Digitale Bibliotheken

Überblick über Metadatenstandards und praktische Anwendung im Bereich der Personenidentifikation

Diplomarbeit

vorgelegt am
Lehrstuhl für Wirtschaftsinformatik III
Prof. Dr. Dr. M. Schader
Universität Mannheim

im
Juli 1999

von
Markus Johannes Richard Klink
aus Bonn – Bad Godesberg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Das Internet stellt dem interessierten Benutzer eine Unmenge von Informationen zur Verfügung, die dieser auswerten kann – vorausgesetzt, daß er sie findet. Bedingt durch die Informationsflut im Internet ist das Auffinden der richtigen, gewünschten Informationen allerdings immer schwieriger geworden. Zahlreiche Suchmaschinen bieten Einstiegspunkte zum „information retrieval“ und sammeln die Informationen mittels unterschiedlicher Strategien. Dazu gehören die Indizierung ganzer Webseiten oder die manuelle Durchsichtung der Informationen und ihre entsprechende Klassifizierung in zugehörige Verzeichnisse.

Der dafür zu treibende Aufwand und die mangelhafte Qualität der letztendlich gefunden Informationen sorgte in den letzten Jahren für ein gestiegenes Interesse an „Digitalen Bibliotheken“ und an Möglichkeiten, der Informationsflut durch Standardisierungsbemühungen besser Herr zu werden.

Digitale Bibliotheken unterscheiden sich einerseits grundlegend von herkömmlichen Bibliotheken, andererseits haben sie auch hinreichend viele Gemeinsamkeiten. Das Bibliothekswesen ist ein alter Wissenschaftszweig mit einen großen Erfahrungsschatz. Diese Erfahrungen sind in Form von Regeln kodiert. Ein Teil der Diplomarbeit widmet sich daher der Vorstellung der wichtigsten bibliothekarischen Standards und ihrer elektronischen Datenformate, um das Problembewußtsein für Fragen der Katalogisierung zu verschärfen. Auf diesen Wissen aufbauend sollen aktuelle Entwicklungen im Bereich des Internets und digitaler Bibliotheken diskutiert werden.

Im bibliothekarischen Sprachgebrauch wird in der Regel zwischen einen Katalog und einer Bibliographie unterschieden. Ein Katalog erschließt einen Bestand, der sich in einer oder

mehreren Bibliothek befindet, und ermöglicht das Auffinden eines bestimmten Werkes, wenn eine bestimmte Anzahl von Suchkriterien bekannt sind (in der Regel Autor und Titel). Eine Bibliographie faßt Werke eines bestimmten Themas oder eines Sachgebietes zusammen, ohne Rücksicht darauf zu nehmen, wo sich diese konkret befinden. Mit Katalog und Bibliographie stehen also zwei Instrumente zur Verfügung, die ein effizientes Benutzen des Bibliotheksbestandes ermöglichen, da sie die gängigsten Fragen („Wo finde ich ein Buch von X und wie finde ich mehr zu einem Thema?“) beantworten können.

Von einer digitalen Bibliothek sollte man also zumindest erwarten können, daß sie katalogisierte und bibliographische Suchfunktionen unterstützt. Andererseits gibt es in digitalen Bibliotheken keinen physikalischen Aufbewahrungsraum und die Dokumente haben oftmals multimedialen Charakter, das heißt sie bestehen aus Texten, Audio- und Videodaten. Des weiteren ist es aufgrund der Hyperlinks in HTML oftmals schwer auszumachen, wo ein Dokument „beginnt“ und wo es „endet“. Die Dokumente im Internet haben also einen grundlegend anderen Charakter als die einer herkömmlichen Bibliothek. Oftmals ist es auch schwierig die Qualität einer Ressource zu bestimmen. Daß Dokumente im Internet meist digital vorliegen, hat aber auch unschätzbare Vorteile. So sind gespeicherte Ressourcen im Volltext durchsuchbar. Suchanfragen sind schneller als ein Heranziehen von Karteikarten, und vor allem sind theoretisch viele Suchkategorien denkbar, das heißt es gibt viele verschiedene Bibliographien. Weiterhin ist der Bestand einer digitalen Bibliothek weltweit abrufbar. Sind die Dokumente auch im Volltext verfügbar ist das ein enormer Vorteil, weil so Wissen einer viel breiteren Schicht von Benutzern zur Verfügung gestellt werden kann.

In der aktuellen Forschung über digitale Bibliotheken und verwandter Themengebiete wie „digital teaching“ lassen sich zwei Grundströmungen festmachen, die folgend kurz umrissen werden:

Der eine Forschungszweig beschäftigt sich mit den technischen Problemen digitaler Bibliotheken. Hier beschäftigt man sich mit Fragen der technischen Realisierung von digitalen Zugriffen auf Dokumente. Unterschiedliche Übertragungsprotokolle und Dateiformate sind hier die dominierenden Themen, ebenso wie Probleme der Identifizierung von Ressourcen oder der Kodierung von Urheberinformationen in nichttextuellen Dokumenten (zum Beispiel bei Graphiken, Videos und Musik).

Der andere große Forschungszweig widmet sich den sozio-ökonomischen Problemen, die digitale Bibliotheken mit sich bringen. Darunter sind unter anderen Fragen der Benutzbarkeit von digitalen Bibliotheken zu verstehen. Wie lernen Menschen optimal mit Hilfe

des Computers? Wie können Informationen so aufbereitet werden, daß ihre Benutzung denen eines herkömmlichen Textes ebenbürtig, wenn nicht sogar überlegen ist? Weitere Fragestellungen des sozio-ökonomischen Problemkreises behandeln insbesondere die Anwendung von modernen Technologien für die Länder der sogenannten dritten Welt, oder stellen Fragen, wer überhaupt für die Pflege der Datenbestände verantwortlich ist, und was ihre Aufbereitung kostet (Thomas and Griffin, 1998).

In dieser Diplomarbeit werden in Kapitel 2 Standards herkömmlicher Bibliotheken und sich entwickelnde Standards zur Internetressourcenbeschreibung vorgestellt. Ziel ist es, dem Leser eine Vorstellung vom Problembereich dieser Aufgaben zu vermitteln. In Kapitel 3 werden dann eine Strategie und dazugehörige Programme entwickelt, um die Suchmöglichkeiten in einer elektronischen Bibliothek für wirtschaftswissenschaftliche Arbeitspapiere zu verbessern. Für Personen wird im Rahmen dieser Diplomarbeit ein webbasierter Service implementiert, der es ermöglicht, sich in der Bibliothek zu registrieren und Bezüge zu erstellten Dokumenten herzustellen. Informationen über Dokumente und mit ihnen assoziierte Personen, wie Autoren und Editoren, sollen relational verknüpft werden.

Dafür ist eine Restrukturierung der zugrundeliegenden Datenstruktur notwendig, um Datenredundanz und Updateprobleme zu beheben, aber ohne die Funktionsweise bereits bestehender Softwarelösungen zu beeinträchtigen. Für Benutzer der Bibliothek wird weiterhin ein Suchdienst implementiert, der ein effizienteres Aufinden von Dokumenten bei Kenntnis des Autors ermöglicht. Dieser Service nutzt die neu geschaffenen Datenstrukturen, muß aber auch mit den „Altlasten“ in Form von nicht-standardisierten Namenseinträgen zurechtkommen.

Kapitel 2

Metadaten

2.1 Warum Metadatenstandards?

Definition

Allgemein ausgedrückt sind „Metadaten“ Daten über Daten. Dabei ist diese Unterscheidung nicht eindeutig, denn sowohl Daten als auch Metadaten, können ihre Rolle wechseln. Zum Beispiel werden für eine Universitätsbibliothek die vorhandenen Bücher katalogisiert. Dieser Katalog kann dann als Metadatensatz über die Bibliothek aufgefaßt werden. Katalogisiert nun jemand wiederum alle ihm bekannten Bücherkataloge, so werden aus den ursprünglichen Metadaten wiederum Daten. Der Begriff Metadaten ist erstmals im Rahmen der Datenbankforschung Mitte der 70er Jahre aufgetreten, um das Datenbankschema von den Daten zu trennen und klarzumachen, daß es sich auch beim Datenbankschema wiederum um Daten handelt.

Interne und externe Metadaten

Man unterscheidet weiterhin für gewöhnlich zwischen „internen“ und „externen“ Metadaten. „Interne“ Metadaten sind mit der Ressource, die sie beschreiben, direkt verknüpft, während „externe“ Metadaten von der Ressource, die sie beschreiben, getrennt sind. Ein Beispiel für „externe“ Metadaten wären Karteikarten einer Bibliothek, denn sie sind mit der zu beschreibenden Ressource nicht verbunden. „Interne“ Metadaten existieren dagegen zum Beispiel in der Form von „Cataloging in Publication“ (CIP) Einträgen, mit denen

Bücher beschrieben werden. CIP-Einträge sind in der Regel ein Teil des Buches (meist auf einer der vorderen Seiten gedruckt) und daher nicht von der Ressource trennbar. Beide Formen sind sowohl im traditionellen Bibliographiewesen als auch im Internet zu finden. Der Vorteil externer Metadaten ist, daß sie zweckgerichtet erstellt werden können, ohne die zu erfassende Ressource zu verändern. Ein Nachteil ist der lose Verbund zwischen Daten und Ressource – geht die Verbindung zwischen beiden verloren, so sind auch die Daten zwecklos (und die Ressource verloren).

Metadatenformate sind im Bereich traditioneller Bibliotheken schon seit langem verbreitet. Die Entwicklung entsprechender Standards für die elektronische Datenverarbeitung und das Internet greift auf die dort bereits gewonnenen Erkenntnisse zurück. Die Forschung und Entwicklung auf diesem Gebiet ist daher weitestgehend interdisziplinär. Darum werden in dieser Diplomarbeit verschiedene, bereits etablierte Standards traditioneller Bibliotheken umrissen und Entwicklungen im Bereich von elektronischen Bibliotheken gegenübergestellt.

Warum überhaupt Standards für Metadaten?

Bevor ein Standard entwickelt wird, sollte man sich fragen, ob es überhaupt sinnvoll ist, einen solchen zu entwickeln. Letztendlich stellt ein Standard oft auch einen Interessenkompromiß dar, der Gefahr läuft, den Bedürfnissen der Benutzer nicht mehr zu genügen, da er zu allgemein oder zu spezifisch ist. Ist ein Standard zu allgemein, so läuft er Gefahr, daß er nicht umgesetzt werden kann, weil er zu vage ist. Ein fiktiver Standard METAKLINK könnte zum Beispiel fordern, daß Dokumente durch Angabe eines Titels und der Autoren zu beschreiben sind. Er läßt die Frage offen, was ein Dokument ist (ein Buch, Artikel, oder ein Bild?), und erfaßt nicht, wer als Autor eines Dokuments zählt. Der Austausch von Daten im METAKLINK-Format ist im besten Falle also schwierig, wenn nicht unmöglich.

Eine Weiterentwicklung von METAKLINK könnte bis ins kleinste Detail festlegen, welche Daten für bestimmte Dokumente erhoben werden müssen, und dabei vernachlässigen, daß bestimmte Anwendergruppen diese Daten entweder gar nicht benötigen oder eben andere Daten erfassen wollen. Ein Ausweg aus diesem Dilemma besteht darin, sich auf einen Kernsatz von Daten zu beschränken und eine Methode zu definieren, die es ermöglicht, den Standard wohldefiniert zu erweitern. Eine Möglichkeit bestünde in der Anwendung des Resource Description Frameworks (siehe Kapitel 2.3.3).

Die Motivation für die Entwicklung von Metadatenstandards im Internet liegt im Internet selbst. Durch das beständige Wachstum des Internet wird es für den Benutzer immer schwieriger, gewünschte Informationen zu finden. Die Menge an Informationen verhindert also eine effiziente Nutzung der Information. Dies liegt zum großen Teil daran, daß die meisten Dokumente des Internet in HTML abgefaßt sind. HTML eignet sich zwar dafür, Dokumente zu strukturieren, aber es trennt nicht zwischen Struktur und Aussehen der Information. Deshalb ist es zwar maschinenlesbar (*machine-readable*), aber eben nicht maschinenverständlich (*machine-understandable*). HTML begünstigt es leider, daß im Dokument enthaltene Information nach erfolgter Kodierung verloren geht. Zum Beispiel kann eine Kapitelüberschrift eines Dokuments in HTML als:

```
<H1>Vorwort</H1>
```

angegeben werden. Es ist nun nicht mehr nachvollziehbar, daß es sich bei dieser Information um eine Überschrift handelt, es ist vielmehr ein in irgendeiner Form hervorgehobener Text.

Die eXtensible Markup Language (XML), welche vom World Wide Web Konsortium standardisiert wird, soll dieses Problem beheben, indem es einen Nachfolger von HTML definiert, der Aussehen und Struktur voneinander trennt. Eine tiefere Diskussion dieses Mechanismus findet sich in Kapitel 2.3.4.

2.2 Standards zur Erfassung von Metadaten in herkömmlichen Bibliotheken

Bereits zu Anfang der Diplomarbeit wurde erklärt, daß Katalog und Bibliographie die Hauptwerkzeuge zur Nutzung einer Bibliothek darstellen. Die folgend vorgestellten Standards bilden das Regelwerk, mit dessen Hilfe ein Bibliothekar diese Werkzeuge erstellt. Die Komplexität der Regeln kann hier nur angerissen werden – es ist allerdings hilfreich, sich den Umfang der Problematik, mit denen Bibliotheken konfrontiert sind, bewußt zu machen.

Der Umfang der zu behandelnden Daten in einer Bibliothek wie der amerikanischen Library of Congress ist gewaltig. Bei mehreren Millionen Datensätzen ist es verständlich, daß die Daten nach strengen Regeln erfaßt werden müssen. Was für einige hunderttausend Datensätze noch funktionieren mag, funktioniert bei großen Datenmengen längst nicht mehr.

Bibliotheken müssen Daten in einer Vielzahl von Sprachen behandeln. Im Internet kommen erst jetzt Standards wie XML zum Zug, die sich dieser Problematik widmen, um zum Beispiel Buchstaben auf beliebigen Rechnersystemen korrekt zu kodieren und darzustellen. Weiterhin sind natürliche Sprachen in stetigem Wandel. Benutzer, die nach „Wasserschifffahrt“ suchen, wollen sich in der Regel nicht mit den Regeln der Rechtschreibreform auseinandersetzen und erwarten auch ältere Texte zum Thema „Wasserschiffahrt“ im Suchergebnis. Englische Bibliotheksbenutzer auf der Suche nach Informationen über die Stadt „Köln“ suchen u.U. unter „Koeln“ oder „Cologne“. In der Regel werden sie dort auch fündig und treffen auf die gewünschten Informationen. Filmliebhaber suchen nach „Woody Allen“ und finden Informationen über sein Filmschaffen, obwohl sein eigentlicher Name „Allen Stewart Konigsberg“ ist.

Weiterhin müssen Bibliotheken Altlasten pflegen. In einer Bibliothek kann man nicht mal eben einen neuen Standard einführen. Der Standard MARC, der später kurz vorgestellt wird, ist der weltweit meistverwendete EDV-Standard zum kontrollierten Datenaustausch. Änderungen an diesen Standards und der Funktionsweise einer Bibliothek sind überhaupt nur möglich, weil alle Daten nach strengen Regeln kodiert sind. Man sollte seine Erwartungshaltung gegenüber Metadaten im Internet daher etwas zurückschrauben, da Kontrolle und Internet bisher zumindest nicht vereinbar sind.

2.2.1 Anglo-American Cataloguing Rules Revision 2

Im anglo-amerikanischen Raum sind die Anglo-American Cataloguing Rules (Gorman and Winkler, 1998) der Metadatenstandard für „Ground floor libraries “ – Bibliotheken, die also tatsächlich Dokumente aufbewahren und verwalten. Wie der Name andeutet, handelt es sich bei den Anglo-American Cataloguing Rules (AACR2R) um einen Regelsatz, der es Bibliothekaren ermöglichen soll, aus einem Dokument die für einen Katalogeintrag benötigten Informationen zu extrahieren. Er beschreibt quasi, welche Informationen für einen Bibliothekar relevant sind und wie diese zu erfassen sind. Es ist daher strenggenommen nicht korrekt, die AACR als Metadatenstandard zu bezeichnen. Vielmehr bilden die AACR die Grundlage für viele der heute benutzten Standards.

Es ist einleuchtend, daß die bloße Erfassung von Daten nur von geringen Nutzen ist, wenn sie nicht auch in standardisierter Form vorgenommen wird. Erst ein solches Vorgehen ermöglicht es späteren Benutzern, die Daten effizient zu nutzen, um zum Beispiel einen Datenbestand zu durchsuchen. Die AACR2R bilden so einen Regelsatz, der auf

Erfahrung aufbaut, um Bibliothekaren die Erstellung von einheitlichen Metadatensätzen zu ermöglichen. Dies bedeutet allerdings nicht, daß zwei Bibliothekare unabhängig voneinander auch identische Informationen über eine Ressource generieren, denn es liegt auf der Hand, daß die Erfassung von Informationen auch mit einen Zweck verbunden ist, der von der erfassenden Institution abhängig ist. Andererseits würden zwei Bibliothekare, die mit den AACR arbeiten, für eine einzelne zu bestimmende Information in der Regel auch zum gleichen Informationsgehalt kommen.

Die Ursprünge der AACR2R gehen auf einen Regelsatz für Bibliothekare zurück, der von der American Library Association im Jahre 1908 erstellt worden ist (American Library Association, 1908). Diese historischen Regeln widmen sich beinahe ausschließlich Material, welches in gedruckter Form zugänglich ist. In den 30er Jahren erwarben Bibliotheken allerdings zunehmend Dokumente und Ressourcen wie Tonträger, die mit den gängigen Katalogisierungsregeln nicht mehr hinreichend zu erfassen waren. Die verbreiteten Standards wurden über die Jahre einige Male erweitert, bis es dann in den 60er Jahren auf Drängen der amerikanischen Library of Congress zur Veröffentlichung der ersten Version der AACR1 kam (American Library Association, 1967). In dieser Version wird erstmals die Wahl der Daten von ihrer Präsentation getrennt. Obwohl die AACR1 erst im Jahre 1967 erschienen, wurde eine Revision bald wieder nötig. Mary Piggott schreibt dazu (Piggot, 1990):

> The technological changes in the presentation of disseminated information and the means of recording and retrieving it that occured between the late 1930s and the late 1960s were so vast, so unprecedented, and the forms they took succeeded each other so rapidly that the code revision committees were unable to legislate for them.

Mit anderen Worten: Bibliotheken wurden nicht nur von einer Dokumentenflut überschüttet, sondern sie mußten auch beständig mit neuen Dokumenttypen (Film, Radio, Schallplatten etc.) umgehen. Des weiteren wurde ihre bisherige Arbeitsweise auch durch das Aufkommen der Computerrevolution in Frage gestellt. Zu dem kam es zu einer beständig größeren Verknüpfung der Sachgebiete untereinander (man denke zum Beispiel an einen Forschungsartikel über „Meeresbiologie zu prähistorischen Zeiten“).

Die Regeln der AACR sind grob in zwei Teile gegliedert. Zum einen befassen sie sich mit der Ermittlung der beschreibenden Daten (engl. *descriptive data*). Unter beschreibenden Daten versteht man den Teil der Daten, der die zu erfassende Arbeit ohne Wertung be-

schreibt, also zum Beispiel die Urheber, die Anzahl der Seiten, die Bindung, die Größe, Anzahl der Abbildungen und sonstige Angaben von beschreibenden Interesse. Eine Erfassung nach rein „deskriptiven“ Kriterien ist allerdings für den Benutzer eines Bibliothekskatalogs nur von sehr geringem, beziehungsweise gar keinem Nutzen. Im zweiten Teil, der die Ansetzungsform (engl. *heading*) beschreibt, geben die AACR Kriterien an, nach denen Zugangspunkte (engl. *points of access*) zu der Ressource definiert werden können. Am üblichsten ist hier die Wahl des Urhebers oder Autors. Ebenso verbreitet ist die Wahl eines Stichwortes wie „Objektorientierte Programmierung“ oder „Geschichte, englische“. Der Sinn der Ansetzungsregeln ist es also, das in der Bibliothek zusammengehörige im Katalog an einer Stelle zusammenzuführen. Für eine Ressource können mehrere Ansetzungsformen generiert werden - in einer zettelbasierten Bibliothek entstehen so mehrere Karteikataloge, um Ressourcen nach unterschiedlichen Ordnungskriterien aufzufinden. Somit wird über die Wahl der Ansetzungsform die Katalogs- und Bibliographiefunktion erfüllt.

Die Regeln für die alphabetische Katalogisierung

Im deutschsprachigen Raum entsprechen die Regeln für die alphabetische Katalogisierung (Deutsches Bibliotheksinstitut, 1993) den anglo-amerikanischen AACR. Die Tendenz geht allerdings immer mehr hin zu einer Harmonisierung der verschiedenen internationalen Standards – insbesondere, um den Austausch von Datenmaterial zu vereinfachen. An einer Übersetzung der AACR ins Deutsche wird zur Zeit gearbeitet. Ein gewichtiger Unterschied zwischen AACR und RAK liegt in der Handhabung von Ressourcen, die sich aus Teilen zusammensetzen, um eine Ganzheit zu bilden. Als Beispiel sei ein Sammelband wissenschaftlicher Artikel genannt. In der Erfassung dieses Sammelbandes betonen die deutschen RAK die Teile, während die amerikanischen AACR die Ganzheit hervorheben. Implizit nimmt ein deutscher Bibliothekar also an, daß Benutzer der Bibliothek eher nach einem Teil suchen, während ein amerikanischer Bibliothekar eher dem Wunsch entspricht, das Ganze zu finden.

Es ist wichtig festzustellen, daß die Ganzheits-/Teilbeziehung nicht unbedingt physikalischer Art sein muß. Während bei einer Festschrift oder einem Sammelband noch unmittelbar einleuchtend ist, daß die Teilartikel auch das Ganze bilden, muß dies zum Beispiel bei einem Sprachkurs mit Tonbandkassette, Vokabelheft und landeskundlichen Begleitvideo nicht mehr der Fall sein. Je nach Auffassung bildet der Sprachkurs ein eigenes Ganzes, oder setzt sich aus mehreren (eigenständigen) Teilen zusammen.

1:1 Regel

Die Art und Weise, wie deutsche Bibliothekare die Ganzheit-/Teilbeziehung handhaben, hat in der Metadatengemeinde einen Namen bekommen: die 1:1 Regel (engl. *1:1 rule*). Sie fordert, daß eine diskrete Ressource (also eine „zerlegbare“ Ressource) in ihren Teilen katalogisiert wird. Dabei wird jeder Teil einzeln erfaßt und katalogisiert (daher der Begriff 1:1). Die 1:1 Regel wird zur Zeit insbesondere in der Internetgemeinde intensiv diskutiert. Der Grund liegt in der gängigen Struktur von Dokumenten im Internet. Durch die Möglichkeit, multimediale Ressourcen zu erzeugen, die oftmals noch nicht einmal physikalisch am gleichen Ort (Server) aufbewahrt werden, ist das Konzept von Dokumenten als Ganzem fragwürdig geworden – sie sind von Natur aus in Teilen vorhanden. Damit einher geht auch die geringere Bedeutung der Urheberschaft eines Dokuments. Man spricht mittlerweile zunehmend vom „geistig oder künstlerisch Verantwortlichen“ anstelle des Autors. Andererseits sind Internetressourcen nicht immer diskret. Es ist oftmals schwierig, zu bestimmen, welcher Teil eines Dokuments eine eigenständige Ressource darstellt.

Diese neue Sichtweise hat einen Grund, der am besten an einem Beispiel veranschaulicht wird. Internetseiten sind oft aus vielen verschiedenen Bestandteilen zusammengesetzt. Bilder, Texte und Audiodateien werden zu einem multimedialen Ganzen zusammengefügt. Häufig wäre es durchaus legitim, den Urheber der Internetseite als Urheber der Ressource zu betrachten, weil der Internetseite durchaus ein eigener künstlerischer oder geistiger Inhalt zugesprochen werden kann. Aus Benutzersicht, also für jemanden, der nach Informationen im Internet sucht, ist diese Sichtweise allerdings ein Irrweg. Er sucht ja nach Informationen über zum Beispiel „Donald E. Knuth“ und nicht über den Gestalter der „Donald E. Knuth – Homepage“. Auf der „Donald E. Knuth – Homepage“ könnten wiederum Bilder abrufbar sein, die von einem bekannten Photographen gemacht worden sind. Wichtig für den Endbenutzer sind der Photograph und der Inhalt der Photographie, nicht derjenige, der die Scans gemacht hat. In der Internetgemeinde setzt sich die Sichtweise durch, den Begriff der Urheberschaft aufzuweichen – in Richtung des „geistig oder künstlerisch“ Verantwortlichen – und die Teile eines Dokuments einzeln zu beschreiben, also das Bild von „Donald E. Knuth“ getrennt vom Einführungstext „Donald E. Knuth – Algorithmen“ , um die Teile dann mittels Relationen zusammenzuführen.

Anhand dieses Beispiel sei nochmals auf die Eigenschaft externer Metadaten hingewiesen, daß sie zielgerichtet erstellt werden können und müssen. Zum einen können über ein und dieselbe Ressource eine Vielzahl von Datensätzen erstellt werden, zum anderen können

sie auch andere Schwerpunkte setzen (zum Beispiel könnte man so den Designer der Internetseite betonen, den ein anderer vernachlässigt).

Die 1:1 Regel macht im übrigen keine Aussage darüber, welche Elemente einer Ressource ein Ganzes bilden, beziehungsweise ob es einzeln zu erfassende Teile sind. Dies liegt im Ermessen und in der Intention des Metadatenerzeugers. Die 1:1 Regel macht allerdings eine Aussage darüber, daß Teile einer Ressource (, wenn sie als solche erkannt werden), auch einzeln erfaßt werden.

Zusammenfassung

Zusammenfassend sei angemerkt, daß die AACR und ähnliche Regelwerke für Bibliotheken einen großen Erfahrungsschatz bereitstellen, der auch für die Ausarbeitung eines entsprechenden Standards für das Internet zu Rate gezogen werden sollte. Viele der Probleme der Katalogisierung im Internet und in traditionellen Bibliotheken unterscheiden sich nicht grundlegend, wobei naturgemäß beide bestimmte Eigenarten haben, die es zu berücksichtigen gilt.

Daten sammeln macht überhaupt nur Sinn, wenn sie auch ausgewertet werden können. Das traditionelle Bibliothekswesen bemüht sich dem Rechnung zu tragen, indem es wo immer möglich, die Verwendung von kontrolliertem Vokabular verlangt (zum Beispiel zur Beschreibung der Ansatzpunkte, der Sortierung und Erfassung von Autorennamen, der Indizierung der Titel usw.). Wo möglich, sollten also Daten in standardisierter Form erfaßt werden, wie dies mit Hilfe der AACR geschieht.

2.2.2 Cataloging in Publication (CIP)

Im Kapitel 2.2.1 wurde bereits erwähnt, daß die AACR ein Regelwerk zur Erfassung von bibliographischen Daten bilden, ohne zu berücksichtigen, in welcher Form diese präsentiert werden sollen. Eine typische Anwendung der AACR zur Präsentation bilden die Cataloging in Publication Regeln. Die CIP-Einträge bilden ein Beispiel interner Metadaten, da sie gewöhnlich zusammen mit der Publikation veröffentlicht werden. Folgendes Zitat gibt einen Überblick über die Motivation der Erstellung von CIP-Einträgen (Library of Congress, 1993):

> The purpose of the Cataloging in Publication (CIP) program is to prepare prepublication cataloging records for those books most likely to be widely

Library in Congress Cataloging in Publication Data

Stearns, Marshall Winslow.
Jazz dance: the story of American vernacular dance / Marshall
and Jean Stearns.
p. cm.
Reprint. Originally published: New York: Macmillan, 1968.
Includes bibliographical references and index.
Includes filmography.
ISBN 0-306-80553-7
1. Afro-Americans-Dancing-History. 2. Dancing-United States-History.
3. Jazz dance-History. I. Title

Abbildung 2.1: Der CIP–Eintrag für „Jazz Dance: the Story of American Vernacular Dance“

> acquired by the nation's libraries. These records (CIP data) are printed in the book and greatly facilitate cataloging activities for libraries. They are also distributed prior to the book's publication in machine readable form via the MARC (MAchine Readable Cataloging) tapes, alerting libraries and other bibliographic services around the world to forthcoming titles.

Damit ist auch schon ein wichtiger Aspekt interner Metadaten angesprochen worden. Interne Metadaten vereinfachen die weitere Erfassung der Publikation für Dritte, da mit ihnen schon ein Rahmenverständnis der Publikation mitgeliefert wird. In der Regel wird der CIP-Eintrag vom Verlag oder Verfasser erstellt, und von diesem Personenkreis kann erwartet werden, daß sie ihr Werk auch beschreiben können. Bibliotheken weichen von der CIP-Beschreibung bei der Erstellung ihrer Kataloge in der Regel gar nicht, oder nur geringfügig ab. Dadurch wird es Benutzern der Kataloge ermöglicht, das gleiche Werk auch in verschiedenen Bibliotheken schnell zu finden. Anhand der Abbildung 2.1 soll die Arbeit mit einen CIP-Eintrag kurz erläutert werden.

