist.

```
<START-TAG> INHALT <END-TAG>
```

Das Start-Tag beginnt mit dem Namen des Elements gefolgt von einem oder mehreren optionalen Attributen. Ein Beispiel für ein Tag mit einem Attribut ist das `body`-Tag aus der HTML-Spezifikation:

```
<body bgcolor="white">
```

Der Elementname ist `body` und eines der möglichen Attribute ist die Hintergrundfarbe der Seite `bgcolor`, wobei in diesem Beispiel der Wert des Attributs `white` ist.

Dem Start-Tag folgt der Inhalt. Woraus der Inhalt besteht, ist in einer DTD oder einem XML-Schema festgelegt. Existiert weder XML-Schema noch DTD, kann der Inhalt aus einer beliebigen Kombination weiterer Elemente, Kommentare, Zeichendaten usw. bestehen. Den

¹⁰ die Spezifikation von XML findet sich unter [25]

Abschluss eines Elementes bildet das End-Tag. Es besteht aus einem Schrägstrich gefolgt von dem Elementnamen. Ein Beispiel passend zu dem oben begonnenen `body`-Tag ist:

```
</body>
```

In XML gibt es auch Elemente ohne Inhalt. Das klassische Beispiel ist das `img`-Tag aus der HTML-Spezifikation, mit dessen Hilfe ein Bild eingebunden wird.

```

```

Bei Elementen ohne Inhalt kann in HTML der End-Tag weggelassen werden. An dieser Stelle unterscheidet sich die Syntax von HTML und XML: Bei XML muss das Suffix des Start-Tags durch einen Schrägstrich darauf hinweisen, dass ein Leer-Tag vorliegt. Das Beispiel des `img`-Tags sähe in korrekter XML-Notation folgendermaßen aus:

```

```

2.5.2.3 Kommentare

Bei der Erstellung von Dokumenten gibt es immer wieder Situationen, in denen einige Passagen des Dokumentes mit Kommentaren versehen werden sollen, z.B. um Mitarbeiter auf etwas hinzuweisen.

```
<!--  
Hallo Wolf, bitte schreib mir eine Mail, was in diesem Kapitel noch ergänzt oder weggelassen werden soll.  
-->
```

Wie in diesem Beispiel zu erkennen, ist die Syntax der Kommentare in XML mit der in HTML identisch. Der Kommentar wird von der Zeichenfolge `<!--` am Anfang und `-->` am Ende umgeben.

2.5.3 RDF (Resource Description Framework)

"Im Web sind alle Informationen maschinenlesbar, aber nur die wenigsten Daten können auch von Maschinen verstanden werden. RDF (Resource Description Framework) als Metasprache soll dies ändern."[23]

Das Resource Description Framework ist die Grundlage für die Beschreibung und den Datenaustausch von Metadaten. Es stellt die Basis für die Zusammenarbeit verschiedener Anwendungen zur Verfügung, die im Web *maschinen-verständliche* Daten austauschen.

RDF kann in den verschiedensten Bereichen eingesetzt werden, z.B. in der *Ressourcensuche*, um bessere Suchmaschinen zu implementieren, in der *Katalogisierung*, um Inhalte und Zusammenhänge zu beschreiben, die auf Webseiten oder in digitalen Büchereien zur Verfügung stehen, oder im Bereich *intelligenter Softwareagenten*, um Wissen gemeinsam zu nutzen und auszutauschen.[26]

Das Hauptziel von RDF ist es, einen Mechanismus zur Beschreibung von Ressourcen zu definieren, ohne jegliche Annahme über die späteren Anwendungen zu machen oder die jeweilige Semantik im Vorfeld festzulegen. Die Definition dieses Mechanismus sollte anwendungsneutral und dennoch passend für die Beschreibung von Informationen jeglicher Art sein.

Das RDF Datenmodell ist ein syntaxneutraler Weg, um RDF Ausdrücke darzustellen. Die Datenmodellardarstellung wird benutzt, um Gleichheiten im Sinne von Aussagen zu finden, wobei zwei Aussagen genau dann gleich sind, wenn ihre Darstellung im Datenmodell identisch ist. Diese Definition der Gleichheit erlaubt einige syntaktische Variationen in Ausdrücken, ohne dass die Aussage verändert wird. [26]

Eines der Ziele von RDF ist es, eine Semantik für Daten, die auf XML basieren, in einer standardisierten und für die Zusammenarbeit geeigneten Weise zu spezifizieren. Für die Kodierung von RDF schlägt das W3C-Konsortium XML vor. RDF und XML ergänzen sich: RDF ist lediglich ein Modell für Metadaten und nimmt nur indirekt Bezug auf die Fragen zur Zeichenkodierung, die ja für Datentransport und Dateiablage essentiell ist (z.B. Internationalisierung, Zeichensätze). In diesen Fragen verlässt sich RDF auf die Unterstützung von XML. An dieser Stelle ist es wichtig, sich darüber im Klaren zu sein, dass XML nur eine mögliche Syntax für die Kodierung von RDF ist und dass alternative Arten der Darstellung erscheinen könnten. [26]

Die Grundlagen des RDF-Modells sind benannte Eigenschaften (Properties) und Werte von Eigenschaften (Values). RDF-Eigenschaften kann man sich als Attribute von Ressourcen und in diesem Zusammenhang als ein klassisches Ressource-Attribut-Wert Tripel vorstellen. Dieses Tripel entspricht in der Logik einer Aussage. Das Tripel wird im Datenmodell mit drei Objekttypen realisiert:

Ressourcen (Resources)

Alles, was mit RDF-Ausdrücken beschrieben wird, wird als Ressource bezeichnet. Eine Ressource kann z.B. ein einzelnes HTML-Dokument einer Web Site sein – etwa das HTML-Dokument <http://www.w3.org/Overview.html> – oder die ganze Web Site "<http://www.w3.org>". Eine Ressource kann aber auch ein Objekt sein, das nicht direkt im Web verfügbar ist; etwa ein gedrucktes Buch. Ressourcen werden mittels einer URI (Uniform Resource Identifier) identifiziert. Alles kann eine URI besitzen; die Erweiterbarkeit der URIs erlaubt die Einführung von Erkennungsmarken (Identifiers) für jede vorstellbare Entität.[26]

Eigenschaften (Properties):

Eine Eigenschaft ist ein spezieller Aspekt, eine Charakteristik, ein Attribut oder eine Beziehung, mit der eine Ressource beschrieben wird. Jede Eigenschaft besitzt eine spezielle Bedeutung, definiert ihre erlaubten Werte, die Typen von Ressourcen, die durch sie beschrieben werden, und ihre Beziehung zu anderen Eigenschaften. RDF-Eigenschaften können auch Beziehungen zwischen Ressourcen darstellen, weshalb ein RDF-Modell einem Entity-Relation-Diagramm ähnelt. Präziser: RDF-Schemata¹¹, die selbst Instanzen von RDF-Modellen sind, sind ER Diagramme. In der Terminologie eines objektorientierten Designs bedeutet dies: Ressourcen entsprechen Objekten und Eigenschaften entsprechen Instanzvariablen.[26]

¹¹ siehe dazu <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>

Aussagen (Statements):

Eine bestimmte Ressource zusammen mit ihrer benannten Eigenschaft und dem Wert, den die Eigenschaft für diese Ressource hat, ist eine RDF-Aussage. Diese drei individuellen Teile der Aussage nennt man passenderweise das Subjekt, das Prädikat und das Objekt.

Das Objekt einer Aussage (z.B. der Wert der Eigenschaft) kann eine andere Ressource sein (spezifiziert durch eine URI) oder ein Literal (eine einfache Zeichenkette oder ein einfacher Datentyp aus XML). In der Sprache von RDF kann ein Literal einen XML Inhalt haben, wird aber nicht weiter vom RDF-Prozessor ausgewertet.[26]

Darüber hinaus besteht RDF aus einem Klassensystem ähnlich wie bei objektorientierter Programmierung, durch das das zugrunde liegende Vokabular festgelegt wird. Eine Menge von Klassen nennt sich ein *Schema*. Die Klassen sind hierarchisch organisiert und bieten eine Erweiterungsmöglichkeit durch Bildung von Unterklassen. Auf diese Weise ist es nicht notwendig, das "Rad neu zu erfinden", um ein neues Schema zu erzeugen, das sich nur geringfügig von einem bereits existierenden unterscheidet.

Durch die Möglichkeit, Schemata gemeinsam zu nutzen, unterstützt RDF die Wiederverwendbarkeit von Metadaten. Entsprechend der Erweiterbarkeit von RDF werden Programme, die Metadaten verarbeiten, in der Lage sein, ihnen unbekannte Schemata anhand der Vererbungshierarchie hinauf zu verfolgen, bis sie auf ein Schema in der Hierarchie stoßen, das sie verarbeiten können. Sie werden dadurch in die Lage versetzt, sinnvolle Aktionen auf Metadaten anzuwenden, für deren Verarbeitung sie ursprünglich nicht entwickelt wurden. Die Möglichkeit der gemeinsamen Nutzung und der Erweiterbarkeit von RDF erlaubt es Metadatenautoren ebenfalls, verschiedene Vererbungen zu verwenden, um Definitionen zu vermischen, wodurch verschiedene Sichtweisen auf ihre Daten möglich werden. So kann z.B. ein Buch, das im Internet veröffentlicht wurde, auch die entsprechenden Eigenschaften einer Internetseite besitzen.

2.5.4 Beispiele

Zu Beginn soll ein einfaches Beispiel betrachtet werden:

Beispiel 4

Der Autor der Ressource <http://www.Helge-Reinsch.de/rdf.htm> ist Helge Reinsch.

Dieser Satz hat die Bestandteile:

Subjekt	http://www.Helge-Reinsch.de/ki2/rdf.htm
Prädikat	Autor
Objekt	Helge Reinsch

Die Aussage lässt sich in einem Diagramm in Form eines gerichteten benannten Graphen darstellen. In diesem Diagramm repräsentieren die Knoten – als Ovale gekennzeichnet – Res-

sources, die Pfeile repräsentieren benannte Eigenschaften und Knoten, die Zeichenketten darstellen, werden als Rechtecke gezeichnet.

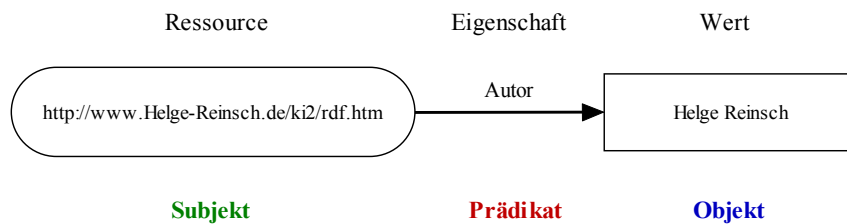


Abbildung 8: RDF-Aussage als Graph

Die Richtung des Pfeils ist wichtig. Der Pfeil beginnt immer am Subjekt der Aussage und endet am Objekt.

Im Folgenden ein Beispiel, das etwas mehr über den Autoren aussagt:

Beispiel 5

Die Person mit dem Namen Helge Reinsch, e-Mail: Helge.Reinsch@stud.uni-hannover.de, ist der Autor der Seite <http://www.Helge-Reinsch.de/ki2/rdf.htm>.

Der Sinn dieses Satzes ist, für den Wert der *Autor*-Eigenschaft ein strukturiertes Entity einzusetzen. Dieses Entity wird in RDF durch weitere Ressourcen repräsentiert. Der Satz gibt keinen Namen für diese Ressource an, d.h. sie ist anonym, sodass sie im folgenden Diagramm durch ein leeres Oval gekennzeichnet wird.

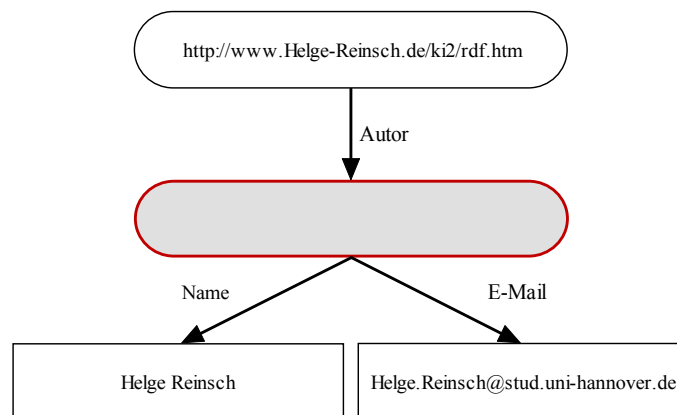


Abbildung 9: RDF-Aussage mit anonymer Ressource

Korrespondierend zu dem eben Gesagten, würde man das Diagramm folgendermaßen lesen:

Beispiel 6

Die Ressource <http://www.Helge-Reinsch.de/ki2/rdf.htm> hat den Autor IRGENDWAS und IRGENDWAS hat den Namen Helge Reinsch und die E-Mail-Adresse Helge.Reinsch@stud.uni-hannover.de.

Dem strukturierten Entity des vorherigen Beispiels kann ebenso ein eindeutiger Verweis zugeordnet werden. Um das Beispiel weiter zu führen, stelle man sich vor, dass die Immatrikulationsnummer (MatrNr) ein eindeutiger Verweis auf einen Studenten ist. Die URIs, die als eindeutige Verweise auf einen Studenten dienen, könnten die Form

<http://www.uni-hannover.de/MatrNr/1813885/>

haben.

Nun könnte die Aussage die Form erhalten, deren Repräsentation als Graph in Abbildung 10 dargestellt ist.

Die Ressource <http://www.Helge-Reinsch.de/ki2/rdf.htm> hat als Autor das Individuum, auf das die Immatrikulationsnummer 1813885 verweist. Dieses Individuum heißt Helge Reinsch und hat die E-Mail Adresse Helge.Reinsch@stud.uni-hannover.de.

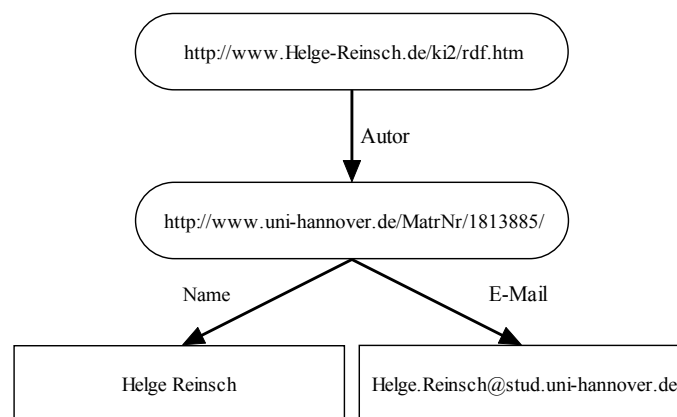


Abbildung 10: RDF-Aussage mit mehreren Ressourcen

2.5.4.1 RDF-Syntax

In RDF gibt es zwei mögliche Arten der Syntax. Zum einen die *Basic Serialization Syntax*, die jede Eigenschaft in einem eigenen Element ablegt, und die *Basic Abbreviated Syntax*, die es erlaubt, die Aussagen kompakter zu formulieren.

2.5.4.1.1 Basic Serialization Syntax

RDF-Aussagen treten selten allein auf. Viel üblicher ist es, dass mehrere Eigenschaften einer Ressource zusammen beschrieben werden. Die RDF/XML-Syntax wurde so entworfen, dass mehrere Aussagen über eine Ressource in einem `description`-Element gruppiert werden können. Das `description`-Element gibt in einem `about`-Attribut an, auf welche Ressource sich die Aussagen innerhalb des Elements beziehen. Sollte die Ressource bisher noch nicht existieren oder eine physikalische Ressource sein, die nicht über eine URI identifizierbar ist, so kann das `description`-Element diese mit einem solchen Verweis versehen, indem sie das

id-Attribut verwendet. Anders gesagt, das *id*-Attribut erzeugt eine neue Ressource, während das *about*-Attribut sich auf eine existierende Ressource bezieht. Ein einzelnes *description*-Element kann mehrere Eigenschaftselemente mit identischem Namen beinhalten. Jedes dieser Elemente fügt dem gerichteten Graphen eine weitere Kante hinzu.[26] Beispiel 4 sähe in dieser Notation wie folgt aus:

Beispiel 7

```
1 <?xml version="1.0"?>
2   <rdf:rdf
3     xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:s="http://description.org/schema">
5     <description about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
6       <s:autor>Helge Reinsch</s:autor>
7     </description>
8   </rdf:rdf>
```

Die erste Zeile (`<?xml version="1.0"?>`) ist die XML-Deklaration. Das `rdf`-Element in der zweiten Zeile ist lediglich eine Art Hülle, die die Grenzen innerhalb eines XML-Dokumentes markiert, zwischen denen sich die Aussagen befinden. In dem `rdf`-Element werden außerdem noch zwei Namensräume angegeben. Zum einen der Namensraum von RDF selbst (Zeile 3) und zum anderen ein fiktiver Namensraum, in dem das Vokabular für Beschreibungen festgelegt sein kann (Zeile 4).

Werte von Eigenschaften können aber nicht nur Zeichenketten, sondern auch wieder Ressourcen sein. Im Beispiel 8 wird von dieser Möglichkeit Gebrauch gemacht. Der Wert von `s:Autor` ist die Ressource `http://www.uni-hannover.de/MatrNr/1813885/` (Zeile 6).

Beispiel 8

```
1 <?xml version="1.0"?>
2   <rdf:rdf
3     xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
4     xmlns:s="http://description.org/schema">
5     <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
6       <s:Autor rdf:resource="http://www.uni-hannover.de/MatrNr/1813885/">
7     </rdf:Description>
8
9     <rdf:Description rdf:about="http://www.uni-hannover.de/MatrNr/1813885/">
10      <s:Name>Helge Reinsch</s:Name>
11      <s:Email>Helge.Reinsch@stud.uni-hannover.de</s:Email>
12    </rdf:Description>
13  </rdf:rdf>
```

Zum Schluss noch einmal ein Beispiel, das eine Verwendung von Dublin Core darstellt (Zeile 8–13). Es werden mehrere Namensräume für die Beschreibung einer Ressource verwendet.

Beispiel 9

```

1  <?xml version="1.0"?>
2  <rdf:rdf
3    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4    xmlns:s="http://description.org/schema"
5    xmlns:dc="http://purl.org/metadata/dublin_core#">
6
7    <rdf:Description rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
8      <s:autor>Helge Reinsch</s:autor>
9      <dc:Title>Semantic Web: RDF (Resource Description Framework)</dc:Title>
10     <dc:Subject>Dublin Core, Metadaten, Warwick Framework, RDF</dc:Subject>
11     <dc:Author>Helge Reinsch</dc:Author>
12     <dc:Date>2001-06-09</dc:Date>
13     <dc:Identifizier>http://www.Helge-Reinsch.de/ki2/</dc:Identifizier>
14     <dc:Language>de</dc:Language>
15   </rdf:Description>
16 </rdf:rdf>

```

2.5.4.1.2 Basic Abbreviated Syntax

Oft ist es wünschenswert, eine kompaktere Schreibweise zu haben. Diese Möglichkeit erhält man durch die *RDF Abbreviated Syntax* (abkürzende RDF-Schreibweise). Es gibt drei Formen dieser abkürzenden Schreibweise.

Die erste Form kann für Eigenschaften verwendet werden, die innerhalb eines `description`-Elements nur einmal vorkommen und deren Werte Zeichenketten sind. In diesem Fall können die Eigenschaften als Attribute des `description`-Elements geschrieben werden.[26]

Beispiel 10

```

1  <rdf:description
2    rdf:about="http://www.Helge-Reinsch.de/ki2/rdf.htm"
3    s:autor="Helge Reinsch"
4    s:title="Semantic Web: RDF (Resource Description Framework)"
5    s:date="2001-06-08" />

```

Diese Schreibweise hat gegenüber der Serialized Syntax einen Vorteil bei der Verwendung in HTML-Dokumenten. Ein Browser ignoriert ihm unbekannte Tags, zeigt jedoch den Inhalt zwischen dem Start- und End-Tag an. Da bei der abgekürzten Schreibweise die Inhalte als Attribute *innerhalb* des Tags annotiert werden, werden sie mitsamt dem Tag vom Browser ignoriert und nicht angezeigt.

Die zweite abkürzende Schreibweise wirkt sich auf verschachtelte `description`-Elemente aus. Diese Schreibweise kann verwendet werden, wenn der Wert einer Eigenschaft eine Ressource ist, die innerhalb der gleichen Instanz beschrieben wird. In Beispiel 6 kann folglich diese abkürzende Schreibweise angewendet werden. Wie man erkennt, wird in Zeile 5 statt der *Referenz* auf die zweite Ressource diese direkt annotiert.[26]

Beispiel 11

```

1 <rdf:rdf
2   xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
3   xmlns:s="http://description.org/schema/">
4
5   <rdf:description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
6     <s:autor rdf.resource="http://www.uni-hannover.de/MatrNr/1813885/" />
7   </rdf:description>
8
9   <rdf:description rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
10    <s:name>Helge Reinsch</s:name>
11    <s:email>Helge.Reinsch@stud.uni-hannover.de</s:email>
12  </rdf:description>
13 </rdf:rdf>

```

```

1 <rdf:rdf
2   xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
3   xmlns:s="http://description.org/schema/">
4
5   <rdf:description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
6     <s:autor>
7       <rdf:description rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
8         <s:name>Helge Reinsch</s:name>
9         <s:email>Helge.Reinsch@stud.uni-hannover.de</s:email>
10        </rdf:description>
11     </s:autor>
12  </rdf:description>
13 </rdf:rdf>

```

Die dritte Form der Abkürzung kann verwendet werden, wenn das `description`-Element eine `type`-Eigenschaft enthält. `Type`-Eigenschaften werden später noch genauer erklärt.