Die erste Zeile informiert den Leser darüber, daß es sich hier um einen CIP-Eintrag gemäß der Konventionen der Library of Congress handelt. Danach folgt die Ansetzungsform unter dem Namen des Hauptautors (*Stearns, Marshall Winslow.*). Ein Bibliothekar würde das Buch also unter dem Namen Stearns einsortieren. Zur Erinnerung: die Ansetzungs-

form bildet einen Zugriffspunkt für die Ressource, ist also eine nicht-deskriptive Angabe. Es folgen Titel und Untertitel, gefolgt von den Namen der beiden Autoren (für Jean Stearns wurde im übrigen keine Ansetzungsform generiert). Die Informationen über die Bindung (*p. cm.*) wurden nicht angegeben. Danach kommen Angaben über relevante Teile der Publikation (*Includes...*). Nach der ISBN-Nummer folgen Angaben über die Nebeneinträge (engl. *tracings*). Zuerst mit arabischen Nummern die Sachansetzungsformen (engl. *subject headings*) und dann arabisch numeriert der eigentliche Nebeneintragungsvermerk (*I. Title.*). Die Publikation soll also neben der Einordnung unter dem Autornamen auch unter den jeweiligen Sachansetzungsformen aufgeführt werden, dort aber unter dem Kriterium des Titels. Aufgrund dieser beiden Informationen (Hauptansetzungsform und Sachansetzungsform) können also die bibliographischen Werkzeuge Katalog und Bibliographie erstellt werden.

2.2.3 Machine Readable Cataloging (MARC)

Der MARC Standard lehnt sich ebenso wie die Cataloging in Publication-Einträge an die AACR an. MARC ist ein Standard, der den Austausch von bibliographischen Daten auf elektronischem Wege ermöglicht. Entwickelt wurde er Ende der 60er Jahre – also zeitgleich mit dem Aufkommen preisgünstiger Rechenanlagen. Die Standardisierungsbemühungen wurden wiederum von der amerikanischen Library of Congress vorangetrieben, um den Austausch von Daten zwischen Bibliotheken zu vereinfachen und insbesondere, um innerhalb von Bibliotheken das Katalogisierungswesen rechnergestützt zu automatisieren. MARC gibt es in vielen verschiedenen Versionen, die meist eine nationale Prägung haben (US-MARC, UK-MARC, UNIMARC). Die Unterschiede sind allerdings geringfügig und für die weitere Betrachtung unerheblich. Es sei aber angemerkt, daß diese nationalen Eigenstandards dem ursprünglichen Ziel von MARC, austauschbare Information zur Verfügung zu stellen, entgegenlaufen. Gegenwärtige Harmonisierungsbemühungen laufen darauf hinaus, nationale Gepflogenheiten auszumerzen, um letztendlich US-MARC als Standard zu benutzen.

Ein MARC-Datensatz besteht aus mehreren Feldern, die alle eine variable Länge haben können und durch Feldseperatoren abgegrenzt werden. Jedes Feld wird dabei wiederum von einem Feldbezeichner eingeleitet, der Auskunft über den Inhalt des nächsten Feldes gibt. Unglücklicherweise wurde der MARC-Standard technologischen Neuentwicklungen nicht angepaßt, so daß alle Feldbezeichner numerisch sind. Diese Eigenart drückt noch den Wunsch aus, Speicherplatz zu sparen und die Verarbeitungszeit mit (damaligen) Band-

laufwerken zu senken. MARC Datensätze sind daher schwer lesbar und nur mit Übung zu erstellen. Letzteres Problem wird heute durch rechnergestützte Applikationen gemildert.

MARC-Felder

Heutzutage definiert der MARC-Standard circa 300 verschiedene Felder, mit einer Vielzahl von Unterfeldern, die einem Bibliothekar zur Verfügung stehen, um Dokumente zu erfassen. Eine Einführung in die gebräuchlichsten Felder zur Beschreibung von Computerressourcen findet sich bei Nancy Olson (Olson, 1997).

Eine Untersuchung der Library of Congress anhand von über 4 Millionen MARC-Datensätzen hat allerdings gezeigt, daß von den über 300 Feldern nur 33 Felder in mindestens 1% der Datensätze vorkommen. Tabelle 2.1 zeigt eine Übersicht der Felder zusammen mit „äquivalenten“ Dublin Core Feldern (siehe dazu die Diskussion zu Dublin Core in Kapitel 2.3.1).

%	Z	Tag	Element	Dublin Core
100%	Z	245	Title	TITLE
	Z	260	Imprint (Place, Publisher, Date)	PUBLISHER
	Z	300	Physical description (nr of pages etc.)	?
		050	LoC Classification	SUBJECT
			(Library of Congress specific)	
	Z	008	Codes,	LANGUAGE
			among others, language of text, country,	TYPE
			timespan of publ., document type	
95%		650	Topical subject heading	SUBJECT
			(difficult to count since repeatable!)	
72%	Z	100	First author (primary responsibility)	CREATOR
67%	Z	020	ISBN	IDENTIFIER
		500	General Note	DESCRIPTION?
63%		082	Dewey Decimal Classification	SUBJECT
50%		043	Geographic Area Code	COVERAGE
49%		504	Bibliography note	DESCRIPTION
43%	Z	700	Added entry personal names	CONTRIBUTOR
25%		651	Geographical Name Subject Heading	COVERAGE?

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite				
%	**Z**	**Tag**	**Element**	**Dublin Core**
18%	Z	250	Edition, Version	?
		710	Added entry corporate name	CONTRIBUTOR
		490	Series (uncontrolled)	RELATION?
17%		440	Series (under authority control)	RELATION
14%		600	Personal name subject heading	SUBJECT
9.4%		740	Other titles	TITLE
8.2%		830	Title of subordinate work (auth. controlled)	RELATION
7.3%		110	Corporate name, primary responsibility	CREATOR
7%		041	Languages of text -¿ 008'35-37	LANGUAGE
6%		610	Corporate name subject heading	SUBJECT
4%		520	brief abstract	DESCRIPTION
3.8%		130	Uniform title main entry heading	TITLE
3.3%		505	Contents note (multivol works)	DESCRIPTION
2%	Z	111	Event main entry (+ Tag 711 : 2.5%)	?
1%		653	Free subject term (no auth. control)	SUBJECT
		655	Genre as Subject	SUBJECT
		630	Title of a work as subject	SUBJECT
		060	National Library of Medicine classification	SUBJECT
		810	Series under corporate name	RELATION
		730	Uniform title added entry	TITLE
		533	Reprint/Reproduction note	RELATION?

Tabelle 2.1: Die 33 meistgebrauchten MARC-Tags

Offensichtlich gehören die Titelangaben, die Angaben zum Verlag und der Ort der Aufbewahrung in der Bibliothek (LoC number) zu den meist gebrauchten Feldern, da sie auch für den Benutzer von größter Wichtigkeit sind. Felder, die in der Tabelle mit einem „Z“ gekennzeichnet sind, stellen Felder dar, die immer vorhanden sind, wenn die zu beschreibende Ressource diese Information bereitstellt. Ein Bibliothekar kann sich also auf das Vorhandensein von mit „Z“ markierten Feldern bei bestimmten Ressourcen verlassen – sie sind „zuverlässig“ (zum Beispiel kommt die Autorangabe nicht mehr in allen Ressourcen vor, wenn es aber ein Buch ist, dann fehlt auch die Autorangabe nicht).

100	1#	$a	Stearns, Marhall Winslow.
245	10	$a	Jazz dance: the story of American vernacular dance/
		$c	Marshall and Jean Stearns
250	##	$a	1st ed.
260	##	$a	New York:
		$b	Da Capo Press, Inc.
		$c	c1994

Tabelle 2.2: Beispiel eines MARC-Datensatzes

MARC-Formate

In Tabelle 2.1 findet man die Angaben zu den Tag-Zahlen, die im MARC-Standard definiert sind. Dies sind die Angaben zu den Haupttags, die sich wiederum in zahlreiche Unterfelder aufspalten.

Anfangs wurde bereits erwähnt, daß das Speicherformat von MARC weitestgehend von der Tatsache geprägt worden ist, daß MARC auf Magnetbändern gespeichert worden ist. Geübte Bibliothekare können diese Rohdaten durchaus verwerten und lesen. Dies unter anderen auch deshalb, weil sich der Aufbau der Daten stark an den Aufbau von Bibliothekskarteikarten anlehnt. Betrachtet man zum Beispiel Tabelle 2.1, so stellt man fest, daß der Titel die Nummer 245 erhält, während „Other Titles“ die Nummer 740 und der „Uniform Title“ die Nummer 130 trägt. Das liegt am Aufbau einer Bibliothekskarte mit ihren verschiedenen Nebeneinträgen und Ansetzungsformen (siehe Kapitel 2.2.2) – für den Laien zusammengehörige Informationen können in MARC also in ganz andere logische Gruppen eingeteilt sein.

Im folgenden wird kurz ein stark gekürzter MARC Datensatz gezeigt und erklärt, wie er für die Publikation in Abbildung 2.1 generiert wird.

Jeder MARC-Datensatz ist logisch in Felder eingeteilt, die durch Nummern gekennzeichnet sind. Sie befinden sich in der ersten Spalte des Datensatzes. Zum Beispiel kennzeichnet die „245“ eine Titelangabe mit „Statement of Responsibility“ (d.h. eine Titelangabe mit einer Angabe der Urheberschaft), wie man aus der Tabelle 2.1 entnehmen kann. Jedes Feld kann wiederum um die Angabe von Indikatoren ergänzt werden. Im Falle der Titelangabe entsprechen die Indikatoren dem Feld „10“. Indikatoren können die Werte $0-9$ annehmen, bei der Angabe „10“ handelt es sich also um zwei Indikatoren mit dem Wert „1“ , beziehungsweise „0“. Nicht benutzte Indikatoren werden durch ein # gekenn-

zeichnet. Die „1“ bedeutet, daß für den Titel ein Nebeneintrag (*tracing*) generiert werden soll. Die „0“ bezeichnet, daß keine Buchstaben beim Sortieren des Titels ignoriert werden sollen. Mittels des zweiten Indikators können also führende Artikel übersprungen werden. Zum Beispiel:
`100 14 $a The C++ Programming Language $c Bjarne Stroustrup` (zweiter Indikator: die `4` in `14`). Die `14` gibt hier also an, daß ein Nebeneintrag generiert wird, der Titel aber unter `C++` und nicht unter `The` einsortiert werden soll.

Felder können wiederum in Unterfelder eingeteilt werden. Unterfelder werden in der textuellen MARC-Darstellung in der Regel durch „$Buchstabe“-Kombinationen markiert und trennen die für ein Feld benötigten Informationen (siehe als Beispiel Feld 245 mit der Titel- und „statement of resposibility“-Angabe). Die restlichen Felder und ihre Bedeutung können Tabelle 2.1 entnommen werden.

2.3 Standards zur Erfassung von Metadaten in digitalen Bibliotheken

In diesen Kapitel werden aktuelle Standards zur Beschreibung von Metadaten und ihre Strukturen erläutert und vorgestellt. Es sei vorangestellt, daß bei der Gliederung der kommenden Kapitel ein „bottom-up“ – Ansatz gewählt wurde. Daher wird zuerst der sich gerade entwickelnde „Dublin Core“-Standard vorgestellt, der wiederum in das „Resource Description Framework“ (RDF) eingebunden werden kann. Das RDF wiederum baut auf der „eXtensible Markup Language“ (XML). XML, welches von vielen Autoren als Nachfolger von HTML angesehen wird, stellt alle Mechanismen zur Verfügung, welche zur Strukturierung eines Dokumentes benötigt werden. Das RDF benutzt XML, um die Daten, also die Metadaten, des Dokuments zu gliedern. RDF selbst stellt zur Beschreibung der Daten nur ein Grundgerüst zur Verfügung, welches Standards wie Dublin Core dann benutzen können. Dieser Zusammenhang ist in Abbildung 2.2 dargestellt.

Dublin Core kann als ein schon relativ spezifischer Standard zur Beschreibung von Metadaten angesehen werden, der wohl vor allem im Bereich der Katalogisierung zum Einsatz kommen wird. Denkbar sind aber auch andere Standards (zum Beispiel zur Beschreibung chemischer Moleküle), deren Daten mit den Variablen von Dublin Core nur ungenügend beschrieben werden können. Das Ziel des RDF besteht nun darin, solche verschiedenen Standards möglichst sinnvoll zusammenzufassen, in der Art, daß Anfragen der Form „Wel-

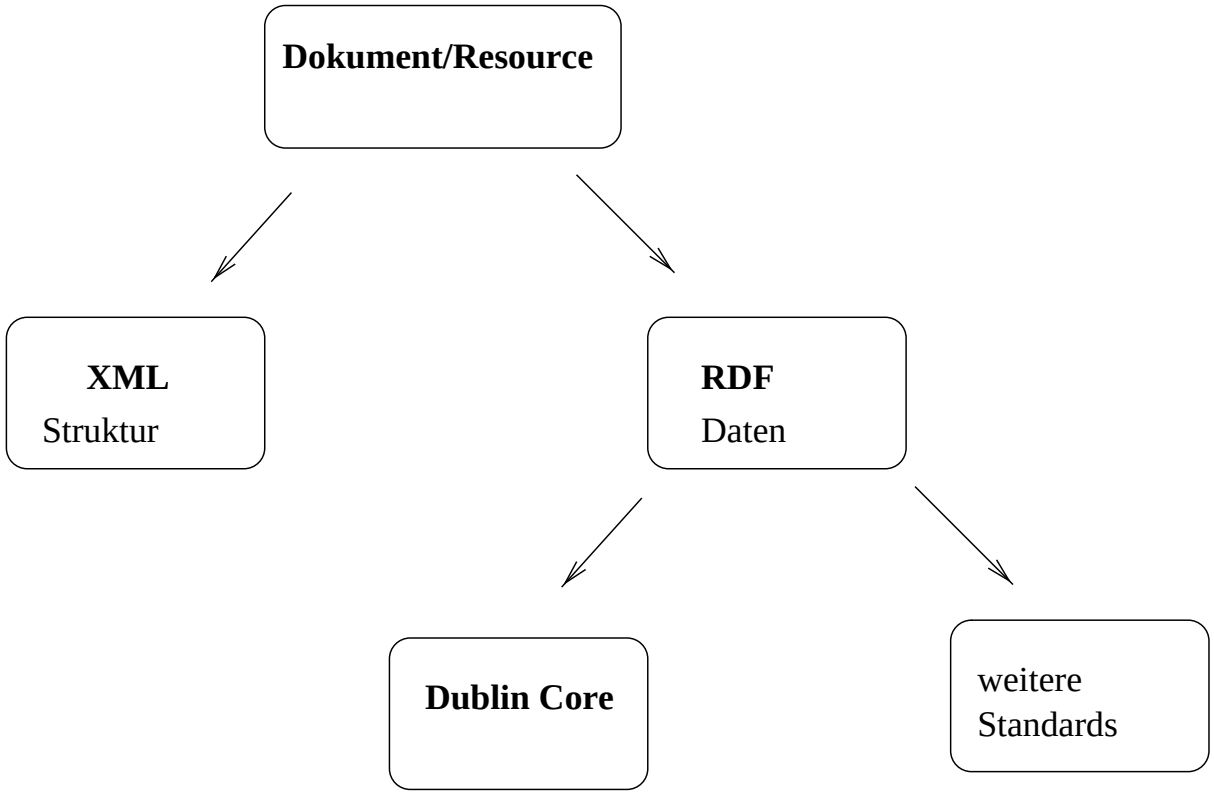


Abbildung 2.2: Der Zusammenhang zwischen XML, RDF und Dublin Core

cher Autor hat etwas über Polymere einer bestimmten Kategorie verfaßt?“ beantwortet werden können.

Neben Dublin Core, XML und RDF werden in diesem Kapitel weiterhin die IAFA-Templates angesprochen. Die IAFA-Templates bilden die konzeptuelle Grundlage der in Kapitel 3 vorgestellten ReDIF-Templates.

2.3.1 Dublin Core

Dublin Core ist eine gemeinsame Anstrengung von Organisationen und Unternehmen, sich auf einen Kernsatz von Metadatenfeldern zu einigen, um Internetressourcen zu beschreiben. Ursprünglich rein auf elektronische Ressourcen beschränkt, zog die Arbeit von Dublin Core allerdings auch schnell das Interesse von Museen, Bibliotheken, Behörden (government agencies) und kommerziellen Organisationen auf sich. Die Arbeit wird allerdings nach wie vor von Bibliotheken dominiert, wie sich auch aus der Liste der Sponsoren von Dublin Core ersehen läßt (Dublin Core, 1999a).

Bisher ist die Entwicklung von Dublin Core noch nicht abgeschlossen, daher liegen dieser Beschreibung nur Working Drafts zugrunde, die noch abschließend zertifiziert werden müssen. Details können sich also jederzeit ändern.

Ziele von Dublin Core

Die Dublin Core Initiative verfolgt mit den Bereitstellen der Dublin Core Kernelemente die 4 folgenden, grundlegenden Ziele, welche im Dublin Core Userguide aufgeführt sind (Dublin Core, 1998c):

Einfache Erzeugung und Wartung: Ein ungelernter Benutzer soll mit geringem Aufwand in der Lage sein, kostengünstig Ressourcen mittels Dublin Core zu beschreiben. Daraus folgt insbesondere der Anspruch, daß die Dublin Core-Elemente auch „human-readable“ sind. Suchmaschinen oder andere Dienste, die diese Metadaten auslesen, sollen trotzdem gehaltvolle Informationen bekommen.

Einheitliche und verständliche Semantik: Über die verschiedenen Anwendungsgebiete von Dublin Core hinweg, sollen die benutzten Elemente eine einheitliche und verständliche Semantik besitzen. Für Dublin Core heißt dies, daß eine zu tiefe Spezialisierung der Metaelemente vermieden werden soll, um zu gewährleisten, daß alle Basiselemente von allen Benutzern von Dublin Core gleichermaßen benutzt werden können (siehe z.B. die spätere Diskussion des „Creator“-Elements).

Internationalisierung: Bisher sind verschiedene Sprachversionen von Dublin Core entwickelt worden, um insbesondere die einfache Erzeugung und Wartung von Metadatensätzen zu erleichtern. Damit soll auch die „einfache Erzeugung und Wartung“ unterstützt werden. Um die Internationalisierung von Dublin Core zu vereinfachen, wird von Dublin Core die Verwendung des Resource Description Framework vorgeschlagen.

Erweiterbarkeit: Da sich Dublin Core selbst nur als Grundgerüst zum Beschreiben von Ressourcen versteht, besteht die Notwendigkeit, die vorhandenen Elemente um neue zu erweitern, beziehungsweise deren Semantik zu verfeinern. Dies kann zum einen wiederum über das Resource Description Framework geschehen, zum anderen auch über Verfeinerungen schon bestehender Elemente (Subelemente) oder unter Zuhilfenahme semantischer Qualifizierer.

Aus Kapitel 2.2 sollte schon klar geworden sein, daß die Erfassung von Ressourcen zum Zwecke der Katalogisierung keineswegs trivial ist. Die Regelwerke, die der Erstellung von

Bibliothekskatalogen zugrunde liegen, sind extrem komplex. Erstaunlicherweise macht man sich in der Dublin Core Gemeinde erst jetzt Gedanken darüber, welche Regelwerke Dublin Core zugrundeliegen sollen. Erstaunlich vor allem deshalb, weil insbesondere Bibliothekare an der Entwicklung von Dublin Core beteiligt sind. Nun sind gängige bibliographische Regeln aber extrem komplex, daher ist es zum Beispiel fragwürdig, ob ein „ungelernter“ Benutzer problemlos Ressourcen beschreiben kann, wenn er gezwungen wird, diese Regeln zu befolgen. Wer will im Internet außerdem die Einhaltung dieser Regeln kontrollieren? Bleibt es aber bei bloßen Richtlinien zur Erstellung von Dublin Core-Datensätzen, so ist zu fragen, inwiefern diese bei der Suche nach Dokumenten im Internet behilflich sind. Auch die Austauschbarkeit der Daten wäre stark behindert, wenn Dublin Core ein bloßes syntaktisches Gerüst ohne dahinterliegende Semantik wäre. Denn dies ist gerade der Grund für den Erfolg von Standards wie MARC, da ihnen die AACR zugrundeliegen.

Ein weiterer starker Kritikpunkt an Dublin Core besteht in der Tatsache, daß ein klare Mission fehlt. Was genau will man mit Dublin Core erreichen und wie? Dies ist eine Frage, die in der Dublin Core Gemeinde noch nicht zufriedenstellend diskutiert worden ist. So will die eine Gruppe letztendlich Suchmaschinen verbessern, die andere ihren Bibliothekskatalog ins Internet stellen, wieder andere scheinen ein „universales Metadatenformat“ im Auge zu haben. Benutzer sollten ihre Erwartungen an Dublin Core also noch etwas dämpfen, bevor in diesen Fragen mehr Klarheit gewonnen worden ist. Andererseits wird von kaum einer Seite die Nützlichkeit eines Standards wie Dublin Core bestritten, wobei es allerdings unterschiedliche Erwartungshaltungen erschweren, den Begriff der „Nützlichkeit“ (Wofür nützlich?) zu definieren.

Die Basiselemente von Dublin Core

Dublin Core besteht in der aktuellen Version aus 15 Basiselementen (zur Erinnerung: in MARC über 300 Felder), die teilweise noch durch Subelemente erweitert werden können. Der User Guide Working Draft (Dublin Core, 1998c) schlägt vor, diese Basiselemente in drei Kategorien aufzuteilen. Tabelle 2.3 gibt diese Elemente in der im Userguide vorgeschlagenen Kategorisierung wieder.

Die Gliederung richtet sich nach Inhalt, Urheberschaft und Ausprägung. Die wichtigsten Elemente werden hier kurz erklärt und, wenn nötig, diskutiert. Eine ausführlichere Erklärung findet sich im Userguide (Dublin Core, 1998c).

Content	**Intellectual Property**	**Instantiation**
Coverage	Contributor	Date
Description	Creator	Format
Type	Publisher	Identifier
Relation	Rights	Language
Source		
Subject		
Title		

Tabelle 2.3: Die 15 Basiselemente von Dublin Core

Creator: Der Dublin Core Standard hat sich bewußt nur auf einen „Creator“ geeinigt. Ein „Creator“ ist demnach diejenige Person oder Organisation, die vorrangig verantwortlich für den geistigen Inhalt („intellectual content“) der Ressource ist. Der „Creator“ ist absichtlich vage gehalten, da er neben Autoren auch Komponisten, Maler, Photographen beschreiben soll. Strittig ist auch die fehlende Unterscheidung zwischen Organisationen und Personen. Es sei angemerkt, daß der Creator mittels Qualifizierern näher bestimmt werden kann (siehe dazu „Die Qualifizierer von Dublin Core“ , Seite 22). Weitere Verwirrung stiftet das „Contributor“-Feld, da man sich hier nicht genau darauf einigen kann, wer als „Contributor“ einer Ressource zählt. Es gab Anregungen – die allerdings wieder verworfen worden sind – alle Namen in einem Feld zu nennen, da sie so in der Regel auch von Suchmaschinen indiziert werden (das heißt in einem Index). Der Vorschlag ist allerdings zu Recht verworfen worden, da sich der Dublin Core-Standard nicht an der Arbeitsweise von Datenbanken orientieren soll.

Andere Bestrebungen liefen darauf hinaus, den Creator, Publisher und Contributor im Creator zusammenzufassen und ein neues Feld `Role` zu schaffen. Namen wären also stets in einem Paar angegeben worden: Zunächst das Creator-Feld, gefolgt von einen Role-Feld, welches Angaben über die Rollenbeziehung des Creators (Author, Contributor, Composer etc.) enthalten hätte. Der Vorschlag wurde aufgrund der Inkompatibilität zu bereits gängiger Implementierungspraxis abgelehnt.

Title: Dieses Element beschreibt den Titel der Ressource, so wie er vom „Creator“ vergeben worden ist. Bisher gibt es noch keine Diskussionen darüber, wie dieser Titel angegeben werden soll, was einerseits erstaunlich ist, denn in der Bibliothekswelt kennt man eine Vielzahl von verschiedenen Titeln. Andererseits ist diese Diskussion, aufgrund der besseren Möglichkeiten mittels EDV nach Titeln zu suchen, auch nicht so wichtig.

Subject: Im Subject-Element sollen Schlüsselwörter vergeben werden, die die Ressource möglichst genau beschreiben. Das Dublin Core-Komittee befürwortet die Verwendung von kontrolliertem Vokabular oder von Schemata. Solch ein Schema könnte zum Beispiel die „Journal of Econometric Literature"-Klassifizierung sein (JEL-Code), oder die im Bibliothekswesen gebräuchliche „Dewey Decimal Classification". Problematisch ist hier die fehlende Definition von "kontrolliert". Wer oder was kontrolliert das Vokabular? Ein Benutzer, der die Metadaten anlegt und zweckspezifisch Dokumente klassifiziert? Oder müssen Spezifikationen nach einem internationalen Standard erstellt werden? Letztere Möglichkeit würde die Austauschbarkeit der Daten vereinfachen, während die erste Lösung dem Ziel der einfachen Erzeugung von Metadaten näherstünde.

Description: Dieses Feld dient zur textuellen Beschreibung einer Ressource, vergleichbar der Zusammenfassung (engl. *abstract*) einer wissenschaftlichen Arbeit.

Date: Das Datumsfeld soll sich syntaktisch an die Konvention der ISO8601 (International Organization for Standardization, 1988) anlehnen. Die ISO8601 ist allerdings relativ komplex, so daß eine Teilmenge für Dublin Core (WTN8601) entwickelt wird. Datumsfelder werden demnach nach der Konvention YYYY-MM-DD belegt. Der 13. November 1972 wäre demnach als 1972-11-13 codiert.

Type: Seit dem 12. März 1999 existiert eine Liste der möglichen Dokumenttypen. Eine Ressource kann nun u.a. den Typ collection, dataset, event, image, interactive resource, physical object, service, software, sound oder text besitzen. Die Definition der einzelnen Typen findet sich in der „List of Resource Types" (Dublin Core, 1998a).

Relation: Mittels einer Relation können Bezüge zwischen verschiedenen Metadatensätzen, beziehungsweise den beschriebenen Ressourcen hergestellt werden. Die Art und Weise, wie dies geschehen soll, ist noch nicht geklärt. Schaut man sich die Liste der möglichen Relationsbezeichner in Tabelle 2.4 an, so sieht man, daß eine eindeutige Zuordnung nicht direkt erkennbar ist. Ist zum Beispiel „Windows 95" eine Version von „Windows 3.1", oder ist „Windows 3.1" die Basis von „Windows 95"? Oder beides?

Die Qualifizierer von Dublin Core

Eine weitere Unterteilung dieser Basiselemente erfolgt anhand zusätzlicher Qualifizierer (engl. *qualifier*), die die Bedeutung der Basiselemente näher spezifizieren. Drei Qualifizierer wurden bisher von Dublin Core anerkannt (Dublin Core, 1998b). Zur Zeit werden diese Qualifizierer allerdings noch stark diskutiert. In der obigen Zusammenfassung wur-

de bereits erwähnt, daß einige der Felder kontrolliertes Vokabular enthalten sollen. Die Qualifizierer sollen diesen Mechanismus ermöglichen. Dublin Core mit Qualifizierern ist allerdings noch im Diskussionstadium, so daß hier nur ganz kurz darauf eingegangen wird.

Subelement: Ein Subelement ermöglicht die Verfeinerung eines Elementinhalts. Zum Beispiel kann so die „Creator"-Information zwischen Personennamen und Namen von Organisationen unterscheiden. Beide können geistige Urheber der Ressource sein, aber die feinere Kennzeichnung ermöglicht eine bessere Verwertung der Metadaten.

Scheme: Schemata ermöglichen es, zu kennzeichnen, daß dieses Attribut sich nach einem bestimmten Schema verhält. Dies ermöglicht zum Beispiel die Angabe eines JEL-Schemas für die Klassifizierung wirtschaftswissenschaftlicher Texte.

Lang: Der LANG-Qualifizierer gestattet es Angaben über die verwendete Sprache des Attributs zu machen.

Element	**Subelement**	**Sub-Subelement**
Title	Alternative Main	
Creator, Publisher, Contributor	PersonalName, CorporateName	Address
Date	Created, Accepted, Acquired, DataGathered	
Relation	IsPartOf / HasPart, IsVersionOf / HasVersion, IsFormatOf / HasFormat, References / IsReferencedBy, IsBasedOn / IsBasisFor, Requires / IsRequiredBy	
Coverage	PeriodName, PlaceName, x, y, z, t, Polygon, Line, 3d	

Tabelle 2.4: Die definierten Subelemente von Dublin Core

Innerhalb der Dublin Core Gemeinde stoßen diese Qualifizierer zum Teil noch auf großen Widerstand. Insbesondere da es Bestrebungen gibt, über ein Subelemente des Creator-Tags auch Angaben über email-Adressen zu ermöglichen. Dies wird als Verstoß gegen die 1:1-Regel betrachtet. Genaugenommen werden auf diese Weise in einem Datensatz Daten über zwei verschiedene Ressourcen beschrieben: der Dokumentressource und der

Creatorressource. Die 1:1 Regel fordert aber einen Datensatz pro Ressource. Besser wäre es daher einen Autor getrennt zu beschreiben und Verbindungen zwischen den Ressourcen über das Relation-Tag herzustellen.

Syntax

Dublin Core soll in einer Vielzahl von Internetstandards darstellbar sein. Zur Zeit arbeiten Arbeitsgruppen an einer Syntax zur Darstellung von DC in HTML und zur Darstellung von DC in RDF/XML. Die Syntax für HTML wird hier näher dargestellt.

In HTML existiert bereits ein Mechanismus, um Dokumente mit Metadaten zu bereichern. Das hierfür verwendete Tag ist `META`.

Prinzipiell ist jedes der 15 Basiselemente optional und läßt sich beliebig oft wiederholen. Diese generell wünschenswerte Vereinfachung kann für Probleme sorgen, weil sie großen Spielraum für Interpretationen läßt. In Abbildung 2.3 ist beispielhaft ein DC-Datensatz dargestellt. Prinzipiell ist die Syntax für Dublin Core also trivial. Problematisch ist allerdings die Anwendung von DC, ohne daß der Standard bisher verabschiedet worden ist. So finden sich zum Beispiel im `Subject`-Feld folgende Angaben:“ `Dublin Core; DC; generator; editor; Warwick Framework; ...`“. Der menschliche Leser wird dies zu Recht als eine durch Semikolons getrennte Aufzählung von Stichwörtern erkennen. Im Standard ist dies aber noch in keiner Weise explizit festgelegt, vielmehr wird solch eine Vorgehensweise empfohlen, da sie bereits allgemein üblicher Implementierungspraxis entspricht. Theoretisch verlangt solch eine Vorgehensweise das Hinzuziehen eines Qualifizierers (Schemas). Die Angabe „dieses Subject-Feld“ ist unter Verwendung des Schemas „semikolon-getrennte Liste“ erstellt, fehlt aber. Im bisher einzigen offiziellen Dokument zur Implementierung von DC in HTML heißt es dazu (Kunze, 1999):

> „Note that the qualifier syntax and label suffixes (which follow an element name and a period) used in examples in this document merely reflect current trends in the HTML encoding of qualifiers. Use of this syntax and these suffixes is neither a standard nor a recommendation.“

Weiterhin ist noch ungeklärt inwiefern solche Aufzählungen semantische Bedeutung haben. Ist das oben aufgeführte Beispiel gleichbedeutend mit einer Wiederholung des Tags wie im folgenden Beispiel?

```
<LINK rel     = "schema.DC"
      href    = "http://purl.org/DC/elements/1.0/">
<META NAME="DC.Title"
      CONTENT="DC-dot">
<META NAME="DC.Creator"
      CONTENT="Andy Powell">
<META NAME="DC.Subject"
      CONTENT="Dublin Core; DC; generator; editor;
               Warwick Framework; SOIF; TEI; USMARC; XML;
               GILS; ROADS; RDF">
<META NAME="DC.Description"
      CONTENT="A CGI based Dublin Core metadata
               generator">
<META NAME="DC.Type"
      CONTENT="Text">
<META NAME="DC.Format"
      CONTENT="text/html">
<META NAME="DC.Identifier"
      CONTENT="http://www.ukoln.ac.uk/metadata/dcdot/">
<META NAME="DC.Publisher"
      CONTENT="UKOLN, University of Bath">
<META NAME="DC.Rights"
      CONTENT="http://www.ukoln.ac.uk/metadata/dcdot/COPYING">
```

Abbildung 2.3: Dublin Core Beispiel in HTML

```
<META NAME="DC.Subject"
      CONTENT="Dublin Core"
<META NAME="DC.Subject"
      CONTENT="DC"
<META NAME="DC.Subject"
      CONTENT="generator"
...
```

Die Frage mag bei Keywordbeschreibungen noch unberechtigt sein, aber wie sieht es im Falle von Autoren aus? Sind durch Semikolon getrennte Autoren gleichgewichtet und

ist eine einzelne Aufzählung im Sinne einer Rangwertung zu verstehen? Wie werden überhaupt Autorennamen angeben? Wie im obigen Beispiel, mittels „Vorname Name“? Oder, wie es im Bibliothekswesen Standard ist, mittels „Nachname, Vorname“? Es sei angemerkt, daß die Feststellung, daß es sich bei „Andy Powell“ um ein „Vorname Name“-Konstrukt handelt, auf reiner menschlicher Intuition basiert. Ein Computer kann solche Informationen nur schlecht auswerten. Aber genau dies ist das erklärte Ziel von Dublin Core: die verbesserte Ressourcenbeschreibung, um „intelligent agents“ die Suche im Internet zu vereinfachen.

Qualifiziertes DC wird, wenn es in HTML ausgedrückt wird, die Mechanismen benutzen, die HTML schon zur Verfügung stellt. Insbesondere sind dies die `LANG` und `SCHEME` Tags, die optional für das `META`-Tag angewandt werden können. Beispielhaft sieht das so aus:

```
<meta name    = "DC.Date" scheme  = "WTN8601"
      content = "1998-05-14">
```

oder im Falle des LANG-Qualifizierers so:

```
<meta name    = "DC.Title" lang    = "en"
      content = "Jazz Dance: the Story of American Vernacular Dance">
```

Die dritte Möglichkeit Qualifizierer (Subelemente) zu verwenden, wird zur Zeit noch diskutiert. Zum einem ist Diskussiongegenstand, wie diese Subelemente kontrolliert werden sollen, beyiehungsweise ob es einen fest akzeptierten Satz von Subelementen geben soll. Zum anderen ist die Verantwortlichkeit von Subelementeinträgen noch zu kontrollieren. Zum Beispiel fordert ein Teil der DC-Gemeinde, Details über Autoren zu erfassen (zum Beispiel Creator.Address, Creator.Email), andererseits gibt es für diese Daten bereits andere Standardisierungsbemühungen in Form von Admin Core (Ianella and Campbell, 1998), die hier nicht näher beschrieben werden.

Zusammenfassung

Dublin Core ist ein Standard, der sich größtenteils noch in der Entwicklungsphase befindet. Die steigende Anzahl von Tools und Implementierungen sollte darüber nicht hinwegtäuschen. Prinzipiell liegt mit den 15 Kernelementen von Dublin Core eine Möglichkeit vor, Internetressourcen zu beschreiben und diese Informationen von „intelligent agents“

auswerten zu lassen. Im momentanen Stadium ist es allerdings fragwürdig, ob die eingangs formulierten Ziele wie einheitliche und verständliche Semantik, einfache Erzeugung tatsächlich eingehalten werden können. Die Katalogisierung von Ressourcen ist sicherlich keine einfache Angelegenheit und erfordert ein detailliertes Expertenwissen. Diese Erkenntnis dringt nun auch in die DC-Gemeinde vor. War das ursprüngliche Schlagwort noch „DC Simple“, welches für einen simplen, schlichten Metadatenstandard stand, mit dem Suchmöglichkeiten im Internet gewaltig verbessert werden können, hat man nun von diesem Begriff Abstand genommen. Die Entwicklung traditioneller bibliothekarischer Standards zeigt, daß große Datenmengen nicht mit simplen Mitteln beherrscht werden können. Ein weiterer Schwachpunkt ist die fehlende Standardisierung der Inhalte der Datenfelder, welche direkt die Semantik und die verbesserten Suchmöglichkeiten betrifft und einschränkt. Wenn sich jeder Implementierer zum Ausfüllen der Felder seine eigenen Regeln schafft, ist es problematisch, die Dublin Core-Elemente automatisiert auszuwerten. Qualifiziertes DC ist ein Weg aus diesem Dilemma, allerdings ist es noch weniger standardisiert als das unqualifizierte Dublin Core. Da das Bewußtsein für Metadaten im Internet aber noch recht jung ist, bleibt zunächst abzuwarten, wie sich die weitere Entwicklung (insbesondere in Richtung XML) abzeichnet.

2.3.2 IAFA-Templates

Die Internet Anonymous FTP Archive (IAFA) Templates wurden noch vom Vorgänger des World Wide Web Konsortiums, der Internet Engineering Taskforce, ins Leben gerufen. Es lohnt sich, im Hinterkopf zu behalten, daß die IAFA-Templates zu einer Zeit ersonnen worden sind, als von einer Explosion des Internets noch nicht die Rede war. Meistgebrauchte Internetdienste waren damals noch telnet- und ftp-Zugänge.

Ziel der IAFA Templates ist es, eine einheitliche Möglichkeit zur Beschreibung von ftp-Servern zu bieten. Diese Informationen können dann von Suchmaschinen ausgewertet und Endbenutzern in Form von Anfragediensten zur Verfügung gestellt werden. Die Verwalter dieser ftp-Server sollten die von ihnen zur Verfügung gestellten Dokumente beziehungsweise Ressourcen in Form von IAFA Templates beschreiben. Templates haben hierbei die Bedeutung von „logical groupings of one or more data elements“ (Deutsch et al., 1995). Die IAFA Templates können also zur Indizierung und Beschreibung von ftp-Servern verwendet werden. Durch das Vorhandensein eines Standards zur Beschreibung der Inhalte ist das Entwickeln von Software, welche mehr als einen ftp-Server in Suchdiensten indi-

ziert, möglich geworden. Dies stellt einen erheblichen Fortschritt gegenüber Insellösungen dar.

Die Templates

Der IAFA Standard stellt verschiedene Templates zur Verfügung, die der Beschreibung von Dokumenten und der Konfiguration der IAFA-Services dienen. Sie sind in Tabelle 2.5 zusammengefaßt.

Templates besitzen mehrere Datenfelder, die alle in der Form `<data element name>: <data element value>` ausgefüllt werden. Datenelemente sind je nach Definition obligatorisch, optional oder wiederholbar. Dies ist im IAFA Standard festgelegt (Deutsch et al., 1995). Zum Beispiel können Templates einen Identifizierer (handle) erhalten, der sie eindeutig bestimmt. Auf Templates kann dann mittels dieses Handles verwiesen werden. Dienste, die die Templates indizieren, müssen dann sicherstellen, daß die Handle eindeutig bleiben. Diese Aufgabe muß an einen globalen Dienst delegiert werden, da die Betreiber einzelner IAFA Archive nichts voneinander wissen, und deshalb nur eindeutige Handle in ihrem eigenen „Namensraum“ vergeben können. Innerhalb des IAFA-Systems ist ein Mechanismus, der von vornherein eindeutige Handles erzeugt, unbekannt.

Die genaue Angabe der „Kardinalität“ von Datenelementen ist schon ein großer Unterschied zu Dublin Core, da es hier pro Templatetyp einen Satz von verläßlichen Daten gibt. Dieser Vorteil ist allerdings durch eine geringeren Spielraum zur Beschreibung von Ressourcen erkauft. IAFA Ressourcen entsprechen Ressourcen für die es einen Templatetyp gibt, wohingegen die zu beschreibenden Ressourcen nicht von vornherein limitiert. Daher sind in Dublin Core auch alle Elemente optional.

Cluster

Wichtige Datenelemente, die gehäuft gruppiert auftreten, werden in der Definition der Templates als Cluster zusammengefaßt. Zum Beispiel treten in den IAFA-Templates häufig Beschreibungen von verantwortlichen Personen (Administratoren) oder Autoren auf. Unabhängig von ihrer Rolle als Administrator oder Autor haben beide einen gemeinsamen Satz von erlaubten Datenelementen, wie zum Beispiel Name, Email, Department, Telephonnummer etc. Diese gemeinsamen Elemente werden zu einen USER-Cluster zusammengefaßt und nachfolgend in den Templatedefinitionen so angegeben:

Template	**Template Name**	**Verwendung**
Templates zur Beschreibung von Ressourcen		
Document	DOCUMENT	beschreibt ein Dokument mitsamt Urheber, Titel, Beschreibung und weiteren Attributen
Dataset	DATASET	wie DOCUMENT, aber für Datensätze
Mailing list archive	MAILARCHIVE	s.o.
Usenet archive	USENET	s.o.
Software package	SOFTWARE	s.o.
Image	IMAGE	s.o.
Sound	SOUND	s.o.
Video	VIDEO	s.o.
Frequently asked questions	FAQ	s.o.
Templates zur Konfiguration oder Beschreibung von Diensten		
Site configuration	SITEINFO	gibt Informationen über den Host und den Administrator
Logical archives configuration	LARCHIVE	ein einzelner Host kann in mehrere logische Gruppen (Archive) aufgeteilt werden. LARCHIVE gibt hierüber Informationen.
Services	SERVICE	Stellt der Host Dienste zur Verfügung, so können diese hier beschrieben werden.
Mirror	MIRROR	MIRROR Templates dienen dazu, den Filetransfer zwischen verschiedenen Hosts abzugleichen. Sie enthalten Informationen über die zu spiegelnden Dateien und Zeitpunkte etc.

Tabelle 2.5: Die IAFA Templates

Admin-(USER*), oder Author-(USER*). Weitere Beispiele für Cluster bilden die ORGANIZATION-Cluster, welche Organisationen beschreiben. Sowohl USER- als auch ORGANIZATION-Cluster besitzen Handles, die die Person/Organisation innerhalb eines Archivs identifizieren sollen. In der Theorie sollen dann Suchmaschinen, die die Informa-

tion mehrerer Archive zusammenfassen, die Eindeutigkeit der Handles weiterhin garantieren, so daß innerhalb eines Archivs mittels des Handles auf die Person/Organisation verwiesen werden kann. Dies hat den Vorteil, daß Templates mit Personen/Autoren-Information nicht ständig mit redundanter, sich wiederholender, Information aufgefüllt werden müssen.In Tabelle 2.6 ist die komplette Definition eines DOCUMENT-Templates angegeben (Deutsch et al., 1995) .

Data Element	**Value**	**Comment**
Template-Type:	DOCUMENT	
Category:	Type of Object.	e.g. User Guide, Technical Report
Title:	Complete title of the object.	
URI-v*:	Description of access to object.	
Short-Title:	Summary title.	if the Title is very long
Author-(USER*):	Description/contact information about the authors/creators of the object.	
Admin-(USER*):	Description/contact information about the administrators of the object.	
Source:	Information as to the source of the object.	
Requirements:	Any requirements for the use of the object.	
Description:	Description/abstract.	
Bibliography:	A bibliographic entry for the object when used in other works	
Citation:	The citation for the object when used in other works.	
Publication-Status:	Current publication status of object	draft, published etc.
Publisher-(ORGANIZATION*):	Description/contact information about the object publisher.	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite		
Data Element	**Value**	**Comment**
Copyright:	The copyright statement.	
Creation-Date:	The creation date for the object.	
Discussion:	Free text description of possible discussion forums (USENET).	
Keywords:	Appropriate keywords for this object.	
Version-v*:	A version designator for the object.	
Format-v*:	Formats in which the object is available.	ASCII, Postscript, LaTeX etc.
Size-v*:	Length of object in bytes.	
Language-v*:	The name of the language in which the object is written.	
Character-Set-v*:	The character set of the object	ASCII, ISO Latin-1 etc.
ISBN-v*:	International Standard Book Number	
ISSN-v*:	International Standard Serial Number	
Last-Revision-Date:	Last date the object was revised.	
Library-Catalog-v*:	Library cataloging information	

Tabelle 2.6: Das IAFA-DOCUMENT Template

Regeln

Die Autoren des Internet Drafts waren sich offenbar darüber im Klaren, daß die erstellten Templates gewissen Regeln genügen müssen, damit sie eine sinnvolle Indizierung von ftp-Servern ermöglichen. Dem Benutzer werden im IAFA Standard (Deutsch et al., 1995) eine Reihe von Daumenregeln genannt, die eine möglichst einheitliche Erstellung von

Datensätzen ermöglichen sollen. Diese Regeln umfassen die Angabe von Namen, Emailadressen, Telephonnummern, Zeitangaben und Ortsangaben (in Form von Longitude und Latitude). Namen werden beispielsweise in einem Format angegeben, welches sich an den Namenformat von BibTex orientiert. Grundsätzlich werden sie in Vorname Nachname Format spezifiziert, wobei es Sonderregelungen für Namenszusätze (von, van den, junior etc.) gibt.

Die genaue Angabe von Schemen ist in IAFA-Templates nicht definiert. Der Internet Draft schlägt vor, diese in der textuellen Beschreibung anzugeben, wie in `Keywords: J51 (JEL classification)`. Ein komplettes Beispiel für ein regelgemäßes DOCUMENT-Template könnte so aussehen:

```
Template-Type: DOCUMENT
Title: Digitale Bibliotheken
Category: Diplomarbeit, diploma thesis
Author-Name: Markus Klink
Author-Email: markus@wotan.econ.surrey.ac.uk
Description: The thesis introduces common standards
   in the library science and evaluates current
   trends in the Internet community concerning
   the standardization of Meta Data for the description
   of online resources.
Keywords: digital library, AACR, XML, RDF, Dublin Core,
   ReDIF, person registratrion
```

Zusammenfassung

Mittels der IAFA-Templates soll ermöglicht werden, ftp-Server effizient zu indizieren und so die dort vorhandene Information optimal zu nutzen. IAFA-Services durchsuchen diese ftp-Server und sammeln so Informationen aus mehreren Quellen, die sie dann mit „Gewinn“ für den Benutzer anbieten können. Es wurde schon früh erkannt, daß ein effektives „information retrieval“ nicht an den einzelnen Archiven beginnen kann, sondern von Servern unterstützt werden muß, die sich speziell dieser Aufgabe widmen.

Die Templates sind leicht verständlich und mit einem einfachen Texteditor zu erstellen. Es hat sich aber gezeigt, daß der Aufwand die Templates zu erzeugen, doch zu groß ist. Dies ist mit ein Grund dafür, daß die IAFA-Templates heute nur noch historische Bedeu-

tung haben. Positiv ist zu bewerten, daß man sich auf einige grundsätzliche Regeln für die Datenfelder einigen konnte, auf die sich Benutzer verlassen können. Es fehlen Schemata und Beziehungen zwischen Templates, wie sie prinzipiell in Dublin Core ausgedrückt werden können. Andererseits sind die Templates semantisch spezialisierter, als dies mit den 15 Elementen von Dublin Core möglich wäre. Von daher sind die IAFA-Templates besser an ihre Problemstellung angepaßt. Weiterhin liegt es in der Intention des Standards, daß Templates dort erzeugt werden, wo sie anfallen. Rachel Heery schreibt dazu (Heery, 1996):

> „The underlying philosophy is that it must be the information providers who create metadata records if indexing of the Internet is to be a viable proposition. Given the instability of network resources the alternative of centrally creating records would be a high cost option."

Weiterhin kann es als Vorteil der IAFA Strukturen angesehen werden, daß die Templates zur Beschreibung der Dienste (LARCHIVE, MIRROR etc.) von denen der Ressourcen (DOCUMENT, FAQ etc.) getrennt sind. Dies hätte einen leichten Ausbau der Dienste ermöglicht. Allerdings konnten sich die IAFA-Templates im Bereich des WWW nie richtig durchsetzen.

2.3.3 Resource Description Framework (RDF)

Das gestiegene Interesse an Metadaten im Internet hat zu einer Vielzahl von Standards geführt, die zur Zeit parallel existieren und teilweise unterschiedliche Zielsetzungen verfolgen. So probiert die Dublin Core Gemeinde (Dublin Core, 1999b), die Katalogisierung von Internetressourcen zu vereinheitlichen und im Kreise der Entwickler von Admin Core (Ianella and Campbell, 1998) ist man bemüht, Informationen über Personen und Organisationen einheitlich zu erfassen. Darüber hinaus benutzen viele Anwender eigene Standards, die ihren eigenen Zwecken entsprechen. Oftmals möchte man aber auch schon bestehende Standards benutzen oder ein bißchen erweitern, damit sie der eigenen Zielsetzung entsprechen.

Das Resource Description Framework erlaubt es, solche Standards mittels eines Schemas in einheitlicher Form zu beschreiben und die so erstellten Strukturen in einem Datenmodell zu verwenden. Die Vorgehensweise erinnert geringfügig an objekt-orientierte Technologien. So werden zusammengehörige Attribute mittels des Schemas in Klassen

zusammengefaßt. Diese Klassen gehorchen auch primitiven Vererbungsmechanismen. Es fehlt allerdings eine Unterstützung für Methoden. Weiterhin stellt das RDF Container zur Verfügung, mit deren Hilfe Datenelemente oder Klassen strukturiert werden können. So gibt es Container der Typen Bag, Sequence und Alternative.

Die Verwendung mehrerer Standards (siehe Abbildung 2.2), die unter Umständen gleichnamige Elemente mit unterschiedlicher Semantik besitzen, wird über einen Namespacemechanismus realisiert, so daß auch gleichnamige Elemente oder Klassen in unterschiedlichen Standards sich nicht gegenseitig überschreiben.

Aus dieser Strategie erwachsen für die Metadatenverwendung einige Vorteile. Zum einen wird die Auswertung vereinfacht, weil die Standards syntaktisch vereinheitlicht werden. Das RDF wird bisher nur in XML dargestellt (, wobei andere Verwirklichungen sicherlich möglich sind). Die Syntax von RDF verwendet daher alle Mechanismen von XML und kann mit einen XML Parser verarbeitet werden. Zum anderen verfügen alle per RDF beschriebenen Standards über gleiche Mechanismen, um die Daten zu strukturieren (Container). Weiterhin lassen sich Standards über Vererbungsmechanismen erweitern. „Intelligent agents“ können so zumindest noch den kleinsten gemeinsamen Nenner der Informationen auswerten, wenn ihnen über die erweiterten Informationen sonst nichts bekannt ist.

Schema und Syntax

Der RDF-Standard besteht aus zwei Dokumenten: der Definition des RDF-Schemas (Brickley and Guha, 1999) und der RDF-Syntax (Lassila and Swick, 1999). Die Dokumente sind noch nicht abschließend ratifiziert, sondern befinden sich im Stadium der „proposed recommendation“ bzw. der „recommendation“. Änderungen sind also noch möglich.

Mittels des RDF werden Ressourcen beschrieben, die bestimmte Eigenschaften haben. Diese Eigenschaften haben Werte, welche Literale oder Objekte sein können. Aussagen im RDF bestehen also aus 3er-Tupeln. Die formale Definition der Tupel stellt sich wie folgt dar:

1. Es gibt eine Menge von Ressourcen (engl. *resource*).
2. Es gibt eine Menge von Literalen (engl. *literal*).
3. Es gibt eine Untermenge der Menge Ressourcen, die Eigenschaften (engl. *properties*).
4. Es gibt eine Menge von Aussagen (engl. *statements*), so daß ein Element der Menge

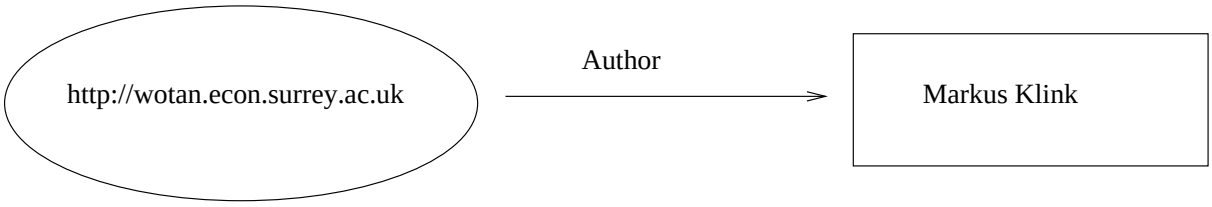


Abbildung 2.4: RDF Beispiel 1: Ein einfaches Statement

Aussagen folgende Form hat: {*pred*, *sub*, *obj*}, wobei *pred* eine Eigenschaft, *sub* eine Ressource und *obj* entweder eine Ressource oder ein Literal ist.

Betrachtet man zum Beispiel folgende Aussage: „`Markus Klink` ist der `Author` der Ressource `http://wotan.econ.surrey.ac.uk/`“. Dann hat das entsprechende Tupel die Form:
(`Author`, `http://wotan.econ.surrey.ac.uk/`, `Markus Klink`)

Oder anders gelesen: Der `Author` der Ressource `http://wotan.econ.surrey.ac.uk/` hat den Wert `Markus Klink`. Unter Verwendung der RDF-Syntax zur Beschreibung von RDF in XML hätte es (verkürzt) die Form:

```
<rdf:RDF>
    <rdf:Description about="http://wotan.econ.surrey.ac.uk">
      <metaklink:Author>Markus Klink</metaklink:Author>
    </rdf:Description>
</rdf:RDF>
```

Die RDF eigene graphische Darstellung findet sich in Abbildung 2.4. Ressourcen oder Objekte werden dabei in Ellipsen dargestellt, Literale durch Rechtecke. Eigenschaften werden durch beschriftete Pfeile angegeben. Das Beispiel wirft schon eine Frage auf: was ist ein `Author`? Und was ist `metaklink`? Das RDF unterstützt selbst keinen Metadatenstandard (wie zum Beispiel Dublin Core), kann also mit einem vorhandenen `Creator`-Element nichts anfangen. Ebensowenig kann es den imaginären Standard `metaklink` verarbeiten. Trotzdem kommt es mit Konstrukten wie `Author` zurecht. Diese werden mittels eines RDF-Schemas (Brickley and Guha, 1999) definiert. Zur Erklärung zunächst ein komplexeres Beispiel:
Die Ressource `http://wotan.econ.surrey.ac.uk` ist von einem `Author` mit `Namen Markus Klink` und `Nationality German` erzeugt worden (siehe Abbildung 2.5). Wie sehen in Beispiel 2 die Tupel aus? Die Ressource hat eine Eigenschaft in Form eines

Autors. Der Wert dieser Eigenschaft muß aber nicht (wie in Beispiel 1) ein Literal sein, sondern kann wiederum eine Ressource (oder allgemein: ein Objekt) sein. In diesem Fall ist es das anonyme Objekt mit der Eigenschaft Autor, welches wiederum die Eigenschaften Name und Nationalität besitzt. Die Tupeldarstellung sieht also wie folgt aus:
(`Author`, `http://wotan.econ.surrey.ac.uk`, (`anonymous`))
(`Name`, `anonymous`, `Markus Klink`)
(`Nationality`, `anonymous`, `German`)

Bei dem Konstrukt `Author` handelt sich also offensichtlich um so etwas wie eine Klasse mit einer bestimmten Anzahl von Attributen. Diese Klassen können in einem RDF-Schema definiert werden. Die Eigenschaften einer Klasse („Attribute“) werden dabei nicht innerhalb einer Klasse definiert, sondern von außen der Klasse zugewiesen:

```
<rdfs:Class rdf:ID="Author">
      <rdfs:comment>The class of Authors.</rdfs:comment>
</rdfs:Class>

<rdf:Property ID="Name">
     <rdfs:comment>The name of the Author.</rdfs:comment>
     <rdfs:domain rdf:resource="#Author"/>
</rdf:Property>

<rdf:Property ID="Nationality">
     <rdfs:comment>The nationality of the Author.</rdfs:comment>
     <rdfs:domain rdf:resource="#Author"/>
</rdf:Property>
```

Die Zuweisung einer Eigenschaft zu einer Klasse erfolgt über das Schlüsselwort `domain`. Die Eigenschaft `Name` darf also nur innerhalb der Klasse `Author` verwendet werden. Eine Eigenschaft darf aber in mehreren Domänen vorkommen (Author, Editor, . . .). Es besteht auch die Möglichkeit, den Wertebereich einer Eigenschaft einzugrenzen. Dies geschieht mittels des Schlüsselwortes `range`. Der Wert eines `range` muß immer eine Klasse sein. Zum Beispiel kann die `Nationality` so eingeschränkt werden:

```
<rdf:Property ID="Nationality">
    <rdf:domain rdf:resource="#Author"/>
    <rdf:range rdf:resource="#CountryNames"/>
```

```
</rdf:Property>
```

Dieses Schema kann dann unter Verwendung eines Namespaces verwendet werden, um mittels RDF Ressourcen zu beschreiben. Syntaktisch sieht das so aus:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:metaklink="http://description.org/schema/metaklink">
    <rdf:Description about="http://wotan.econ.surrey.ac.uk">
        <metaklink:Author>
           <rdf:Description>
                <metaklink:Name>Markus Klink</metaklink:Name>
                <metaklink:Nationality>German</metaklink:Nationality>
           </rdf:Description>
        </metaklink:Author>
    </rdf:Description>
</rdf:RDF>
```

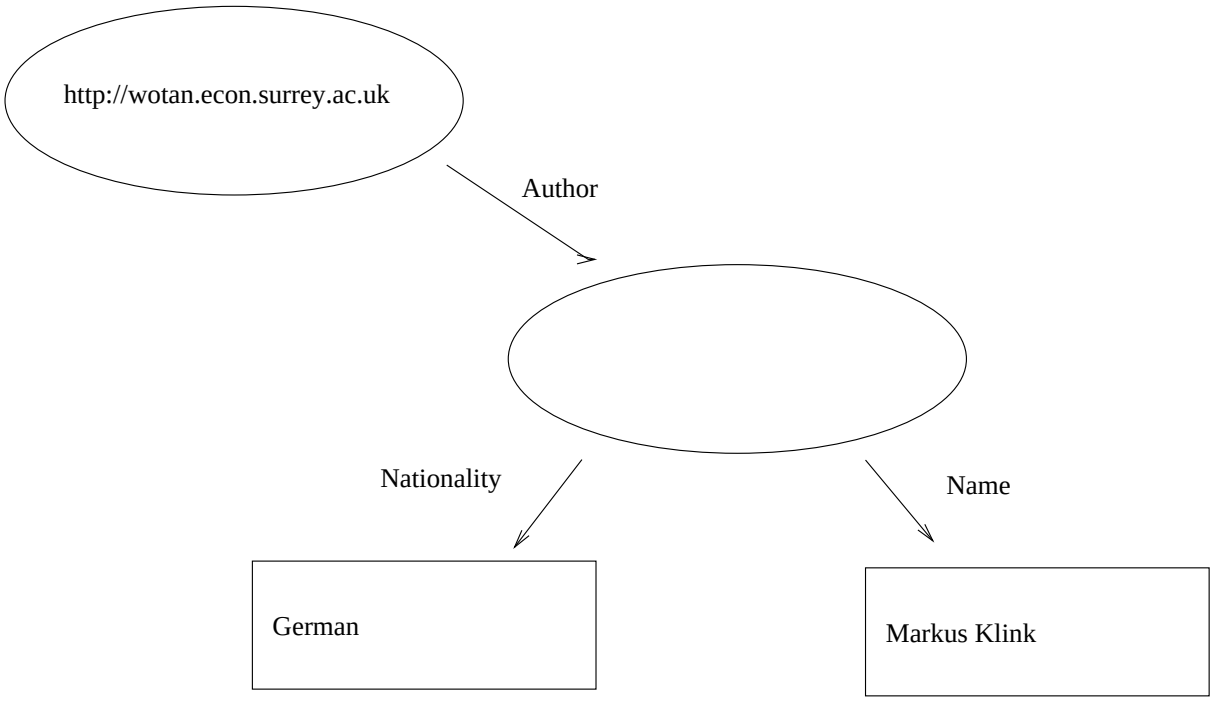


Abbildung 2.5: RDF Beispiel 2: Statement mit anonymer Ressource

Container

Das RDF stellt auch Mechanismen in Form von Containern zur Verfügung, die es erlauben, Elemente und Klassen zu strukturieren. Es gibt folgende Container:

Bag: Ungeordnete Liste, welche doppelte Elemente enthalten darf.

Sequence: Geordnete Liste, welche doppelte Elemente enthalten darf.

Alternative: Liste von alternativen Möglichkeiten, von denen ein Wert angenommen werden darf.

Mittels Containern können komplexe Sachverhalte ausgedrückt werden, die sich durch ein bloßes Wiederholen der Elemente nicht ausdrücken lassen. Zum Beispiel sollen folgende Aussagen durch das RDF ausgedrückt werden: Zum einen „Die Vorstandsmitglieder Meier, Schmidt und Heinz stimmten jeder dem Vorschlag zu.“ sowie „Die Vorstandsmitglieder Meier, Schmidt und Heinz stimmten dem Vorschlag zu.“ Dieser Sachverhalt ist in Abbildung 2.6 ausgedrückt.

In RDF-Syntax gestaltet sich die Beschreibung der zweiten Aussage wie folgt:

```
<rdf:RDF>
    <rdf:Description
         about="http://unternehmen.de/Entscheidungen/Vorschlag">
      <ns:zugestimmt>
        <rdf:Bag>
          <rdf:li resource="http://unternehmen.de/Vorstand/Meier"/>
          <rdf:li resource="http://unternehmen.de/Vorstand/Schmidt"/>
          <rdf:li resource="http://unternehmen.de/Vorstand/Heinz"/>
        </rdf:Bag>
      </ns:zugestimmt>
    </rdf:Description>
  </rdf:RDF>
```

Der RDF-Parser ist verantwortlich für die automatische Durchnumerierung des Elementes `rdf:li`, und ersetzt es durch Elemente der Art `rdf:_1`, `rdf:_2` und so weiter. Innerhalb der Dublin Core Gemeinde liegt ein Dokument vor, welches die Verwendung des RDF zur Darstellung von Dublin Core aufzeigt (Miller et al., 1999). Dort wird die Verwendung

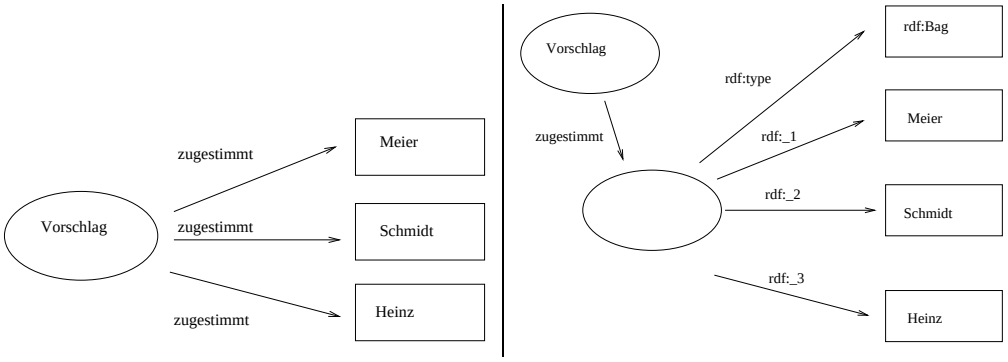


Abbildung 2.6: RDF Beispiel 3: Wiederholte Elemente vs. Container

von Containern empfohlen. Es zeigt sich hier allerdings auch, daß die Verwendung von RDF in der Regel strengere semantische Konstrukte erlaubt, als dies bei der Darstellung von Dublin Core in HTML möglich ist.

Aussagen über Aussagen

Bisher wurden nur einfache Statements vorgestellt. Im RDF ist es aber auch möglich, Statements über Statements auszudrücken. Möglich ist dieser Mechanismus, weil ein Objekt im Datenmodell ein Literal oder eine Ressource sein kann. Dadurch können Aussagen der Art "Markus Klink hat gesagt, daß Thomas Krichel der Autor der Homepage http://a.website.org/Thomas ist" ausgedrückt werden. Weiterhin können dadurch auch bereits vorhandene Metadaten weiterverarbeitet werden. Zum Beispiel könnte jemand primäre Daten sammeln, die in RDF ausgedrückt worden sind, und diese um eigene Aussagen ergänzen. Diese sekundäre Bewertung ist als ein Statement über ein Statement anzusehen. Durch diese Vorgehen werden originale Daten und Ergänzungen aber nicht gemischt, so daß stets die originale Quelle der Daten zu bestimmen ist.

Syntaktisch lassen sich Statements über Statements so begreifen: In der Regel liegt ein Statement in Form einer Description vor (Thomas Krichel ist der Autor der Homepage ...”). Descriptions selbst sind RDF-intern nichts anderes als ein Sammlung von Statements, also vom Typ ”Bag”. Dieser Bag kann eine BagID zugewiesen werden, mit der alle Statements der Sammlung identifiziert werden können.

Eine Aussage über diese ursprüngliche Aussage kann dann also mittels dieser bagID vorgenommen werden:

```
<rdf:RDF>
```

```
<!-- Aussage 1 -->
    <rdf:Description about="http://a.website.org/Thomas" bagID="myID">
      <s:Creator>Ora Lassila</s:Creator>
      <s:Title>Ora's Home Page</s:Title>
    </rdf:Description>

<!-- Aussage über Aussage 1 -->
    <rdf:Description aboutEach="#myID">
      <a:attributedTo>Markus Klink</a:attributedTo>
    </rdf:Description>
</rdf:RDF>
```

2.3.4 SGML und XML

Unternehmen, Universitäten und Informationsanbieter haben in den letzten Jahren zunehmend den Nutzen des Internets erkannt, um dort Informationen anzubieten oder zu erhalten. Oftmals wird das Internet dabei als ein weiterer Markt genutzt, um Informationen darzustellen, die bereits im Unternehmen in anderer Form gespeichert sind. Die „Sprache des Internets“ ist bisher die Hypertext Markup Language (HTML), welche, wie man nun sieht, einige Nachteile hat. Zunächst entwickelt, um einfache Dokumente elektronisch darzustellen (und dies unabhängig von der verwendeten Hardware), zeigten sich die Schwächen von HTML recht schnell, als es nicht nur auf die reine Präsentation der Daten/Dokumente ankam, sondern auch Wünsche nach einer Gestaltung der Seiten aufkamen. Weiterhin bewahren die Tags von HTML nicht die Struktur eines Dokumentes, da sich die meisten Möglichkeiten zur Beeinflussung des Textes auf das Aussehen desselbigen beziehen. So werden Kapitelüberschriften als fett markiert, wichtiger Text kursiv dargestellt und so weiter. Die Information „Kapitelüberschrift“ oder „wichtig“ geht in dem fertigen HTML-Dokument verloren. Dies erschwert insbesondere die spätere Suche nach Informationen, da diese nicht gezielt ausgewertet werden können. HTML bietet selbst aber keine Mechanismen, um die vorhandene Menge von Tags zu erweitern, so daß Dokumentteile als Kapitel, Autorname, Überschrift etc. markiert werden könnten. Genau an diesem Problempunkt setzt die SGML (Structured General Markup Language) und die XML (eXtensible Markup Language) an.

SGML ist bereits Mitte der 70er Jahre entwickelt worden, und wird vor allem im Verlagsbereich eingesetzt – einem Gewerbe, welches traditionell Dokumente in großer Zahl und

auch multi-lingual verwalten muß. Die weiteren Vorteile bei der Verwendung von SGML sind die gleichen, wie sie im XML-Standard (Bray et al., 1998) angegeben sind:

> „XML will
>
> 1. Enable internationalized media-independent electronic publishing
> 2. Allow industries to define platform-independent protocols for the exchange of data, especially the data of electronic commerce
> 3. Deliver information to user agents in a form that allows automatic processing after receipt
> 4. Make it easy for people to process data using inexpensive software
> 5. Allow people to display information the way they want it
> 6. Provide metadata – data about information – that will help people find information and help information producers and consumers find each other“

Insbesondere die Ziele der automatischen Verarbeitung, der Trennung von Inhalt und Formatierung sowie der Unterstützung von Suchmaschinen heben die Vorteile von XML gegenüber HTML hervor. XML ist dabei eine vereinfachte Version von SGML. Da SGML historisch gewachsen ist und sich auch an andere Zielgruppen wendet, finden sich in SGML einige Ausnahmen, Vereinfachungen und Sonderregeln, die es erschweren einen vollkompatiblen SGML-Parser zu entwickeln. Vereinfacht gesagt ist XML unkompliziertes SGML, mit weniger syntaktischen Ausnahmen und einer besseren Anpassung an die Bedürfnisse des Internet.

XML-Syntax

Ebenso wie in HTML (HTML ist eine SGML Anwendung) werden in XML verfaßte Dokumente in Blöcke unterteilt, die durch XML-Tags gegliedert werden. Der große Unterschied besteht darin, daß diese Tags von jedermann frei definiert werden können und in der Zahl nicht begrenzt sind (HTML: festgelegter, nicht erweiterbarer Satz von Elementen mit spezifischer semantischer Bedeutung). XML stellt dabei größere Anforderungen an die syntaktische Korrektheit der so erstellten Dokumente als HTML. Jedes Anfangstag muß durch ein gleichnamiges Endetag abgeschlossen werden. Zum Beispiel:

```
<person>
   <name> Markus Klink </name>
   <email>markus@wotan.econ.surrey.ac.uk</email>
</person>
```

Anfangstags müssen in der umgekehrten Reihenfolge beendet werden, in der sie aufgetreten sind. Weiterhin wird im Gegensatz zu HTML nach Groß- und Kleinschreibung unterschieden. Folgendes Konstrukt ist daher illegal [1]:

```
<person>
   <name> Markus Klink <email> markus@wotan.econ.surrey.ac.uk
   </name></email>
</Person>
```

Ein XML-Dokument, welches sich an diese einfachen Regeln hält, heißt wohlgeformt. Wohlgeformte Dokumente sind syntaktisch korrekt – es ist aber noch keine Aussage darüber gemacht worden, ob die Struktur auch semantisch korrekt ist. Diese Aussage ist mittels der Document Type Definition (DTD) möglich. Die DTD enthält Aussagen über die erlaubten Tags und deren Beziehungen zueinander. Solch eine Aussage wäre zum Beispiel: „Ein Personendokument kann mehrere Personenbeschreibungen enthalten. Eine Person ist korrekt definiert, wenn genau eine Namensangabe erfolgt. Die Information über die email-Adresse ist optional und wiederholbar.“ In einer Document Type Definition sähe diese Definition wie folgt aus:

```
<!ELEMENT person (name,email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Die DTD kann wesentlich kompliziertere Formen annehmen und wird hier nur aus Veranschaulichungszwecken unkompliziert gehalten. Die `<!ELEMENT ...>` Tags sind spezielle Tags, die in der DTD zugelassen sind (kein Endtag!). Mit ihnen können Elemente der Dokumentstruktur definiert werden, die sich entweder wiederum aus weiteren Elementen zusammensetzen oder aus Daten. Die XML kennt dabei verschiedene Formen von Enddaten, die sich generell darin unterscheiden, ob XML die Daten weiter analysiert oder sie

[1] Die Fehler bestehen in der falschen Reihenfolge der Abschlusstags von email und name, sowie in der unterschiedlichen Schreibung von <person> (</Person>)

Kardinalität	**Bedeutung**
*Element**	0..n
Element?	0..1
Element+	1..n
keine Angabe	1..1
Selektierer	**Bedeutung**
Element1,*Element2*	Element1 muß vor Element2 vorkommen
Element1 | *Element2*	Element1 oder Element2
Gruppierungen	**Bedeutung**
(*Element1* , oder | *Element2* ...)	Zusammenfassen von mehreren Elementen

Tabelle 2.7: Die Spezifizierer in der DTD

ohne weitere Bearbeitung übernimmt (PCDATA steht für „parsed character data“ , nicht etwa für „personal computer data“. CDATA steht für ungeparste „Character Data“). Weiterhin können die Kardinalitäten eines Elements sowie Reihenfolgen der Elemente untereinander angegeben werden (siehe Tabelle 2.7).

Ein Dokument, welches die Vorgaben einer Document Type Definition erfüllt, ist gültig (engl. *valid*). Es wird deshalb auch zwischen zwei verschiedenen Formen von XML-Parsern unterschieden: validierende und nicht validierende. Erstere berücksichtigen die Informationen der DTD, letztere überprüfen nur, ob das Dokument wohlgeformt ist. Es ist allerdings eine falsche Annahme, wenn man davon ausgeht, daß nicht-validierende Parser die DTD nicht zu berücksichtigen brauchen. Zum einen können dort sogenannte ENTITIES definiert werden, die im Prinzip Konstanten entsprechen, die nachher natürlich im Text durch den entsprechenden Wert ersetzt werden müssen. Zum anderen muß auch eine DTD auf Wohlgeformtheit untersucht werden.

Damit ein Dokument anhand einer DTD validiert werden kann, muß eine Verbindung zwischen diesen beiden Dokumenten hergestellt werden. Dies kann einmal intern erfolgen, indem die DTD im zu validierenden Dokument spezifiziert wird, oder extern. Bei der externen Angabe wird dem XML-Parser mitgeteilt, an welcher Stelle er die DTD finden kann. So können DTDs im Web abgelegt werden und beliebige Dokumente können auf sie verweisen. Auf diese Art und Weise entsteht ein mächtiges Verfahren, um Dokumente jederzeit erweitern zu können. Die Erklärung erfolgt in der XML-Präambel mittels des DOCTYPE-Elements:

```
<?xml version='1.0'?>
<!DOCTYPE redif  SYSTEM
          "http://wotan.econ.surrey.ac.uk/~markus/redif.dtd">
<redif>
 ...
</redif>
```

Diese Präambel vereinbart mit einem validierden XML Parser, daß das Dokument der Struktur gemäß der DTD entsprechen muß. Ein Beispiel für diese DTD findet sich in Anhang A. Sie stellt einen Prototyp zur Darstellung von ReDIF-Templates (Kapitel 3.3) in XML dar.

Von der Struktur zur Präsentation

In der kurzen Einführung in den vorherigen Abschnitten sollte deutlich geworden sein, daß mittels der XML Dokumente durch benutzerdefinierte Tags strukturiert sind. Definierte Elemente dürfen dabei wiederum aus weiteren Elementen bestehen, die gewissen Kardinalitäten entsprechen müssen. Ob solch ein Dokument eine gültige Struktur abbildet wird durch Vergleich mit der DTD bestimmt. Ein gültiges Dokument läßt sich also in einer Baumstruktur abbilden, in der jedes Kind genau einen Vater hat, oder aber die Wurzel des Baumes ist.

Ein strukturiertes Element ist allerdings nutzlos, wenn es nicht in eine Form gebracht werden kann, in der die Struktur präsentiert wird. Für die XML gibt es eine Vielzahl von Möglichkeiten, ein XML-Dokument in „präsentable“ Form zu bringen.

Schwerpunkt dieser Diskussion ist die „eXtensible Stylesheet Language (XSL)“, welche zur Zeit vom W3C-Konsortium definiert wird (Deach, 1999; Clark, 1999). Der große Vorteil der XSL ist es, daß sie syntaktisch ebenfalls in XML ausgedrückt wird. Andere Lösungen, wie die Cascading Stylesheets (CSS) bieten diesen Vorteil nicht. Die XSL kann weiterhin ein XML Dokument in jedes beliebige Ausgabeformat umwandeln. Cascading Stylesheets hingegen sind an eine Ausgabe in HTML gebunden. Weiterhin kann im Gegensatz zu CSS der Elementbaum in beliebiger Reihenfolge durchlaufen werden, in CSS nur in der Reihenfolge des Auftretens der Elemente.

Von der XSL sagt man im allgemeinen, sie sei templategesteuert. Dies hat folgende Bedeutung: ein Eingabebaum (in Form eines XML-Dokuments) durchläuft einen XSL-

Parser. Der Benutzer spezifiziert dabei mittels Templates (im Sinne von Schablonen), wie bestimmte Teilbäume oder Blätter des Baumes in einen Ausgabebaum transformiert werden sollen. Die Templates können dabei wiederum die Suche nach anderen Elementen des Baumes anstoßen. Diese Elemente werden selektiert („pattern match“). An einem Beispiel wird dies deutlicher. Sei folgendes XML-Dokument als Eingabedokument gegeben und ein HTML-Dokument als gewünschtes Ausgabedokument:

```
<products>
   <product>
      <name>Regenschirm</name>
      <price>12.00 DM</price>
   </product>
   <product>
      <name>Regenjacke</name>
      <price>35.00 DM</price>
   </product>
   <product>
      <name>Regenstiefel</name>
      <price>7.00 DM</price>
   </product>
</products>
```

```
<HTML>
<HEAD>
<TITLE>Produkttabelle</TITLE>
</HEAD>
<BODY>
<TABLE>
<TR><TD>Name</TD>
    <TD>Preis</TD></TR>
<TR><TD>Regenschirm</TD>
    <TD>12.00 DM</TD></TR>
...
</TABLE>
</BODY>
```

Dieses XML-Fragment kann in einer Baumstruktur dargestellt werden (siehe Abbildung 2.7). Sei folgendes Beispiel ein XSL-Stylesheet:

```
<?xml version='1.0'?>
<!-- Namespace für xsl und html(default) deklarieren -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
                xmlns="http://www.w3.org/TR/REC-html40"
                indent-result="yes">

<!-- Template für Root Element
     bereitet Dokumentoutput vor und "taucht" dann in die product
     Elemente -->
<xsl:template match='/'>
```

```
<HTML>
<HEAD>
<TITLE>Produkttabelle</TITLE>
</HEAD>
<BODY>
<TABLE>
       <xsl:apply-templates/>
</TABLE>
</BODY>
</xsl:template>

<!-- Template für product Elemente
     Kreiiert Reihe in der Tabelle mit Werten der Kinder -->
<xsl:template match="product">
  <TR><TD><xsl:value-of select="name"/></TD>
      <TD><xsl:value-of select="price"/></TD>
  </TR>
</xsl:template>
```

Die Abarbeitung erfolgt nach folgendem Schema: Zunächst wird der „Root Node“ gefunden (in der Abbildung „/“). Der „Root Node“ ist dabei wie folgt definiert (Clark, 1999):

> The root node is the root of the tree. It does not occur anywhere else in the tree. It has a single child which is the element node for the document element of the document. The value of the root node is the value of the document element.

Der Wert des „Root Nodes“ wird gemäß Definition durch „products“ ersetzt. Weiterhin erkennt der XSL-Parser, daß eine Schablone auf das Rootelement paßt: `<xsl:template match="/">`. Die Anweisungen dort werden in den Ergebnisbaum geschrieben, bis der Parser auf die Direktive `<xsl:apply-templates/>` stößt. Die Anweisung bedeutet den Parser weiter in den Ausgangsbaum vorzudringen – es werden die Elemente `product` erreicht. Der Parser probiert wieder Elemente und Templates in Übereinstimmung zu bringen, und tut dies erfolgreich mit den zweiten Template. Dort werden die Tabellenzeilen erzeugt und die Werte der Kinder von `product` ausgelesen: `name` und `price`. In

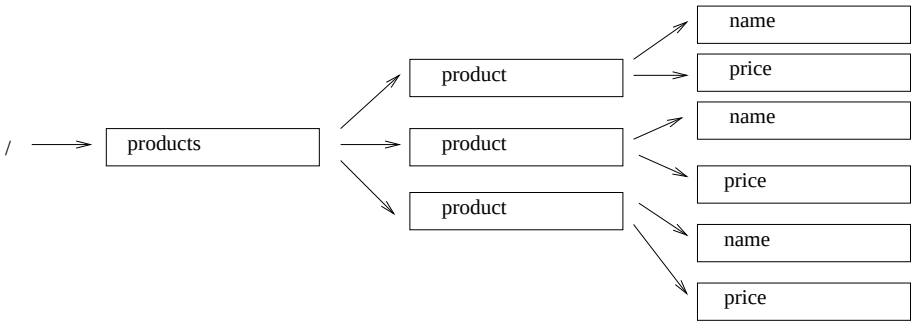


Abbildung 2.7: XML Dokument als Baumstruktur

diesen Beispiel wird also der Ausgangsbaum in rekursiver Weise durchschritten, um den Ergebnisbaum zu erstellen. XSL kann aber auch mehr: so können die zu verarbeitenden Teilbäume gezielt angewählt werden, es gibt logische Schleifen und if-then-else Konstrukte.

Kritik an der eXtensible Stylesheet Language

Beide Schlüsseldokumente der XSL befinden sich noch im Workingdraftstadium. Für eine abschließende Bewertung der XSL ist es daher noch zu früh. Dennoch kann schon ausgesagt werden, daß die XSL andere Zielsetzungen verfolgt, als die anderen Sylesheetsprachen des W3C. So ist das Ausgabeformat der XSL vollkommen unabhängig und frei wählbar. Für die Anwendung der XSL spielt es keine Rolle, ob die Ausgabe in HTML, XML , postscript, in einen Highendformat der druckverarbeitenden Industrie oder in Braillezeilen erfolgen soll. Daher ist die XSL insgesamt komplexer als die anderen Sylesheetsprachen. Ein gängiger Kritikpunkt ist, daß sie zu schwierig zu erlernen ist, wobei es fraglich ist, wie hierbei „schwierig“ definiert ist.

Wesentlich gravierender ist der Umstand, daß bereits kommerzielle Entwicklungen der XSL im Einsatz sind. So unterstützt der Microsoft Internet Explorer in der Version 5 bereits die XSL. Dies allerdings mit eigenen Erweiterungen und auf einen veralteten Workingdraft aufbauend. Die XSL läuft also bereits vor Standardisierung in Gefahr in vielen verschiedenen Versionen angeboten zu werden, wie dies schon mit HTML oder javascript geschehen ist.

Ein anderer Gedankengang wird ebenfalls häufig gegen die XSL angebracht: Da die meisten XML-Dokumente mit Hilfe der XML in HTML konvertiert werden, geht genau der Vorteil verloren, weswegen man zunächst auf XML umgestellt hat: die Strukturierung des

Dokuments. Hätten sich die Ressourcen auf die Entwicklung und Unterstützung der CSS für XML konzentriert, dann hätte man bereits heute ein “strukturiertes, XML-basiertes Internet“. Dem kann entgegnet werden, daß es mittels der XML problemlos möglich ist, per XSL auch XML mit CSS Dokumente zu erstellen. Es ist eben eine der vielen Möglichkeiten und nicht nur die einzige.

2.4 Crosswalking

Das Umwandeln von Datensätzen eines Metadatenstandards in einen weiteren wird als Crosswalking bezeichnet. Crosswalking wird mit den zunehmenden Bedarf an Integration von verschiedenen Metadatenstandards und in Hinblick auf den Austausch von Datenmaterial zunehmend wichtiger.

Eine 1:1-Umwandlung ohne Informationsverlust ist in der Regel allerdings nie zu realisieren. Daher ist ein methodisches Vorgehen notwendig, um diesen Informationsverlust so gering wie möglich zu halten. Folgende Daumenregeln können dafür einen Anhaltspunkt bieten (Pierre and LaPlant, 1998). Für eine erfolgreiche Umwandlung ist es zunächst unerläßlich, sich über beide Metadatenstandards ein ausreichendes Wissen zu verschaffen. Oftmals gibt es mit den verschiedenen Metadatenstandards auch eine dazugehörige Terminologie. Beide gilt es zu erlernen. Weiterhin besitzen die meisten Metadatenstandards eine implizite oder explizite Struktur. Gemeinsamkeiten und Unterschiede sind demnach zu erarbeiten, um ein grobes Verständnis der Struktur der Datensätze zu bekommen. Im nächsten Schritt werden die Elemente der Standards betrachtet. Oftmals gibt es Vorschriften darüber, ob Elemente optional oder Pflichtbestandteile eines Datensatzes sind. Unter Umständen ist zu überlegen, ob sich Elemente im Zieldatensatz künstlich aus anderen Elemente des Quelldatensatzes erzeugen lassen. Umgekehrt müssen Quelldaten eventuell auf mehrere Elemente des Zieldatensatzes verteilt werden.

Dabei muß beachtet werden, daß verschiedene Felder auch Daten in bestimmten Formaten erwarten. Im günstigsten Fall wird die Konvertierung durch die mögliche Angaben von Qualifizierern erleichtert, ansonsten müssen Konvertierer erstellt werden. Diese Aufgabe ist um so einfacher zu bewerkstelligen, je qualitativ hochwertiger die Datensätze sind. Das heißt, wenn sie möglichst einheitlich und standardkonform erstellt worden sind. Deswegen ist auch ein grundsätzliches Verständnis des zugrundeliegenden Regelwerkes notwendig (zum Beispiel der AACR für MARC), um einen Crosswalk erfolgreich durchzuführen.

Die gefällten Entscheidungen sollten dokumentiert werden, damit zukünftige Crosswalks gleichartig durchgeführt werden können.

Der Aufwand der für ein erfolgreichen Crosswalk betrieben werden muß, kann durchaus enorm sein. Insbesondere, wenn mehrere Standards benutzt werden, kann die Zahl der möglichen Kombinationen schnell wachsen (bei N Metadatenstandards folgt, daß es $N^2 - N$ Kombinationsmöglichkeiten gibt). Ein Ausweg könnte in einen formellen Modell liegen, welches einen Crosswalk spezifiziert.

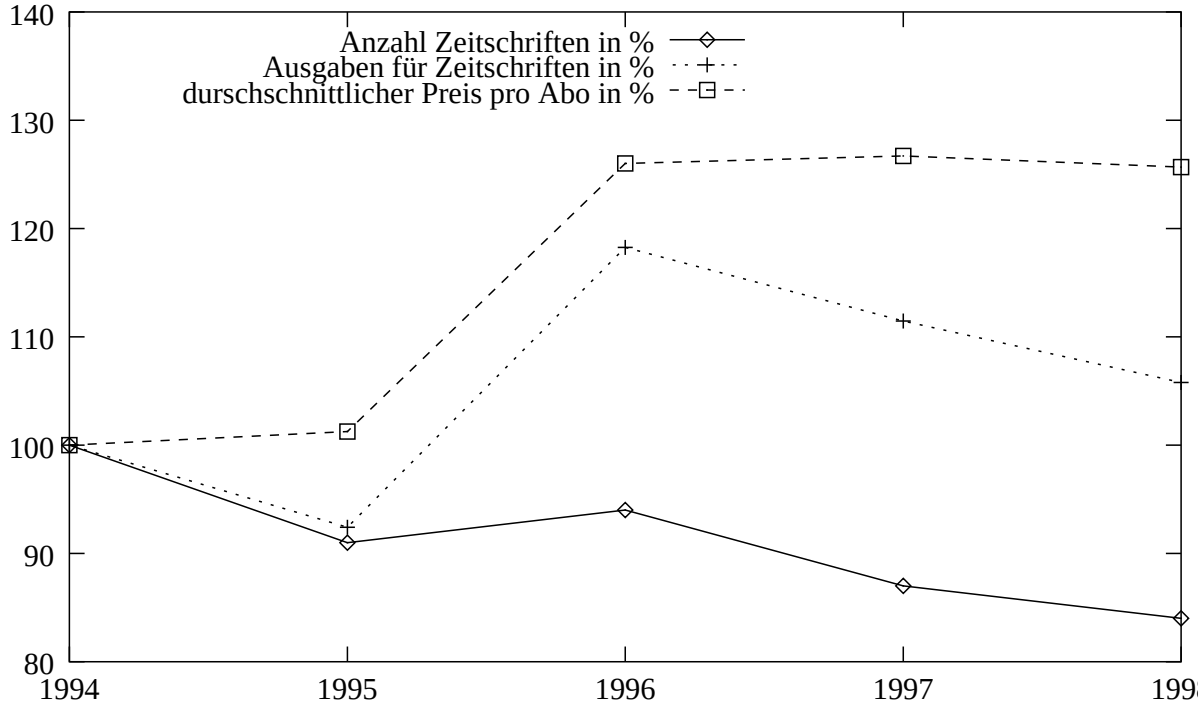
Kapitel 3

Personenidentifikation in einer digitalen Bibliothek

3.1 Einführung

Im wissenschaftlichen Bereich ist der Bedarf an aktuellem Informationsaustausch traditionell sehr groß. In der Regel wird dieser Austausch von Informationen über Fachzeitschriften und Konferenzen bewerkstelligt. Mit der zunehmenden Verbreitung des Internets, welches meist Universitäten zuerst zugänglich war, kommt die digitale Bibliothek als weiteres Informationsmedium hinzu. Insbesondere im Bereich der Informatik, Mathematik und Physik gibt es solche Bibliotheken schon seit langem. Andere Wissenschaftszweige taten sich mit der Akzeptanz des neuen Mediums schwerer. Im wirtschaftswissenschaftlichen Bereich gibt es das Projekt „RePEc“ (Research Papers in Economics), welches sich zum Ziel gesetzt hat, Arbeitspapiere und Preprints im Bereich der VWL und BWL einem breiten Publikum zur Verfügung zu stellen.

Fachzeitschriften und Konferenzen sind sicherlich ein unentbehrliches Hilfsmittel für Wissenschaftler, um sich mit Informationen zu versorgen. Andererseits verursachen sie enorme Kosten, die sich insbesondere Universitäten in Entwicklungsländern nicht leisten können. Ein Zugriff auf die Informationen im Internet ist daher mehr als wünschenswert. Abbildung 3.1 zeigt einen Überblick über die Kostenentwicklung im Zeitschriftenbereich der Universitätsbibliothek Mannheim. Es sind dabei nur die Kosten der Zeitschriftenerwerbung dargestellt. Unberücksichtigt geblieben sind die Folgekosten für Gebäude und Lagerung sowie Verwaltung der Bestände.