Die Aussage:

Beispiel 12

```

1 <rdf:rdf
2   xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
3   xmlns:s="http://description.org/schema/">
4
5   <rdf:description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
6     <s:autor>
7       <rdf:description rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
8         <rdf:type rdf.about="http://www.uni-hannover.de/MatrNr/1813885/person"/>
9         <s:name>Helge Reinsch</s:name>
10        <s:email>Helge.Reinsch@stud.uni-hannover.de</s:email>
11      </rdf:description>
12    </s:autor>
13  </rdf:description>
14 </rdf:rdf>

```

kann mit Hilfe der dritten Abkürzung geschrieben werden als:

```
1 <rdf:rdf
2   xmlns:rdf="http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
3   xmlns:s="http://description.org/schema/"
4   <rdf:description rdf.about="http://www.Helge-Reinsch.de/ki2/rdf.htm">
5     <s:autor>
6       <s:person rdf.about="http://www.uni-hannover.de/MatrNr/1813885/">
7         <v:name>Helge Reinsch</v:name>
8         <v:email>Helge.Reinsch@stud.uni-hannover.de</v:email>
9       </s:person>
10    </s:autor>
11  </rdf:description>
12 </rdf:rdf>
```

2.5.4.2 Aussagen über Aussagen

Zusätzlich zu der Eigenschaft, dass mit RDF Aussagen über Ressourcen gemacht werden können, bietet RDF auch noch die Möglichkeit an, Aussagen über Aussagen zu treffen. Diese sollen als *Aussagen höherer Ordnung* (High-order Statements) bezeichnet werden.[26]

Dazu betrachte man die Aussagen:

Helge Reinsch ist der Autor der Seite <http://www.Helge-Reinsch.de/ki2/rdf.htm>.

und die Aussage:

Wolf Siberski sagte, dass Helge Reinsch der Autor der Seite <http://Helge-Reinsch.de/ki2/rdf.htm> ist.

Mit der zweiten Aussage wird nichts über die Ressource <http://Helge-Reinsch.de/ki2/rdf.htm> ausgesagt. Stattdessen wurde eine Aussage über eine Aussage von Wolf Siberski gemacht. Um diese Fakten in RDF auszudrücken, muss die ursprüngliche Aussage als eine Ressource mit vier Eigenschaften modelliert werden. Im Bereich Wissensrepräsentation wird dies als *Vergegenständlichung* (reification) bezeichnet. Um Aussagen zu modellieren, stellt RDF folgende Eigenschaften zur Verfügung:

subject

Die *subject* Eigenschaft identifiziert die Ressource, die durch das modellierte Statement beschrieben wird. Das bedeutet, der Wert des Subjekts ist die Ressource, über die die ursprüngliche Aussage gemacht wurde. (<http://www.Helge-Reinsch.de/ki2/rdf.htm>).[26]

predicate

Die *predicate* Eigenschaft verweist auf die ursprüngliche Eigenschaft in der modellierten Aussage. Der Wert des Prädikats ist die Ressource, die durch die spezielle Eigenschaft in der ursprünglichen Aussage spezifiziert wurde (hier: `author`).[26]

object

Die *object* Eigenschaft verweist auf den Eigenschaftswert in der modellierten Aussage. Der Wert der *object* Eigenschaft ist das Objekt der ursprünglichen Aussage (hier: Helge Reinsch).[26]

type

Der Wert von *type* beschreibt den Typ der neuen Ressource. Alle "Vergegenständlichungen" sind Instanzen von `RDF:Statement`. Das bedeutet, dass sie eine *type* Eigenschaft haben, deren Objekt ein `RDF:Statement` ist. Die *type* Eigenschaft wird allgemeiner dazu benutzt, Typen beliebiger Ressourcen zu deklarieren. [26]

Notiert sieht die Beispielaussage so aus:

Beispiel 13

```
<rdf:rdf
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://description.org/schema/">
  <rdf:description>
    <rdf:subject resource="http://www.Helge-Reinsch.de/ki2/rdf.htm" />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Helge Reinsch</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
    <a:attributedTo>Wolf Siberski</a:attributedTo>
  </rdf:description>
</rdf:rdf>
```

2.6 Datenverteilung

2.6.1 Peer-to-Peer-Netzwerke

Das Internet, wie es im Allgemeinen verwendet wird, basiert auf der Client/Server-Architektur. Darunter versteht man den Aufbau von Internetanwendungen, der die Funktionalität in zwei separate Komponenten trennt.[24]

Client:

Der Client wird durch eine Client-Anwendung repräsentiert, die von Internet-Anwendern genutzt wird, um einen Dienst eines Servers in Anspruch zu nehmen.

Server:

Der Server wird durch eine Server-Software realisiert und stellt die vom Client genutzte Funktionalität (Dienst) zur Verfügung.

Client und Server verständigen sich dabei über das Internet im Rahmen eines Protokolls, das sie selbst definieren. Z.B. „sprechen“ Web-Server und Web-Browser – als deren Clients – die Sprache des HTTP-Protokolls, Mail-Server und Mail-Clients die des SMTP-Protokolls. In der Regel läuft die Kommunikation zwischen Client und Server so, dass der Client vom Server bestimmte Dienste in Anspruch nimmt, beispielsweise eine Datei oder eine andere Information anfordert. Der Server antwortet mit der gewünschten Information oder einer Fehlermeldung, falls er die Anfrage des Clients nicht beantworten kann. Charakteristisch ist hier, dass der Server keinen Versuch unternimmt, in gleicher Weise einen Dienst vom Client anzufordern. Dies kennzeichnet das besondere Verhältnis zwischen Client und Server und drückt sich auch in einer gewissen Asymmetrie des Protokolls aus.[24]

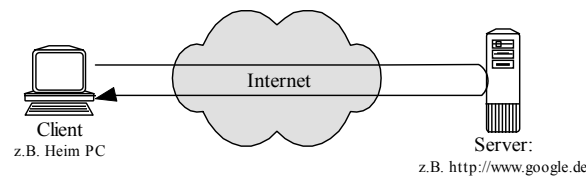


Abbildung 11: Client/Server-Architektur

Im Gegensatz zur Client/Server-Architektur können in einem Peer-to-Peer-Netzwerk die Computer – die in diesem Zusammenhang Peers genannt werden – zeitgleich sowohl als Client als auch als Server fungieren. Alle Rechner innerhalb des Netzwerkes sind gleichberechtigt und tauschen die Daten dezentral aus. Dabei stellt jeder Computer individuelle Daten bereit. Die Daten liegen folglich nicht auf einem zentralen Server vor, sondern verteilt auf den einzelnen Rechnern, die mit dem Netzwerk verbunden sind. Die Tatsache, dass die Daten nicht auf zentralen Servern liegen, sondern direkt auf den einzelnen Rechnern ist der entscheidende Unterschied gegenüber der Client/Server-Architektur. Der Aufwand, Daten in einem solchen Netzwerk zur Verfügung zu stellen, ist sehr gering. Während in der Client/Server-Architektur die Daten auf den zentralen Server kopiert werden müssen, reicht in der Peer-to-Peer-Architektur bereits die Installation einer Peer-Software, die die Verbindung zu dem Netzwerk herstellt. Die Bereitstellung der Informationen kann anschließend direkt vom lokalen Rechner geschehen.

Ein Beispiel für ein Peer-to-Peer-Netzwerk ist das dezentrale Gnutella¹², das anfänglich im Jahr 2000 von Nullsoft – einer Tochter von AOL – später dann von Gene Kan zum heutigen Gnutella-Netzwerk weiterentwickelt wurde.

¹² siehe auch <http://www.gnutella.com/>

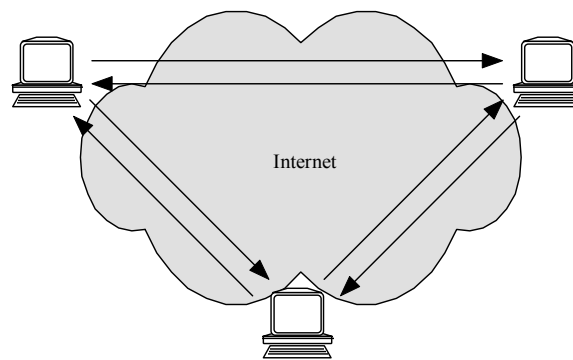


Abbildung 12: Peer-to-Peer-Architektur

2.6.2 Edutella

2.6.2.1 Allgemeines

Jede Universität besitzt eine Vielzahl von Lehrmaterialien verteilt über die Institute. Diese Lehrmaterialien werden von den Instituten nur ungern zentralen Einrichtungen überlassen, aus Angst, die Kontrolle darüber zu verlieren. Um die Lehrmaterialien bereitzustellen und trotzdem die Kontrolle zu behalten, könnten die Institute diese auf eigenen Servern innerhalb eines Peer-to-Peer-Netzwerkes bereitstellen, auf die anfragegesteuert zugegriffen werden kann.[27]

In einem typischen Peer-to-Peer E-Learning-Szenario fungieren die einzelnen Universitäten nicht nur als Anbieter, sondern ebenfalls als Nachfrager von Inhalten. Als Anbieter in einem Peer-to-Peer-Netzwerk verlieren sie die Kontrolle über ihre eigenen Lehrmaterialien nicht und stellen sie dennoch innerhalb des Netzwerkes den anderen Mitgliedern zur Verfügung.[28]

Als Nachfrager ziehen sowohl Lehrende als auch Studierende Nutzen aus dem Zugang nicht nur zu lokalen als vielmehr zu einem ganzen Netzwerk von Lehrmaterialien. Sie können durch die Verwendung von Anfragen über die gesamten Metadaten des Netzwerkes die benötigten Quellen finden und so mit einer großen und aktuellen Auswahl von Lehrmaterialien arbeiten. Um Lehrmaterialien in einem solchen Netzwerk zu finden, werden die Veröffentlichungen mit Metadaten versehen, um das Thema und den Inhalt der Arbeit zu charakterisieren. Da Lehrmaterialien oftmals in einem hohen Maße fachgebiets- und quellenspezifisch sind, wird eine Auszeichnungssprache für Metadaten benötigt, die die Interoperabilität und die Wiederverwendbarkeit von Lehrmaterialien und einen großen Bereich der Quellen von Lehrmaterialien unterstützt.[27]

2.6.2.2 Edutella Peer-to-Peer

Das Edutella-Netzwerk ist in der Lage, heterogene Peers mit unterschiedlichen Datenquellen, verschiedenen Anfragesprachen und unterschiedlichen Metadaten-Schemata zu integrieren. [27] Darüber hinaus sind die Peers sehr verschieden bzgl. der zeitlichen Abschnitte, in denen sie mit dem Netzwerk verbunden sind. Z.B. könnte ein Edutella-Peer eine große Datenbank repräsentieren, die zu 99% verfügbar ist, oder einen anderen Peer, der nur gelegentlich online ist und eine geringe Menge an dateibasierten Metadaten zur Verfügung stellt.[29]

Eines der wichtigen Charakteristika von RDF-Metadaten ist die Fähigkeit, verteilte Annotationen für ein und dieselbe Ressource zu verwenden. Im Gegensatz zu klassischen Datenbanksystemen ist es nicht notwendig, dass alle Annotationen für eine Ressource auf einem Server gespeichert sind. Ein Server könnte Metadaten speichern, die Eigenschaften aus dem Dublin Core Metadaten Schema enthalten, ein anderer könnte für die gleiche Ressource Metadaten bereitstellen, die andere Eigenschaften verwenden. Die Fähigkeit der verteilten Bereitstellung von Metadaten macht RDF besonders nützlich für die Konstruktion eines verteilten Repositories.[29]

2.6.2.2.1 Schemabasierte Peer-to-Peer-Netzwerke

Gegenwärtige Peer-to-Peer-Netzwerke unterstützen nur begrenzte Metadatensätze, wie zum Beispiel einfache Dateinamen. Auf diese Weise kann man zwar relativ einfach innerhalb von Gnutella nach einem Lied von Madonna suchen, aber alle Symphonien zu finden, die von Beethoven komponiert wurden, wird zu einer wesentlich schwierigeren Aufgabe. Um dieses Problem mit eingeschränkten und festen Metadaten der herkömmlichen Peer-to-Peer-Netzwerke zu umgehen und verteilte Repositories zu ermöglichen, muss man in Richtung komplizierterer, den so genannten *schemabasierten Peer-to-Peer-Netzwerken* gehen. Schemabasierte Peer-to-Peer-Netzwerke beruhen auf Peers, die explizite Schemata verwenden, um ihre Dateninhalte zu beschreiben und deren Metadaten auf heterogenen Schemata basieren können. Edutella ist bestrebt, einen Zugang zu verteilten Ressourcen über ein Peer-to-Peer-Netzwerk bereitzustellen. Innerhalb des Edutella-Netzwerkes werden die Ressourcen nicht durch willkürliche Begriffe beschrieben, sondern es werden RDF-Schemata und RDF-Metadaten verwendet. Um auf die Inhalte, die im Edutella-Netzwerk gespeichert sind, zuzugreifen, wird die Anfragesprache RDF-QEL verwendet, die im Abschnitt 2.6.2.2.15 näher beschrieben wird.[29]

2.6.2.2.2 Super-Peers-Topologie

In Peer-to-Peer-Netzwerken, die ein einfaches Broadcast für die Verteilung der Anfragen verwenden, verbrauchen die Anfragen sehr viel Bandbreite, da sie zu allen Peers geschickt werden. Außerdem steigt der Verbrauch von Prozessorzeiten mit der Anzahl der zu verarbeitenden Anfragen. Um die Bandbreitenauslastung und die Verarbeitungszeiten von Anfragen innerhalb der Peers zu verbessern, verwendet das Edutella-Netzwerk eine andere Strategie zur Verteilung von Anfragen. Grundlage der Verteilung der Anfragen an die einzelnen Peers bilden die verwendeten Metadatenschemata. Anfragen und Antworten werden durch RDF-Metadaten repräsentiert. Diese Metadaten, zusammen mit den inhaltsbeschreibenden Metadaten, können dazu verwendet werden, explizite Routingindizes zu bestimmen, die eine komplexere Verteilung von Anfragen innerhalb des Netzwerkes erlauben. Diese Routingindizes werden nicht in den Peers abgelegt, da dies wieder eine zusätzliche Belastung des Peers bedeuten würde, sondern vielmehr unterstützt Edutella die sog. *Super-Peer-Topologie*. [29]

In der Super-Peer-Topologie verbindet sich jeder Peer mit genau einem Super-Peer. Dieser Super-Peer verbindet sich mit anderen Super-Peers und bildet so den Backbone des Super-Peer-Netzwerkes. Die Super-Peers sind für die Verwaltung der Routingindizes verantwortlich und entscheiden, welche Anfrage an welchen Peer oder an welche Super-Peers weitergeleitet werden soll.[29]

2.6.2.2.3 HyperCuP-Topologie

Die Super-Peers sind untereinander in der sog. *HyperCuP-Topologie* angeordnet. Der HyperCuP-Algorithmus ist in der Lage, Peers in einem Peer-to-Peer-Netzwerk in einer rekursiven Graphenstruktur, aus der Familie der *Cayley-Graphen*, anzuordnen. Diese Anordnung erlaubt, das Routing von Anfragen mit einer minimalen Anzahl von Zwischenschritten an die einzelnen Peers weiter zu leiten. Abbildung 13 zeigt eine HyperCuP-Topologie für acht Super-Peers. Eine ausführliche Beschreibung der HyperCuP-Topologie findet sich unter [30] und [29].

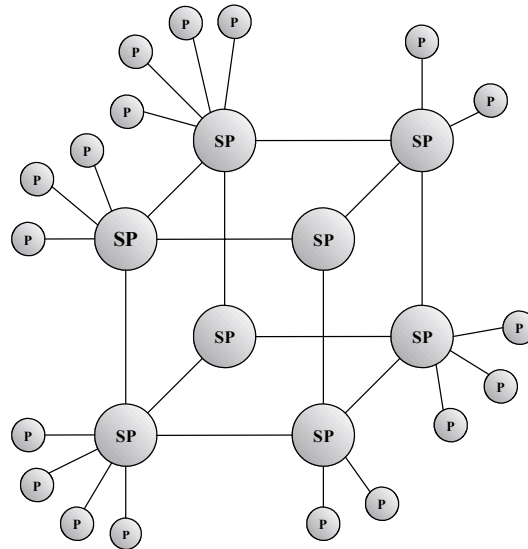


Abbildung 13: Peer Verbindungen im Super-Peer Backbone¹³

2.6.2.2.4 Edutella-Peers

Die Edutella-Peers sind in hohem Maße heterogen in den Funktionalitäten, die sie anbieten. Um es den Peers zu ermöglichen, sich mit dem Edutella-Netzwerk zu verbinden, werden Edutella-Wrapper verwendet, die Anfragen und Ergebnisse aus dem Edutella Anfrage- und Ergebnis-Austauschformat in die lokalen Formate des Peers übersetzen und umgekehrt. Darüber hinaus verbinden die Wrapper den Peer über eine JXTA¹⁴ basierte Bibliothek mit dem Edutella-Netzwerk.[27]

Um die Anfrage zu handhaben, verwendet der Edutella-Wrapper das Edutella-interne Anfrage-Austauschformat und -Datenmodell für die Anfragen- und Ergebnisrepräsentation. Zur Kommunikation mit dem Edutella-Netzwerk übersetzt der Wrapper das lokale Datenmodell in das Edutella-Datenmodell *EQM (Edutella Query Model)*. [27]

¹³ entnommen aus [29]

¹⁴ siehe dazu Abschnitt 2.6.2.3

2.6.2.2.5 Edutella Query Model (EQM)

Das *Edutella Query Model (EQM)* ist das Datenmodell, das Anfragen und deren Ergebnisse innerhalb des Edutella-Netzwerkes repräsentiert. Es liegt der Kommunikation zwischen einem Edutella-Consumer – einem Peer, der Daten anfragt – und einem Edutella-Provider – ein Peer, der seine Daten zur Verfügung stellt – zugrunde und enthält sowohl die Anfrage an einen Peer wie auch die vom Edutella-Provider zurückgegebene Antwort. Dem Edutella-internen Anfragemodel liegt das *Datalogkalkül* zugrunde, das der nächste Abschnitt beschreibt.

2.6.2.2.6 Datalog

Datalog ist eine Sprache für logische Daten und Regeln, mit der eine Anfrage an eine Wissensbasis mit Fakten oder Aussagen gestellt werden kann. Eine Wissensbasis von Fakten enthält lediglich binäre Relationen, beschrieben durch `predicate(subject, object)` oder als ternäre Aussage `s(subject, predicate, object)`.

Datalog ist eine nicht-prozedurale Anfragesprache, die auf Horn-Klauseln basiert. Eine Horn-Klausel ist eine Disjunktion von Literalen mit höchstens einem nicht-negativen Literal. Zusätzlich zu reinen Fakten sind Literale auch Prädikate, die eine beliebige Relation zwischen einer beliebigen Anzahl von Konstanten und Variablen ausdrücken.[28]

2.6.2.2.7 Prädikate

Prädikat-Ausdrücke sind die elementaren Konstrukte in *Datalog*. Ein Ausdruck für ein Prädikat besteht aus einem *Prädikat-Symbol*, gefolgt von einer Liste von *Argumenten*. Zum Beispiel:

```
equals(X, Y)
title(book, "Artificial Intelligence")
```

Argumente können Werte (wie "*Artificial Intelligence*"), Konstanten (wie `book`) und Variablen (wie `X`) sein. Werte werden immer in Anführungszeichen gesetzt, Konstantenbezeichnungen immer in kleinen und Variablen immer in großen Buchstaben geschrieben. Prädikate werden gelegentlich auch als atomare Formeln bezeichnet, ebenso die Namen von Konstanten.[32]

2.6.2.2.8 Fakten

Daten werden in *Datalog* durch Prädikat-Ausdrücke beschrieben, die nur Werte und Konstantenbezeichnungen als Argumente haben.¹⁵

```
father(matthias, peter).
father(emma, peter).
name(matthias, "Matthias").
name(peter, "Peter").
```

¹⁵ Das Beispiel wurde aus [32] entnommen

Hier wird festgelegt, dass Peter der Vater von Matthias und Emma ist, und dass Matthias den Namen "Matthias" und Peter den Namen "Peter" trägt. Diese Art der Prädikat-Ausdrücke werden *Fakten* genannt.[31]

2.6.2.2.9 Anfragen

Wenn ein Prädikat-Ausdruck eine oder mehrere Variablen enthält, dann wird dieser Ausdruck als Anfrage-Literal bezeichnet.

```
father(matthias,X)
```

Anfrage-Literale werden als wahr angesehen, wenn sich die Variable durch Konstanten oder Werte so ersetzen lässt, dass sich ein Tupel ergibt, das mit den Bedingungen der Anfrage übereinstimmt. Dieser Prozess wird *Variablen-Bindung* genannt.[32] Um eine Anfrage darzustellen, wird die folgende Notation verwendet:

```
?- father(matthias,X)
```

Diese Anfrage bedeutet: "Suche alle Väter von Matthias". Unter Verwendung der obigen Fakten würde die Anfrage das Ergebnis *wahr* ergeben, wenn die Variable X an Peter gebunden ist. Im Gegensatz zu Prolog liefert eine Anfrage in Datalog alle möglichen Variablenbindungen zurück und nicht nur die erste mögliche. Eine Liste von Anfrage-Literalen, die durch Kommata getrennt ist, wird als Konjunktion aufgefasst.

```
?- name(Person, "Peter"), father(Child, Person)
```

Diese Anfrage findet alle Personen, deren Name Peter ist *und* die Väter sind.[32]

2.6.2.2.10 Regeln und datenfreie Prädikate

In Datalog ist es möglich, Prädikate zu erzeugen, die nicht Teil der Daten, aber von ihnen abgeleitet sind. Dies geschieht, indem für diese Prädikate Regeln erzeugt werden, die ihre Validität prüfen.[32]

```
Regel:          sibling(X, Y) :- father(X, Z), father(Y, Z).
Anfrage:       ?- sibling(X, Y)
```

Diese Regel besagt, dass X und Y Geschwister sind, wenn sie den gleichen Vater haben. Der linke Teil dieser Regel wird *Regelkopf* genannt, der rechte *Regelinhalt*. Die Anfrage aus dem Beispiel gibt als Ergebnis Matthias und Emma zurück. Allerdings wurde bei der Formulierung der Regel übersehen, dass es sich bei Personen ebenfalls um Geschwister handelt, wenn sie die gleiche Mutter haben. Daher muss noch eine weitere Regel ergänzt werden, die diesen Umstand berücksichtigt.

```
sibling(X, Y) :- father(X, Z), father(Y, Z).
sibling(X, Y) :- mother(X, Z), mother(Y, Z).
```

Regeln desselben Prädikats verhalten sich zueinander, als wären sie durch ein logisches ODER verbunden. Bei zwei Personen handelt es sich also um Geschwister, wenn sie den gleichen Vater *oder* die gleiche Mutter haben.[32]

2.6.2.2.11 Rekursionen

Es ist erlaubt, dass sich Prädikate rekursiv selbst aufrufen. So könnten Vorväter durch folgende Regeln definiert werden:

```
forefather(X, Y) :- father(X, Y).
forefather(X, Y) :- forefather(X, Z), father(Z, Y).
```

Auf diese Weise würde die Anfrage

```
?- forefather(matthias, Y)
```

sämtliche Vorväter von Matthias finden.[32]

2.6.2.2.12 Auswertung

Anfragen können mit einem Satz von Regeln kombiniert werden, um ihre Ausdrucksfähigkeit zu erhöhen. Eine solche Anfrage wird als *Prozedur* bezeichnet. Um einen besseren Überblick über die Ausdrucksfähigkeit von Datalog-Prozeduren und den Verlauf der Auswertung zu bekommen, soll das folgende Beispiel untersucht werden:

```
h2 ('Artificial Intelligence')
    :- s(Y,title , 'Artificial Intelligence').
h2 (Z)      :- s(Y, title, Z).
h1 (Y)      :- s(X, type, Z), h2 (Y).
?- h1(X).
```

Dies ist eine Datalog-Prozedur mit verschachtelten Regeln. Sie besteht aus einer Menge von Regeln und der eigentlichen Anfrage. Bei der Ausführung der Anfrage beginnt Datalog mit dem ersten Anfrage-Literal der Anfrage. Dieses sagt aus, dass nach allen Regeln gesucht werden soll, die mit $h1(X)$ benannt sind. Daraufhin findet Datalog $h1(Y)$ als einzige Regel, deren Kopf mit dem Anfrage-Literal $h1(X)$ übereinstimmt. Jedes Auftauchen der Variable Y innerhalb der Regel wird durch die Variable X ersetzt. Dieser Prozess wird Unifikation genannt.