Inflationsbereinigt, Basisjahr 1994, Quelle Nominaldaten Deutsches Bibliotheksinstitut Berlin (Deutsches Bibliotheksinstitut, 1999)

Abbildung 3.1: Kostenentwicklung im Zeitschriftenbereich der Universitätsbibliothek Mannheim

Ein realer Preisanstieg der Zeitschriften von über 25% im Jahr 95/96 ist deutlich zu erkennen. Die Ausgaben für alle Zeitschriften folgen diesem Trend. Von 1994 bis 1998 nimmt die Zahl der jährlich beschafften Zeitschriften beinahe kontinuierlich ab. Es kann aus diesen Zahlen allein kein Rückschluß auf eine mögliche Motivation getroffen werden. Denkbar wäre aber entweder eine Streichung der Mittel für Zeitschriften, so daß bei sinkenden Mitteln auch weniger Zeitungen gekauft werden können, oder aber ein Versuch, den Verlust an Zeitschriftenbestand mittels anderer Medien zu kompensieren (zum Beispiel durch Kauf von CD-ROMs).

3.2 Ziele von RePEc

Der Begriff „RePEc“ ist nicht präzise definiert. Zum einen wird darunter eine Sammlung von Archiven verstanden, die wirtschaftswissenschaftliche Daten zur Verfügung stellen. Andererseits wird mit RePEc auch der komplette Datensatz selbst bezeichnet. Zugu-

terletzt wird RePEc auch als Bezeichnung der, in das Projekt involvierten, Personen verwandt. Aufgrund dieser Begriffsvieldeutigkeit empfiehlt es sicht nicht, das Akronym aufzulösen.

Mittels RePEc werden im Bereich der Wirtschaftswissenschaften zwei Ziele verfolgt: Zum einen soll eine Bibliotheksfunktion erfüllt werden. Das heißt, daß Benutzer wie in einer normalen Bibliothek nach Artikeln oder Sachgebieten suchen können. Das zweite (bisher nur zum Teil verwirklichte) Ziel ist es, den vorhandenen Datensatz so zu nutzen und zu erweitern, daß die Disziplin in ihrer Gesamtheit erfaßt wird. Das muß näher erklärt werden: Zum einen besitzt RePEc mit ReDIF ein eigenes Metadatenformat, welches sich an die IAFA-Templates anlehnt, aber mehr auf die spezifischen Zielsetzungen Bibliothek und Disziplinerfassung zugeschnitten ist. Zum anderen arbeitet es mit den sogenannten Guildford-Protokoll, welches dem Informationsaustausch innerhalb des Systems regelt – insbesondere, wie Informationen einheitlich in einer Filestruktur abgelegt werden.

ReDIF definiert weit mehr Templates als dies für eine bloße Katalogisierung von Arbeitspapieren notwendig wäre. So unterscheidet es u.a. zwischen Arbeitspapieren, Kollektionen, Institutionen und (als Projekt dieser Diplomarbeit) Personen als eigenständigen Entitäten. Gelingt es zwischen diesen Grundpfeilern Verbindungen herzustellen, so gelangt man schon wesentlich näher an das Ziel, eine Disziplin insgesamt zu erfassen. Mögliche Suchanfragen gehen dann über die Bibliographie- und Katalogsfunktion einer Bibliothek weit hinaus, da sie auch Anfragen der Art
– Wer arbeitet an der Institution XYZ?
– Was wurde an der Universität ABC publiziert?
– Wer arbeitete an der Serie S?
– Was publizierte die Person Q?
erlauben.

Je nach Sichtweise bietet dieses Archivsystem auch noch weitere Vorteile gegenüber der Verbreitung von Artikeln in Fachzeitschriften. Im RePEc-System gibt es bisher noch kein „peer-review“ System. Darunter versteht man in der Regel die Bewertung eingegangener Artikel durch Editoren von Zeitschriften, um die Eignung zur Veröffentlichung zu prüfen. Idealerweise erfüllen diese „peer reviews“ eine wünschenswerte Vorselektionsfunktion, indem sie Informationen filtern und so die Wissenschaftler einer Disziplin vor einer Informationsflut bewahren. Andererseits scheint Qualität nicht das einzige Merkmal zu sein, anhand dessen ein Arbeitspapier beurteilt wird. Ansehen und Bekanntheitsgrad des Autors, welche bei Veröffentlichung im Gegenzug auch Ansehen und Bekanntheitsgrad

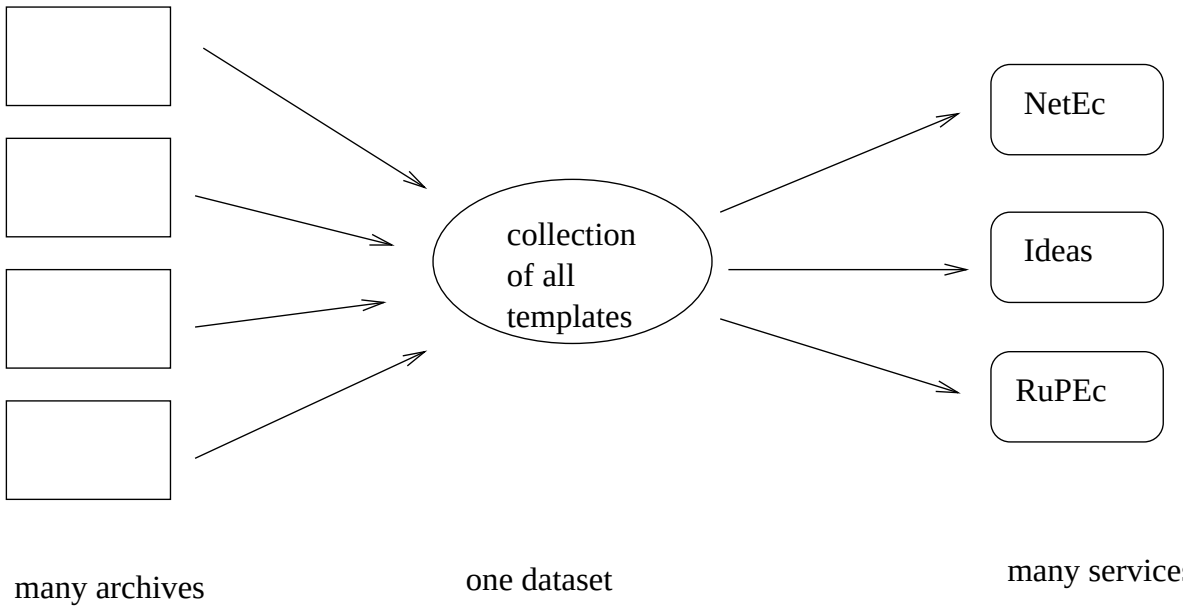


Abbildung 3.2: Aufbau der RePEc Dienste

der Zeitschrift steigern, scheinen auch eine Rolle zu spielen. Weiterhin liegen zwischen Prüfung zur Veröffentlichung und tatsächlicher Publikation oftmals ein bis zwei Jahre. Relativ zu einer angenommenen „Lebensdauer“ eines Arbeitspapiers von vier bis fünf Jahren ist dies eine lange Zeit.

Ein Autor könnte sein Arbeitspapier auf seiner Homepage oder der seines Instituts ablegen und so den Text auch schon vor Publikation zugänglich machen. Allerdings wäre es für interessierte Kollegen eine Sisyphusarbeit, auf diese Art und Weise Informationen zu suchen, da sie an unzähligen Stellen nach Informationen suchen müßten. Eine Suche über Internetsuchmaschinen wird in der Regel auch nicht den gewünschten Erfolg bringen. Vielversprechender ist es, einige bibliographische Informationen so aufzubereiten, daß sie von einem spezialisierten Service verstanden werden können. Das zugrundeliegende Denkmodell dafür ist in Abbildung 3.2 dargestellt. Die einzelnen Services verwenden alle die gleichen Daten, benutzen aber verschiedene Applikationen, die sie je nach Zielsetzung, Hardware und Dienstleistung variieren können.

Einer dieser Service wäre zum Beispiel NEP (New Economics Papers), welcher über Mailinglisten über neue Arbeitspapiere in der Datenbank berichtet. Den Berichten liegt dabei kein wertendes „peer-review“ System zugrunde.

Template	Verwendung
Paper	beschreibt ein Workingpaper bzw. Arbeitspapier
Article	beschreibt einen Artikel, der in einen Journal erschienen ist
Chapter	beschreibt Teile einer Publikation (Proceedings, Festschriften) mit jeweils unterschiedlichen Autoren
Book	beschreibt ein Buch
Software	beschreibt Software bzw. Code
Institution	beschreibt eine Organisation bzw. einen Arbeitsplatz
Person	beschreibt eine Person und Relationen zu Institutionen und Dokumenten (neu implementiert)
Archive	beschreibt ein physikalisches Archiv
Series	beschreibt eine Sammlung (Kollektion) von gleichartigen Dokumenten
Mirror	dient zur Verwaltung des Datenaustausch zwischen verschiedenen RePEc Servern

Tabelle 3.1: Die definierten ReDIF-Templates

3.3 ReDIF-Templates

Von den bisher vorgestellten Metadatenstandards sind die ReDIF-Templates (Krichel and Cruz, 1997) die am meisten spezialisierten. Das ReDIF Format hat als Zielsetzung, wissenschaftliche Dokumente (Workingpaper, Journalartikel, Bücher etc.) zu beschreiben und diese Informationen in einer einheitlichen Datensprache zur Verfügung zu stellen. Durch diese Spezialisierung können Probleme vermieden werden, denen sich zum Beispiel Dublin Core stellen muß. So entfällt zum größten Teil die Problematik der 1:1 Regel (Workingpaper sind im Rahmen von ReDIF immer ein Ganzes), und auch die Probleme der geistigen Urheberschaft werden durch eine Spezialisierung der Datenfelder in Autoren, Editoren und Administratoren umgangen. Weiterhin ist der Umfang der zu beschreibenden Ressourcen stark eingegrenzt. Man hat erst gar nicht den Anspruch, „alles“ im Netz zu katalogisieren und erfaßt nur Daten über Artikel, Arbeitspapiere, Serien in Journals etc.

Von der technologischen Seite bilden lehnen sich die ReDIF-Templates stark an die bereits in Kapitel 2.3.2 beschriebenen IAFA-Templates an.

In den ReDIF-Templates sind viele der Ideen aus den IAFA-Templates übernommen worden. So teilen sie sich zum Beispiel die gleiche Syntax für Felder (<*Data Element:*> <*Data*

Value>) und die Idee der Cluster. Im Gegensatz zu den IAFA-Templates finden sich aber wesentlich mehr Templates zur Beschreibung von verschiedenartigen textuellen Dokumenten (vergleiche Tabelle 3.1 mit Tabelle 2.5). Weiterhin besitzt jedes Template einen im gesamten Datensatz eindeutigen Identifizierer, welcher hier ebenfalls Handle heißt. Mittels des Handles kann jedes Template aufgefunden und verarbeitet werden. Da jedes Template einen solchen Handle besitzt, ermöglichen die Handles auch Verweise von einem Template auf ein anderes. Diese Idee wird nun in ReDIF verwirklicht. Darauf wird später noch eingegangen.

Paper-Template

Da das Paper-Template das mit Abstand wichtigste Template ist, werden seine Elemente tabellarisch zusammengefaßt. Im WWW ist eine Liste und Beschreibung aller Templates mitsamt der Elemente verfügbar (Krichel and Cruz, 1997).

Es folgt ein Beispiel für ein Paper-Template:

```
Template-Type: ReDIF-Paper 1.0
Title: Digitale Bibliotheken
Author-Name: Klink, Markus
Author-Email: markus@wotan.econ.surrey.ac.uk
Author-Workplace-Name: University of Surrey
Author-Workplace-Postal: Guildford GU2 5XH
                         England
Abstract: This thesis introduces various metadatastandards which
   evolved in the library community, and more recently in the internet
   cataloguing community. In the second part an application to register
   authors in an electronic library is introduced and developed.
File-URL: http://wotan.econ.surrey.ac.uk/diploma.pdf
File-Format: application/pdf
Handle: RePEc:zzz:zserie:99-01
```

Data Element	**Value**	**Comment**
Template-Type:	ReDIF-Paper 1.0	(**mandatory**)
Title:	Complete title of the working paper.	(**mandatory**)
Author-(USER*):	Description/contact information about the authors/creators of the object.	(**mandatory and repeatable**)
Contact-Email:	Contact email for this email	in case the author does not wish to provide his own email
Abstract:	Description/abstract.	
Classification-*Scheme*:	Classification for this document	currently only the JEL-Scheme is supported
Keywords:	Appropriate keywords for this object.	semicolon seperated list
Note:	Any other relevant information about the paper	
Length:	The length of a printed versions of the document	usually in pages
Availability:	Can be used to give information on how to get hold of the paper if it is not available online	
Creation-Date:	Specifies the creation date of the object	YYYY-MM[-DD]
Revision-Date:	Speicifies optional revision dates	YYYY-MM[-DD](**repeatable**)
Price:	Proce and currency of the object	
Publication-Status:	Current publication status of the paper.	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite		
Data Element	**Value**	**Comment**
Notification:	To specify how new versions of this paper are announced.	
Restriction:	If the paper is available online, but the acccess is restricted, information can be given here about the nature of these restrictions	
File-(FILE*)	Information about the URL and format of an electronic version of the workingpaper	(**repeatable**)
Handle:	Unique identifier with a specified value structure	(**mandatory**)

Tabelle 3.2: Das ReDIF-Paper Template

3.3.1 Strukturen von RePEc

Logische Struktur

RePEc ist ein Datensatz, der Beschreibungen wirtschaftswissenschaftlicher Dokumente und dazugehöriger Entitäten, wie Institutionen und Personen, enthält. Dieser Datensatz erhält durch die "Beschreibungssprache" ReDIF eine logische Struktur. In ReDIF sind für jedes Template gewisse Felder Pflichtfelder, andere sind optional oder wiederholbar. ReDIF selbst wird innerhalb des RePEc Projektes als Beschreibung der logischen Struktur der Daten angesehen. ReDIF ist also mehr als die Ausprägung eines Datensatzes in ASCII-Beschreibung, vielmehr ist diese ASCII-Beschreibung ein mögliches Datenformat zur Beschreibung von ReDIF. Es heißt fortan ReDIF-ASCII. Weitere mögliche Ausprägungen ein und desselben Templates könnten auch in XML oder RDF vorgenommen werden. Da sich ReDIF stark an die Bedürfnisse der Wirtschaftswissenschaften anlehnt, sich das Projekt aber zunehmend auch anderen wissenschaftlichen Disziplinen zuwendet, wird momentan darüber nachgedacht, eine Strukturbeschreibungssprache zu entwickeln, die es erlaubt, logische Strukturen flexibel darzustellen, so daß Parser beliebige Templates auslesen und validieren können. Die Sprache modelliert also ein Metamodell, welches Beziehungen von Entitäten beschreibt. Programme sollen dann in der Lage sein, diese

Beziehungen in ein Format, welches ReDIF-ASCII ähnlich ist, abzubilden. So soll der Wartungsaufwand für Programme verschiedener Disziplinen gering gehalten werden und gleichzeitig den einzelnen Disziplinen Entscheidungsspielraum eingeräumt werden.

Zur Zeit existiert mit dem perl-Modul `rr.pm` ein Parser, der ReDIF-ASCII Templates ausliest und auf Gültigkeit überprüft. Seine Information über gültige Datenelemente und zu einem gewissen Grad auch über gültige Dateninhalte bezieht er aus einer Datei namens `redif.spec`. Neben den logischen Strukturen innerhalb eines Templates existieren auch Verbindungen zwischen verschiedenen Templates. Diese werden zur Zeit entwickelt und in Teilen auch durch diese Diplomarbeit verwirklicht (siehe Kapitel 3.4). Sowohl der Parser als auch die Konfigurationsdatei können aber keine Relationen zwischen Entitäten abbilden. Dies erfolgt in Programmen, die die Templates weiterverarbeiten.

Jedes Template ist weiterhin Teil eines Archivs, welches wiederum aus einer oder mehreren Serien besteht. Ein Archiv ist als übergeordnete Einheit einer publizierenden Institution (Universität, Verlagshaus, Sonderforschungsbereich, Zentralbank, think tanks ...), welche in RePEc Informationen zur Verfügung stellt, anzusehen. Archive müssen dann wiederum logisch in Serien unterteilt werden. Zu diesen Zweck gibt es das Archive- und Seriestemplate, welches RePEc mit Informationen über diese Struktur versorgt.

Physikalische Struktur

In herkömmlichen Bibliotheken erfolgt die Informationsbeschaffung und Informationsverwaltung in der Regel zentral durch ausgebildete Bibliothekare (, was auch das Einhalten von Regeln vereinfacht). Dies ist für kleine digitale Bibliotheken durchaus vorstellbar, oder für solche, wo ein bestehender Bibliotheksbestand elektronisch zur Verfügung gestellt wird.

Für eine Bibliothek wirtschaftswissenschaftlicher Arbeitspapiere (insbesondere bei einem Freiwilligenprojekt wie RePEc) ist solch ein Weg aber nicht gangbar, weil der Aufwand für Informationsbeschaffung immens wäre. Im Rahmen des RePEc-Projektes stellen Administratoren der jeweiligen Institution ihre Informationen jeweils lokal zur Verfügung. Der gesamte RePEc-Datensatz ist somit eine Kollektion vieler kleiner Einzeldatensätze. Das Prinzip ist dabei wiederum den IAFA-Templates entnommen, wo die Verantwortung der Informationsbeschreibung ebenfalls bei den jeweiligen Administratoren der ftp-Server liegt. Die Informationsaufbereitung erfolgt demnach dezentral.

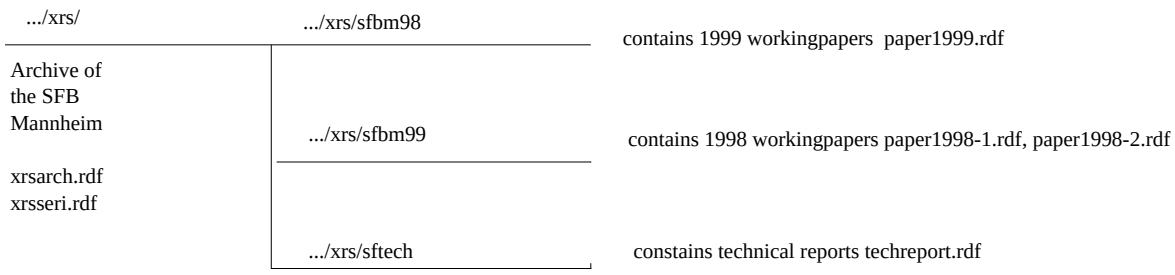


Abbildung 3.3: Archivstruktur eines lokalen Archivs

Anders als bei IAFA [1] ist bei den ReDIF-Templates aber eine physikalische Struktur (Filestruktur) zur Speicherung der Daten vorgeben. Jedes Archiv erhält einen eindeutigen 3-Buchstabencode (den Archividentifier). Diesem Code muß ein gleichnamiges Verzeichnis im ftp (oder http) Server entsprechen. Das Archivverzeichnis enthält wiederum Unterverzeichnisse, die den Serien entsprechen. Serien bekommen 6-Buchstabencodes und können vom Archivadministrator frei gewählt werden. Archive und Serien werden dabei im Archivverzeichnis durch Archiv- und Serientemplates beschrieben. Innerhalb der Serienverzeichnisse befinden sich dann die Dateien, die die Dokumente mittels ReDIF beschreiben, wobei mehrere Templates in einer Datei zusammengefaßt werden dürfen. Der Dateiname entspricht dabei meist. nicht dem Namen der beschriebenen Dokumente. Abbildung 3.3 gibt solch einen Verzeichnisbaum wieder. Der Aufbau der physikalischen Struktur wird im sogenannten „Guildford Protocol“ definiert (Krichel, 1997). Da die Archivcodes zentral vergeben werden (und daher eindeutigen Verzeichnissen entsprechen), können alle lokalen Archive heruntergeladen werden und bei einem zentralen Server in einem Abbild der Dateistruktur zusammengefaßt werden. Ein Archivadministrator muß also nur einmal einen Archivhandle beantragen und daraufhin den Standort seines Servers bekanntgeben, damit er seine Daten zur Verfügung stellen kann. Die Updates erfolgen von da an periodisch, eine weitere Benachrichtigung der RePEc-Administratoren ist nicht mehr notwendig.

Das Verfahren hat aber auch einige Nachteile: So kann in der Regel nur sehr aufwendig bestimmt werden welche Templates neu in der Datenbank sind. Dies geschieht in der Regel dadurch, daß geprüft wird, ob der Handle bereits vorhanden war oder nicht. Fehlerquellen liegen bei diesem Verfahren darin, daß ein Archivadministrator auch den Handle verändert haben könnte, ohne daß die Information tatsächlich einem neuen Template entspricht. Der Vergleich von Dateizeiten bringt auch keinen Aufschluß, da mehrere Templates in einer

[1] IAFA-Dienste finden Informationen, in dem sie einen ftp-Server rekursiv nach Dateien mit der Endung .AFA durchsuchen

Handle:	Authority	:	Archive	:	series	:	free text	structure
	5 letters	:	3 letters	:	6 letters	:	unlimited	length information
	RePEc	:	xrs	:	sfbmaa	:	97-01	example

Tabelle 3.3: Die RePEc Handle Struktur

Datei abgelegt werden dürfen. Der zweite und weit beeinträchtigerende Nachteil liegt in der Tatsache, daß der Informationsfluß immer nur vom lokalen in das globale Archiv erfolgt. Ein Korrigieren von Templates „vor Ort“ ist nicht möglich. Auch ist es nicht möglich, die Templates im globalen Archiv zu korrigieren, weil die Änderungen beim nächsten Update wieder verloren gehen würden.

Struktur der Handles

Jeder vergebene Handle innerhalb des RePEc-Datensatzes muß einer bestimmten Struktur entsprechen. Jedes Template beginnt mit den Namen der Autorität, innerhalb derer die Autorität ein Template definiert. Unter der Autorität ist dabei die Menge der Datensätze zu verstehen, die zu einem bestimmten logischen Gebiet gehören. Momentan sind das RePEc (als Satz wirtschaftswissenschaftlicher Daten), ReLIS (Informationen über Library and Information Science) und ReSOS (Social Sciences) [2]. Autoritäten definieren daher Namespaces, um eine doppelte Vergabe von Handles zu verhindern. Es folgt der 3-stellige zentral vergebene Archivecode, gefolgt vom Seriencode. Der Archivecode ist der einzige Bestandteil, der nicht frei gewählt werden darf, sondern zugewiesen wird. Seriencodes unterliegen bereits der Verantwortung des lokalen Archivadministrators. Der freie Teil dient dann letztendlich dazu, das individuelle Template eindeutig zu bezeichnen. Ausnahmen dieser Regel bilden nur die Handle für Serientemplates (kein freier Teil) und Templates für Archive (kein Serien- und kein freier Teil).

3.3.2 Vorhandene Services

RePEc steht als Datensatz einer Vielzahl von Services zur Verfügung, die damit unterschiedliche Dienste anbieten können. Es gibt keinen Dienst für RePEc, der eine offizielle

[2]RePEc bezeichnet zur Erinnerung sowohl einen Datensatz der Wirtschaftswissenschaften als auch das Projekt, welches die Dienste betreibt. So kommt es zu den unglücklichen Umstand, daß RePEc einerseits eine Autorität ist, zum anderen aber konzeptuell darüber steht. In letzterer Funktion wird RePEc bald ACMES (Academic Metadata System) heißen.

Anzahl Archive	93	
Anzahl Serien	957	
Anzahl Workingpaper	58236	84,5%
Anzahl Artikel	10299	14,9%
Anzahl Softwarekomponenten	390	0,6%
Gesamt	**68925**	**100%**
davon mit JEL Klassifizierung	21793	31,6%
online verfügbar	15545	22,5%

Stand: 1. Juli 1999

Tabelle 3.4: Der Aufbau des RePEc Datensatzes (wichtigste Templates)

Funktion hätte. Zugleich steht jedes neue Template allen diesen Diensten zur Verfügung, so daß mit minimalem Arbeitsaufwand für einen Archivadministrator größtmögliche Verbreitung der Information gewährleistet ist. Viele verschiedene Dienste bieten (mit unterschiedlichen Methoden) Suchdienste innerhalb des Datensatzes an.

So gibt es an der Universität Manchester NetEc (http://netec.mcc.ac.uk). NetEc bietet unter anderem eine datenbankorientierte Suche an (mittels mysql, einer SQL Variante). Andere Suchlösungen basieren auf ROADS (wertet IAFA Templates aus)

IDEAS an der Universität von Quebec, Kanada (http://ideas.uqam.ca) stellt ebenfalls Suchdienste zur Verfügung, die mittels der Internetsuchmaschine excite verwirklicht werden. IDEAS beherbergt ebenfalls ein Archiv ökonomischer Institutionen, die mittels der Institution Templates wiederum anderen Diensten zur Verfügung stehen.

An der Handelshochschule Stockholm steht mit SWOPEC (http://swopec.hhs.se) ein Dienst zur Verfügung, der mittels internetbasierter Software die Aufnahme insbesondere schwedischer Arbeitspapiere in RePEc erfolgreich fördert.

NEP (New Economic Papers) ist ein System, welches interessierten Personen bibliographische Informationen über neue Veröffentlichungen (in RePEc) nach Themengebieten getrennt zusendet. NEP berücksichtigt dabei nur den Teil der Daten, der auch im Volltext online zur Verfügung steht.

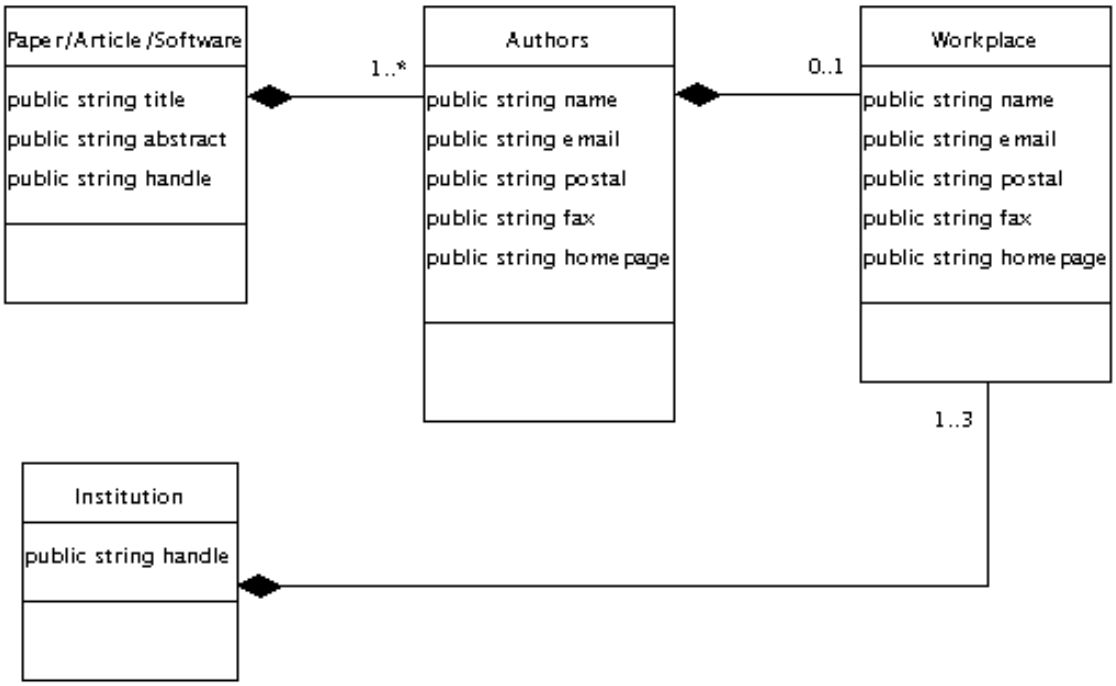


Abbildung 3.4: Kompositionelles RePEc-Datenmodell (Teilausschnitt)

3.4 Problemvorstellung

Eines der großen Ziele von RePEc ist es, eine Forschungsdisziplin als ganzes abzubilden. Vor allem sollten mittels des vorhandenen Datensatzes nicht nur die Bibliographie- und Katalogfunktion einer Bibliothek erfüllt werden, sondern darüber hinaus auch andere Informationen von primärem Interesse erfaßt werden. Die Bemühungen laufen nun verstärkt darauf hinaus, die in den Templates vorhandenen Informationen in Relation zueinander zu setzen. Abbildung 3.4 zeigt das Datenmodell von RePEc zu Anfangszeiten, das heißt als relationale Beziehungen noch nicht verwirklicht worden sind.

Dokumenttemplates sind dabei dargestellt als eine Komposition von Autoren, die wiederum eine Workplace-Information haben können. Weder Autoren noch Workplaces haben dabei eigene Handles. Die Nachteile eines solchen Systems liegen auf der Hand:

1. Wenn ein Autor seine Kontaktinformationen bereitstellen möchte, so muß er das in jedem Paper-Template tun. Das führt zu Datenredundanz.

2. Wenn sich die Autoreninformationen ändern, so müssen die Änderungen in jedem Template vorgenommen werden.

3. Mit dem alten Datenmodell ist es nicht möglich festzustellen, welche Dokumente ein Autor verfaßt hat. Die einzige Möglichkeit besteht im Vergleich der Namen, aber dies berücksichtigt nicht die Möglichkeit, daß zwei Autoren den gleichen Namen haben. Weiterhin könnte ein Autor seinen Namen unterschiedlich angegeben haben (Markus Klink, M. Klink) oder zwischendurch seinen Namen geändert haben (Heirat).

Die Institutionen in ReDIF sind nichts anderes als abgestufte Workplacecluster, die die Organisationstiefe wiederspiegeln. Ein Institution-Template könnte so aussehen (Universität Nantes, Forschungslabor Ökonomie und Management, Abteilung zur Beobachtung und Erforschung von Wasserressourcen und Küstenindustrien):

```
Template-Type: ReDIF-Institution 1.0
Primary-Name: Université de Nantes
Primary-Location: Nantes
Secondary-Name: Laboratoire d'Économie
                et de Gestion de Nantes (LEN)
Secondary-Name-English: Nantes Economics and Management Laboratory
Tertiary-Name: Centre d'Observation et de Recherche
               sur les Ressources Aquatiques
               et les Industries du Littoral (CORRAIL)
Tertiary-Name-English: Center for the Obervation and Research
                       of Aquatic Resources and Coastal Industries
Tertiary-Phone: (33) 02 40 14 17 41
Tertiary-Email: corrail@sc-eco.univ-nantes.fr
Tertiary-Fax: (33) 02 40 14 17 40
Tertiary-Postal: Chemin de la Censive du Tertre,
                 BP 52231 44322 NANTES Cedex 3
Tertiary-Homepage:
         http://www.sc-eco.univ-nantes.fr/LEN/corrail/corrail.html
Handle: RePEc:edi:conatfr
```

Wichtig ist dabei die Feststellung, daß Institutionen Handles besitzen. Daraus folgt, daß sie innerhalb von RePEc eindeutig identifizierbar sind. Autoren und Arbeitsplätzen fehlt

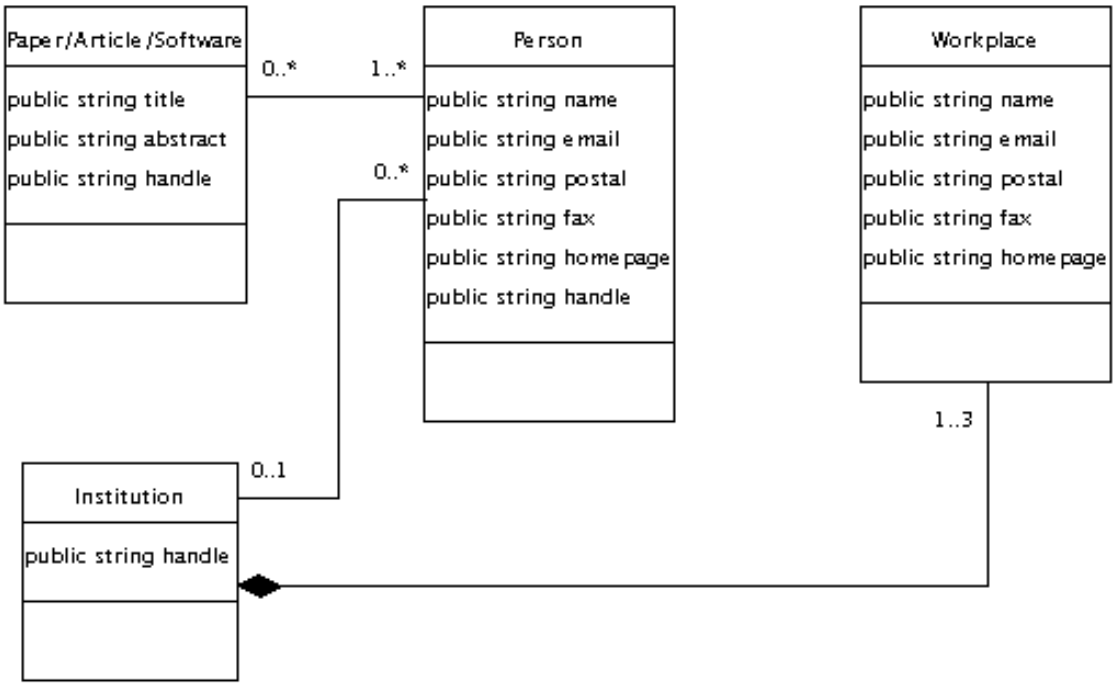


Abbildung 3.5: Assoziatives RePEc-Datenmodell (Teilausschnitt)

ein Handle, daher sind sie auch nicht zu identifizieren und es ist keine Relation zwischen ihnen und anderen Entitäten herstellbar.

3.5 Analyse

3.5.1 Aufbau eines personenorientierten Service

Endziel der Implementationsarbeit soll es sein, einen Service für RePEc zu implementieren, der es ermöglicht, Personen, die innerhalb des System vorhanden sind, eine eigene Identität zu vergeben. Dieses Ziel heißt fortan das Registrierungsziel. Für die Erfüllung des Registrierungszieles ist ein Wandel von einem Modell in dem Personen als Cluster der Dokumenttemplates verwaltet werden, hin zu einen assoziativen Modell notwendig. Abbildung 3.5 zeigt ein solches Modell, wie es in der Analysephase erstellt worden ist.

Im assoziativen Modell besitzt eine Person einen Handle und ist damit eindeutig identifizierbar. Weiterhin geht sie eine Assoziation mit Dokumenten und Institutionen ein.

Ebenso wie im kompositionellen Modell, in dem die Angabe eines Arbeitsplatzes nicht Pflicht war, ist nun die Assoziation mit einer Institution optional. Weiterhin können Personen mit Dokumenten assoziiert sein, sie müssen es aber nicht. Workplaces bestehen weiterhin als kompositionelle Bestandteile der Institution.

Das assoziative Modell ermöglicht es also, über die Handles anzugeben, welche Dokumente von welcher Person verfaßt worden sind. Darüber hinaus sind Assoziationen mit Institutionen möglich. Die in den Zielen (Kapitel 3.2) formulierten Fragestellungen sind nun beantwortbar.

Eine weitere Vorgabe war die bessere Auswertung der vorhandenen Personendaten in den Templates, so daß eine Suche nach Autoren gezielter vorgenommen werden kann. Dieses Ziel kann als Suchfunktion bezeichnet werden. Bisherige Suchen in den vorhandenen RePEc-Diensten arbeiten meist als Volltextsuche über alle oder einige Felder und liefern dementsprechend oftmals überflüssige Ergebnisse, da sie die Informationsstruktur nicht gut ausnutzen.

Benutzerfokus oder Administratorenfokus?

Eine weitere weitere Frage, für die in der Analysephase eine Antwort gefunden werden mußte, ist die Frage, wer für den Aufbau der Assoziationen verantwortlich ist. Solche Assoziationen können immer nur in den Templates aufgebaut werden, weil nur darauf alle Services von RePEc gemeinsam Zugriff haben. Weiterhin kann der Informationsfluß immer nur von den Archiven zum Dienst laufen und niemals umgekehrt (siehe Abbildung 3.2, Informationsfluß von links nach rechts).

Letztendlich ist man zu der Entscheidung gelangt, daß den Archivadministratoren die Arbeit nicht aufgebürdet werden kann, für die Pflege der Assoziationen in den Templates zu sorgen. Diese Daten ändern sich ständig und würden einen hohen Arbeitsaufwand verursachen. Weiterhin können auch von einer Person verfaßte Dokumente und ihre Templatebeschreibungen über mehrere Archive verteilt sein – also auch physikalisch an ganz anderen Orten aufbewahrt werden. Es wäre äußerst fraglich, ob sich jemand die Mühe machen würde, die Daten auf unterster Ebene (also auf Ebene der Archive, die voneinander nichts wissen) zusammenzutragen und stets zu aktualisieren.

Andererseits müssen die Autoren von Dokumenten in die Arbeit miteingebunden werden, denn nur sie wissen genau, welche Dokumente sie verfaßt haben, wie ihre email-Adresse lautet und wo sie gerade arbeiten (Assoziation zur Institution). Autoren haben weiterhin

den größten Anreiz ihre Daten zu aktualisieren, denn so erreichen sie selbst und ihre Dokumente einen höheren Bekanntheitsgrad.

Die gefundene Lösung besteht darin, daß ein RePEc-Dienst die Personenverwaltung übernimmt und die erstellten Templates den anderen Diensten zur Verfügung stellt. Über ein Benutzerinterface wird den Autoren die Möglichkeit gegeben, sich zu registrieren und sich mit den gewünschten Informationen zu assoziieren. Die so gewonnenen Informationen werden als Templates den anderen Diensten zur Verfügung gestellt.

Die Entscheidung für einen Benutzerfokus hat aber auch einen entscheidenden Nachteil: Angenommen Markus Klink registriert sich beim Personendienst und entscheidet sich dafür, daß er Autor der Diplomarbeit „Digitale Bibliotheken“ ist. Der Dienst schreibt ein Template, welches besagt, daß Person Markus Klink mit Handle X die Diplomarbeit mit Handle Y verfaßt hat und verteilt die Information an andere Dienste. Jeder Dienst ist nun in der Lage, ein Verbindung von der Person X zu den Dokument Y (Y1, Y2 ..., wenn die Person mehrere Dokumente verfaßt hat) nachzuvollziehen. Mittels der technischen Möglichkeiten ist es aber nicht möglich, eine Verbindung von Y nach X zu finden (welche registrierte Person hat Y verfaßt?). Zur Erinnerung: Die Templates zur Beschreibung der Dokumente befinden sich nicht in der Kontrolle der Dienste. Dort sind also keine Änderungen auf Initiative des Dienstes hin möglich. Allerdings können die Dienste die Frage nach dem registrierten Verfasser von Y an den zentralen Personendienst weiterleiten (siehe dazu Kapitel 3.6).

3.5.2 Handlestruktur des Persontemplates

Einen eindeutigen Identifizierer für Personen zu erstellen ist eine bisher noch nicht gelöste Aufgabe. Für das RePEc-Projekt mußte eine dem Problem angepaßte Lösung gefunden werden, die nach Möglichkeit nicht mnemonisch und auch nicht kryptisch ist. Reine Identifikationsnummern fallen damit aus. Eine Kombination von Namen und Geburtsdatum, die auch häufig in der Bibliothekswelt benutzt wird, wurde als ausreichend stabil erachtet (Tabelle 3.5). Als Nachteil ergibt sich, daß einige Personen unter Umständen nicht bereit sind, ihr Geburtsdatum preiszugeben, und daß dadurch die Akzeptanz des Service gemindert wird.

Die so gefundene Lösung eines Handles für registrierte Personen berücksichtigt dabei auch die gängige Praxis, daß die ersten drei Komponenten eines Handles stets von konstanter Länge sein sollen (Kompatibilitätsgesichtpunkt mit vorhandener Software). Das Archiv

Person-Handle für registrierte Personen									
Handle:	Authority	:	Archive	:	Birthday	:	fname_lname		logic
	5 letters	:	3 letters	:	10 letters	:	unlimited		length
	RePEc	:	per	:	1972-06-06	:	MARKUS_KLINK		example
Person-Handle für andere Personen									
Handle:	Authority	:	Archive	:	Series	:	Document free part + Name Information		logic
	5 letters	:	3 letters	:	6 letters	:	unlimited		length
	RePEc	:	xrs	:	sfbmaa	:	97-01:MARKUS_KLINK		example

Tabelle 3.5: Die RePEc Person-Handle Struktur

ist dabei fix („per“) und ist neu einzurichten. Damit dient es dem Informationsaustausch zwischen den Diensten, die über eine dem Guildfordprotokoll entsprechende Struktur auf die Daten zugreifen können.

Intern werden für alle anderen Personen (nicht-registrierte Personen) ebenfalls Handles vergeben. Diese Handles können nicht die gleiche Struktur haben wie die Handles für registrierte Personen, da deren Geburtsdatum nicht bekannt ist. Der Handle sollte aber folgenden Anforderungen genügen: zum einen sollte er eindeutig sein und aus rein lokaler Information erzeugbar sein (d.h. nur aufgrund der Informationen aus dem Dokumenttemplate). Zum anderen muß der Handle auch stabil sein (das heißt bei jedem Erzeugen das gleiche Resultat liefern). Diese Anforderungen werden durch die gewählte Struktur für das Handle erfüllt, wenn die Informationen aus Dokumenthandle und Autorname kombiniert werden. Zum Beispiel werden für das Template

```
Template-Type: ReDIF-Paper 1.0
Title: Stock market bubbles
Author-Name: Markus Klink
Author-Name: Thomas Krichel
Handle: RePEc:xrs:sfbmaa:96-01
```

folgende Handles generiert:

```
Handle: RePEc:xrs:sfbmaa:96_01:MARKUS_KLINK
```

```
Handle: RePEc:xrs:sfbmaa:96-01:THOMAS_KRICHEL
```

Die Methode schlägt nur dann fehl, wenn ein Dokument von zwei oder mehr Autoren verfaßt wird, die exakt den gleichen Namen haben, was eher auf einen Eingabefehler schließen läßt.

3.5.3 Dokumenten- und Rollenanalyse

Die Templateübersicht (Tabelle 3.1) wurde daraufhin untersucht, inwiefern die definierten Templates für den betrachteten Problemausschnitt relevant sind. Dabei wurde erkannt, daß das Buch- und Chaptertemplate nicht mit in die Betrachtung aufgenommen werden müssen. Beide Templates sind noch in der Testphase und werden noch nicht offiziell verwendet. Unter Umständen werden sie auch wieder aus der Spezifikation entfernt. Übriggeblieben sind das Paper-, Article-, Software- und Serientemplate. Das Archiv- und Mirrortemplate brauchen ebenfalls nicht berücksichtigt zu werden, da ihre Funktion rein technisch ist und keine Beziehungen zwischen Personen und Dokumenten ausdrückt.

Im weiteren Analyseprozeß wurde noch festgestellt, daß die Assoziation zwischen Personen und Dokumenten auf einem Rollenmodell basiert. Eine Person kann zum Beispiel Autor eines Workingpapers sein und Editor einer Serie. Eine Übersicht über die gefundenen Rollen findet sich in Tabelle 3.6. Die Rollen wurden über eine Analyse der ReDIF-Templates gefunden. Dabei wurde entschieden, daß die Rollen „Contributor“ und „Maintainer“ nicht aufgenommen werden. Der „Contributor“ entfällt, da er nie innerhalb eines RePEc-Datensatzes verwendet worden ist und mittlerweile wieder aus der Spezifikation genommen wurde, und der „Maintainer“ tritt nur innerhalb von Archiv-Templates auf, liegt damit also außerhalb des Problembereiches.

Diese Liste ist allerdings nicht endgültig, da es durchaus möglich ist, daß zukünftig neue Rollen in die ReDIF-Spezifikation aufgenommen werden. Als Beispiel könnte ein Illustrator dienen, wenn es um die Erfassung von geographischen Kartenmaterial geht. Da RePEc sich in Zukunft auch anderen Disziplinen zuwenden möchte, ist mit solch einer Erweiterung zu rechnen.

3.5.4 Aufbau des Personentemplates

Nachdem Klarheit über die zu leistenden Funktionen des Registrierungsdienstes gewonnen worden ist, kann mit der Definition eines entsprechenden Templates für Personen

Templates und zugehörige Rollen				
Templatetype:	Paper	Article	Software	Series
Rolle:	Autor	Autor	Autor	Editor

Tabelle 3.6: Übersicht über mögliche Dokument- / Rollenkombinationen

begonnen werden. Das Template ist in Tabelle 3.7 zusammengefaßt. Es enthält neben den Informationen über persönliche Daten auch die Verweise auf eine Institution sowie auf die assoziierten Dokumente in Form der Datenelemente `Author-Paper`, `Author-Article`, `Author-Software` und `Editor-Series`. Die Benennung der Datenfelder erfolgt dabei nach dem Schema, daß zuerst die Rolle und dann der Dokumenttyp genannt werden müssen. Nach diesem Schema lassen sich flexibel neue Dokument-/Rollenbeziehungen aufbauen.

3.6 Design

Neben der Klärung der grundsätzlichen Vorgehensweise, mußten noch Entscheidungen getroffen werden, die den Aufbau des verarbeitenden Systems betreffen. Diese Entscheidungen gliedern sich in die Erstellung eines Klassenmodells, die Wahl einer Datenbankkomponente, die Verarbeitung der vorhandenen Templates, des Benutzerinterfaces und der Informationsausgabe sowie der Systemkonfiguration.

3.6.1 Klassenmodell

Damit die Registrierungs- und Suchfunktion von den gleichen Klassen verwendet werden können, müssen das kompositionelle (Abbildung 3.4) und das assoziative Datenmodell (Abbildung 3.5) zusammengefaßt werden. Weiterhin dürfen die Erkenntnisse aus Kapitel 3.5.3 (Rollen und Dokumenttypen) nicht unberücksichtigt bleiben. Ein entsprechendes Diagramm der Beziehungen stellt Abbildung 3.6 dar.

Die Klassen `Person` und `Researchpaper` sind dabei über die Klasse `PersonRole` assoziiert. Jede dieser Beziehungen muß genau eine Assoziation mit der Klasse `PersonRole` eingehen, um die Art der Assoziierung näher anzugeben. Weitere Rollen können durch Spezialisierung der Klasse `PersonRole` erzeugt werden.

Data Element	**Value**	**Comment**
Template-Type:	ReDIF-Person 1.0	(**mandatory**)
Name-Full:	full name in lastname, firstname notation	(**mandatory**)
Name-First:	first name	
Name-Last:	last name	
Workplace-(ORGANIZATION*)	workplace cluster	provides workplace information
Workplace-Insitution:	association with Institution	value must be a valid Institutionhandle
Email:		
Fax:		
Postal:		
Phone:		
Homepage:		
Author-Paper:	association with Paper	valid Paper-Handle (**repeatable**)
Author-Article:	association with Article	valid Article-Handle (**repeatable**)
Author-Software:	association with Software	valid Software-Handle (**repeatable**)
Editor-Series:	association with Series	valid Series-Handle (**repeatable**)
Handle:	unique Person-Handle	(**mandatory**)

Tabelle 3.7: Das ReDIF-Person Template

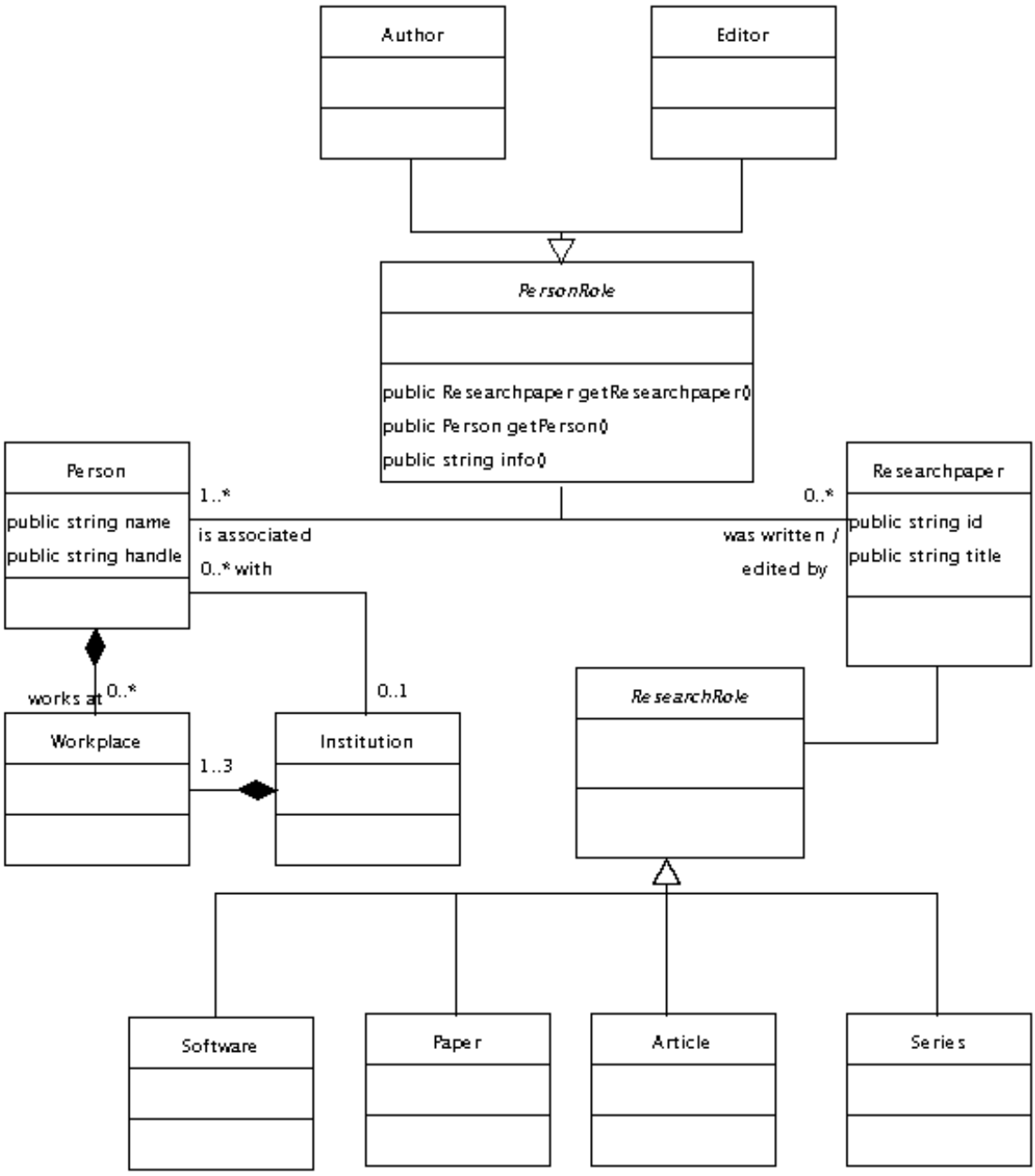


Abbildung 3.6: Basisklassen

`Person` kann nun ebenfalls Beziehungen zu `Workplace` und zu `Institution` aufnehmen, wobei erstere nachwievor mittels einer Komposition verwirklicht ist.

Die Klasse `ResearchRole` ist eine Rollenklasse für das Dokument in `Researchpaper`. Im Rahmen des Systems interessieren die individuellen Eigenschaften eines Dokumentes we-

niger (das heißt Attribute wie Creation-Date, Revision-Date. . . , siehe dazu auch nochmals Tabelle 3.2). Daher wurden in der Klasse `ResearchRole` nur diejenigen Funktionen zur Verfügung gestellt, die für einige Dokumenttypen getrennt verwirklicht werden müssen. Eine Beziehung zwischen `Person` und `Researchpaper` kann im Designstadium auch als Assoziation verwirklicht werden, da durch das künstliche Handlekonzept für Personen sichergestellt ist, daß diese eine eindeutige Identität haben.

3.6.2 Datenbankkomponente

Eine Analyse des Datenbestandes (Tabelle 3.4) ergab schnell die Notwendigkeit einer datenbankgestützten Verwaltung der Daten. Die Wahl fiel auf mySQL (DatakonsultAB, 1999), einer frei erhältlichen SQL-Implementation, die auch schon auf der Zielplattform, einer Sun Solaris Workstation, lief.

Es wurden auch schon die elementaren Tabellen entworfen, die notwendig sind, um die zu verwaltenden Daten zu erfassen. Eine Übersicht findet sich in Tabelle 3.6.2. Die in der Designphase erstellten Tabellen befinden sich dabei im Abschnitt der Basistabellen, die Hilfstabellen wurden entweder aus Performancegründen während der Implementation erstellt, oder aus der Notwendigkeit der Erweiterung des Modells heraus.

mySQL arbeitet intern mit einer perl oder einer C API. Um eine bessere Anbindung an C++ zu ermöglichen, wurde eine einfache Klassenbibliothek entwickelt (siehe Abbildung 3.7). Die Datenbank wird über einen Konstruktoraufruf an `SQLdatabase` initialisiert und kann dann mittels der Methode `query` abgefragt werden. `query` liefert als Ergebnis ein `SQLresult`, welches per Iteratoren durchlaufen werden kann. Die Spalten der Reihen sind dabei immer vom Typ `char *`. Auf die einzelnen Spalten wird dabei wiederum per Indexoperator zugegriffen. Im folgenden ein kleines Anwendungsbeispiel zur Veranschaulichung:

```
const char* name="Klink";
try {
  SQLdatabase db;
  SQLresult res=db.query("SELECT lname, fname FROM authors WHERE
                         lname=\"%s\"", name);
  for (SQLresult::iterator i(res.begin(); i!=res.end(); ++i)
    cout << (*i)[0] << ", "<< (*i)[1] << endl;
}
```

Tabellenname	**Verwendung**
Basistabellen	
`authors`	enthält Namensinformationen einer Person
`author_info`	enthält speziellere Informationen, wie email-Addressen, die Homepage ...
`author_wp`	enthält Informationen über den Arbeitsplatz einer Person
`author_map`	beinhaltet die Schlüssel zur Realisierung einer Relation zwischen Personen in `authors` und Dokumenten in `researchpaper` mitsamt der Rolleninformation
`researchpaper`	enthält Informationen über ein Dokument und die Rolle (Software, Paper, ...)
Hilfstabellen	
`researchpaper_article`	enthält „Article“ spezifische Information
`researchpaper_paper`	enthält „Paper“ spezische Informationen
`researchpaper_series`	enthält „Series“ spezifische Informationen
`researchpaper_software`	enthält „Software“ spezifische Informationen
`person_confirmation`	enthält Details über laufenden Registrierungen (z.B. die Bestätigungsnummer)
`person_mapto_temp`	ähnlich wie `author_map`, allerdings für temporäre Beziehungen während des Registrierungsprozesses
`database_status`	enthält Statusinformationen über den Datenbankzustand (bereit oder inaktiv)

Tabelle 3.8: Die verwendeten SQL-Tabellen

```
catch(SQLerror err) { cout << err.what(); }
```

Eine Prüfung des Wertebereichs des Indexoperators findet dabei aus Performancegründen nicht statt. Ebensowenig werden Konvertierungsfunktionen unterstützt, um Ergebnisse

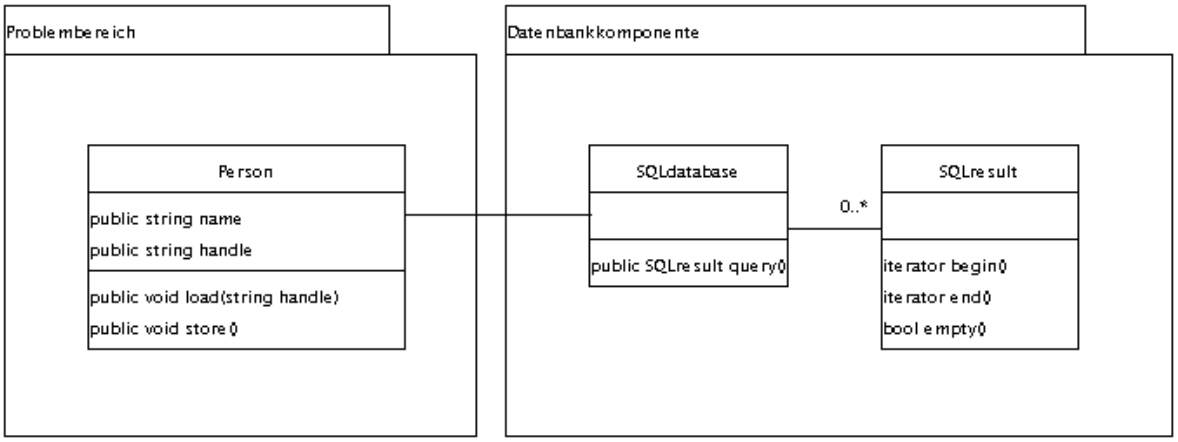


Abbildung 3.7: Datenbankkomponente (Ausschnitt)

in andere Datentypen zu verwandeln. Innerhalb der Anwendungen war das auch so gut wie nie nötig, da alle Daten als strings verarbeitet worden sind.

Andere bereits entwickelte C++ Klassen zur Unterstützung von mySQL wurden nicht herangezogen, da sie sich zumeist entweder nicht auf der Entwicklungsumgebung oder auf der Zielplattform kompilieren ließen. Weiterhin waren einige der Klassenbibliotheken extrem komplex, so daß in Hinblick auf die Weiterentwicklung der erstellten Programme durch Dritte auf die Anpassung an die vorhandenen Compiler verzichtet wurde.

3.6.3 Dateneingabe

Die Datenbank ist innerhalb von RePEc immer nur ein sekundäres Medium. Alle Daten müssen innerhalb von Templates im Dateisystem vorliegen und werden von dort verarbeitet. Zum Validieren der Syntax dieser Templates steht ein perl-Modul zur Verfügung (`rr.pm`), welches einen Verzeichnisbaum nach ReDIF-Templates durchsucht und die syntaktisch gültigen Templates in einer Hashstruktur ablegt.

Damit nicht zwei verschiedene Parser geschrieben werden müssen, wurde entschieden, eine Komponente zu entwickeln, die den Aufruf von `rr.pm` gestattet und daraufhin Zugriff auf die Speicherstrukturen von perl erlaubt. Dieser low-level Mechanismus ist in den Manpages von perl beschrieben (in den Sektionen perlcall, perlembed und perlguts). Um einen verbesserten Zugriff von C++ zu erlauben, wurden Klassen für gewöhnliche Perlvariablen (perlvar), Hashtables und Arrays (perlhash, perlarray) und eine rudimentäre Klasse zur Unterstützung von Funktionsaufrufen (perlfunc) entwickelt.

Variablen können dabei sowhl im Namensbereich von C++ als auch in dem von perl erzeugt werden. Letzteres ist notwendig, um zur Laufzeit auf Variablen innerhalb eines perl-Programms zugreifen zu können.