Als nächstes versucht Datalog, die Regel zu verifizieren, indem es alle Anfrage-Literale aus der Regel verarbeitet. Das erste ist ein Anfrage-Literal, das auf keinen weiteren Regelkopf zutrifft. Das zweite Literal stimmt allerdings mit zwei Regelköpfen überein. Zum einen $h2('Artificial Intelligence')$ und zum anderen $h2(Z)$. Daher wird $h2(Y)$ in der Regel $h1(Y)$ durch die Disjunktion der Regelinhalte der beiden $h2$ Regeln unter Anwendung der Unifikation ersetzt. Dieser Prozess wird Substitution genannt.

Am obigen Beispiel konnte man erkennen, wie Datalog arbeitet: Es verarbeitet die Anfrage-Literale eines nach dem anderen. Jedes Mal, wenn übereinstimmende Regelköpfe gefunden werden, werden diese verarbeitet. Während Datalog dies tut, folgt es dem Konzept von Substitution und Unifikation.

2.6.2.2.13 Systematisierung verschiedener Syntax-Level

In einem heterogenen System wie dem Edutella-Netzwerk stellt sich bei der Informationssuche die Frage, wie Anfragen zwischen den einzelnen Peers ausgetauscht werden sollen. Da in einem heterogenen Netzwerk nicht davon ausgegangen werden kann, dass alle Peers die glei-

che Anfragesprache verwenden, bedarf es einer standardisierten Anfrage-Austausch-Sprache, die als Vermittler zwischen den einzelnen Peers fungiert.

Darüber hinaus führen systembedingte Unterschiede der Peers dazu, dass nicht jeder Peer in der Lage ist, den gleichen Sprachumfang für Anfragen zur Verfügung zu stellen. So können Peers, denen eine Datenbank zugrunde liegt, komplexere Anfragen beantworten, als zum Beispiel Peers, die Stichwörter aus einer Datei verwenden. Damit ein Peer entscheiden kann, ob er in der Lage ist, eine Anfrage zu beantworten, muss diese Anfrage innerhalb der Anfrage-Austausch-Sprache in geeigneter Form charakterisiert werden.

Im Folgenden sollen nun zunächst die verschiedenen Kriterien für Anforderungen an eine Anfrage-Austausch-Sprache vorgestellt werden und anschließend ein System, das es ermöglicht, eine Anfrage in standardisierter Form einzuordnen.

2.6.2.2.14 Kriterien zur Systematisierung

Für die Charakterisierung von Anfrage-Austausch-Sprachen haben sich die folgenden Kriterien als nützlich erwiesen:

Standard Semantik und fehlerfreie RDF-Serialisierung

Eine einfache und standardisierte Semantik einer Anfrage-Austausch-Sprache ist wichtig, da Übersetzungen von und in diese Sprache innerhalb eines Edutella-Peers die Semantik der originalen Anfrage beibehalten müssen. Darüber hinaus muss eine fehlerfreie Darstellung von Anfragen in RDF möglich sein, um diese zwischen den Peers zu transportieren und innerhalb der Peers zu speichern.[27]

Ausdrucksfähigkeit

Die Ausdrucksfähigkeit gibt darüber Auskunft, inwieweit Anfragen verstanden werden können (einfache graphbasierte Anfragen, SQL-Anfragen oder vielleicht sogar Anfragen an Inferenzmaschinen). Es ist also wichtig, dass die Austauschsprache sowohl in der Lage ist, einfache Anfragen zu formulieren, die von Anbietern direkt verwendet werden können, sowie einem fortschrittlichen Peer die Möglichkeit bietet, seine gesamten Fähigkeiten einzusetzen.[27]

Anpassungsfähigkeit (an unterschiedliche Formalismen)

Eine Anfragesprache muss neutral bezüglich unterschiedlicher semantischer Repräsentationen sein. Sie muss in der Lage sein, ein Prädikat mit vordefinierter Semantik (wie `rdfs:subClassOf`) zu verwenden, ohne eine eingebaute Semantik zu besitzen.[27]

Umformbarkeit

Das zugrunde liegende Modell für eine Anfrage-Austausch-Sprache muss auf einfache Weise in viele verschiedene Anfragesprachen sowohl importiert als auch exportiert werden können.[27]

2.6.2.2.15 RDF-QEL

Um die zuvor dargestellten Kriterien zu handhaben, wurden verschiedene Stufen von Austauschsprachen definiert. RDF-QEL basiert auf der Datalogsemantik und definiert fünf Sprachstufen. Die einfachste Anfrage ist eine Rule-less Query; sie erlaubt reine konjunktive Anfragen und kann als einfacher RDF-Graph dargestellt werden, wobei die komplexeren Anfragen mehr ausdrücken können, als es RDF vermag und deshalb in vergegenständlichten (reified) RDF-Aussagen dargestellt werden müssen (wie z.B. disjunktive Anfragen, die das relationale Kalkül beinhalten oder linear-rekursive Anfragen, die nicht kooperatives Datalog darstellen). Alle Sprachen können jedoch durch das EQM dargestellt werden.

Regelfreie Anfrage (Rule-less Query)

Regelfreie Anfragen werden motiviert durch Einfachheit und Lesbarkeit. Anfragen werden durch einfache RDF-Graphen repräsentiert, die exakt die gleiche Struktur haben wie der Antwortgraph. Jede Anfrage in Form eines RDF-Graphen kann als Konjunktion interpretiert werden, die gegen die Wissensbasis geprüft wird.[32]

Konjunktive Anfrage (Conjunctive Query)

Bei diesem Sprachlevel werden die regelfreien Anfragen, die in sich bereits konjunktiv sind, um Regeln erweitert. Dabei ist die Ausdrucksfähigkeit insoweit eingeschränkt, dass nur eine Regel pro Prädikat erlaubt ist. Auf diese Weise können auch weiterhin nur Konjunktionen dargestellt werden.[32]

Disjunktive Anfrage (Disjunctive Query)

Auf diesem Anfragelevel werden mehrere Regeln pro Prädikat zugelassen, aber keine Rekursionen. Diese Sprache enthält nicht länger reine Aussagen und kann nicht mehr direkt in RDF ausgedrückt werden ohne über die logische Kombination von RDF-Tripeln zu sprechen. Zu diesem Zweck wird ein RDF Konstrukt mit Namen reification verwendet. Das "Vergegenständlichen" einer RDF-Aussage bezieht die Erstellung eines Modells der RDF-Tripel in Form von RDF-Ressourcen vom Typ Statement mit ein. Diese Ressource hat als Eigenschaften `rdf:subject`, `rdf:predicate`, `rdf:object` der modellierten RDF-Tripel. Diese reified statements sind die Bausteine für jede Anfrage und werden mit Hilfe eines UNDOER-Baumes verbunden.[32]

Linear-Rekursive Anfrage (Linear Recursive Query)

Bei diesem Anfragetyp sind Rekursionen erlaubt, die allerdings linear sein müssen. Auf diese Weise können Definitionen für transitive Abschlüsse und lineare rekursive Anfragen ausgedrückt werden, und die Anfragen sind kompatibel zu den Möglichkeiten von SQL99.[32]

Generelle rekursive Anfrage (General Recursive Query)

Die Rekursion dieser Anfragen muss nicht mehr linear sein. Sie erfordern daher ein Äquivalent zu Datalog- oder Prolog-Prozessoren für die Ausführung.[32]

Eine Einführung in RDF-QEL und die darin enthaltenen Datalog Elemente sind unter [28] zu finden. Semantik und Syntax können unter [32] nachgelesen werden.

2.6.2.3 JXTA-Project

Als Grundlage für das Peer-to-Peer-Netzwerk, auf das Edutella aufbaut, wurde das Projekt JXTA ausgewählt. JXTA ist ein Open-Source-Projekt von Sun Microsystems, dessen Ziel es ist, eine Grundlage für die Implementierung von Peer-To-Peer-Anwendungen zu schaffen. Sun arbeitet dabei mit Firmen zusammen, die sich dem Open-Source-Modell verpflichtet haben und in Peer-To-Peer-Technologie investieren. Sie teilen ihre Ergebnisse miteinander und ermöglichen es Entwicklern, auf diese Weise von Erkenntnissen anderer zu profitieren und dadurch eigene neue fachgebietspezifische Peer-Services und -Anwendungen zu entwickeln.[33]

2.6.2.4 Das JXTA Peer-to-Peer System

Das Projekt JXTA stellt eine Sammlung einfacher, kleiner und flexibler Mechanismen bereit, die den Aufbau eines Peer-to-Peer-Netzwerkes auf jeder Plattform unterstützen. JXTA verwendet offene Standards wie XML, Java und Schlüsselkonzepte, wie die Möglichkeit, in Shells Kommandos miteinander durch Pipes zu verbinden, um komplexe Aufgaben zu lösen. Die Abbildung 14 stellt das JXTA zugrunde liegende Layerkonzept dar.

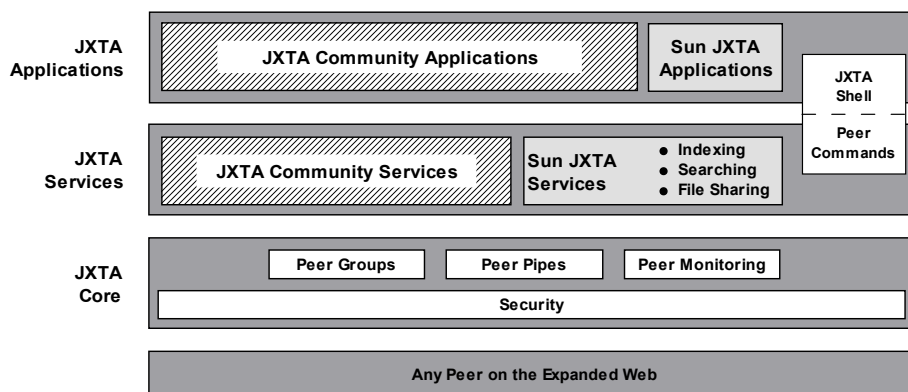


Abbildung 14: JXTA-Layerkonzept¹⁶

Im Kern – JXTA Core – der Funktionalitäten stehen die Möglichkeiten, Peer-Gruppen zu erzeugen und zu löschen, die Gruppen potentiellen Mitgliedern bekannt zu geben und anderen zu ermöglichen, Gruppen zu finden, sich ihnen anzuschließen bzw. sie zu verlassen.[35]

¹⁶ entnommen aus [34]

Der JXTA Core besteht aus:

Peer groups

stellen eine Zusammenfassung von Peers dar. Es werden Mechanismen zum Erzeugen und Löschen von Peers, Erlangung einer Mitgliedschaft, Veröffentlichung und Auffinden anderer Peer-Gruppen und Peer-Knoten zur Verfügung gestellt sowie Funktionen zur Kommunikation, für Sicherheit und gemeinschaftliche Benutzung von Inhalten.[35]

Peer pipes

stellen Kommunikationskanäle zwischen Peers bereit. Nachrichten, die innerhalb von Pipes verschickt werden, werden in XML formuliert.[35]

Peer monitoring

ermöglicht das Überwachen von Verhalten und Aktivität eines Peers in einer Peer-Gruppe und kann dazu verwendet werden, Peer-Management-Funktionen zu implementieren inklusive Zugriffskontrolle, Prioritätenfestlegung, Messung von Datentransfer und Bandbreitenausgleich.[35]

So wie die verschiedenen Bibliotheken in UNIX-Systemen Funktionen auf einem höheren Level als den des Kernels anbieten, erstrecken sich JXTA-Services über die Fähigkeiten des Kernels hinaus und erleichtern die Anwendungsentwicklung. Die Möglichkeiten, die in diesem Layer zur Verfügung gestellt werden, beinhalten Mechanismen zur Suche, gemeinsamen Nutzung und Indizierung von Inhalten. Darüber hinaus können Peer-Anwendungen erstellt werden, die auf diesen Services basieren.[35]

Im Application Layer sind die eigentlichen Anwendungen des Peer-to-Peer-Netzwerkes – wie auch Edutella – angesiedelt.[35]

2.7 Datentransformation

2.7.1 XSL (eXtensible Style Language)

XSL ist eine Sprache zur Formulierung von Stylesheets. Das Stylesheet legt fest, wie das Layout des Inhaltes des Quelldokumentes für verschiedene Ausgabemedien – wie Webbrowser, PDAs, gedruckte Seiten oder Bücher gestaltet werden soll. Ein XSL-Prozessor wendet das XSL-Stylesheet auf die strukturierten Daten eines XML-Dokuments an und erzeugt so eine Datei im gewünschten Repräsentationsformat.[38]

XSL stellt zwei Funktionen bereit. Zum einen die *Transformation von XML-Dokumenten* und zum anderen einen *Sprachschatz zur Formatierung von XML-Dokumenten*, wobei Transformation und Formatierung voneinander unabhängig sind. So kann man z.B. ein XML-Dokument in ein HTML-Dokument konvertieren, und dabei das XSL-Formatobjekt völlig außer Acht lassen.

Bei XSL können im Wesentlichen drei Komponenten unterschieden werden:

1. **XSL** (eXtensible Style Language): XML-Elemente zur Formatierung
2. **XSLT** (XML Style Language Transformation): Überführung eines hierarchischen XML-Dokumentes in ein anderes hierarchisches Dokument
3. **XPath** (XML Path Language): dient zum Zugriff auf bestimmte Teile eines XML-Dokumentes

In dieser Arbeit wird lediglich vom XSL-Transformationsmechanismus Gebrauch gemacht, um ein XML-Dokument in ein RDF-Dokument zu übersetzen. Es wird daher auf eine ausführliche Beschreibung von XSL verzichtet. Daher sollen in diesem Zusammenhang nur einige Erläuterungen gebracht werden, die zum Verständnis der in dieser Arbeit verwendeten Stylesheets beitragen. Eine ausführliche Beschreibung des Sprachumfanges findet sich unter [36].

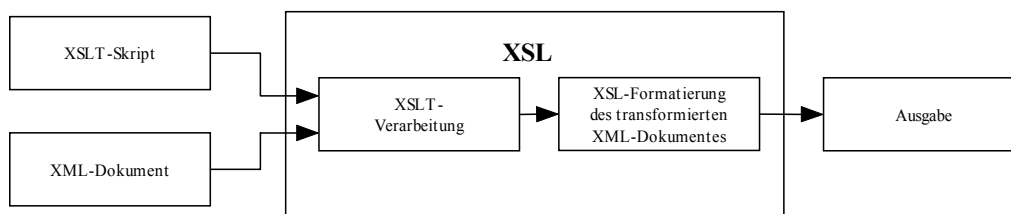


Abbildung 15: Ablauf einer XSL-Transformation

Die Transformation – formuliert in XSLT – beschreibt die Umwandlung eines hierarchisch organisierten Dokumentes in ein neues hierarchisch organisiertes Dokument. Bei der Transformation liest der Prozessor sowohl das XML-Quelldokument als auch das XSLT-Stylesheet ein (Abbildung 15). Anschließend werden die in dem Stylesheet enthaltenen Anweisungen zur Transformation auf das Quelldokument angewendet und so ein neues Dokument erstellt, das im abschließenden Schritt formatiert und ausgegeben werden kann.

Stylesheets können als eine Sammlung von Vorlageregeln betrachtet werden, die jeweils aus einem *Muster* und aus einer *Schablone* bestehen. Das Muster beschreibt, welche Elemente des XML-Quelldokumentes von der Regel verarbeitet, und die Schablone, welche neuen Strukturen aus dem Element erzeugt werden sollen. Als Beispiel soll die folgende Erweiterung von Beispiel 2 dienen.

Beispiel 14

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<books>
  <item>
    <author>
      J.R.R. Tolkien
    </author>
    <title lang="de">
      Der Herr der Ringe
    </title>
    <title lang="en">
      Master of the ring
    </title>
  </item>
</books>
```

```

</title>
<publisher>
  Klett-Cotta
</publisher>
<isbn>
  3608935444
</isbn>
</item>
<item>
  <author>
    J.R.R. Tolkien
  </author>
  <title lang="de">
    Der Hobbit
  </title>

  <title lang="en">
    The Hobbit
  </title>
  <publisher>
    Klett-Cotta
  </publisher>
  <isbn>
    3608938052
  </isbn>
</item>
</books>

```

Zunächst wird die Selektion einzelner Elemente des Quelldokuments beschrieben und anschließend die Überführung der selektierten Elemente in eine neue Darstellungsform demonstriert.

Das Element auf der höchsten Hierarchiestufe soll im Weiteren als *Wurzelement* bezeichnet werden. Elemente, die auf einer nächst niedrigeren Hierarchiestufe stehen, erhalten die Bezeichnung *Kindelemente*; als *Elternelemente* werden diejenigen Elemente bezeichnet, die genau eine Stufe höher in der Hierarchie eingeordnet sind.

2.7.1.1 Selektion von Tags

Im ersten Schritt sollen die Autoren aus Beispiel 14 extrahiert werden. Das dazu verwendete Stylesheet hat die Form:

Beispiel 15

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xsl:stylesheet version="1.0"
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4   <xsl:template match="item">
5     <xsl:apply-templates select="author"/>
6   </xsl:template>
7 </xsl:stylesheet>

```

Das Stylesheet beginnt nach dem XML-Prolog und wird durch das `xsl:stylesheet`-Tag gekapselt. Es besteht aus dem XSLT-Befehl `xsl:template`, der ein Muster definiert, das auf

das Quelldokument angewendet werden soll (Zeilen 4–6). Das Attribut *match* legt das zu suchende Element fest – in diesem Fall *item*.

Das *template*-Element beinhaltet den Befehl `xsl:apply-template` mit dem Attribut *author*. Dieser Befehl selektiert alle Kindknoten des Elements *item*, die vom Typ *author* sind.

Eine Übersetzung mit diesem Stylesheet führt zu dem Ergebnis:

```
J.R.R. Tolkien  
J.R.R. Tolkien
```

In einer leichten Abwandlung des Stylesheets können alle Daten des Dokumentes extrahiert werden:

Beispiel 16

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>  
2 <xsl:stylesheet version="1.0"  
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  
4   <xsl:template match="/">  
5     <xsl:apply-templates />  
6   </xsl:template>  
  
7 </xsl:stylesheet>
```

Es werden alle Kindelemente des Wurzelementes – definiert durch den „/“ – herausgefiltert.

Ein angenehmes Werkzeug bei der Suche nach Tags ist die Möglichkeit, Wildcards (d.h. Platzhalter) zu verwenden. Eine Suche mit dem Stylesheet aus Beispiel 17 filtert, ausgehend von der Ebene von *item* und sucht sämtliche Kindelemente von *item* heraus.

Beispiel 17

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>  
2 <xsl:stylesheet version="1.0"  
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  
4   <xsl:template match="item">  
5     <xsl:apply-templates select="**"/>  
6   </xsl:template>  
  
7 </xsl:stylesheet>
```

Der Befehl `xsl:value-of` wird verwendet, wenn Elemente auf bestimmte Attribute und Attributwerte getestet oder ihr Wert ausgegeben werden sollen.

Möchte man Titel selektieren, die einen *lang*-Attribut besitzen, sieht das entsprechende Stylesheet folgendermaßen aus:

Beispiel 18

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xsl:stylesheet version="1.0"
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4   <xsl:template match="books/item">
5     <xsl:value-of select="title[@lang]"/>
6   </xsl:template>
7 </xsl:stylesheet>

```

Es wird jeweils das *erste* Auftreten eines Attributes unterhalb des *item*-Elementes angezeigt. In diesem Fall ist das Ergebnis:

Der Herr der Ringe
Der Hobbit

Möchte man *alle* Titel herausfinden, die mit einem Sprachattribut versehen sind, benötigt man eine kleine Schleife, die alle Titel ausliest:

Beispiel 19

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xsl:stylesheet version="1.0"
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4   <xsl:template match="/books/item">
5     <xsl:for-each select="title[@lang]">
6       <xsl:value-of select="." />
7     </xsl:for-each>
8   </xsl:template>
9 </xsl:stylesheet>

```

Wenn der Wert des Attributes bereits bekannt ist, kann man das obige Stylesheet erweitern und beispielsweise nur alle englischen Titel auswählen.

```
<xsl:value-of select="title[@lang='en']"/>
```

Für den Fall, dass nach denjenigen Titeln gesucht wird, denen keine Sprache zugeordnet ist, besteht die Möglichkeit einer Negation.

```
<xsl:value-of select="title[not(@lang)]"/>
```

Dieser Ausdruck `title[not(@lang)]` ist ein *XPath-Lageausdruck*. Zum besseren Verständnis dieser Ausdrücke soll der nächste Abschnitt beitragen.