Die Strukturen, die `rr.pm` anlegt, um ein ReDIF-Template im Speicher abzulegen, sind recht komplex. Vereinfacht handelt es sich dabei um eine Kombination von Hash- und Arraytabellen. Jedes nicht wiederholbare Element ist dabei in einer Hashtabelle abgelegt. Cluster (wie Author-(USER*)) befinden sich in Arrays, wobei die Elemente der Arrays wiederum Hashtabellen sind, um auf die nicht wiederholbaren Elemente zu zeigen. Befindet sich in einem Cluster ein weiterer Cluster (Author-Workplace-(ORGANIZATION*)), so verweist der Eintrag wiederum auf ein Array, welches wiederum auf Hashtabellen verweist und so weiter. Das Auslesen einer solchen Struktur wird an folgendem Beispiel kurz erläutert (die Variable `HashT` ist dabei eine globale Variable der perl-Domäne, die das gesamte Template beinhaltet):

```
string templatetype;
// HashT auslesen
perlhash h1("HashT",TRUE);
templatetype=(const char *)h1["template-type"];
if (templatetype=="ReDIF-Paper 1.0")
{
   // nicht-Cluster Elemente
   cout << templatetype << endl;
   cout << (const char *)h1["handle"] << endl;
   cout << (const char *)h1["title"] << endl;
   if (h1.exists["author"]) {
      // Array der Author-Cluster
      perlarray author(h1["author"]);
      for (int nr=0; nr<=author.len();nr++) {
         // Endknoten oder weitere Cluster in author_info
         perlhash author_info((SV*)author[nr]);
         cout << (const char *)author["name"] << endl;
         // Abfrage für Author-Workplace Cluster
         if (author_info.exists["workplace"]) {
           // wieder ein Array und Hash erzeugen und so weiter
         }
```

```
      }
   }
}
```

Die verschiedenen Casts sind dabei notwendig, weil perl intern Variablen anders behandelt als C++. In Perl kann eine Variable einen beliebigen Typ annehmen und ihn während der Laufzeit eines Programmes auch ändern. Man kann dann entweder abfragen, was für ein Typ gerade in der Variablen gespeichert ist, oder wenn der Typ bekannt ist, direkt darauf casten. Die entsprechenden Operatoren sind dafür in den Klassen überladen worden. Weiterhin können Skalare auch Referenzen enthalten. Eine Referenz darf auch auf Hashtabellen oder Arrays zeigen. Daher kommt es zu der (unschönen) Konstruktion von `perlhash author_info((SV*)author[nr])`. Im Array author befindet sich ein Verweis auf eine Hashtabelle, die als Referenz vorliegt. Um auf die Hashtabelle zuzugreifen, muß zuerst vom Typ perlarray runtergecastet werden (auf ein Skalar), dieses Skalar dereferenziert werden und dann der Hashtabelle author_info zugewiesen werden.

3.6.4 Benutzerinterface

Die Kommunikation mit dem Servicebenutzer erfolgt über das WWW. Dafür gibt es eine Reihe von technischen Möglichkeiten, um in Interaktion mit dem Benutzer zu treten. Alle Methoden haben Vor- und Nachteile, die gegeneinander abgewägt werden müssen. Im Rahmen des Projektes wurde vor allem Wert darauf gelegt, daß auch mit primitivsten Methoden auf die Webseiten zugegriffen werden kann und die volle Funktionalität zur Verfügung steht. Daher wurde auf den Einsatz von Javaskript und Java verzichtet und eine Lösung über die Realisierung von CGI-Skripts angestrebt, die in C++ programmiert werden. Gegenüber einer java-orientierten Lösung hat dies den Nachteil, daß die Kompatibilität auf Serverseite eingeschränkt ist. Zumindest müssen beim Ändern der Plattform unter Umständen Systembibliotheken neu hinzugelinkt werden, die vorher nicht benötigt worden sind, oder die Programme neu kompiliert werden.

Als Ausgangsbasis für eine CGI-Implementation wurde das Headerfile CGI.h von Stephen Martin verwendet (Martin, 1998). Der generelle Aufbau wird in Abbildung 3.8 dargestellt. Das Design verfolgt die Idee, daß sich eine HTML-Seite für ein Programm aus drei Elementen aufbaut: aus einem konstanten Rahmen, bestehend aus Kopf- und Fußelement, sowie aus einem variablen Innenteil. Die Ausgabe der Rahmenteile wird von den Methoden `display_header` und `display_header` übernommen. Die Darstellung

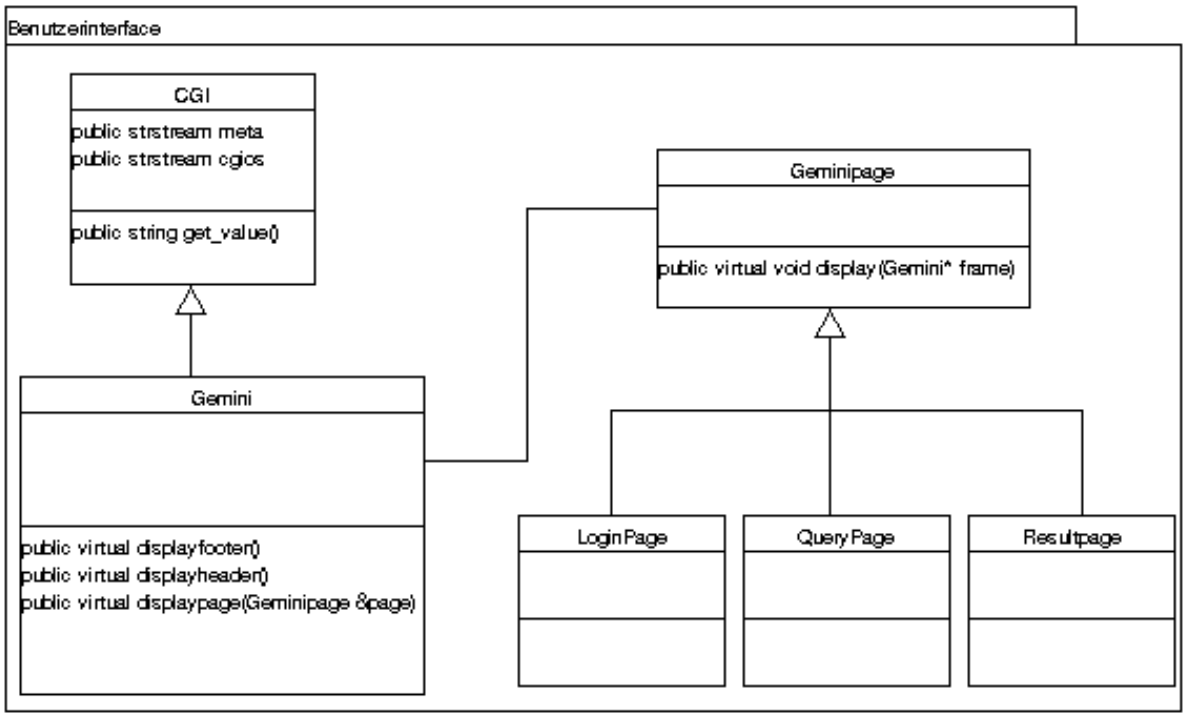


Abbildung 3.8: Benutzerinterface

des Seiteninhalts erfolgt über die virtuelle Methode `Geminipage::display_page`. Eine beispielhafte Anwendung der Klassen im Programmablauf könnte wie folgt aussehen:

```
Gemini frame;
GeminiPage *body;
string mode=frame.get_value("submit");

// ueberpruefen welche Seite dargestellt werden soll
if (mode=="login"||mode=="") body=new LoginPage(&frame);
if (mode=="query") body=new QueryPage(&frame);
if (mode=="result") body=new ResultPage(&frame);

// Kopf, Seite, Fuss darstellen
frame.display_header();
frame.display_page(*body);
frame.display_footer();
```

```
delete body;
```

Des weiteren ist es in der Klasse Gemini vorgesehen, daß der herkömmliche Ausgabestrom zweigeteilt wird. Zum einem existiert ein `strstream cgios`, der für normale Ausgaben verwendet wird, zum anderen ein `strstream meta`, der zur Ausgabe von Informationen verwendet wird, die sich im Header einer HTML-Seite befinden. Beide Streams können unabhängig voneinander benutzt werden. Insbesondere ermöglicht es diese Aufteilung, daß Daten nicht vorher gesammelt werden müssen, wenn sie als Metatags in das Dokument eingearbeitet werden sollen.

3.6.5 Informationsausgabe

Die mit einer Person zusammenhängenden Daten müssen in einer Vielzahl von verschiedenen Ausgabeformaten ausgegeben werde. Die Ausgabefunktion soll dabei möglichst getrennt von der eigentlichen Personenklasse realisiert werden, damit es einfacher möglich ist, neue Ausgabeformate hinzuzufügen, oder bestehende zu ändern. Ein solches Designmuster existiert in Form eines Builderpatterns (Gamma et al., 1995). Die Intention dieses Patterns ist es

> [to] Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Die Verwirklichung des Designmusters ist in Abbildung 3.9 dargestellt. Die Anwendung in der Implementation gestaltet sich folgendermaßen:

```
// Person aus der Datenbank laden
Person p(db,"RePEc:per:1972-06-06:MARKUS_KLINK");
// Ausgabemodus setzen
PersonOutputHTML html;
Person::setOutput(&html);
// ausgeben
cout << p << endl;
```

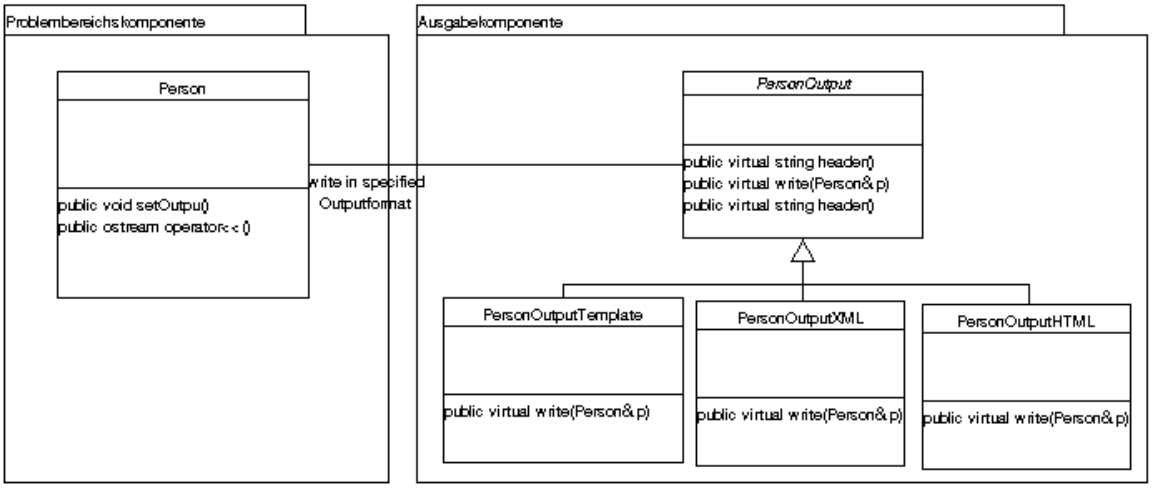


Abbildung 3.9: Ausgabekomponente

3.7 Implementierung

Nachdem in der Designphase die grundsätzlichen Klassen und Abläufe identifiziert worden sind, wurden in der Implementierungsphase die verschiedenen Klassendiagramme zusammengefaßt und die fehlenden Methoden implementiert. Dabei handelt es sich fast ausschließlich um Methoden zum Setzen und Lesen der Attribute sowie um einige Hilfsklassen. Ein konsolidiertes Klassendiagramm wird hier nicht abgebildet – einen Überblick kann man sich aber anhand der Dokumentation im Anhang verschaffen.

Die erstellten Programme gliedern sich in drei Gruppen: die Suchfunktion für Personen (`gemini.cgi`), die Registrierungsfunktion für Personen (`register.cgi`) und die Initialisierung der Datenbank (`addauthors`). Ein Prototyp zur verbesserten Suche in Dokumentdaten ist ebenfalls erstellt worden (`coolsearch.cgi`).

3.7.1 Initialisierung der Datenbank

Die Datenbankanbindung hat zur Aufgabe, die in den ReDIF-Templates gespeicherten Informationen bezüglich Personen, Autoren und Editoren sowie der mit ihnen verbundenen Dokumente (Paper, Article, Software, Series, Institutionen) auszulesen und in einer Datenbank zu speichern. Die Datenbank stellt damit alle Daten zur Verfügung, die von den anderen Programmen benötigt werden (Abbildung 3.10).

Da die ReDIF-Templates von den einzelnen beteiligten Archiven heruntergeladen werden und gemäß dem Guildfordprotokoll auf einem Serviceserver abgespeichert werden, kennt der RePEc-Datensatz keine Zustände. Damit ist gemeint, daß nicht zwischen neuen und alten Templates unterschieden werden kann. Die Datenbank muß also bei jeden Durchlauf vollkommen neu initialisiert werden. Dies bedeutet gleichzeitig für alle anderen Dienste, die die Datenbank benutzen und manipulieren, daß persistente Daten in Form eines Templates erzeugt werden müssen.

Das im Rahmen der Diplomarbeit entwickelte Programm `addauthors` benutzt zur Erfüllung der zugeteilten Aufgabe alle Komponenten, die bisher in der Designklasse vorgestellt worden sind. Das perl-Modul liest die Informationen aus, die in der von `rr.pm` erstellten Hash-Array Struktur vorliegen und erzeugt daraufhin entsprechende Klassen für Personen und Dokumente. Besonderes Augenmerk ist dabei auf die Verarbeitung der Personen gerichtet. Zum einen gibt es die im `per`-Archiv gespeicherten Informationen registrierter Personen, die keine weiteren Probleme verursachen, da ihre Informationen bereits in standardisierter Form vorliegen. Zum anderen werden Namen verarbeitet, die in den Autor-, bzw. Editor-, Clustern vorliegen. Diese sind im Regelfall nicht standardisiert. Standardisiert ist hier in dem Sinne zu verstehen, daß die Namen nicht in einer gängigen Art und Weise (z.b. Nachname Komma Vorname) abgelegt sind. Soll eine Suche über Autorennamen allerdings mehr sein als ein bloßer Vergleich von Zeichenketten, so ist eine solche Struktur unabdingbar. Des weiteren wurden etliche Regelverstöße gegen die Spezifikation von ReDIF entdeckt, die die Verarbeitung weiter erschwerten. So kam es häufig vor, daß in einem `Author-Name` Feld mehrere Personen genannt worden sind. Diese Strings müssen aufgeteilt werden und in „Vorname, Nachname“-Strukturen gebracht werden.

Namensstandardisierung

Der Algorithmus, der die Namen in eine logische Form bringen soll, arbeitet wie folgt: Zunächst wird ein kompletter Textstring eingelesen. Von diesen String wird angenommen, daß er mehrere Personennamen enthalten könnte (z.B. Markus Klink and Thomas Krichel). Dieser String wird zunächst gesäubert, das heißt, es werden alle potentiell störenden Elemente (jr., Prof., Dr., Ph.D. etc.) entfernt. Der String wird dann rekursiv zerlegt (Funktion `splitnames, addpersons.cc`), bis sich keine Namenstrenner mehr in den Teilstrings befinden. Zu diesen Trennern gehören Seperatoren wie Semikolon, Komma und gehäuftes Auftreten von Leerzeichen, sowie „trenn-anzeigende“ Wörter wie

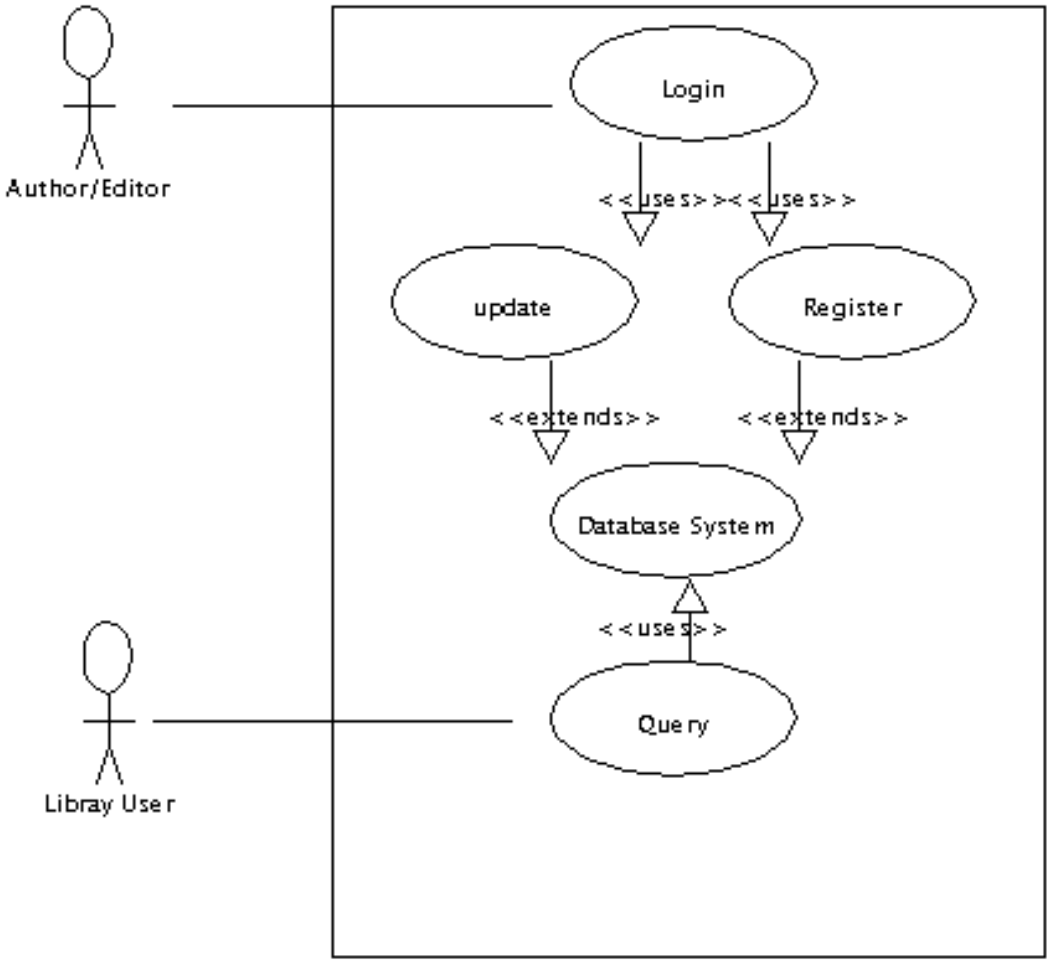


Abbildung 3.10: Datenbanksystem

„and“. Jeder dieser Strings wird dann weiter unterteilt, um eine Gliederung in Vor- und Nachnamen zu erwirken (`Person::parsename, author.cc`). Die Methode erfüllt die Funktion, Initialien zu vereinheitlichen (M Klink $\longmapsto$ M. Klink, Klink M.J.R. $\longmapsto$ Klink M. J. R.). Weiterhin werden Namenselemente, die in Klammern stehen, gelöscht (Markus Klink (editor) $\longmapsto$ Markus Klink) und letztendlich der gesamte Name in Vor- und Nachname unterteilt. Dabei wird bei Vorhandensein eines Kommas der Name dort aufgetrennt (unter Annahme einer Nachname, Vorname Struktur), ansonsten wird das letzte Wort als Nachname bewertet. Ist der Nachname zuguterletzt „viel“ kürzer als der Vorname, so wird unter der Annahme einer „Klink M“ Struktur die Wahl von Vorname und Nachname umgedreht. Der ganze Algorithmus wurde so entwickelt, daß er die in den

Templates auftretenden Fälle von Namensformatierungen möglichst gut berücksichtigt. Letztendlich müssen und sollen Archivadministratoren dazu angehalten werden, Daten korrekt zu erstellen.

Für jeden so gewonnenen Namen wurden die Informationen in den entsprechenden SQL-Tabellen gespeichert. Weiterhin wurde nach dem in Tabelle 3.5 beschriebenen Schema ein Personhandle für alle nicht registrierten Personen gebildet und die Verbindung zwischen Dokument und Person hergestellt.

Die Erkenntnisse aus der ungenügenden Vereinheitlichung der Namensfelder wurden dem RePEc-Team mit den Wunsch vorgetragen, die Dokumentation so zu verbessern, daß eine einheitliche Regel aufgestellt wird. Der Vorschlag lief auf eine Empfehlung zur Adaptierung einer „Nachname, Vorname“-Regel hinaus. Selbst so eine einfache Regel ist nicht unproblematisch: so gibt es Kulturkreise wie Vietnam in denen das Konzept eines Vor- und Nachnamen unbekannt ist. Es ist jedoch zu erwarten, daß der Großteil der Daten durch solch eine Daumenregel einheitlich erzeigt wird. Weiterhin wurden Archive identifiziert, die besonders viele Regelverstöße verursachten und deren Administratoren verständigt. So wurde die Zahl der Autorenfelder, die mehrere Autorennamen beinhalten von über 10.000 auf unter 1.000 gesenkt.

Objektpersistenz

Zum Laden und Speichern der Personen und Dokumente stehen Methoden zur Verfügung, die eine einheitliche Verwaltung der Persistenz von Objekten gewährleisten. Der Aufruf zum Speichern von Objekten geht stehts von einer Methode `store` der Klassen `Person` oder `Researchpaper` aus:

```
...
\\ Standarddatenbank öffnen
SQLdatabase db;
\\ Transientes Objekt erzeugen und ein paar
\\ Daten setzen
Person p;
p.lname("Klink");
p.fname("Markus");
p.archive("per");
p.handle("RePEc:per:1972-06-06:MARKUS_KLINK");
```

```
\\ Persistenz
p.store(db);
...
```

Umgekehrt können transiente Objekte aus der Datenbank über einen Konstruktoraufruf geladen werden. Dabei kann jeweils entschieden werden, ob alle mit den Objekt assoziierten Objekte ebenfalls geladen werden. Dies ist manchmal aus Performancegründen unerwünscht, wenn zum Beispiel für eine Liste von Personen nur die Namen, nicht aber die assoziierten Dokumente benötigt werden:

```
...
try{
   SQLdatabase db;
   // Person und assoziierte Dokumente laden
   Person *p=new Person(db,"RePEc:per:1972-06-06:MARKUS_KLINK",
                           YES_RESEARCHPAPERS);
   Person::const_role_iterator rolle(p.const_role_begin());

   // alle Dokumente über die Rollenassoziation durchlaufen
   // und den Titel ausgeben
   for (; rolle!=p.const_role_end(); ++role)
       cout << (*rolle)->getResearchpaper()->title()<<endl;
}
// Fehler abfangen:
// entweder die Person existiert gar nicht,
// oder es gibt Probleme mit der Datenbank
catch(const PersonError& err) { cerr << err.what<<endl;}
catch(const SQLerror& err) { cerr << err.what()<<endl; }
...
```

Für die Klassen `Person`, `Researchpaper` und die Datenbankkomponente stehen zur Fehlerbehandlung die Klassen `PersonError`, `ResearchpaperError` und `SQLerror` zur Verfügung, die jeweils über die Methode `what()` Informationen über die Art des Fehlers bereit stellen.

	absolut	**prozentual**
A. Anzahl Personeninformationen	114.247	100%
davon		
registriert	0	0,0%
mit (USER*)-Cluster	4487	3,9%
mit (WORKPLACE*)-Cluster	10.317	9,0%
B. Größe der Teilmenge von A mit identischen		
Nachnamen	19.332	16,9%
Vor- und Nachnamen	36.768	32,2%
Vor- und Nachnamen und Archiv	42.489	37,2%

Stand: 1. Juli 1999

Tabelle 3.9: Struktur der Personeninformationen

3.7.2 Suchfunktion

Ziel der Suchfunktion ist es, möglichst schnellen Zugriff über Personeninformationen zu bekommen, wenn man nur nach dem Namen sucht. Dabei sollen zusammengehörige Informationen möglichst auch an gleicher Stelle dargestellt werden. Eine Analyse des Datenbestandes nach den ersten Durchläufen der Dateneingabe ergab ein in Tabelle 3.9 dargestelltes Bild.

Die geringe Zahl der zusätzlichen Personeninformationen (email, homepage) und der Arbeitsplatzinformationen läßt den Schluß zu, daß eine Identifizierung der Autoren über diese Information zu keinem nennenswerten Ergebnis führt. Im besten anzunehmenden Fall, in dem alle (USER*)-Cluster und alle (WORKPLACE*)-Cluster einer einzigen Person zugeordnet werden könnten, würde die Zahl der Personen nur um ca. 15.000 abnehmen. Interessanter ist die Analyse der Personen mit identischen Namensstrukturen. Daraus lassen sich folgende Strategien ableiten (siehe zum Datenmaterial Tabelle 3.9):
1. Bei der Suche nach Personen ist nur der Nachname zu berücksichtigen. Dies führt zu einer großen Aggregation von Daten.
2. Selbst bei Berücksichtigung der Vornamen (wobei ein Initial und ein ausgeschriebener Vorname verschieden sind) ist eine große Aggregation der Daten möglich.
3. Wird die „Lokalität“ der Daten in Form der Archive auch berücksichtigt, so steigt die Datenmenge nicht wesentlich (von 32,2% auf 37,2% der Ursprungsdaten). Dies läßt den Schluß zu, daß nur wenige Personen mit unterschiedlichen Namen in mehreren Archiven vorkommen.

Mangels anderer Informationen für die Identifizierung einer Person wurde Strategie 3 adaptiert. Das heißt, bei der Suche nach einer nichtregistrierten Person werden die Ergebnisse so aggregiert, daß eine Person als „identisch“ gilt, wenn sie den gleichen Namen (Vor- und Nachnamen) hat sowie im gleichen Archiv lebt. Natürlich sind dabei Fehler möglich, denn innerhalb eines Archives können natürlich zwei Personen vorkommen, die den gleichen Namen haben, aber physikalisch (in der realen Welt) unterschiedlich sind. Es ist aber genau diese mögliche Fehlerquelle, die für Strategie 3 spricht, denn zum einen ist der Fehler „lokal“, das heißt, er tritt innerhalb eines Archives auf und ist von einem einzigen Administrator durch Vergabe eindeutigerer Namen zu verhindern, zum anderen wird ein Administrator, der von der Existenz solcher Personen weiß, von vornherein eindeutige Namen vergeben.

Die Realisierung einer Teilmenge aller Personen kann weiterhin mittels eines einzigen SQL-Kommandos vorgenommen werden (siehe `addpersons.cc`):

```
db.query("INSERT INTO author_link (lname,fname,archive,birth)
   SELECT DISTINCT lname, fname, file, birth FROM authors");
```

Die Hinzunahme des „Birth“-Feldes, dient der Identifizierung von registrierten Personen, die ja alle im Archiv „per“-Leben und ggf. auch gleiche Namen, aber unterschiedliche Geburtsdaten haben können. In der Spalte „file“ befindet sich der Name des Archivs. Die Tabelle `author_link` hat dabei den Charakter einer Hilfstabelle, die bei der Suche nach Namen den Zugriff auf die Informationen beschleunigt. Jede Suchanfrage schaut zunächst in der Tabelle `author_link` nach und dann in der Tabelle `authors`, um die Personen mit den gewünschten Eigenschaften zu finden. Über die Tabelle `authors` stehen dann auch die Informationen zur Verfügung, die ein Auffinden der assoziierten Dokumente ermöglichen. Die Erzeugung der eigentlich redundanten Informationen in `author_link` dient auch dem Zweck, die Strategie möglichst ohne großen Aufwand wechseln zu können, wenn dies erwünscht sein sollte.

Darstellung der Suchergebnisse

Der Benutzer kann auswählen, welche der Personeninformationen, die ihm getrennt nach Name, Vorname und Archiv präsentiert werden, er näher anschauen möchte. In der letztendlichen tabellarischen Aufbereitung werden dabei gegebenenfalls Informationen mehrerer Personen aggregiert. Dies geschieht genau in dem Fall, in dem mehrere Personen mit

gleichem Namen in demselben Archiv gespeichert sind. Man beachte, daß nicht registrierte Personen pro Dokumentassoziation einen künstlichen Handle erhalten und diese Informationen dann zusammengefaßt werden sollen, sofern die Informationen aus demselben Archiv stammen. Nach erfolgter Aggregation der Informationen werden diese abschließend präsentiert. Die Personeninformationen kann man sich dann abschließend im vCard Format zusenden lassen (vCard, 1996). Die Assoziationen der Person mit gespeicherten Dokumenten werden in einer separaten Tabelle dargestellt. Eine graphische Darstellung der Ergebnisseite findet sich in Abbildung 3.11.

3.7.3 Registrierungsfunktion

Die Registrierungsfunktion erfüllt die Aufgaben der Informationserfassung von Personeninformationen und der Assoziierung von Personen mit Dokumenten, die bereits im RePEc Datensatz vorliegen. Nicht zum Problembereich gehört hingegen die Aufnahme neuer Dokumente zum RePEc–Datensatz.

Eine Person wird dabei über eine Kombination von Nachnamen, Vornamen und Geburtsdatum eindeutig identifiziert. Dies (zusammen mit der email-Adresse) sind die Mindestangaben, die für eine Registrierung notwendig sind. Im Fall, daß zwei physikalisch unterschiedliche Personen diese gleichen Eigenschaften besitzen, werden sie als identisch betrachtet. Es wird nahegelegt, den Vornamen in einem solchen Fall durch Hinzufügen des Zweit- oder Drittnamens zu ergänzen. Auf diese Weise wird eine eigenständige Identität erreicht.

Nach Angabe der persönlichen Daten hat ein Benutzer die Möglichkeit, sich mit in der Datenbank gespeicherten Dokumenten zu assoziieren. Alternativ kann er sich wieder aus der Datenbank löschen. Der generelle Programmablauf ist in Abbildung 3.12 dargestellt.