2.7.1.2 Auswahl mit XPath

XSLT bietet weitergehende Möglichkeiten, eine Regelbedingung zu definieren. Die dafür verwendete Sprache heißt *XPath*. Ausgehend vom Kontext-Knoten, lässt sich hier eine Reihe relativer Ausdrücke – sog. *Pfade* – aneinander fügen. Der Kontextknoten ist bei einer XPath-

Lokalisierung innerhalb eines Knotenbaums des XML-Quelldokumentes das Element, von dem aus eine Operation gestartet wird.

Das grundlegende Konstrukt in XPath ist der Ausdruck. Er wird verwendet, um entweder eine Knotenmenge, einen Wahrheitswert (boolean), eine Zahl oder eine Zeichenkette zu erhalten.[37]

Ein wichtiger Teil von XPath ist der *Pfad* (location path). Dieser Ausdruck wählt ein Element oder Knoten relativ zum jeweiligen Kontextknoten aus. Das Ergebnis der Auswertung eines solchen Pfades ist eine Knotenmenge, die die durch den Ausdruck ausgewählten Knoten enthält. Die Pfade können rekursiv ineinander geschachtelt werden, um das Filtern von Knotenmengen zu erweitern.

Jeder Pfad kann durch eine gradlinige, allerdings auch gelegentlich sehr lange Syntax dargestellt werden. Zur Abkürzung dieser Pfade stehen einige Verkürzungsschreibweisen zur Verfügung, auf die später noch eingegangen wird. Zunächst sollen ein paar Beispiele die Funktionsweise von Pfadausdrücken verdeutlichen, die aus [38] entnommen sind.

<code>child::para</code>	selektiert die Kindknoten vom Typ <code>para</code> des aktuellen Kontextknotens.
<code>child::*</code>	selektiert alle Kindknoten des aktuellen Kontextknotens.
<code>child::text()</code>	selektiert alle Kindknoten des aktuellen Kontextknotens, die Textknoten sind.
<code>child::node()</code>	selektiert alle Kindknoten des aktuellen Kontextknotens, egal von welchem Knotentyp sie sind.
<code>attribute::name</code>	selektiert das <code>name</code> -Attribut des aktuellen Kontextknotens.
<code>attribute::*</code>	selektiert alle Attribute des aktuellen Kontextknotens.
<code>descendant::para</code>	selektiert das <code>para</code> -Element, wenn es ein Nachkomme des aktuellen Kontextknotens ist.
<code>self::para</code>	selektiert den Kontextknoten, wenn er vom Typ <code>para</code> ist, andernfalls wird nichts selektiert.
<code>child::* / child::para</code>	selektiert alle Elemente vom Typ <code>para</code> , die Enkel des aktuellen Kontextknotens sind.
<code>/</code>	selektiert das Wurzelement des Dokumentes
<code>child::para[position()=1]</code>	selektiert das erste <code>para</code> -Element, das ein Kind des aktuellen Kontextknotens ist.
<code>child::para[position()=last()-1]</code>	selektiert das vorletzte <code>para</code> -Element, das ein Kind des aktuellen Kontextknotens ist.

Bei der vorangegangenen Tabelle handelt es sich nicht um eine abschließende Aufzählung der Möglichkeiten. Es soll lediglich ein Eindruck vermittelt werden, wie Pfade erzeugt werden.

Es gibt zwei Arten von Pfaden:

- relative Pfade
- absolute Pfade

Die *relativen Pfade* bestehen aus einer Sequenz von einem oder mehreren Lageschritten, die durch einen „/“ getrennt sind. Sie werden von rechts nach links zusammengesetzt, wobei jeder Schritt Knoten selektiert, die relativ zu dem vorherigen Kontextknoten liegen. Anders ausgedrückt: jeder einzelne Lageausdruck innerhalb der Sequenz selektiert eine Knotenmenge, die als Kontextknoten für den jeweils nächsten Lageausdruck verwendet werden. Z.B. selektiert `child::div/child::para` die `para`-Element Kinder eines `div`-Elementes. Oder, vom aktuellen Kontextknoten aus betrachtet, werden die `para`-Element Enkel selektiert, die `div`-Elemente als Eltern haben.

Ein *absoluter Pfad* besteht aus einem „/“, der optional von einem relativen Pfad gefolgt wird. Der „/“ an sich selektiert das Wurzelement des Dokuments, also den obersten Knoten in der Hierarchie. Der relative Pfad, der sich dem „/“ anschließt, selektiert demnach Elemente, die direkt zum Wurzelement des Dokumentes in Beziehung gesetzt sind.

Ein Lageschritt besteht aus drei Teilen.

- der **Achse**, die die Beziehung zwischen dem selektierten Knoten und dem aktuellen Kontextknoten spezifiziert.
- dem **Knotentest**, der den Knotentyp des durch den Lageschritt selektierten Knotens spezifiziert.
- keines oder mehrere **Prädikate**, die beliebige Ausdrücke verwenden, um die selektierte Knotenmenge zu verfeinern.

Ein Beispiel wäre `child::para[position()=1]`, bei dem `child` der Name der Achse ist, `para` der Knotentest und `[position()=1]` das Prädikat. Eine vollständige Beschreibung der Lageausdrücke findet sich unter [39].

Zum Schluss soll noch einmal auf die verkürzende Schreibweise eingegangen werden, da diese auch später bei der Transformation von Dokumenten verwendet wird.

Eine der wichtigsten Vereinfachungen ist, dass `child::` im Ausdruck als Standard angesehen wird und daher entfallen kann. Eine Abkürzung für `attribute::` ist `@`.

Als Beispiel soll `para[@type="warning"]` dienen. Dies ist eine Abkürzung für `child::para[attribute::type="warning"]` und selektiert alle `para`-Kinder mit einem `type`-Attribut, dessen Wert `warning` ist.

// steht für `/descendant-or-self::node()`. Z.B. steht `//para` für `/descendant-or-self::node()/child::para` und selektiert jedes `para`-Element im Dokument,

Der Ausdruck „.“ ist eine Abkürzung für `self::node()`, welches den Kontextknoten an sich auswählt. Besonders nützlich ist dieser Ausdruck zusammen mit //. So ist `./para` die Kurz-

form für `self::node()/descendant-or-self::node()/child::para` und selektiert alle `para`-Element-Nachkommen des aktuellen Kontextknotens.

Zuletzt soll noch „`..`“ erwähnt werden, eine Kurzform für `parent::node()`. So wählt `../title` alle `title`-Elemente des Elternknotens aus.

2.7.1.3 Einige wichtige Befehle

Zum besseren Verständnis der Stylesheets sollen an dieser Stelle noch einige Befehle erläutert werden, die zur Anwendung kommen.

2.7.1.3.1 xsl:value-of

Das Tag `xsl:value-of` fügt einen bestimmten Wert ein, der mit dem `select`-Attribut festgelegt wird.

`<xsl:value-of select=".">` fügt den Inhalt des aktuellen Kontextknotens ein. `select` akzeptiert aber nicht – wie oben gesehen – nur einen Punkt oder XPath-Ausdruck, sondern vielmehr Ausdrücke im Allgemeinen. Folglich akzeptiert `select` auch eine Zahl, eine Berechnung, einen booleschen Ausdruck oder eine Zeichenkette.

`<xsl:value-of select="3+8+9"/>` z.B. fügt die Zahl 20 ein. Es wird eine Großzahl mathematischer Operationen unterstützt.

2.7.1.3.2 Bedingungen

XSLT unterstützen bedingte Verzweigungen, z.B.

```
1 <xsl:template match="">
2   <xsl:if test="1 = 2">
3     Eins ist gleich zwei!
4   </xsl:if>
5   <xsl:if test="1 != 2">
6     Eins ist nicht gleich zwei!
7   </xsl:if>
8 </xsl:template>
```

Das `test`-Attribut muss einen Ausdruck enthalten, der entweder nach *wahr* oder *falsch* ausgewertet werden kann. Zeichenketten sind wahr, sofern sie nicht leer sind, Zahlen sind wahr, sofern sie ungleich 0 sind, und Knotensätze sind wahr, sofern sie Knoten enthalten.

Daneben gibt es noch Fälle, in denen zwei Werte verglichen werden. XSLT stellt alle wichtigen Vergleichsoperatoren (gleich `=`, größer `>`, kleiner `<`, größer-oder-gleich `>=`, kleiner-oder-gleich `<=`, ungleich `!=`) zur Verfügung. Es findet keine Unterscheidung zwischen Zeichenketten- und Zahlenvergleichen statt. Die Operatoren müssen dabei, wie in XML-Dokumenten erforderlich, maskiert werden – z.B. `<=` anstelle von `<=`. [38]

2.7.1.3.3 Elemente und Attribute erzeugen

Bei der Übersetzung eines XML-Dokumentes in ein neues XML-Dokument ist häufig erforderlich, neue Tags zu erzeugen, die nicht im Ursprungsdokument enthalten sind. Dies geschieht mit Hilfe des `xsl:element`-Tags. Dieses Tag erzeugt im resultierenden Dokument ein neues Element, dessen Name im Attribut `name` festgelegt werden kann.

Das folgende Beispiel transformiert das oben angegebene Quelldokument in ein neues Dokument, bei dem `item` durch `eintrag`, `autor` durch `schriftsteller` ersetzt und `publisher` sowie `isbn` entfernt werden. Die `Publisher`- und die `ISBN`-Information sollen als Attribut in das `buchtitle`-Tag geschrieben werden. Dazu wird das `xsl:attribute`-Tag verwendet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:template match="books">
    <xsl:element name="buecher">
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
```

```
  <xsl:template match="author">
    <xsl:element name="schriftsteller">
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:template>
```

```
  <xsl:template match="title">
    <xsl:element name="buchtitel">
      <xsl:attribute name="verlag">
        <xsl:value-of select="normalize-space(..publisher)"/>
      </xsl:attribute>
      <xsl:attribute name="isbn">
        <xsl:value-of select="normalize-space(..isbn)"/>
      </xsl:attribute>
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:template>
```

```
  <xsl:template match="publisher"/>
  <xsl:template match="isbn"/>
```

```
</xsl:stylesheet>
```

Der erste Abschnitt beinhaltet, wie bei allen Stylesheets, die Deklaration als XML-Dokument und anschließend das umgebende `stylesheet`-Tag.

Das erste Template wählt den Haupteintrag `books` aus und ersetzt diesen mit Hilfe des `element`-Tags durch `buecher`. Innerhalb des `element`-Tags werden die anderen Templates über `apply-templates` angewendet, so dass das Ergebnis der Übersetzung innerhalb dieses Elementes eingefügt wird.

Das zweite `template`-Tag übersetzt die Elemente vom Typ `author` in Elemente vom Typ `schriftsteller` und fügt mit dem `value-of`-Tag die Werte der `author`-Tags ein.

Bei den `title`-Tags im dritten Abschnitt soll zunächst die Übersetzung des Typs von `title` nach `buchtitel` geschehen. Dazu wird erneut das `element`-Tag verwendet. Innerhalb des `element`-Tags wird anschließend das `attribut`-Tag verwendet, um die Attribute `verlag` und `isbn` in das `buchtitel`-Tag zu schreiben. Bei diesem Beispiel sieht man noch einmal die Verwendung des `.. /` Ausdrucks bei der Knotenauswahl. Innerhalb der `value-of`-Tags im `attribut`-Tag treten bisher noch nicht verwendete Kernfunktionen auf. Die Funktion `normalize-space()` entfernt überflüssige Füllzeichen, die aus der Formatierung des Originaldokumentes stammen, aus der Zeichenkette des Wertes. Nach den `attribut`-Tags im dritten Abschnitt wird mit Hilfe des `value-of`-Tags mit `select="."` der Wert des `title`-Tags in das `buchtitel`-Tag geschrieben.

Zum Schluss weisen die beiden `template`-Tags im vierten Abschnitt noch den `publisher`- und `isbn`-Tags keinen neuen Wert zu und löschen sie auf diese Weise. Als Ergebnis erhält man:

```
<?xml version="1.0" encoding="UTF-8"?>
<buecher>
  <schriftsteller>
    J.R.R. Tolkien
  </schriftsteller>
  <buchtitel verlag="Klett-Cotta" isbn="3608935444">
    Der Herr der Ringe
  </buchtitel>
  <buchtitel verlag="Klett-Cotta" isbn="3608935444">
    Master of the ring
  </buchtitel>

  <schriftsteller>
    J.R.R. Tolkien
  </schriftsteller>
  <buchtitel verlag="Klett-Cotta" isbn="3608938052">
    Der Hobbit
  </buchtitel>
  <buchtitel verlag="Klett-Cotta" isbn="3608938052">
    The Hobbit
  </buchtitel>
</buecher>
```

2.7.1.3.4 Variablen

Ein wichtiges Element ist das `xsl:variable`-Tag. Mit Hilfe dieses Tags ist es möglich, ermittelte Werte zwischenspeichern oder als globale Variable festzulegen.

```
<xsl:variable name="title">
  Titel:
</xsl:variable>
<xsl:template match="books/item">
  <xsl:value-of select="$title"/>
  <xsl:value-of select="title"/>
</xsl:template>
```

In diesem Beispiel wird die Variable *title* mit dem Wert „*Titel:*“ belegt. Es ist auch möglich, innerhalb des `xsl:variable`-Tags Werte aus einem `xsl:value-of`-Tag an die Variable zuzuweisen. Auf den Wert der Variable wird mit dem `xsl:value-of`-Tag zugegriffen, indem innerhalb des `select` Attributes ein `$` gefolgt von dem Variablennamen angegeben wird.[38] Als Resultat des obigen Stylesheets erhält man:

```
Titel:  Der Herr der Ringe
Titel:  Der Hobbit
```

2.7.1.3.5 Kernfunktionen

Über das Auswählen und Erzeugen von Elementen hinaus stellt XSLT noch eine Reihe von Funktionen zur Verfügung, die bei der Auswertung und Manipulation von Ausdrücken verwendet werden können. Eine Auswahl stellt die folgende Tabelle dar, deren Einträge aus [38] entnommen sind.

<code>last()</code>	gibt einen Wert gleich der Kontextgröße des ausgewerteten Ausdrucks zurück.
<code>position()</code>	gibt die Position innerhalb des ausgewerteten Ausdrucks zurück.
<code>count(node-set)</code>	gibt die Anzahl der Knoten des Argumentes zurück.
<code>local-name(node-set?)</code>	gibt den lokalen Anteil des gesamten Namens des Knotens wieder, der in der Dokumentenhierarchie als erstes auftritt.
<code>namespace-uri(node-set?)</code>	gibt die Namespace URI des Knotens zurück, der in der Dokumentenhierarchie als erstes auftritt.
<code>name(node-set?)</code>	gibt den bedingten Namen des Knotens wieder, der in der Dokumentenhierarchie als erstes auftritt.
<code>string(object?)</code>	überführt das angegebene Objekt in eine Zeichenkette.
<code>contains(string, string)</code>	gibt Wahr zurück, wenn die erste Zeichenkette die zweite enthält.
<code>not(boolean)</code>	negiert den angegebenen booleschen Ausdruck.
<code>true()</code>	gibt wahr zurück.
<code>false()</code>	gibt falsch zurück.
<code>number(object?)</code>	überführt das angegebene Objekt in eine Zahl.
<code>sum(node-set)</code>	überführt alle Knoten in eine Zahl und bildet die Summe.

Ein Stylesheet, das die Namen aller verwendeten Tags wiedergibt, ist:

```
<xsl:template match="/books/item/*">  
  <xsl:value-of select="name()"/>  
</xsl:template>
```

Das `template` selektiert alle Kindelemente von `item` und `value-of`, liest die Elementnamen aus und gibt sie zurück.

```
author  
title  
title  
publisher  
isbn  
author  
title  
title  
publisher  
isbn
```

3 Konzeption

Das Ziel der Diplomarbeit ist es, Bestandsdaten, die von digitalen Bibliotheken bereitgestellt werden, innerhalb des Edutella-Peer-to-Peer-Netzwerks verfügbar zu machen.

Um eine solche Anbindung herzustellen, müssen Daten zwischen der digitalen Bibliothek und dem Edutella-Netzwerk transferiert werden. Zu diesem Zweck muss ein Peer des Edutella-Netzwerkes in der Lage sein, über eine Intranet/Internet-Verbindung die Daten einer digitalen Bibliothek abzufragen. Der hieraus resultierende Aufbau des Netzwerks ist in Abbildung 16 zu sehen.

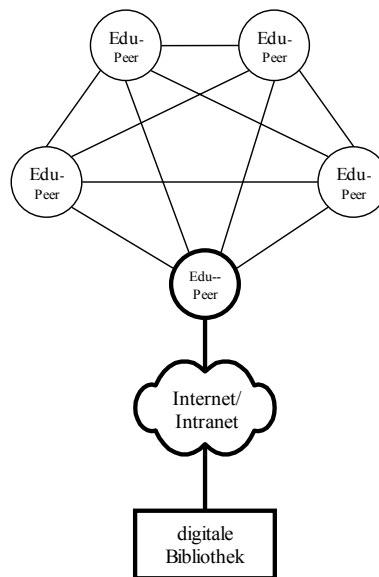


Abbildung 16: Prinzipieller Aufbau der Anbindung einer digitalen Bibliothek an das Edutella-Netzwerk

Um dieses Ziel zu erreichen, müssen mehrere Teilprobleme gelöst werden. Die Daten müssen von der digitalen Bibliothek über eine Schnittstelle abgefragt und zum Edutella-Peer übertragen werden. Die aus der Anfrage resultierenden Ergebnisse müssen, sofern sie nicht bereits in einer entsprechenden Form vorliegen, in eine Datenform übersetzt werden, die es erlaubt, auf Anfragen aus dem Edutella-Netzwerk hin untersucht zu werden.

Als Grundlage für die Übertragung von Daten erscheint es sinnvoll, einen verbreiteten Standard zu verwenden, der von möglichst vielen digitalen Bibliotheken unterstützt wird. Hier fiel die Entscheidung auf das *Open Archive Initiative Protocol of Metadata Harvesting* (OAI-PMH). Es erlaubt, auf einfache Weise digitale Bibliotheken über das HTTP-Protokoll abzufragen, sofern sie diesen Standard (OAI-PMH) unterstützen. Als Antwort liefern die digitalen Bibliotheken (Data Provider im Sinne des OAI-PMH-Protokolls), wie in den Grundlagen bereits vorgestellt, ein XML-Dokument zurück.

Auf die Frage nach einem geeigneten Format für die Repräsentation der Daten liegt RDF nahe. Als ein auf XML basierendes Datenformat lässt sich RDF mit Hilfe von XSL-Stylesheets

unkompliziert aus anderen XML-Dokumenten erzeugen und kann darüber hinaus mit der Anfragesprache RDQL¹⁷ untersucht werden.

Daraus ergibt sich als möglicher Edutella-Peer eine Abwandlung des bereits vorhandenen RDQL-Providers, der in der Lage ist, dem Edutella-Netzwerk ein RDF-Modell als Datenbasis zur Verfügung zu stellen. Dieser Provider nimmt eine Anfrage aus der Edutella-internen Anfragesprache entgegen und transformiert sie in eine RDQL-Anfrage, mit der er das RDF-Modell untersucht. Das Anfrageergebnis wird daraufhin vom RDQL-Provider in das Edutella-interne Antwortformat übersetzt und anschließend an das Edutella-Netzwerk zurückgesendet.

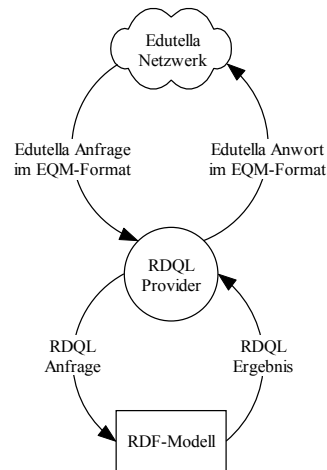


Abbildung 17: Anfrageformate im RDQL-Provider

Um eine vollständige Anbindung des RDQL-Providers an einen OAI Data Provider sicher zu stellen, muss der RDQL-Provider derart ergänzt werden, dass er befähigt wird, einen OAI Data Provider abzufragen. Um dies zu ermöglichen, muss ein OAI-Adapter entwickelt werden, der die Kommunikation mit einem Data Provider übernimmt und die Antwortdaten übersetzt.

Es ist wünschenswert, dass der zu implementierende Edutella-Peer nicht ausschließlich auf OAI festgelegt ist. Vielmehr soll der Peer prinzipiell in der Lage sein, auch andere Server als OAI Data Provider in das Edutella-Netzwerk zu integrieren. Hierzu weist der Edutella-Provider eine Schnittstelle auf, an den der OAI Server Adapter und gegebenenfalls andere Serveradapter angebunden werden können.

¹⁷ nähere Informationen zu RDQL findet man unter <http://www.hpl.hp.com/semweb/doc/tutorial/RDQL/>

Es ergibt sich die folgende Komponentenstruktur:

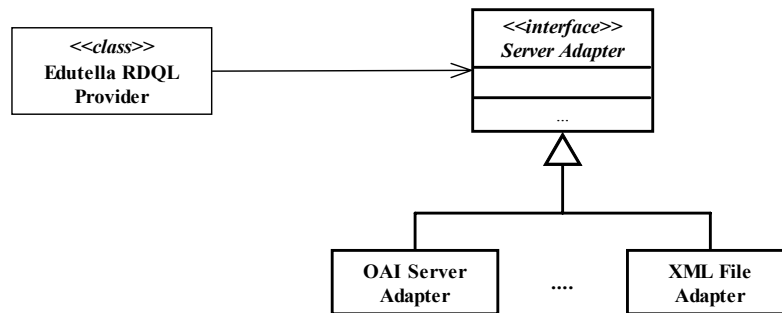


Abbildung 18: Server Adapter Klassendiagramm

Jeder der Adapter soll ein *Server Adapter* Interface implementieren, das es dem Edutella-Peer ermöglicht, unabhängig von der konkreten Art des Servers, der sich hinter dem Adapter befindet, auf dessen Daten in Form eines RDF-Modells zuzugreifen, ohne Annahmen darüber zu treffen, wie der Server Adapter mit der Datenquelle verbunden ist. Im einfachsten Fall könnte es sich bei der Datenquelle um eine lokal vorliegende XML-Datei handeln, aber ebenso so gut könnte es sich um einen entfernten Datenbankserver handeln, der über eine aufwändige Datenbankanfrage abgefragt wird.

Das RDF-Modell, das dem Edutella-Peer zur Verfügung gestellt wird, soll als lokale Replik der Daten vorliegen, die der OAI Data Provider oder ein anderer Daten-Provider zur Verfügung stellt. Auf diese Weise werden die Antwortzeiten des Peers verkürzt; genau genommen werden die Fehlzeiten eingespart, die bei der Übermittlung der Daten vom Data Provider zum Edutella-Provider und durch die Transformation in ein RDF-Modell entstehen. Andererseits können auf diese Weise auch zeitweilige Verbindungsschwierigkeiten mit dem Server des Data Providers überbrückt werden.

Als weitere Anforderung sollten sich die Server Adapter als eigenständige Prozesse starten lassen, die in festgelegten Zeitintervallen ihre Daten aktualisieren. Der Vorteil ist, dass eine – bis auf eine dem Aktualisierungsintervall entsprechende kurze Verzögerung – aktuelle Versionen der Daten dem Edutella-Provider direkt zur Verfügung steht und der Edutella-Peer nicht durch die Anforderung der Daten blockiert wird.

Die Aktualisierung der Daten soll im Hintergrund erfolgen. Dabei werden die aktualisierten Daten des Data Providers in ein neues Modell gespeichert. Nach Abschluss der Aktualisierung wird das bisher verwendete Modell durch das aktualisierte Modell ersetzt. Während dieser Zeit soll sich der Server Adapter für eine kurze Zeit sperren, um die Datenkonsistenz zu gewährleisten.

3.1 OAI/DSpace-Anbindung

3.1.1 OAI Server Adapter

Da das DSpace-System eine OAI-Schnittstelle zur Verfügung stellt, kann zur Anbindung von DSpace das gleiche Verfahren verwendet werden, das auch für andere OAI Provider verwendet wird. Die Kommunikation des Edutella-Peers mit dem OAI Data Provider läuft über einen

OAI Server Adapter. Der OAI Server Adapter ist sowohl für die Beschaffung der Daten von einem OAI Data Provider verantwortlich wie auch für die Transformation des Ergebnisses in ein RDF-Modell und dessen Bereitstellung für den Edutella-Peer.

Die Funktionalität des Protokolls soll nicht innerhalb des Server Adapters liegen, sondern in einem separaten Protokoll Adapter. Auf diese Weise wird die Bereitstellung der Daten von der eigentlichen Kommunikation mit dem Data Provider getrennt. Der Protokoll Adapter übernimmt die Kommunikation mit dem Server und liefert als Ergebnis einer Anfrage ein XML-Dokument-Objekt zurück, das die Daten des Providers enthält.

Die als XML-Dokument empfangenen Daten müssen anschließend in RDF übersetzt werden. Der ursprünglichen Überlegung, eine auf den Einsatz in diesem Projekt zugeschnittene Übersetzungskomponente zu entwickeln, wurde der Einsatz eines XSL-Transformers vorgezogen. Zum einen ist XSL ein verbreiteter Standard, zum andern ist es auf diese Weise leicht möglich, die Daten ineinander zu überführen und gegebenenfalls die Übersetzung durch einfache Änderung am Stylesheet zu steuern, ohne dass die *Implementierung* innerhalb des Projektes geändert werden muss. Aus diesen Überlegungen ergibt sich der interne Aufbau des Server Adapters, wie ihn Abbildung 19 zeigt.

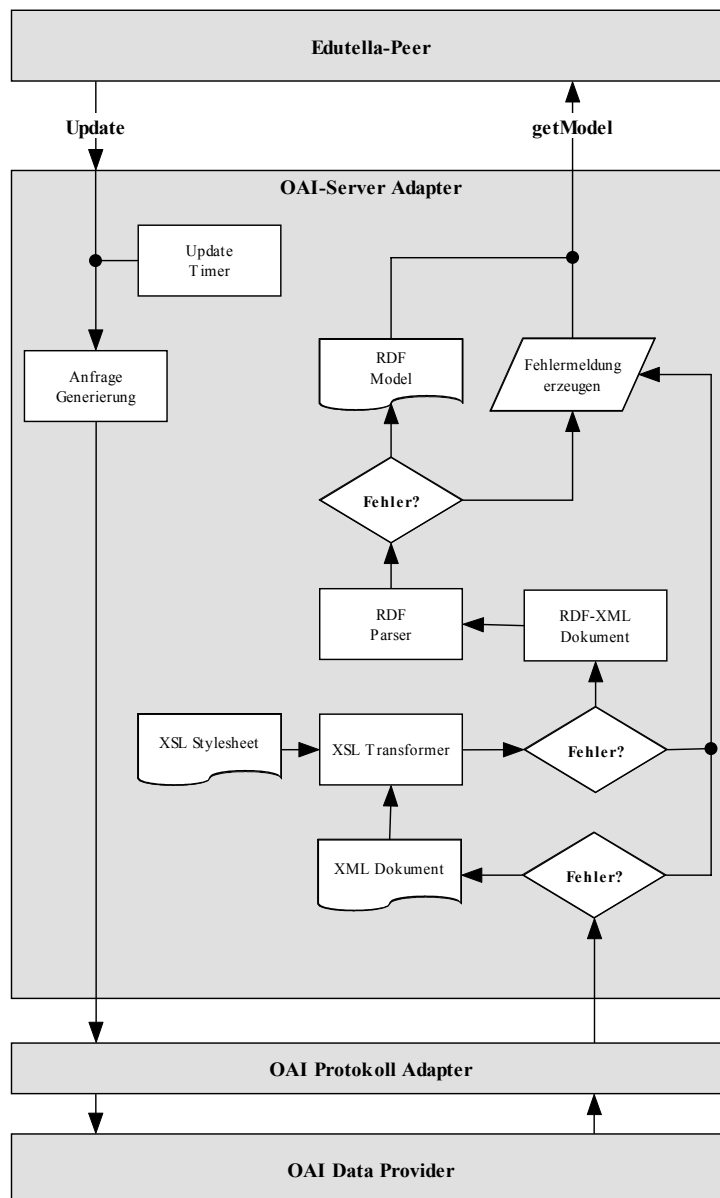


Abbildung 19: Interner Aufbau der des OAI Server Adapter

In Abbildung 19 erkennt man die internen Zusammenhänge der Adapter. Der Edutella-Peer ruft die Update-Methode auf, um die Aktualisierung der Daten auszulösen. Alternativ ist der OAI Server Adapter in der Lage, nach Ablauf eines Aktualisierungsintervalls das Datenmodell selbstständig zu aktualisieren. Nach Aufruf der Update Methode soll der Adapter sich mit dem Server verbinden und die aktuellen Daten abfragen.

Die Antwort des Data Providers soll anschließend aus XML in RDF mit Hilfe eines XSL-Stylesheets übersetzt werden und dem Edutella-Peer als RDF-Modell zur Verfügung gestellt werden. Mögliche Fehler, die während der HTTP-Übertragung entstehen können, weil z.B. der Server des Providers vorübergehend nicht erreichbar ist, oder Fehler, die durch das OAI-PMH begründet sind, werden während der Verarbeitung behandelt und dem Edutella-Peer gemeldet.

3.1.2 OAI Protokoll Adapter

Ein OAI Data Provider stellt seine Daten entsprechend dem OAI Metadata Harvesting Protokoll zur Abfrage bereit. Dieses Protokoll basiert auf dem HTTP-Protokoll, das für die Kommunikation zwischen Server und Clients im Internet verwendet wird. Es ist also notwendig, einen Adapter zu implementieren, der einerseits in der Lage ist, eine Anfrage an einen OAI Data Provider mit Hilfe dieses Protokolls zu formulieren, und andererseits das HTTP-Protokoll unterstützt. Eine umfangreiche Beschreibung des OAI Metadata Harvesting Protokolls findet sich im Abschnitt 4. Abbildung 20 zeigt den internen Aufbau des OAI Protokoll Adapters und des Informationsflusses.

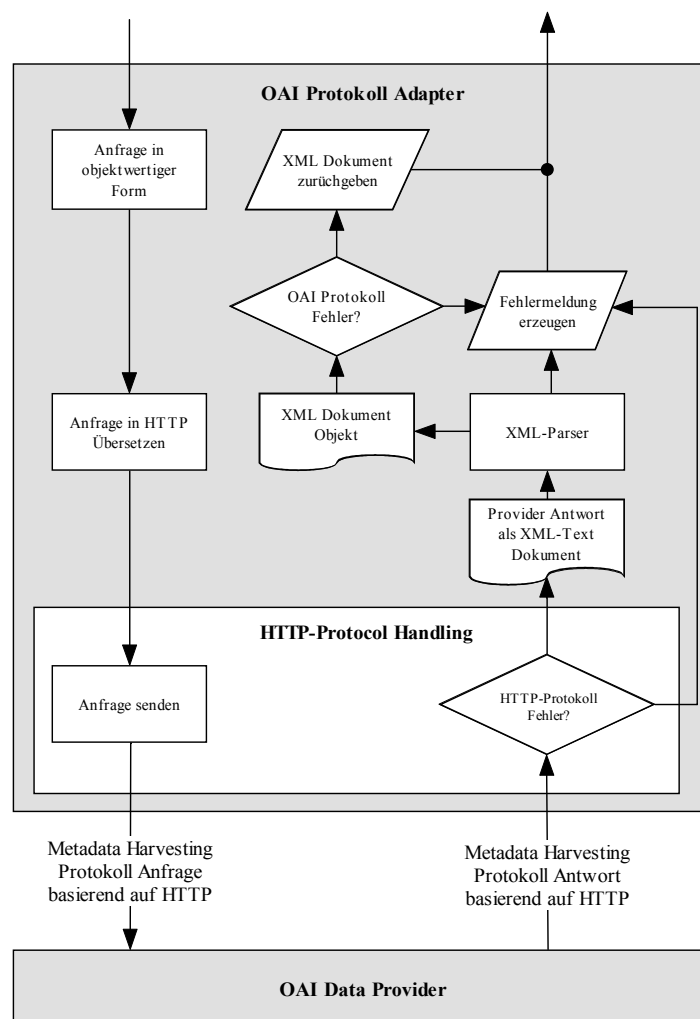


Abbildung 20: Informationsfluss im OAI Protokoll Adapter

Da das OAI Metadata Harvesting Protokoll verschiedene Anfragen unterstützt, soll der OAI Protokoll Adapter für jede dieser Anfragen eine Funktion zur Verfügung stellen, die es ermöglicht, die entsprechenden Parameter mittels üblicher Java-Objekte zu formulieren. Anschließend soll die objektwertige Anfrage in eine HTTP konforme Anfrage übersetzt werden. Diese Anfrage wird an einen HTTP-Protocol Handler übergeben, der die Kommunikation zwischen Provider und Adapter übernimmt.

Dieser Handler kann intern vorliegen, da er lediglich von dem Adapter verwendet wird. Im Falle eines Protokollfehlers, der auf das HTTP-Protokoll zurückzuführen ist, soll der HTTP-Protocol Handler eine Fehlermeldung generieren und diese zurückgeben. Andernfalls soll die Antwort des OAI Data Providers anschließend an einen XML-Parser übergeben werden, der das textbasierte Ergebnis in ein XML-Dokument-Objekt übersetzt. Falls während der Übersetzung Fehler auftreten, weil das XML-Dokument z.B. nicht wohlgeformt ist, soll eine Fehlermeldung erstellt und zurückgegeben werden. Nach der Übersetzung wird das Ergebnis auf Fehler, die auf das OAI-Protokoll zurückzuführen sind – z.B. weil die Argumente nicht in der richtigen Form angegeben wurden – untersucht und, falls keine Fehler aufgetreten sind, als Antwort zurückgegeben.

3.2 ILIAS-Anbindung

ILIAS unterstützt keine direkte Anbindung über eine Online Schnittstelle. Stattdessen bietet das System die Möglichkeit eines Im- und Exports der Daten aus und in ein XML-Format, dem die ILIAS DTD zugrunde liegt. Eine ausführliche Beschreibung der DTD findet sich im Abschnitt 4.3.1.

3.2.1 XML File Adapter

Zusätzlich zum OAI Server Adapter soll ein weiterer Server Adapter implementiert werden, um beliebige XML-Dateien an das Edutella-Netzwerk anzubinden. Im Gegensatz zum OAI Server Adapter verbindet sich der XML File Adapter nicht mit einem Server, sondern liest die Daten aus einer Datei oder einem Verzeichnis aus. Dabei soll konfigurierbar sein, ob einzelne Dateien und/oder rekursiv Verzeichnisse ausgelesen werden sollen.

Die aus der oder den Dateien ausgelesenen Daten sollen in einem RDF-Datenmodell zur Verfügung gestellt werden. Da die Daten als XML-Dokument vorliegen, werden diese auch hier mit Hilfe eines XSL-Transformers in einen RDF-Graphen übersetzt. Eine Voraussetzung ist, dass für jede zur Laufzeit vorliegende Instanz dieses XML File Adapters alle vorgesehenen XML-Dokumente konform zu einem gemeinsamen Schema sind. Dies ermöglicht die Transformation dieser XML-Dokumente mit einem einheitlichen XSLT-Stylesheet.

Der interne Aufbau des XML File Adapters ist in Abbildung 21 dargestellt. Der XML File Adapter führt – gesteuert durch einen Updatetimer – in festen Intervallen eine Aktualisierung der Modelle durch (alternativ kann das Aktualisieren auch durch den Peer über die `update()` Methode eingeleitet werden). Wenn die Aktualisierung durchzuführen ist, liest der XML File Adapter eine Konfigurationsdatei ein, in der angegeben ist, welche Dateien und Verzeichnisse als Grundlage für die zur Verfügung gestellten Daten dienen sollen. Anhand der Konfigurationsdatei wird eine Liste erstellt, in der alle Dateien, die ausgelesen werden sollen, aufgeführt sind.

Anschließend beginnt der XML File Adapter, die einzelnen Dateien auszulesen. Sollte beim Auslesen einer Datei ein I/O-Fehler auftreten, weil etwa die angegebene Datei nicht vorhanden ist, nicht gelesen werden darf oder ähnliches, wird das Lesen der restlichen Dateien aus der Liste nicht unterbrochen, sondern eine Fehlermeldung an `stdout` ausgegeben und mit der nächsten Datei fortgefahren. War das Lesen einer Datei erfolgreich, wird das gelesene XML-Dokument mit Hilfe eines XSL-Transformers in ein RDF-Dokument übersetzt. Auch hier

können wieder Fehler auftreten, die z.B. darauf zurückzuführen sind, dass das Dokument nicht wohlgeformt ist. Auch in diesem Fall wird lediglich eine Fehlermeldung an stdout ausgegeben.

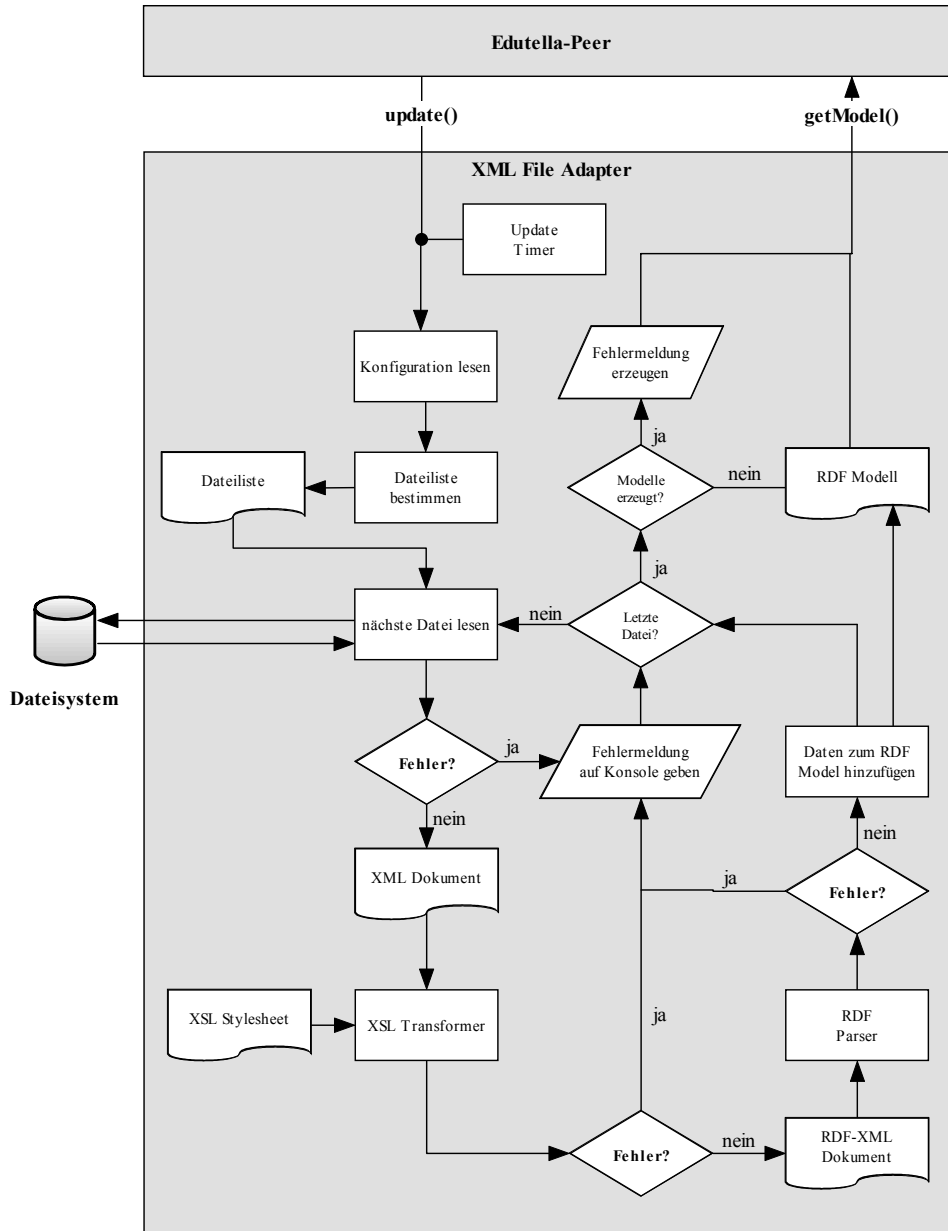


Abbildung 21: Informationsfluss im XML File Adapter

War das Übersetzen erfolgreich, wird das RDF-Dokument einem RDF-Parser übergeben und in ein RDF-Modell übersetzt. Im Fehlerfall wird auch hier wieder eine einfache Fehlermeldung ausgegeben. War das Parsen erfolgreich, wird das Modell dem bereits vorhandenen Modell hinzugefügt. Sind noch weitere Dateien vorhanden, werden diese auf die gleiche Weise eingelesen. Der Vorteil dieses Verfahrens ist, dass in dem Fall, dass eine von möglicherweise hundert Dateien fehlerhaft ist, die restlichen Dateien trotzdem dem Modell hinzugefügt werden können. Nur, wenn gar kein Modell erzeugt wurde, weil keine der angegebenen Dateien

vorhanden oder fehlerfrei war, wird dem Edutella-Peer eine Fehlermeldung zurückgegeben, die angibt, dass kein Modell erzeugt werden konnte.

4 Implementation

4.1 RDF Server Adapter Interface

Das Server Adapter Interface soll die Kommunikationsschnittstelle zwischen dem Edutella-Provider und der Anbindung an eine Datenquelle festlegen. Um die Art der zur Verfügung gestellten Daten zu charakterisieren, soll die Klasse des Server Adapters mit *RDF Server Adapter* bezeichnet werden.

Es soll keine Annahme darüber getroffen werden, welche Art von Datenquelle angesprochen wird. Prinzipiell sollte ein Server Adapter in der Lage sein, sämtliche Datensätze bereitzustellen, die der angebotenen Datenquelle zugrunde liegen. Um die Daten aus dem Server Adapter auslesen zu können, sollte es eine Funktion `getModel()` geben, die ein RDF-Modell der Daten zurückliefert. Darüber hinaus sollte es eine `update()` Funktion geben, die es gestattet, bei Bedarf eine Aktualisierung der Daten einzuleiten.

Über dieser Grundfunktionalität hinaus ist der Server Adapter so zu konzipieren, dass er als eigenständiger Prozess in der Lage ist, ein Update nach einem konfigurierbaren Intervall selbstständig durchzuführen. Um das Intervall festzulegen, ist die Funktion `setUpdateIntervall()` zu verwenden. Aus diesen Überlegungen ergibt sich das folgende UML-Klassendiagramm für den RDF Server Adapter.

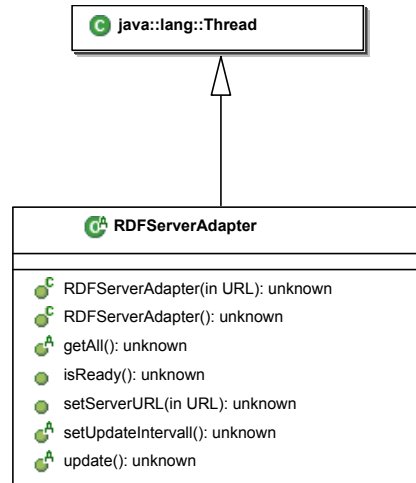


Abbildung 22: UML-Klassendiagramm RDF Server Adapter

4.2 OAI/DSpace Anbindung

4.2.1 OAI Server Adapter

Der OAI Server Adapter stellt eine konkrete Implementierung des *RDF Server Adapters* dar. Er verbindet einen OAI Data Provider mit einem Edutella-Peer. Der Server Adapter selbst besteht aus mehreren Teilen. Dem OAI Server Adapter, der die Implementierung des Interfaces darstellt, und dem OAI Protokoll Adapter, der für die Kommunikation mit einem OAI Data Provider zuständig ist. Die Kommunikation zwischen dem OAI Server Adapter und dem OAI Data Provider geschieht mittels dem OAI Metadata Harvesting Protokoll, das im nächsten Abschnitt genauer beschrieben wird.

4.2.2 OAI-Datenübertragungsprotokoll

Das *OAI Protocol for Metadata Harvesting* – auch kurz OAI-PMH – stellt ein anwendungsneutrales Framework zur Sammlung von Metadaten zur Verfügung. Es gibt zwei Arten von Teilnehmern an diesem Protokoll. Zum einen die Data Provider, die ein System bereitstellen, das das OAI-PMH zur Veröffentlichung von Metadaten unterstützt, zum anderen die Service Provider, die die bereitgestellten Daten auslesen, um damit weitere Dienste zur Verfügung zu stellen.

Eine Übersicht über das strukturelle Modell[41] von OAI zeigt Abbildung 23.

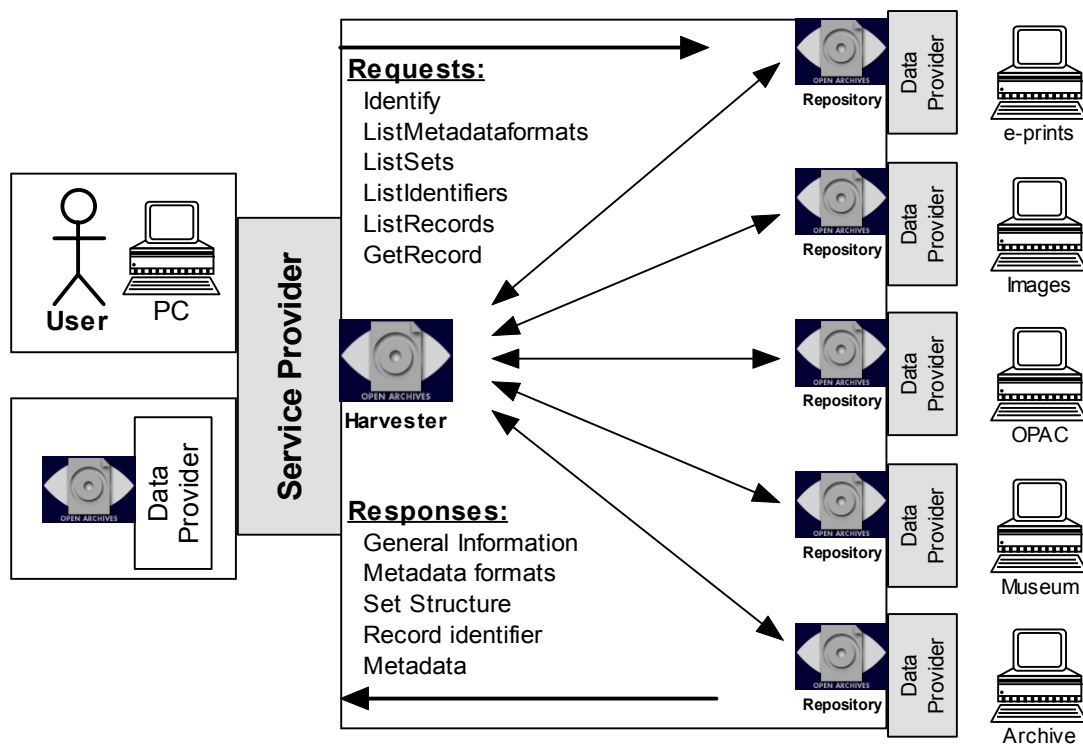


Abbildung 23: Übersicht des strukturellen Modells

Das OAI-PMH unterstützt sechs verschiedene Anfragen, die als *verbs* bezeichnet werden. Auf die verschiedenen Anfragen hin können verschiedene Antworten gegeben werden, die sich von generellen Informationen des Data Providers bis hin zu der gesamten Datenmenge des Providers erstrecken können.

Eine OAI-PMH Anfrage wird mittels des HTTP-Protokolls übermittelt. Als Schnittstelle zum Data Provider wird eine Basis-URL verwendet, an die alle Anfragen gesendet werden. Alle Repositories müssen sowohl die GET als auch die POST Methode des HTTP-Protokolls unterstützen. Zusätzlich zur Basis-URL bestehen alle Anfragen aus einer Liste von Schlüsselwert-Attribut Paaren in der Form Schlüssel=Wert, wie sie im HTTP-Protokoll definiert sind.

Jede Anfrage, die an die Basis-URL gesendet wird, muss zumindest den Schlüssel *verb* mit einem Wert enthalten, der angibt, um welche Art von Anfrage es sich handelt. In der folgenden Tabelle sind alle möglichen Anfragen und ihre Anwendungen aufgeführt.

Anfrage	Beschreibung
GetRecord	Die GetRecord Anfrage wird verwendet, um einen bestimmten Record aus dem Repository auszulesen.
Identify	Identify gibt Informationen über das Repository zurück, an das die Anfrage gerichtet wurde.
ListIdentifiers	Ist eine abgekürzte Form der ListRecords-Anfrage, die lediglich die Header der Records zurückliefert anstelle des gesamten Records.
ListMetadataFormats	Dieses Verb wird dazu verwendet, die Metadatenformate, die das Repository zur Verfügung stellt, zu erhalten.
ListRecords	Dieses Verb wird verwendet, um alle oder eine Teilmenge der Records aus dem Repository auszulesen.
ListSets	Gibt eine Struktur der Mengen des Repositories zurück.

Tabelle 2: Verbs des OAI-Protokolls

Ein Beispiel für eine Anfrage gemäß des HTTP-Protokolls ist:

`http://an.oa.org/OAI-script?verb=GetRecord&identifier=oai:arXiv.org:hep-th/9901001&metadataPrefix=oai_dc`

In diesem Fall handelt es sich um eine *GetRecord*-Anfrage. Das Verb, das den Anfragetyp angibt, wird dabei um einige Attribute ergänzt. Das Attribut *identifier* gibt an, welcher Datensatz zurückgegeben werden soll, und das *metadataPrefix*-Attribut gibt an, in welchem Metadatenchema der Record zurückgeliefert werden soll. Je nachdem, um welchen Anfragetyp es sich handelt, sind Attribute erforderlich optional oder exklusiv anzugeben. Eine Übersicht über die möglichen Attribute und ihre Bedeutung findet sich in der Tabelle 3.

Argumente	Beschreibung
metadataPrefix	gibt an, welcher Metadaten-Präfix in den Antwort-Records verwendet werden soll. Es werden nur Records zurückgegeben, die das entsprechende Format unterstützen. Die Anfrage ListRecords gibt alle unterstützten Formate des Data Providers zurück.
identifizier	spezifiziert einen Unique Identifier des Eintrags des Repositories, der angefordert werden soll.
from	Ist ein im UTC-Zeitformat angegebener Zeitpunkt, der den frühesten Zeitstempel bei der Selektion von Einträgen angibt.
until	Ist ein im UTC-Zeitformat angegebener Zeitpunkt, der den spätesten Zeitstempel bei der Selektion von Einträgen angibt.
set	beinhaltet einen setSpec-Wert, der angibt, aus welchen Mengen Einträge selektiert werden sollen.
resumptionToken	ist ein Argument zur Flusskontrolle des Datenstroms. Der Wert wurde von einer vorherigen Anfrage angegeben und weist auf weitere Records hin.

Tabelle 3: Attribute für OAI-Anfragen

Eine Übersicht, ob und in welchem Umfang die Attribute bei den einzelnen möglichen Anfragen erforderlich sind, findet sich im Anhang A.

Der zu implementierende OAI Protokoll Adapter soll zu jeder der möglichen Anfragen eine entsprechende Funktion zur Verfügung stellen, die testet, ob die erforderlichen Attribute angegeben wurden, und gegebenenfalls eine Fehlermeldung zurückgibt. Die Funktionen werden mit elementaren Java-Objekten, wie String, als Parameter aufgerufen und erzeugen daraus eine HTTP konforme Anfrage. Diese Anfrage wird dann an den Server gesendet.

4.2.2.1 OAI-PMH-Antworten

Exemplarisch für die Antworten im OAI-PMH soll hier die Antwort auf eine ListRecords-Anfrage dargestellt werden, da sich die spätere Übersetzung des Ergebnisses auf eine solche Anfrage bezieht. Zum besseren Verständnis der Antwort stellt Abbildung 24 die hierarchische Struktur der Antwort dar.

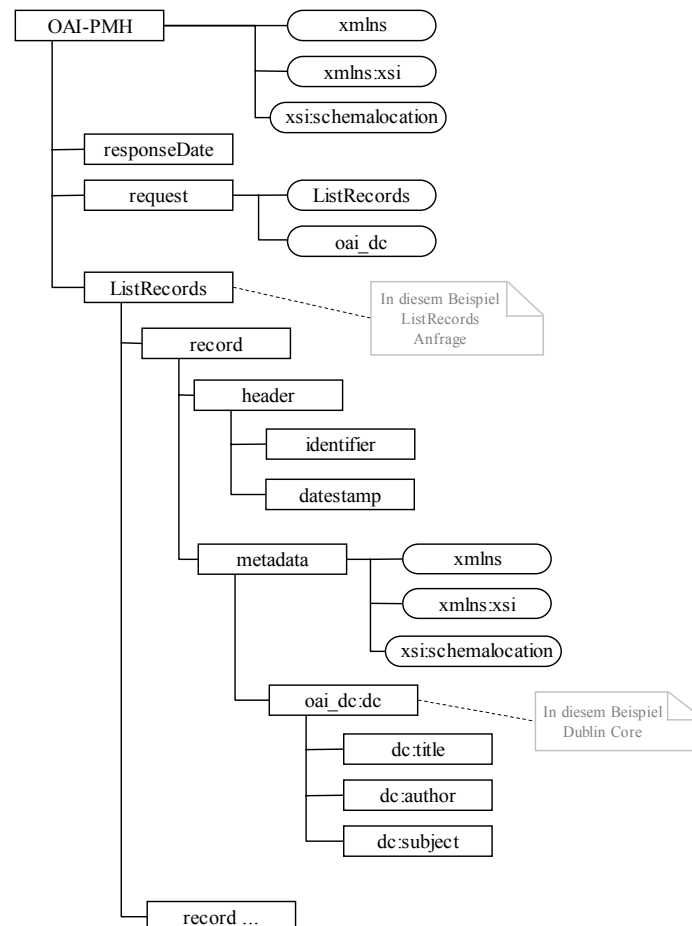


Abbildung 24: Prinzipieller Aufbau einer Antwort eines OAI Data Providers

Der Inhalt der Antwort ist in das OAI-PMH-Wurzelement eingeschlossen. Dieses Element muss drei Attribute beinhalten, die die XML-Namensräume definieren, die im weiteren Antwortdokument verwendet werden, und den Ort des Schemas, das zur Validierung verwendet werden soll.

`xmlns`

Der Wert für den Standard-Namensraum muss die URI des Namensraumes des OAI-PMH sein (<http://www.openarchives.org/OAI/2.0/>).

`xmlns:xsi`

Der Wert muss die URI für das XML-Schema sein (<http://www.w3.org/2001/XMLSchema-instance>).

`xsi:schemaLocation`

ist ein Paar, wobei der erste Teil die URI für den Namensraum des OAI-
PHM darstellt (<http://www.openarchives.org/OAI/2.0/>) und der zweite Teil
die URL für das XML-Schema, das zur Validierung des Ergebnisses ver-
wendet werden soll ([http://www.openarchives.org/OAI/2.0/OAI-
PMH.xsd](http://www.openarchives.org/OAI/2.0/OAI-
PMH.xsd)).

Im `reponseDate`-Tag wird ein UTC-Zeitwert angegeben, der aussagt, zu welcher Zeit die Antwort gesendet wurde. Zeit- und Datumsangaben werden einheitlich gemäß der ISO 8601¹⁸ angegeben. Das `request`-Tag gibt an, auf welche Protokollanfrage hin diese Antwort generiert wurde. Der Wert des `request`-Tags muss immer die Basis-URL des OAI Data Providers sein. Als Attribute des Tags müssen die Schlüssel=Wert-Paare der Anfrage angegeben werden. Im Fall eines Verarbeitungsfehlers können die Attribute jedoch weggelassen werden. Das dritte Kindelement der Rückgabe ist entweder ein `verb`-Tag, ein Element, das den gleichen Namen hat wie das Verb der Anfrage. Oder im Falle eines Fehlers ein `error`-Tag, das verwendet werden muss, wenn ein Fehler aufgetreten ist. In diesem Beispiel befindet sich die eigentliche Antwort des Servers unterhalb des `ListRecords`-Tags und besteht aus einer Reihe von `record`-Tags.

Die `record`-Tags beinhalten die eigentlichen Daten des OAI Data Providers. Ein Record besteht aus drei Teilen:

`header`

`metadata`

`about`

Der `header` beinhaltet Informationen, die alle Records gemeinsam haben. Er ist unabhängig vom Metadatenformat innerhalb des Records und ist notwendig für den Verbreitungsprozess. Die Information, die im Header definiert wird, ist eine eindeutige Kennung für den Record sowie ein Zeitstempel, der das Datum der Erstellung, Löschung und letzten Änderung der Metadaten des Record beschreibt.

Das `metadata`-Tag beinhaltet Metadaten in einem einzigen Format. Alle OAI Data Provider müssen zumindest in der Lage sein, Records auszugeben, die Dublin Core Metadaten enthalten. Andere Metadatenformate sind optional. Das `metadata`-Tag enthält mehrere Attribute.

¹⁸ siehe <http://www.w3.org/TR/NOTE-datetime>

Zunächst sind dort die Namensraum-Attribute, die sich in zwei Kategorien aufteilen:

Metadaten-spezifische Namensräume

Jeder Metadatenteil muss eine oder mehrere Namensraum-Prefix-Attribute beinhalten, die den Zusammenhang zwischen Metadata Format Prefix – im Beispiel Dublin Core: `dc` – und der Namensraum-URI des zugehörigen Metadatenformats definieren.

XML-Schema-Namensraum

Jeder Metadatenteil muss ein `xml:xsi`-Attribut enthalten, dessen Wert immer wie im Beispiel gesetzt werden muss. Es handelt sich um die URI des Namensraumes für das XML-Schema.

Als zweites ist noch das Schema zu nennen, das zu Validierung verwendet werden kann:

`xsi:schemaLocation`

Der Wert ist ein URI-URL-Paar. Die erste ist die URI des Namensraumes der Metadaten, die andere ist die URL des Schemas der Metadaten für die Validierung.

Das `about`-Tag ist ein optionaler Container, der Daten über den Metadatenteil des Records bereitstellt. Üblicherweise kann der About-Container dazu verwendet werden, Rechteinformationen oder Bedingungen über die Nutzung zu notieren.[42] Wichtig ist dabei, dass das Wurzelement des `about`-Teils die gleiche Struktur wie das Wurzelement des Metadatenteils aufweist.

4.2.2.2 XML-Notation

Eine in XML notierte Antwort auf eine `ListRecords`-Anfrage zeigt das folgende Beispiel. Da die Antwort auf eine solche Anfrage sehr lang ist, wurden bei diesem Beispiel alle Records bis auf den ersten entfernt und durch „...“ ersetzt.¹⁹

```
<?xml version="1.0" encoding="UTF-8" ?>
<OAI-PMH
  xmlns="http://www.openarchives.org/OAI/2.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
    http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2004-02-22T11:07:09Z</responseDate>
  <request verb="ListRecords" metadataPrefix="oai_dc">
    http://www.compsci-preprints.com/comp/OAI/index.htm</request>
  <ListRecords>
    <record>
```

```
      <header>
        <identifier>oai:CompSciPreprints:Conferences/0201001</identifier>
        <datestamp>2002-01-03</datestamp>
        <setSpec>M00</setSpec>
```

¹⁹ Ein vollständiges Ergebnis erhält man unter der URL:

http://www.compsci-preprints.com/comp/OAI/index.htm?verb=ListRecords&metadataPrefix=oai_dc

```

</header>

<metadata>
  <oai_dc:dc xmlns:oai_dc=http://www.openarchives.org/OAI/2.0/oai_dc/
    xmlns:dc=http://purl.org/dc/elements/1.1/
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
      http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
    <dc:creator>Eugene Izhikevich</dc:creator>
    <dc:title>
      Resonance and Selective Communication Via Bursts in Neurons
      Having Subthreshold Oscillations
    </dc:title>
    <dc:description>
      Revealing the role of bursts of action potentials is an important step toward understanding
      how the neurons communicate. The dominant point of view is that bursts are needed to in-
      crease the reliability of communication between neurons (Lisman 1997). Here
      we present an alternative but complementary hypothesis. We consider the effect a short
      burst on a model postsynaptic cell having damped oscillation of its membrane potential. The
      oscillation frequency (eigenfrequency) plays a crucial role. ...
    </dc:description>
    <dc:subject>Computational Neuro Science</dc:subject>
    <dc:date>2002-01-03</dc:date>
    <dc:type>text</dc:type>
    <dc:identifier>http://www.compsci-preprints.com/Conferences/0201001</dc:identifier>
  </oai_dc:dc>
</metadata>

<about>
  <oai_dc:dc
    xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
      http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
    <dc:publisher>Elsevier</dc:publisher>
  </oai_dc:dc>
</about>

</record>
<record> ...</record>
...
</ListRecords>
</OAI-PMH>

```

Die Antwort wurde auf eine Anfrage am 2004-02-22T11:07:09Z erstellt, wie aus dem `responseDate`-Tag hervorgeht. Da die Anfrage, auf die diese Antwort hin generiert wurde, eine `ListRecords` Anfrage mit den Attributen

```
verb=ListRecords&metadataPrefix=oai_dc
```

war, werden diese als Attribute im `response`-Tag angegeben. `oai_dc` bedeutet dabei, dass die Antwort im Dublin Core Metadatenformat zurückgegeben werden soll. Aus diesem Grund werden auch die eigentlichen Datensätze in ein `ListRecords`-Tag eingebunden. Der erste Datensatz, der innerhalb des `record`-Tags steht, beinhaltet im `header` den Unique Identifier

des ersten Records `oai:CompSciPreprints:Conferences/0201001` und den Zeitstempel `2002-01-03`. `setSpec` gibt an, dass die Daten zu der Menge *M00* des Repositories gehören.

Der Metadata-Teil beinhaltet ein einziges Wurzelement, in diesem Fall `oai_dc:dc`. Im Beispiel ist das Metadata-Format Dublin Core, weshalb Elemente wie `dc:title` enthalten sind.

Im `about`-Tag dieses Beispiels wird schließlich eine Aussage über den Herausgeber im Dublin Core Format getroffen: Elsevier.

4.2.3 Aufbau des OAI Protokoll Adapters

Das folgende UML-Klassendiagramm zeigt den internen Aufbau der OAI Protokoll Adapter.

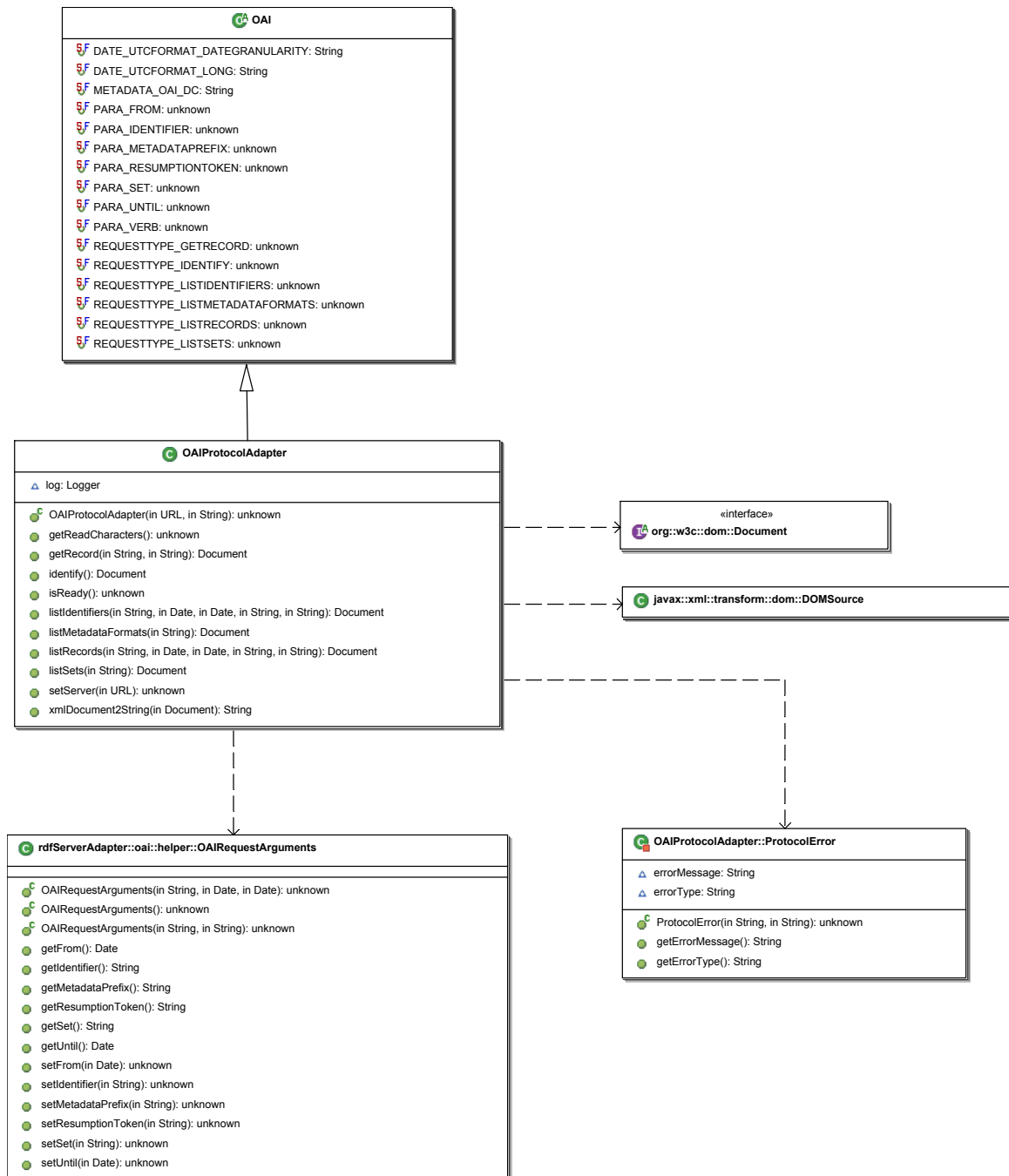


Abbildung 25: UML-Diagramm des OAI Protokoll Adapters

4.2.4 Übersetzung der OAI-Datensätze in RDF

Die Übersetzung des Ergebnisses der OAI-Anfrage soll mit Hilfe eines XML-Transformers und eines XSL-Stylesheets geschehen.

4.2.4.1 Aufbau des resultierenden RDF-Modells

Bevor das Stylesheet entworfen werden kann, muss der Aufbau des RDF-Modells, das als Resultat entstehen soll, festgelegt werden. Zur Darstellung der Dublin Core Elemente in RDF soll der Ansatz von Dave Becket, Eric Miller und Dan Brickley verwendet werden.

Zunächst muss die Verwendung von XML deklariert werden:

```
<?xml version="1.0"?>
```

Anschließend wird der RDF-Container erzeugt, der die Daten beinhaltet.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
```

Die einzelnen Ressourcen sollen durch das `rdf:description`-Tag beschrieben werden. Das `about`-Attribut soll in diesem speziellen Fall die Basis-URL des OAI Data Providers und den OAI Identifier des Datensatzes durch ein `#` getrennt enthalten.

```
<rdf:Description rdf:about="http://an.oa.org/OAI-script#oai:celebration:stowe/foiks">
...
</rdf:Description>
```

Innerhalb des `description`-Containers werden die Dublin Core Elemente zusammen mit ihrem Namespace-Prefix notiert.

```
<rdf:Description>
  <dc:title>Internet Ethics</dc:title>
  <dc:creator>Duncan Langford</dc:creator>
  <dc:format>Book</dc:format>
  <dc:identifier>ISBN 0333776267</dc:identifier>
</rdf:Description>
```

Es kann der Fall eintreten, dass ein oder mehrere Identifier eine Ressource mit URI enthalten. In diesem Fall sollte die Ressource in einem `rdf:resource`-Attribut annotiert werden, wie in folgendem Beispiel:

```
<rdf:Description rdf:about="http://.../">
  <dc:source rdf:resource="http://.../">
</rdf:Description>
```

Aus diesen Überlegungen ergibt sich ein Aufbau der Datensätze nach der Transformation, wie in Abbildung 26 dargestellt.

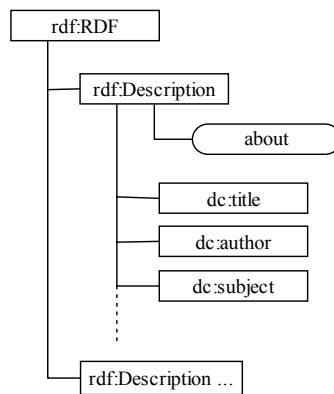


Abbildung 26: Prinzipieller Aufbau eines Datensatzes in RDF

4.2.4.2 Aufbau des XSL-Stylesheets

Aus dem Aufbau des Ergebnisdokumentes des OAI Data Providers und dem zu erreichenden Aufbau des RDF-Dokumentes ergibt sich eine Übersetzung nach dem Schema in Abbildung 27. Für jeden Record aus der OAI-Antwort soll ein `rdf:Description`-Tag erzeugt werden, dessen `about`-Attribut sich auf eine URI bezieht, die sich aus der Basis-URI des OAI Data Providers und dem OAI Identifier zusammensetzt. Anschließend sollen die Dublin Core Einträge aus den Metadaten in dieses `Description`-Tag eingetragen werden.

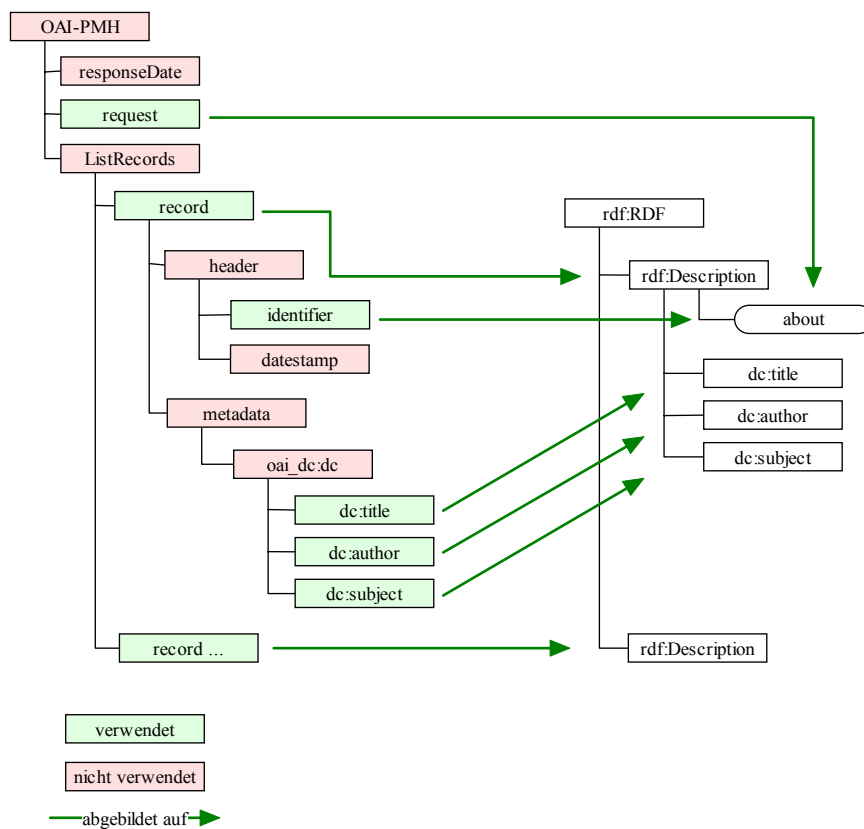


Abbildung 27: Prinzipieller Aufbau der Übersetzung von OAI nach RDF

Bei der Übersetzung der Records ist darauf zu achten, dass alle benötigten Namespaces gesetzt werden. Dies sind:

xsl

ist der Namensraum des XSL-Stylesheets

oai

ist der Namensraum, der die Tags des OAI Protokolls – wie OAI-PMH oder request – enthält.

oai_dc

Dieser Namensraum wird verwendet, um das Metadatenformat zu kennzeichnen

dc

ist der Namensraum der Dublin Core Elemente

rdf

ist der Namensraum von RDF als Zielformat

Das Wurzelement des Stylesheets hat daher den Aufbau:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oai="http://www.openarchives.org/OAI/2.0/"
  xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

Um die Datensätze als Ressource zu kennzeichnen, bedarf es einer URI. Die Basis-URL des Servers befindet sich als Wert des request-Tag als direktes Kind des OAI-PMH-Tags in der Antwort. Um die Basis-URL im gesamten Übersetzungsprozess zur Verfügung zu haben, wird diese in der globalen Variable *Server* abgelegt.

```
<!-- Speichern der BaseURI in der Variable Server -->
<xsl:variable name="Server">
  <xsl:value-of select="oai:OAI-PMH/oai:request"/>
</xsl:variable>
```

Im ersten Schritt muss das Ergebnisdokument als RDF-Dokument gekennzeichnet werden. Dazu wird ein entsprechendes Element erzeugt. Innerhalb dieses Elements werden durch *apply-templates* die einzelnen Records ausgewählt.

```
<xsl:template match="/">
  <xsl:element name="rdf:RDF">
    <xsl:apply-templates select="oai:OAI-PMH/oai:ListRecords/oai:record"/>
  </xsl:element>
</xsl:template>
```

Als nächstes werden die einzelnen Records übersetzt. Wird ein record-Tag gefunden, wird ein *rdf:Description*-Tag erzeugt, dem ein *about*-Attribut zugewiesen wird. Das *about*-Attribut enthält die bereits beschriebene URI, die aus der Servervariablen und dem Wert des *oai:identifier*-Tags zusammengesetzt ist.

```

<!-- creating the Records -->
<xsl:template match="oai:record">
  <!-- creating the rdf.Description tag -->
  <xsl:element name="rdf:Description">
    <xsl:attribute name="rdf:about">
      <xsl:value-of select="$Server" />#<xsl:value-of select="./oai:header/oai:identifier" />
    </xsl:attribute>

```

Zum Schluss werden die Dublin Core Elemente ausgelesen. Um nicht für jedes Element ein eigenes Template schreiben zu müssen, wird an dieser Stelle mit dem Platzhalter * innerhalb der Selektion gearbeitet.

```

  <!-- creating the dublin core elements -->
  <xsl:apply-templates select="./oai:metadata/oai_dc:dc/dc:*/">
</xsl:element>
</xsl:template>

```

Auf diese Weise werden sämtliche Elemente aus dem Dublin Core Namensraum ausgewählt. Das folgende `template`-Tag enthält die Verarbeitung der Elemente aus dem Dublin Core Namensraum. Zunächst wird ein neues Tag mit dem entsprechenden Namen erzeugt und anschließend der entsprechende Wert eingetragen.

```

  <!-- creating the dc-elements -->
  <xsl:template match="dc:*">
    <xsl:element name="{name()}">
      <xsl:value-of select="." />
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>

```

4.3 ILIAS-Anbindung

Das ILIAS System erlaubt einen Im- und Export der enthaltenen Daten in ein XML-Dokument. Dieses Dokument muss der ILIAS DTD entsprechen. Um die Daten als RDF-Modell bereitstellen zu können, muss das XML-Dokument in ein RDF-Dokument übersetzt werden. Für die Übersetzung muss ein XSL-Stylesheet geschrieben werden, das in der Lage ist, ein Dokument, das konform zur ILIAS DTD ist, in ein RDF-Dokument zu übersetzen.

4.3.1 ILIAS DTD

Zunächst soll der Aufbau der ILIAS-Dokumente²⁰ beschrieben werden. Nicht alle Elemente aus der ILIAS DTD werden für die Erstellung des RDF-Modells der Metadaten benötigt. In der folgenden Beschreibung soll daher nur der Teil der DTD beschrieben werden, der für die Bereitstellung der Metadaten notwendig sind.

²⁰ Die aktuelle Version der ILIAS DTD findet man unter http://www.ilias.uni-koeln.de/download/dtd/ilias_lm.dtd

4.3.1.1 ContentObject

Das ContentObjectTag ist das Wurzelement eines ILIAS-Dokumentes.

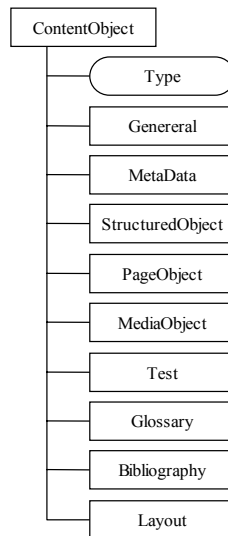


Abbildung 28: Aufbau des ContentObjects-Elementes

In diesem Element wird der gesamte Inhalt des XML-Dokumentes untergebracht. Als notwendiges Attribut des ContentObjectes muss *type* angegeben werden. Als Typ sind die Werte *LearningModule* oder *LibObject* möglich.

Als Inhalt des ContentObjects kommen General, MetaData, StructuredObject, PageObject, MediaObject, Test, Glossary, Bibliografy und Layout in Frage. Das MetaData Objekt muss genau einmal angegeben werden. Von den StructuredObject, PageObject, MediaObject muss zumindest eines vorhanden sein, die übrigen Elemente sind optional.

4.3.1.2 General

General- beinhaltet Informationen über die Ressource als Ganzes.

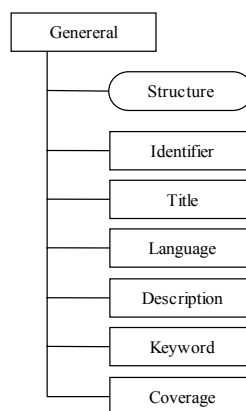


Abbildung 29: Aufbau des General-Elementes

Als notwendiges Attribut muss *Structure* angegeben werden, welches über den Aufbau des Inhaltes des Elementes Aufschluss gibt. Als mögliche Werte stehen *Atomic*, *Collection*, *Network*, *Hierachical* und *Linear* zur Verfügung. Als Kindelemente kann das *General*-Element Identifier, Title, Language, Description, Keyword und Coverage aufnehmen. Diese Elemente sollen bei der Übersetzung des ILIAS-Dokumentes als allgemeine Beschreibung übernommen werden. Der Identifier ist ein notwendiges Element und eine eindeutige Bezeichnung des Lernobjektes. Language, Description und Keyword müssen mindestens einmal angegeben werden. Coverage hingegen ist ein optionales Element.

4.3.1.3 MetaData

Das *MetaData*-Element beinhaltet die Metadaten, die sich auf das übergeordnete Element beziehen. Es besteht aus neun Elementen.

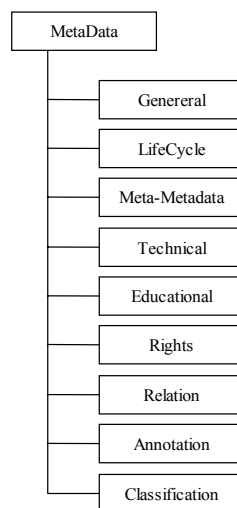


Abbildung 30: Aufbau des *MetaData* Elements

General wurde bereits in Abschnitt 4.3.1.2 beschrieben. *LifeCycle* gibt die bisherige Historie und den gegenwärtigen Status der Ressource an. Der gegenwärtige Status wird im *Status*-Attribut angegeben. Als mögliche Werte für den Status kommen *Draft*, *Final Revised* und *Unavailable* in Frage.

Das *Meta-Metadatas*-Element beinhaltet Informationen über die Metadaten an sich. Es ist optional, doch wenn es angegeben wird, muss es mindestens ein Identifier- und ein *Contribute*-Element enthalten.

Im *Technical*-Element werden die technischen Anforderungen und Charakteristika der Ressource angegeben. Es beinhaltet die Elemente *Format*, *Size*, *Location*, *Requierement*, *OrComposite*, *InstallationRemarks*, *OtherPlatformRequierments* und *Duration*.

Das *Educational*-Element beschreibt die pädagogischen Charakteristika der Ressource. Das *Rights*-Element gibt Auskunft darüber, wer Eigentümer der Rechte an der Ressource ist und unter welchen Bedingungen die Ressource verwendet werden darf.

Relation zeigt den Zusammenhang dieser Ressource mit anderen Ressourcen auf. Hier können Angaben wie *IsPartOf*, *HasPart*, *BasedOn* o.ä. angegeben werden.

Das Annotation-Element beinhaltet Informationen über die pädagogische Verwendung der Ressource.

Zuletzt noch das Classification-Element, das angibt, wo die Ressource in einem speziellen Klassifikationssystem eingeordnet werden kann.

4.3.1.4 StructureObject

Das StructureObject-Element strukturiert die Lehrmaterialien des Lernmoduls. Es besteht aus einem MetaData-Element (Abschnitt 4.3.1.3) zusammen mit mindestens einem StructureObject oder einem PageObject (Abschnitt 4.3.1.5). Alternativ kann auch ein StructureAlias-Element angegeben werden.

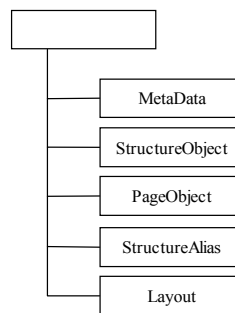


Abbildung 31: Aufbau des StructureObject Elementes

Das StructureAlias-Element dient als Alias für ein bereits existierendes StructureObject. Auf diese Weise ist es möglich, die StructureObject-Elemente wieder zu verwenden, anstatt sie neu anzugeben. Optional ist die Angabe eines Layout-Elementes möglich, das über das zu verwendende Layout des Inhaltes Auskunft gibt, wenn die Formatierung von der Standardformatierung abweichen soll.

4.3.1.5 PageObject

Das PageObject-Element ist das Objekt zur Anzeige des Inhaltes auf einem Bildschirm. Es besteht aus einem MetaData-Element und optional aus PageContent-Elementen oder einem PageAlias-Element. Darüber hinaus kann noch ein Layout-Element angegeben werden. Das PageContent-Element beinhaltet einen Paragraphen, ein MediaObject, eine Tabelle, Liste oder Dateiliste.

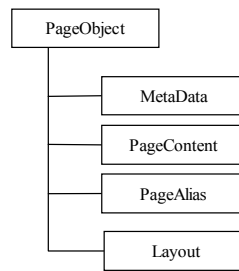


Abbildung 32: Aufbau des PageObject-Elementes

4.3.1.6 Sonstige Objekte

Das `MediaObject`-Element ist das Element mit der kleinsten Komplexität. Es beinhaltet lediglich Rohdaten, wie Bilder oder Applets, jedoch keine Textdaten. Das `Test`-Element beinhaltet einfache Tests für den Benutzer im Sinne von Multiple-Choice-Tests. Im `Glossary`-Element können Glossareinträge abgelegt werden. Das `Bibliography`-Element schließlich beinhaltet bibliografische Referenzen.

4.3.2 Übersetzung der ILIAS-Datensätze

Das Resultat der Übersetzung soll ein RDF-Modell sein, wie es in Abschnitt 4.2.4.1 beschrieben ist. Um die Übersetzung zu entwerfen, muss festgelegt werden, welche Daten aus dem ILIAS bereitgestellt werden sollen. Die Entscheidung fiel dabei auf die `StructureObject`-Elemente, die die eigentliche Struktur des Inhaltes wiedergeben. Als URI für die Daten soll die URL des ILIAS-Systems, erweitert um den Identifier des `StructureObject`-Elementes, verwendet werden. Auf diese Weise ist ein Zugriff auf die Daten mittels eines Browsers möglich.

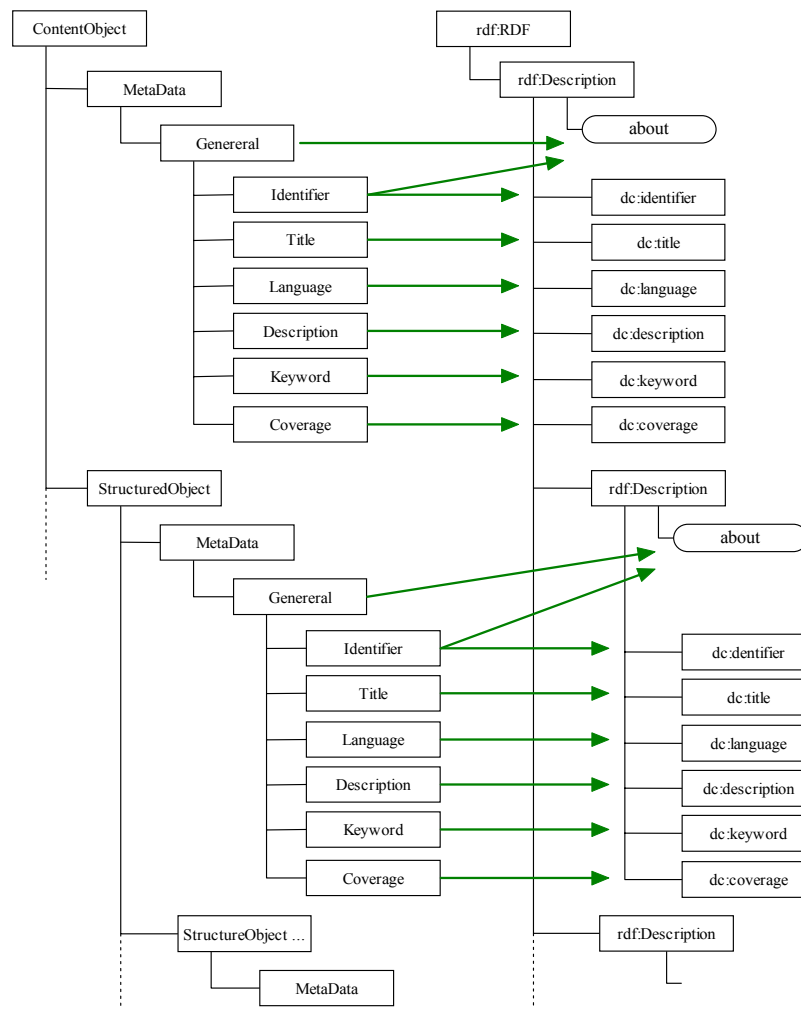


Abbildung 33: Prinzipieller Aufbau der Übersetzung von ILIAS nach RDF

Der prinzipielle Aufbau der Übersetzung der ILIAS-Datensätze nach RDF ist in Abbildung 33 dargestellt. Als Wurzelement soll das `ContentObject` als Ressource erzeugt werden. Die URI der Ressource ist die URL des ILIAS-Systems erweitert um den Identifier aus den Metadaten des `ContentObject`-Elementes. Innerhalb dieses Elementes sollen die Ressourcen für die einzelnen `StructuredObject`-Elemente abgelegt werden. Auf diese Weise lassen sich die einzelnen `StructuredObject`-Elemente eindeutig einem `ContentObject` zuordnen, wenn mehrere dieser Modelle zusammengefügt werden. Bei der Konvertierung der Metadaten werden die einzelnen Elemente auf die gleichnamigen Elemente aus dem Dublin Core Metadaten Schema übersetzt. Aus diesen Überlegungen ergibt sich das folgende XSL-Stylesheet für die Übersetzung.

4.3.2.1 Aufbau des XSL-Stylesheets

Da die Elemente des ILIAS keinen eigenen Namensraum haben, braucht bei der Auswertung für sie kein Namensraum angegeben zu werden. Es bleiben lediglich die Namensräume von RDF und Dublin Core übrig, die im Vorspann des Stylesheets angegeben werden müssen.

```

1  <xsl:stylesheet version="1.0"
2    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3    xmlns:dc="http://purl.org/dc/elements/1.1/"
4    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

```

Die URL des ILIAS-Servers kann nicht aus den Daten extrahiert werden. Deshalb muss diese URI innerhalb des Stylesheets festgelegt werden. Daraus ergibt sich die Einschränkung, dass das Stylesheet nur für ein einziges ILIAS System verwendet werden kann. Allerdings lässt sich bei der Verwendung einer globalen Variablen das Stylesheet leicht an andere Server anpassen.

```

5      <!--Base URL of the ILIAS Servers -->
6      <xsl:variable name="Server">http://www.test.org/ilias</xsl:variable>
7      <xsl:template match="/">

```

Im nächsten Abschnitt wird in den Zeilen 11 und 12 die URI für das ContentObject bestimmt. Dazu wird eine Ressource erzeugt, die den Namen des Servers, gefolgt von einem # und dem Eintrag aus dem Identifier aus den Metadaten enthält.

```

8      <xsl:element name="rdf:RDF">
9        <xsl:element name="rdf:Description">
10         <xsl:attribute name="rdf:about">
11           <xsl:value-of select="$Server" />#<xsl:value-of
12             select="/ContentObject/MetaData/General/Identifier/@Entry" />
13         </xsl:attribute>
14         <!-- creating the dublin core elements -->
15         <xsl:apply-templates select="/ContentObject/MetaData/General/Title"/>
16         <xsl:apply-templates select="/ContentObject/MetaData/General/Language"/>
17         <xsl:apply-templates select="/ContentObject/MetaData/General/Description"/>
18         <xsl:apply-templates select="/ContentObject/MetaData/General/Keyword"/>
19         <xsl:apply-templates select="/ContentObject/MetaData/General/Coverage"/>
20         <!--getting the StructureObject Elements -->
21         <xsl:apply-templates select="/ContentObject/StructureObject"/>
22       </xsl:element>
23     </xsl:element>
24   </xsl:template>

```

In diese Ressource werden zunächst die Metadaten über das ContentObject geschrieben, die sich im Metadatenteil des ContentObjectes befinden (Zeilen 15–19). Nachdem das Erzeugen der ContentObject-Ressource abgeschlossen ist, werden die Einträge für die einzelnen StructureObjects in Zeile 21 angestoßen. Bei jedem dieser Elemente wird versucht, das MetaData-Element auszulesen. Mit Hilfe der Informationen aus diesem Element wird dann eine Ressource erzeugt. Beim ContentObject wird aus der ServerURL und dem Identifier die URI für die Ressource erzeugt. Anschließend werden dann in den Zeilen 31–36 die Templates zum Erzeugen der Metadaten aufgerufen.

```

25     <!-- creating the Records -->
26     <xsl:template match="MetaData">
27       <xsl:element name="rdf:Description">
28         <xsl:attribute name="rdf:about">
29           <xsl:value-of select="$Server" />#<xsl:value-of select="./General/Identifier/@Entry" />
30         </xsl:attribute>
31         <xsl:apply-templates select="./General/Title"/>
32         <xsl:apply-templates select="./General/Language"/>
33         <xsl:apply-templates select="./General/Description"/>

```

```
34         <xsl:apply-templates select="./General/Keyword"/>
35         <xsl:apply-templates select="./General/Coverage"/>
36     </xsl:element>
37 </xsl:template>
```

Anders als beim OAI-Stylesheet kann hier nicht ein einziges Template für die Metadaten erzeugt werden, da sich kein Platzhalter angeben lässt, der sich auf alle Metadateneinträge bezieht und die Auswahl ermöglicht. Die Templates für Title, Language und Coverage übersetzen die Einträge aus den Metadaten eins zu eins in Dublin Core Metadaten. Die Einträge für Description und Keyword lesen außerdem noch die *Language*-Attribute der Einträge aus und schreiben diese in ein *Language*-Attribut in die Dublin Core Metadaten.

```
38     <!-- creating the dc-elements -->
39     <xsl:template match="Title">
40         <xsl:element name="dc:{name()}">
41             <xsl:value-of select="." />
42         </xsl:element>
43     </xsl:template>
44     <xsl:template match="Language">
45         <xsl:element name="dc:{name()}">
46             <xsl:value-of select="." />
47         </xsl:element>
48     </xsl:template>
49     <xsl:template match="Description">
50         <xsl:element name="dc:{name()}">
51             <xsl:attribute name="Language"><xsl:value-of select="./@Language" /></xsl:attribute>
52             <xsl:value-of select="." />
53         </xsl:element>
54     </xsl:template>
55     <xsl:template match="Keyword">
56         <xsl:element name="dc:{name()}">
57             <xsl:attribute name="Language"><xsl:value-of select="./@Language" /></xsl:attribute>
58             <xsl:value-of select="." />
59         </xsl:element>
60     </xsl:template>
61
62     <xsl:template match="Coverage">
63         <xsl:element name="dc:{name()}">
64             <xsl:value-of select="." />
65         </xsl:element>
66     </xsl:template>
67
68 </xsl:stylesheet>
```

5 Ausblick

Innerhalb dieser Arbeit wurde gezeigt, wie mit Hilfe vorhandener Standards digitale Bibliotheken, DSpace und das ILIAS-System an das Edutella-Peer-to-Peer-Netzwerk angebunden werden können.

Da die Konzeption des Server-Adapters, der einen Informationsanbieter mit dem Edutella-Netzwerk verbindet, auf verbreiteten Standards beruht, ist es leicht möglich weitere Informationsanbieter in das Edutella-Netzwerk zu integrieren. Die nahezu einzige Voraussetzung ist die Unterstützung eines XML-Exportes der zugrunde liegenden Daten, um anschließend aus diesen Daten ein RDF-Modell zu erzeugen.

Ein Beispiel für eine zukünftige Anbindung wäre ein relationales Datenbank Management System (DBMS). Die Realisierung einer Anbindung könnte gemäß des folgenden Schemas geschehen.

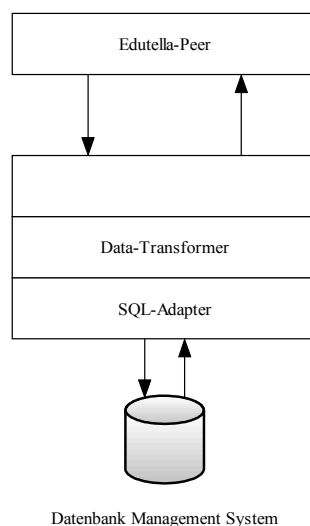


Abbildung 34: Anbindung eines Datenbank Management Systems

Ein SQL-Adapter fragt die Daten des DBMS ab. Die erhaltene Antwort könnte in ein XML-Dokument übersetzt werden, das mittels eines Data-Transformers in ein RDF-Modell transformiert wird. Dieses RDF-Modell kann anschließend über die Server-Adapter-Schnittstelle dem Edutella-Peer zur Verfügung gestellt werden.

Durch die Anbindung digitaler Bibliotheken und anderer Informationsanbieter ergibt sich eine wesentliche Bereicherung im Bereich der Informationsbereitstellung für Benutzer des Edutella-Netzwerkes. Sie können auf diese Weise auf eine Vielzahl von Veröffentlichungen zurückgreifen und diese in ihren täglichen Arbeitsprozess integrieren. Bei der Suche nach Informationen ist es nicht mehr notwendig, die Homepages einzelner digitaler Bibliotheken zu besuchen, sondern vielmehr wird durch die Anbindung eine zentrale Zugriffs- und Suchmöglichkeit über das Edutella-Netzwerk geschaffen.

Es bleibt zu hoffen, dass sich in Zukunft möglichst viele Informationsanbieter an das Edutella-Netzwerk anbinden werden und dadurch das Informationsangebot bereichern.

Anhang

A OAI-Anfrage-Attribute

GetRecord

Diese Anfrage wird verwendet um einen bestimmten Record aus dem Repository auszulesen.

	erforderlich	optional	exklusiv
Argumente	metadataPrefix identifier		
Fehlermeldungen	badArgument, cannotDisseminateFormat, idDoesNotExist		

Ein Beispiel ist:

- `http://arXiv.org/oai2?verb=GetRecord
&identifier=oai:arXiv.org:cs/0112017
&metadataPrefix=oai_dc`

Identify

Identify gibt Informationen über das Repository zurück, an das die Anfrage gerichtet wurde.

	erforderlich	optional	exklusiv
Argumente			
Fehlermeldungen	badArgument		

Ein Beispiel ist:

- `http://memory.loc.gov/cgi-bin/oai?verb=Identify`

ListIdentifiers

Ist eine abgekürzte Form der ListRecords Anfrage, die lediglich die Header der Records zurückliefert anstatt den gesamten Record.

	erforderlich	optional	exklusiv
Argumente	metadataPrefix	identifier from until set	resumptionToken
Fehlermeldungen	badArgument, cannotDisseminateFormat, badResumptionToken, noRecordsMatch, noSetHierarchy		

Ein Beispiel ist:

- `http://an.oa.org/OAI-script?
verb=ListIdentifiers&from=1998-01-15&metadataPrefix=oldArXiv&set=physics:hep`

ListMetadataFormats

Dieses Verb wird dazu verwendet, die Metadatenformate, die das Repository zur Verfügung stellt zu erhalten.

	erforderlich	optional	exklusiv
Argumente		identifier	
Fehlermeldungen	badArgument, idDoesNotExist, noMetadataFormats,		

Ein Beispiel ist:

- <http://www.perseus.tufts.edu/cgi-bin/pdataprov?verb=ListMetadataFormats&identifier=oai:perseus.tufts.edu:Perseus:text:1999.02.0119>

ListRecords

Dieses Verb wird verwendet, um Records aus dem Repository auszulesen.

	erforderlich	optional	exklusiv
Argumente	metadataPrefix	from until set	resumptionToken
Fehlermeldungen	badArgument, cannotDisseminateFormat, badResumptionToken, noRecordsMatch, noSetHierarchy		

Ein Beispiel ist:

- http://an.oa.org/OAI-script?verb=ListRecords&from=1998-01-15&set=physics:hep&metadataPrefix=oai_rfc1807

ListSets

Gibt eine Struktur der Mengen des Repositories an den Anfrager zurück.

	erforderlich	optional	exklusiv
Argumente			resumptionToken
Fehlermeldungen	badArgument, badResumptionToken, noSetHierarchy		

Ein Beispiel ist:

- <http://an.oa.org/OAI-script?verb=ListSets>

B XML-Stylesheets

B.1 OAI

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:oai="http://www.openarchives.org/OAI/2.0/"
  xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!-- Speichern der BaseURI in der Variable Server -->
  <xsl:variable name="Server">
    <xsl:value-of select="oai:OAI-PMH/oai:request"/>
  </xsl:variable>
  <xsl:template match="/">
    <xsl:element name="rdf:RDF">
      <xsl:apply-templates select="oai:OAI-PMH/oai:ListRecords/oai:record"/>
    </xsl:element>
  </xsl:template>
  <!-- creating the Records -->
  <xsl:template match="oai:record">
    <!-- creating the rdf.Description tag -->
    <xsl:element name="rdf:Description">
      <xsl:attribute name="rdf:about">
        <xsl:value-of select="$Server" />#<xsl:value-of select="." />oai:header/oai:identifier" />
      </xsl:attribute>
      <!-- creating the dublin core elements -->
      <xsl:apply-templates select="." />oai:metadata/oai_dc:dc:*/>
    </xsl:element>
  </xsl:template>
  <!-- creating the dc-elements -->
  <xsl:template match="dc:*">
    <xsl:element name="{name()}">
      <xsl:value-of select="." />
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

B.2 ILIAS

```

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <!--Base URL of the ILIAS Servers -->
  <xsl:variable name="Server">http://www.test.org/ilias</xsl:variable>
  <xsl:template match="/">
  <xsl:element name="rdf:RDF">
    <xsl:element name="rdf:Description">
      <xsl:attribute name="rdf:about">
        <xsl:value-of select="$Server" />#<xsl:value-of
          select="/ContentObject/MetaData/General/Identifier/@Entry" />
        </xsl:attribute>
      <!-- creating the dublin core elements -->
      <xsl:apply-templates select="/ContentObject/MetaData/General/Title"/>
      <xsl:apply-templates select="/ContentObject/MetaData/General/Language"/>
      <xsl:apply-templates select="/ContentObject/MetaData/General/Description"/>
      <xsl:apply-templates select="/ContentObject/MetaData/General/Keyword"/>
      <xsl:apply-templates select="/ContentObject/MetaData/General/Coverage"/>
      <!--getting the StructureObject Elements -->
      <xsl:apply-templates select="/ContentObject/StructureObject"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
<!-- creating the Records -->
<xsl:template match="MetaData">
  <xsl:element name="rdf:Description">
    <xsl:attribute name="rdf:about">
      <xsl:value-of select="$Server" />#<xsl:value-of select="./General/Identifier/@Entry" />
    </xsl:attribute>
    <xsl:apply-templates select="./General/Title"/>
    <xsl:apply-templates select="./General/Language"/>
    <xsl:apply-templates select="./General/Description"/>
    <xsl:apply-templates select="./General/Keyword"/>
    <xsl:apply-templates select="./General/Coverage"/>
  </xsl:element>
</xsl:template>
<!-- creating the dc-elements -->
<xsl:template match="Title">
  <xsl:element name="dc:{name()}">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
<xsl:template match="Language">
  <xsl:element name="dc:{name()}">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
<xsl:template match="Description">
  <xsl:element name="dc:{name()}">
    <xsl:attribute name="Language"><xsl:value-of select="./@Language" /></xsl:attribute>
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
<xsl:template match="Keyword">
  <xsl:element name="dc:{name()}">
    <xsl:attribute name="Language"><xsl:value-of select="./@Language" /></xsl:attribute>
    <xsl:value-of select="." />
  </xsl:element>

```

```
</xsl:template>
<xsl:template match="Coverage">
  <xsl:element name="dc:{name()}">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Literaturverzeichnis

- [1] D. Otto. „Informationsportale und digitale Bibliotheken“. Konrad-Zuse-Zentrum für Informationstechnik. Berlin 2001.
http://webdoc.gwdg.de/ebook/e/2003/zib_2/reports/ZR-01-21.pdf
- [2] M. O. Will. Aufbau und Nutzung einer digitalen Bibliothek in einer universitären Ausbildungsumgebung. Medien in der Wissenschaft Band 22, Gesellschaft für Medien in der Wissenschaft e.V. Waxmann Verlag Münster. 2002
- [3] Napster. <http://www.napster.com/>.
- [4] Gnutella, <http://www.gnutella.com/>
- [5] Organisation und Nutzung von verteilten Inhalten und Lehrmaterialien. November 2001.
<http://www.learninglab.de/padlr/>.
- [6] W. Arms. Digital Libraries“; MIT Press, Cambridge, Massachusetts. 2000
- [7] T. Meyer. Seminar Internet-Mehrwertdienste. Oldenburg, Februar 2000
<http://www.informatik.uni-oldenburg.de/~totti/metadaten/>,
- [8] T. Berners-Lee. Metadata Architecture. W3C. Januar 1997
<http://www.w3.org/DesignIssues/Metadata.html>
- [9] People involved in the Dublin Core Metadata Initiative. 2004
<http://www.dublincore.org/about/participants/>
- [10] History of the Dublin Core Metadata Initiative. 2004
<http://www.dublincore.org/about/history/>,
- [11] Dublin Core Metadata Initiative. <http://www.dublincore.org/>. 2004
- [12] C. Lagoze, H. Van de Sompel. The Open Archives Initiative: Building a low-barrier interoperability framework. Cornell University. Ithaca, NY, JCDL. Juni 2001
www.openarchives.org/documents/jcdl2001-oai.pdf,
- [13] S. Ohme. Konzeption von Dokumentenservern für Digitale Bibliotheken im Hinblick auf Langzeitarchivierung und Retrieval. Institut für Informatik Potsdam, Oktober 2003.
<http://www.informatikdidaktik.de/Examensarbeiten/Ohme2003.pdf>
- [14] Smith, MacKenzie et all. DSpace – An Open-Source Dynamic Digital Repository. D-Lib Magazine. Volume 9 Number 1. Januar 2003,

-
- [15] DSpace Federation Functional Overview. 2003
<http://dspace.org/technology/function.html>
- [16] DSpace Federation. MIT Libraries & Hewlett-Packard Company. Februar 2004.
<http://www.dspace.org/>
- [17] M. Kunzel. Virtuelle Universitätssysteme. April 2002
<http://www.virtus.uni-koeln.de/>
- [18] M. Kunkel. ILIAS open source. Februar 2004
<http://www.ilias.uni-koeln.de/ios/index.html>
- [19] Funktionalitäten von ILIAS.
http://www.ilias.uni-koeln.de/ios/docs/i3_funktionsuebersicht.pdf
- [20] CampusSource. CampusSource · Software · ILIAS. CampusSource. Januar 2004
<http://www.campussource.de/software/ilias/>
- [21] ILIAS User Help. 2004
<http://www.homer.ilias.uni-koeln.de/ilias/help/en/user/index.php?z=120.html>
- [22] ILIAS DOC. 2004
<http://www.homer.ilias.uni-koeln.de/iliasdoc/doc/html/1.html>
- [23] D. Hoppmann. ITWissen – XML. 2003
<http://www.hoppmann-online.de/ITWissen/XML.html>
- [24] Tischer, Jennrich. Internet intern – Technik und Programmierung. Data Becker. 1997
- [25] F. Yergeau, J. Cowan, T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler. Extensible Markup Language (XML) 1.1, W3C Recommendation. February 2004
<http://www.w3.org/TR/2004/REC-xml11-20040204/>
- [26] O. Lassil, R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation. February 1999
<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>
- [27] W. Nejdl, B. Wolf, W. Siberski, C. Qua, S Decker, M. Sintekc, A. Naeved, M. Nils-sond, M. Palmérd, T. Rische. EDUTELLA: P2P Networking for the Semantic Web. Technical Report, 2003.
- [28] B. Wolf. Peer-to-Peer Networking for Distributed Learning Repositories. Diplomarbeit. Dezember 2001
- [29] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Löser, SuperPeerBased Routing and Clustering Strategies for RDFBased PeerToPeer
-

Networks, WWW2003, May 2003, Budapest, Hungary.

- [30] M. Schlosser, M. Sintek, S. Decker, W. Nejdl. HyperCuP – Hypercubes, Ontologies and Efficient Search on P2P Networks, In International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, Juli 2002.
- [31] Anfang Datalog Quellen
- [32] M. Nilsson, W. Siberski. RDF Query Exchange Language (QEL) - concepts, semantics and RDF syntax. März 2004
<http://edutella.jxta.org/spec/qel.html> ,
- [33] Sun Microsystems, Inc. Project JXTA: An Open, Innovative Collaboration. April 2001
<http://www.jxta.org/project/www/docs/OpenInnovative.pdf>
- [34] L. Gong. Project JXTA: A Technology Overview. Sun Microsystems, Inc. Oktober 2002
<http://www.jxta.org/project/www/docs/TechOverview.pdf>
- [35] JXTA v2.0 Protocols Specification. Sun Microsystems, Inc. February 2003
<http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html>
- [36] H. Thompson, The Extensible Stylesheet Language Family (XSL). 2004
<http://www.w3.org/Style/XSL/>
- [37] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation. November 1999
<http://www.w3.org/TR/xpath>
- [38] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation. November 1999
<http://www.w3.org/TR/xslt>
- [39] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation. November 1999
<http://www.w3.org/TR/xpath#NT-LocationPath>
- [40] W. Nejdl, B. Wolf, S. Staab, J. Tane, EDUTELLA: Searching and Annotating Resources within an RDFbased P2P Network, Semantic Web Workshop, at the 11th International World Wide Web Conference (WWW2002), Hawaii, USA, May 2002
- [41] L. Carpenter, Main Technical Ideas of OAI-PMH, University of Bath, Oktober 2003
<http://www.oaforum.org/tutorial/english/page3.htm>
- [42] W. von Keitz. „Vom Buch zum Bit?? – Analoges zur Digitalen Bibliothek“. Novem-

ber 1997. <http://v.hdm-stuttgart.de/~keitz/digilib.html>

- [43] R. Böhme. Resource Description Framework. Problemseminar "*Datenbankeinsatz im Internet*". Juni 1999
<http://dbs.uni-leipzig.de/de/seminararbeiten/semSS99/arbeit5/Rdf.html>,
- [44] D. Brickley, R.V. Guha. RDF Schema Specification 1.0. W3C. März 2000
<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>

Abbildungsverzeichnis

Abbildung 1: Schematischer Netzwerkaufbau einer digitalen Bibliothek.....	7
Abbildung 2: Prinzipielle Funktionsweise des OAI-PHM.....	12
Abbildung 3: DSpace Information Model.....	13
Abbildung 4: DSpace Datenmodell.....	15
Abbildung 5: DSpace technische Architektur.....	16
Abbildung 6: ILIAS Persönlicher Schreibtisch.....	17
Abbildung 7: ILIAS Masthead.....	17
Abbildung 8: RDF-Aussage als Graph.....	25
Abbildung 9: RDF-Aussage mit anonymer Ressource.....	25
Abbildung 10: RDF-Aussage mit mehreren Ressourcen.....	26
Abbildung 11: Client/Server-Architektur.....	32
Abbildung 12: Peer-to-Peer-Architektur.....	33
Abbildung 13: Peer Verbindungen im Super-Peer Backbone.....	35
Abbildung 14: JXTA-Layerkonzept.....	41
Abbildung 15: Ablauf einer XSL-Transformation.....	43
Abbildung 16: Prinzipieller Aufbau der Anbindung einer digitalen Bibliothek an das Edutella-Netzwerk.....	54
Abbildung 17: Anfrageformate im RDQL-Provider.....	55
Abbildung 18: Server Adapter Klassendiagramm.....	56
Abbildung 19: Interner Aufbau der des OAI Server Adapter.....	58
Abbildung 20: Informationsfluss im OAI Protokoll Adapter.....	59
Abbildung 21: Informationsfluss im XML File Adapter.....	61
Abbildung 22: UML-Klassendiagramm RDF Server Adapter.....	63
Abbildung 23: Übersicht des strukturellen Modells.....	64
Abbildung 24: Prinzipieller Aufbau einer Antwort eines OAI Data Providers.....	67
Abbildung 25: UML-Diagramm des OAI Protokoll Adapters.....	72
Abbildung 26: Prinzipieller Aufbau eines Datensatzes in RDF.....	74
Abbildung 27: Prinzipieller Aufbau der Übersetzung von OAI nach RDF.....	74
Abbildung 28: Aufbau des ContentObjects-Elementes.....	77
Abbildung 29: Aufbau des General-Elementes.....	77
Abbildung 30: Aufbau des MetaData Elements.....	78
Abbildung 31: Aufbau des StructureObject Elementes.....	79

Abbildung 32: Aufbau des PageObject-Elementes	80
Abbildung 33: Prinzipieller Aufbau der Übersetzung von ILIAS nach RDF	81
Abbildung 34: Anbindung eines Datenbank Management Systems	84