Bei der Assoziation trifft der Registrierungsservice eine Vorauswahl von möglichen Dokumenten. Zum einen sind dies Dokumente, mit denen sich ein Benutzer in vorherigen Registrierungsdurchgängen bereits assoziiert hat, zum anderen werden Dokumente vorgeschlagen, die von Personen des gleichen Namens und mindestens gleichen Anfangsbuchstaben des Vornamens erstellt worden sind. Weiterhin besteht die Möglichkeit, sich mit beliebigen Dokumenten durch Eingabe des Templatehandles zu verbinden. Alle diese Assoziation werden temporär in der Tabelle `person_mapto_temp` gespeichert.

Neben der Assoziation mit den Dokumenten ist auch die Wahl der Rolle zwischen Person und Dokument notwendig. Zur Zeit ist dies eine injektive Abbildung: Ist

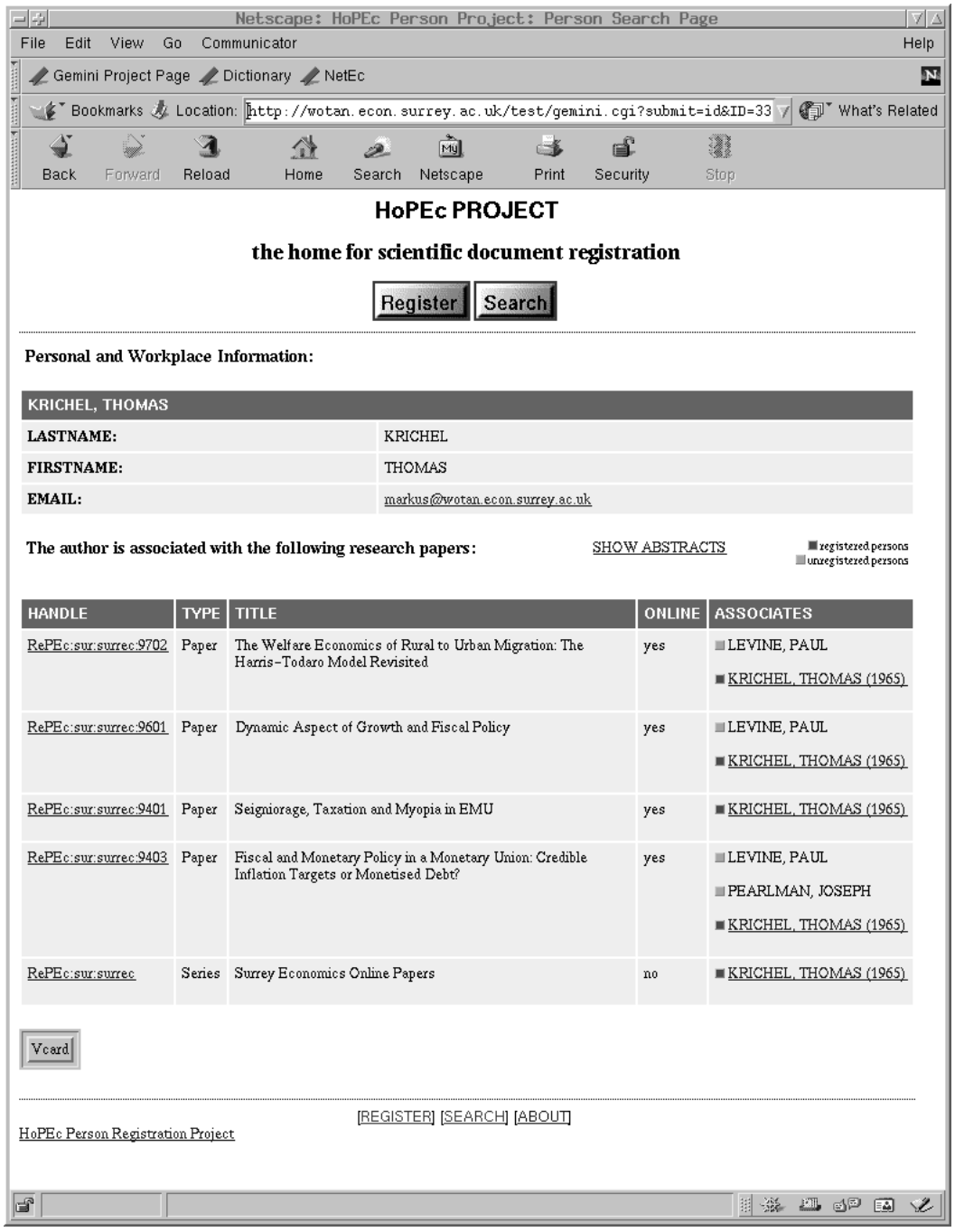


Abbildung 3.11: Bildschirm: gemini – Surchergebnis

der Dokumenttyp bekannt, so kann auch die Rolle bestimmt werden ({Article, Paper, Software}⟼{Author} und {Series}⟼{Editor}). Eine Umkehrung dieser Abbildungsfunktion ist aber nicht möglich. Für den Fall, daß neue Rollen für Personen in das ReDIF-Datenmodell aufgenommen werden, steht die Methode `string ResearchRole::-`

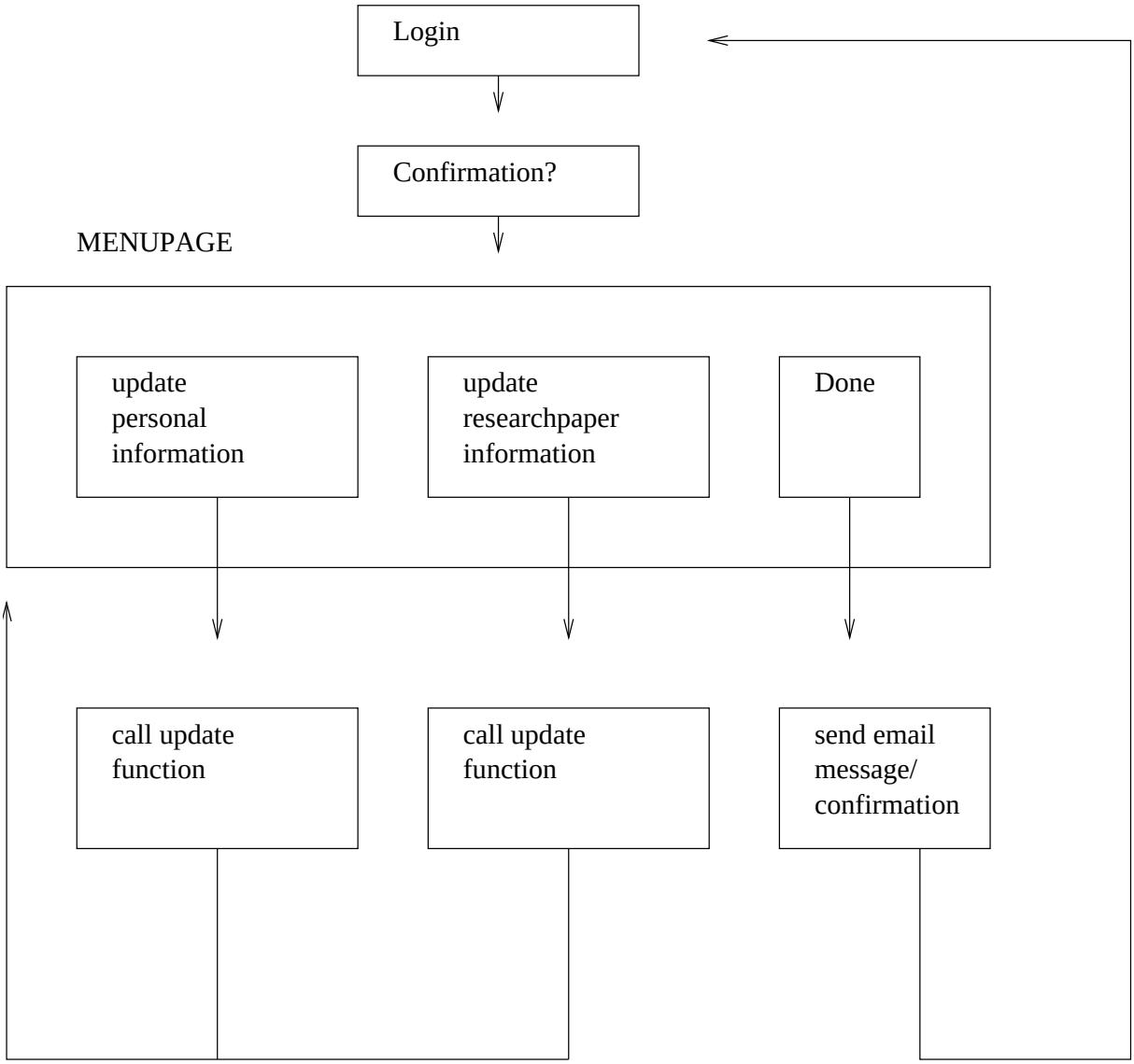


Abbildung 3.12: Vorgang einer Personenregistrierung

`possible_roles()` zur Verfügung, die Auskunft über mögliche Personenrollen für ein Dokument gibt.

Abbildung 3.13 zeigt den Vorgang der Dokumentauswahl. Auf der linken Seite befinden sich die Dokumente, mit denen ein Autor bereits assoziiert ist, rechts befinden sich die Dokumente, die vom System vorgeschlagen worden sind. Bei der Anzeige der bereits assozierten Dokumente wird auch die Information ausgegeben, welche Rolle die Person eingenommen hat. Links finden sich die Dokumente mit denen sich die Person noch assoziieren kann. Besonderes Augenmerk ist auf die letzte Zeile zu richten: Hier können

Abbildung 3.13: Bildschirm: Personenregistrierung

Personen sich mit Dokumenten verbinden, die der Registrierungsservice nicht als ihre Dokumente erkannt hat. Der Grund dafür könnte zum Beispiel in einer Namensänderung des Autors liegen. Durch Eingabe eines Handles kann man sich dennoch mit solchen Dokumenten assoziieren. Der Registrierungsservice macht dann deutlich, daß er von solch einer Assoziation nichts weiß (durch rechtsseitige Darstellung des Rollenbezeichners „unknown“).

Jede Änderung der Daten durch den Benutzer muß anschließend bestätigt werden. Dafür wird dem Benutzer vom Registrierungssystem per email eine 4-stellige Codenummer zugesandt, mittels derer er seine Änderungen bestätigen kann. Im Rahmen des Projektes wurde bewußt auf eine Lösung mit permanenten Passwörtern verzichtet, da diese in der Regel einen hohen administrativen Aufwand verursachen. Benutzer tendieren dazu, diese Passwörter zu vergessen und registrieren sich unter Umständen erneut, so daß Personen doppelt in der Datenbank gespeichert wären. Da aber trotzdem ein Mindestmaß an Schutz geboten werden soll, wurde die Kombination von Namen und Geburtsdaten gewählt, die zusätzlich über die email-Codes abgesichert wird.

3.8 Unterstützung für Metadatenstandards

Das Eingangs- und Ausgangsdatenformat für alle Dienste, die den RePEc Datensatz benutzen, ist zur Zeit ReDIF-ASCII. Unter ReDIF-ASCII wird dabei die Ausprägung der logischen Struktur von ReDIF-Datensätzen im Templateformat verstanden. Zur Zeit laufen Bestrebungen, die dahin gehen, ReDIF-ASCII durch eine XML Ausprägung zu ergänzen, wenn nicht sogar langfristig zu ersetzen.

Diese Umstellung kann mehrere Formen haben, über deren Verwirklichung noch abgestimmt werden muß. Zum einen kann ReDIF-XML schon auf Benutzerseite (Benutzer im Sinne von Archivadministratoren) verwendet werden. Die Daten würden dann in XML auf die Zielserver transferiert. Zum anderen könnten Benutzerdaten erst auf Dienstseite in XML konvertiert werden.

Beide Verfahren haben als gemeinsamen Vorteil, daß für XML (im Gegensatz zu ReDIF) Standardsoftware zur Verfügung steht, die Dokumente auf eine gültige Struktur testen kann. Eine Eigenentwicklung solcher Software, wie sie mit dem Perl-Modul `rr.pm` vorliegt, kann also entfallen. Für eine Entscheidung der Unterstützung von XML auf Benutzerseite spricht also, daß Benutzer ihre Daten einfach auf Korrektheit überprüfen können. Dagegen spricht die notwendige Konvertierung der Altdaten und die höhere Komplexität von XML. Ein wichtiger Grund für die rasche Akzeptanz eines Datenformates wie ReDIF ist die Einfachheit der Datenbeschreibung. Oftmals müssen die Archive von Universitätssekretärinnen betreut werden, die von Detailwissen nicht behelligt werden können. In Anhang A wird eine exemplarische Beschreibung der ReDIF-Struktur in XML dargestellt.

Diese Struktur orientiert sich dabei an der vom Guildfordprotokoll vorgegebenen Filestruktur für ReDIF-Daten. Insbesondere wird durch die Elementdefinition des Rootelements `redif` dafür gesorgt, daß innerhalb eines gültigen Dokumentes nur Templates eines Typs (also nur `archive`, `series`, `paper`, ...) vorkommen dürfen. Implizit wird damit also unterstellt, daß es wieder drei Dateien geben wird, die Dokumente in der Struktur, wie in Abbildung 3.3 gezeigt, beschreiben. Problematisch ist an dieser Struktur, daß sie dem Wesen von XML nicht direkt entspricht. So würde man in der XML eigentlich ausdrücken wollen, daß Serien Bestandteile eines Archives sind, also etwa in der Form:

```
<archive handle="RePEc:xrs">
   <name>Sonderforschungsbereich 504 Workingpaper Archive</name>
   <url href="http://my.archive.edu/RePec/xrs" />
   <maintainer>
      <email>markus@wotan.econ.surrey.ac.uk</email>
   </maintainer>
   <series handle="RePEc:xrs:sfbmaa">
      <name>Financial Market Workingpapers</name>
   </series>
</archive>
```

Diese grundsätzlichen Überlegungen haben bisher den Einsatz von XML erschwert. Weiterhin ist die Rolle des Resource Description Frameworks noch nicht abschließend behandelt. Die noch in der Entwicklung befindlichen Standards müssen zunächst noch näher geprüft werden, damit entschieden werden kann, wo eigene Entwicklungen notwendig sind und inwiefern Elemente bestehender Standards mittels des RDF vereint werden können, um die vorhandenen Informationen effizient auszunutzen. Ein bisher implementierter Prototyp eines RDF-Schemas definiert zwar Klassen für die verschiedenen Templatetypen mit den dazugehörigen Eigenschaften, allerdings werden fremde Datenstandards dabei nicht berücksichtigt. Es ist fraglich, inwiefern eine solche Insellösung den Zielen von RePEc dient und somit erwünscht ist.

Für Personentemplates existieren bereits einige Testimplementierungen. Als Ausgabeformat wurde alternativ eine reine XML-Darstellung der Personendaten und eine Darstellung in RDF mit Dublin Core Bezeichnern gewählt. Aus den XML-Dateien wird dann noch per XSL-Stylesheets ein HTML File erzeugt. Das externe Programm `generate_xml` generiert dabei aus den in der Datenbank gespeicherten Informationen jeweils neue XML-Dateien. Das Programm kann so konfiguriert werden, daß es jeweils die neueste Version einer DTD

berücksichtigt. Als momentane DTD wird die DTD in Anhang A benutzt. `generate_xml` benutzt zum Generieren der XML-Dateien die Klasse `PersonOutputXML`, welche angepaßt werden muß, wenn sich das XML-Ausgabeformat ändert. Dieselbe Methode wird auch von der Registrierungsdatei verwendet, um neue XML-Dateien zu erzeugen, wenn sich eine Person neu im System registriert.

Weiterhin werden auf den dynamisch generierten HTML-Seiten der Suchfunktion RDF Daten eingebettet. Diese sind in der Klasse `PersonOutputHTMLDubCore`, welche wie `PersonOutputXML` Teil der Ausgabekomponente ist, generiert.

Im folgenden ist eine Ausgabe im RDF-Format dargestellt. Das Format von RDF ist dabei geringfügig anders, als es in Kapitel 2.3.3 dargestellt wurde. Es handelt sich hier um die sogenannte „abbreviated syntax“, die notwendig ist, um die RDF-Informationen vor Webbrowsern zu verstecken, da diese unter Umständen Informationen in Tags darstellen, die sie nicht kennen. Da in der „abbreviated syntax“ aber jedes Elementtag leer ist, weil alle Informationen in Form von Attributen „versteckt“ sind, kann auf diese Art und Weise RDF in ein HTML Dokument eingefügt werden.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:DC="http://purl.org/dc/elements/1.0/"
xmlns:HOPEC="http://wotan.econ.surrey.ac.uk/~markus/hopec-schema#"
xmlns:rdfs="www.w3.org/TR/1999/PR-rdf-schema-19990303#">

<rdf:Description about="http://netec.mcc.ac.uk/BibEc/data/Papers/
                                xrssfbmaa99-28.html" bagID="bag_0"
        DC:Title="Tax incentives, bequest motives and the demand
                  for life insurance: evidence from Germany">
<DC:Creator>
<rdf:Bag
        rdf:_0="WALLISER, JAN"
        rdf:_1="WINTER, JOACHIM"
/>
</DC:Creator>
</rdf:Description>
<rdf:Description aboutEach="#bag_0"
        HOPEC:attributedto="Hopec Person Registration Project">
```

```
        <rdf:type resource="http://www.w3.org/
                                    1999/02/22-rdf-syntax-ns#Statement" />
</rdf:Description>
</rdf:RDF>
```

3.9 Ausblick

3.9.1 Multiautoritäten

Unter einer Autorität wird im Sinne von RePEc ein Namensraum verstanden, der in der Regel Metadaten einer bestimmten wissenschaftlichen Disziplin von anderen Autoritäten abgrenzt. Zur Zeit existiert als einzige Disziplin eine Sammlung wirtschaftswissenschaftlicher Arbeitspapiere. ReDIF als Datenformat stößt aber im wissenschaftlichen Bereich auf zunehmend größeres Interesse. So arbeitet das „xxx"-Archiv (ein zentralisiertes Archiv für Physik, Mathematik und Informatik) zur Zeit an einer Konvertierung des eigenen proprietären Datenformats nach ReDIF. Die Datensätzes (ca. 100.000 Arbeitspapiere) müssen dann noch neu zu schaffenden Autoritäten zugewiesen werden. Ebenso stellt das CogPrints-Archiv seine Daten in ReDIF zur Verfügung.

Die Programme zur Registrierung und Suche von Autoren wurden so gestaltet, daß eine Anwendung auf andere Autoritäten problemlos verwirklicht werden kann. Unter Umständen ändert sich allerdings etwas an dem Format der Institutionen, so daß hier vielleicht weitere Entwicklungsarbeit notwendig ist. Es muß dabei noch geklärt werden, ob der Begriff der Institution in der Verantwortung einer Autorität liegt oder ob Institutionen durch eine übergeordnete Schicht verwaltet werden müssen. Der Grund dafür liegt in der Art der Daten von Institutionen, die zumindest auf oberster Ebene („Universität Mannheim") disziplin-unabhängig sind. Erst auf zweiter Ebene kommen in der Regel fachspezifische Abteilungen ins Spiel (Fakultät für Volkswirtschaftslehre, Fakultät für Mathematik und Informatik etc.). Leider können Institutionen aber noch nicht in einer Baumstruktur dargestellt werden, sondern organisieren sich in einer Form von Listen, wobei das übergeordnete Element stets neu angegeben werden muß. Der unterschiedliche Sachverhalt ist in Abbildung 3.14 abgebildet. Um eine Baumstruktur zu verwirklichen, müßten die einzelnen Cluster, die eine Institution bilden, über Handle auf die nächsthöhere Ebene verweisen können. Zur Zeit nehmen die Programme zur Suche und Registrierung

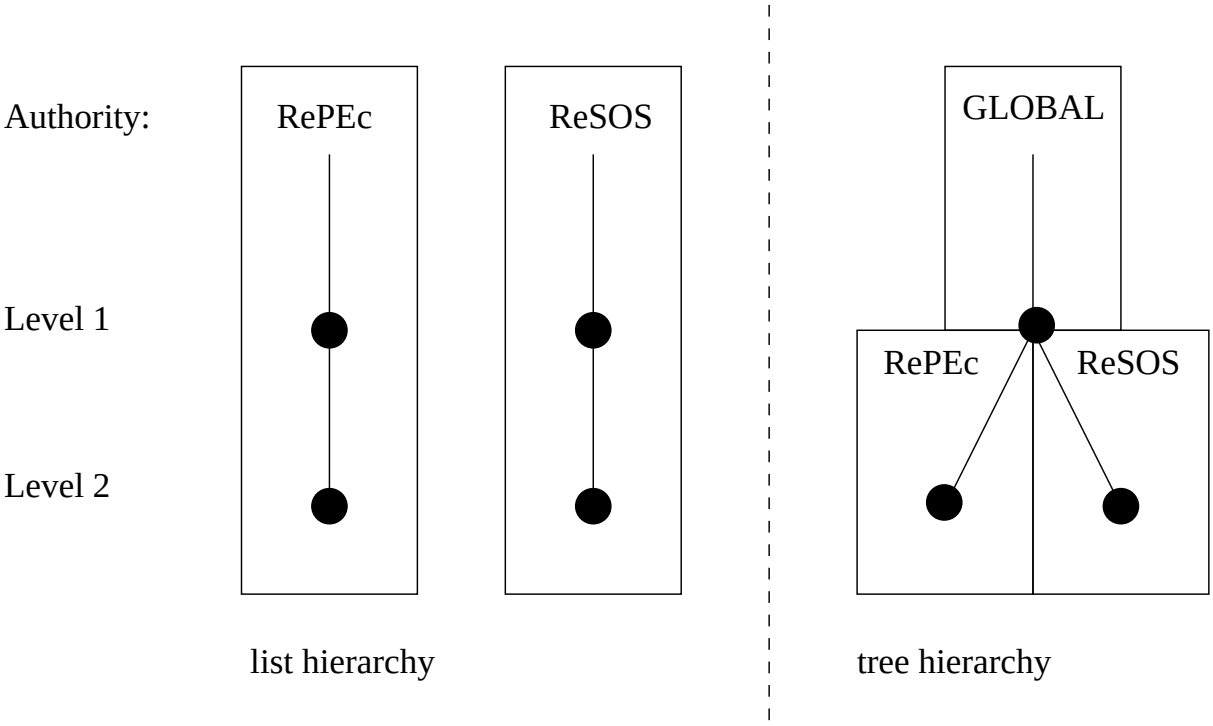


Abbildung 3.14: Listen-hierarchische und baum-hierachische Institutionsstruktur

noch an, daß es sich bei Institutionen um lokale, autoritätsspezifische Informationen handelt, die als Liste hierarisch geordnet sind.

3.9.2 Authority Control

Im bibliothekarischen Sinne versteht man unter dem Begriff „Authority Control“ die Verwendung und Verwaltung eines kontrollierten Vokabulars. So sind in der Regel die Sachansetzungsformen eines CIP-Eintrages mittels eines kontrollierten Vokabulars zu erstellen. Dies ist extrem wichtig, da ansonsten eine funktionierende Katalogsfunktion nicht gewährleistet werden kann.

In RePEc gibt es bisher keine Authority Control, da der Verwaltungsaufwand dafür zu groß ist und die Einhaltung eines kontrollierten Vokabulars auf Seiten der Archivadministratoren nicht durchsetzbar ist. Andererseits zeigen viele Benutzeranfragen, daß eine bessere Katalogsfunktion dringend notwendig ist. Zur Zeit könnte diese im Rahmen der wirtschaftswissenschaftlichen Daten über eine JEL-Klassifizierung verwirklicht werden. Es haben aber nur ca. 31% der Dokumente eine entsprechende Klassifizierung (siehe Tabelle 3.4). Das nachträgliche Hinzufügen ist nur über die Archivadministratoren möglich,

POLICY	MODEL	ECONOMIC	EVIDENCE	ANALYSIS	MONETARY	TRADE	GROWTH	MODELS
3471	3076	2891	2601	2325	2225	2111	2076	2009
Cluster:					MONETARY	TRADE		
					MONETAIRE	RATE		
					MONETARIA	TRADERS		
					MONETARIO	RACE		
					MENETARY	TALE		
					MOMENTARY	TRAP		
					MONETRAY	TRADES		
					MONTARY	...		

Tabelle 3.10: Die häufigsten Wörter in Dokumenttiteln

so daß alternative Wege gefunden werden müssen, einen brauchbaren Katalog der Daten zu erstellen. Ein gangbarer Weg, besteht in der Erstellung eines automatischen Registers (French et al., 1997b; French et al., 1997a). Dieses Verfahren arbeitet mit einem ungefähren Zeichenkettenvergleich innerhalb eines Datensatz. Dabei wird ein zu bearbeitender Datensatz so sortiert, daß Einträge nach Häufigkeit sortiert in einer Tabelle stehen. Im RePEc Datensatz würde sich zum Erstellen eines Registers der Dokumenttitel anbieten. Tabelle 3.10 zeigt die 10 häufigsten Wörter in Dokumententiteln. Dabei sind sogenannte Füllwörter wie „the“, „a“, „der, die, das“ schon herausgefiltert worden.

Offensichtlich ist ein Benutzer, der an „Model“ interessiert ist, auch an „Models“ interessiert. Der von French et al. vorgeschlagene Algorithmus geht davon aus, daß es sich bei dem häufigsten Wort um das übergeordnete Wort handelt. Andere Wörter wie „MODELS“ sind entweder davon abgeleitete Formen oder auch einfach Tippfehler. Für das häufigste Wort wird dann der Abstand zu allen anderen Wörtern in der Tabelle berechnet. Liegt der Abstand unter einem bestimmten Schwellenwert, so kann man davon ausgehen, daß die Wörter im Register zusammengehören. Sie werden aus der Häufigkeitstabelle entfernt und in einen „Cluster“ abgelegt. Der Algorithmus fährt fort, bis die Tabelle entweder leer ist oder kein neuer Cluster gebildet werden kann. Als Maß des Abstands zweier Wörter hat French die sogenannte „editing distance“ verwendet (Wagner and Fischer, 1974). Die „Editing Distance“ gibt dabei die Zahl der benötigten Einfüge-, Lösch- oder Änderungsoperationen an, um Zeichenkette1 in Zeichenkette2 zu überführen. Demgemäß hätten die Zeichenketten „MODEL“ und „MODELS“ den Abstand 1 (eine Löschoperation). In der Regel wird der Algorithmus zur Bestimmung der „edit distance“ so implementiert, daß Einfüge- und Löschoperationen sowie Änderungsoperationen unterschiedliche Kosten verursachen, wobei sinnvollerweise gelten soll, daß eine Änderungsoperation mehr kostet, als eine Lösch- oder Einfügeoperation, aber weniger als zwei Lösch- oder Änderungsoperationen. Zum Beispiel kann eine Änderungsoperation 15 Kosteneinheiten (KE) verursachen und Lösch- oder Einfügeoperation 10 Kosteneinheiten ($KE_{Loeschen} = KE_{Einfuegen} < KE_{Aendern} <$

($KE_{Loeschen} + KE_{Einfuegen}$), da sich sonst jede Änderungsoperation durch eine kombinierte Einfüge- und Löschoperation ersetzen läßt.

Bei French, der Zeichenketten zur Beschreibung von Orten untersucht hat, war die Schwelle des Abstands zwischen zwei Zeichenketten fix. Die Schwelle ist dabei der Abstand, der unterschritten werden muß, damit zwei Zeichenketten im gleichen Cluster enden. Dies ist bei entsprechend langen Zeichenketten ein gangbarer Weg. Eine erste Testuntersuchung anhand der Wörter in Titelbeschreibungen brachte dabei allerdings unterschiedliche Ergebnisse. Wie in Tabelle 3.10 zu sehen ist, ist der gebildete Cluster für das Wort „MONETARY“ durchaus zufriedenstellend, wohingegen der Cluster des kürzeren Wortes „TRADE“ Begriffe enthält, die gar nichts mit den Wort zu tun haben. Daraus folgt, daß der Schwellenwert in Abhängigkeit der Wortlänge stehen muß, wobei der Schwellenwert umso kleiner ist, je kürzer das zu untersuchende Wort ist.

Eine Katalogfunktion kann mittels der erstellten Cluster dann wie folgt realisiert werden: Ein Algorithmus erstellt die Cluster anhand der primären Wörter der Titelangaben. Dabei wird abgespeichert, welches Wort in welchem Dokument vorkommt, und es wird eine Struktur erzeugt, die es erlaubt jedes Wort einem Cluster zuzuweisen. Suchanfragen werden dann so behandelt, daß zunächst für jedes Wort der Suchanfrage geschaut wird, in welchem Cluster es sich befindet. Die Suche erfolgt dann über alle Worte der Cluster, wobei Wörter innerhalb eines Clusters mittels einer “oder“-Verknüpfung gesucht werden und zwischen den Clustern mittels einer „und“-Verknüpfung. Die Ergebnismenge enthält dann Verweise auf alle Dokumente, die zumindest ein Wort eines jeden Clusters enthalten.

Ein Prototyp, der wiederum die in der Designphase erstellten Klassen verwendet, wurde bereits implementiert. Als einzige Neuentwicklung mußte der Algorithmus zum Berechnen der „editing distance“ erstellt werden (`edit_distance.cc`. Die Suchergebnisse sind schon recht gut, allerdings müssen noch Optimierungsarbeiten an den SQL-Anfragen vorgenommen werden, um das Laufzeitverhalten zu verbessern. Als besonderes Merkmal der Suchmaschine ist hervorzuheben, daß der Benutzer Feedback über die verwendeten Suchbegriffe erhält und die Suche interaktiv weiter eingrenzen kann, wenn die Ergebnismenge zu umfangreich war. In Abbildung 3.15 wird ein Screenshot der Applikation dargestellt. Im unteren (nicht sichtbaren) Teil, werden die Ergebnisse noch tabellarisch zusammengefaßt. Als Ausgabefunktion wird hier wiederum der Mechanismus verwendet, der bereits vorgestellt vorgestellt worden ist.

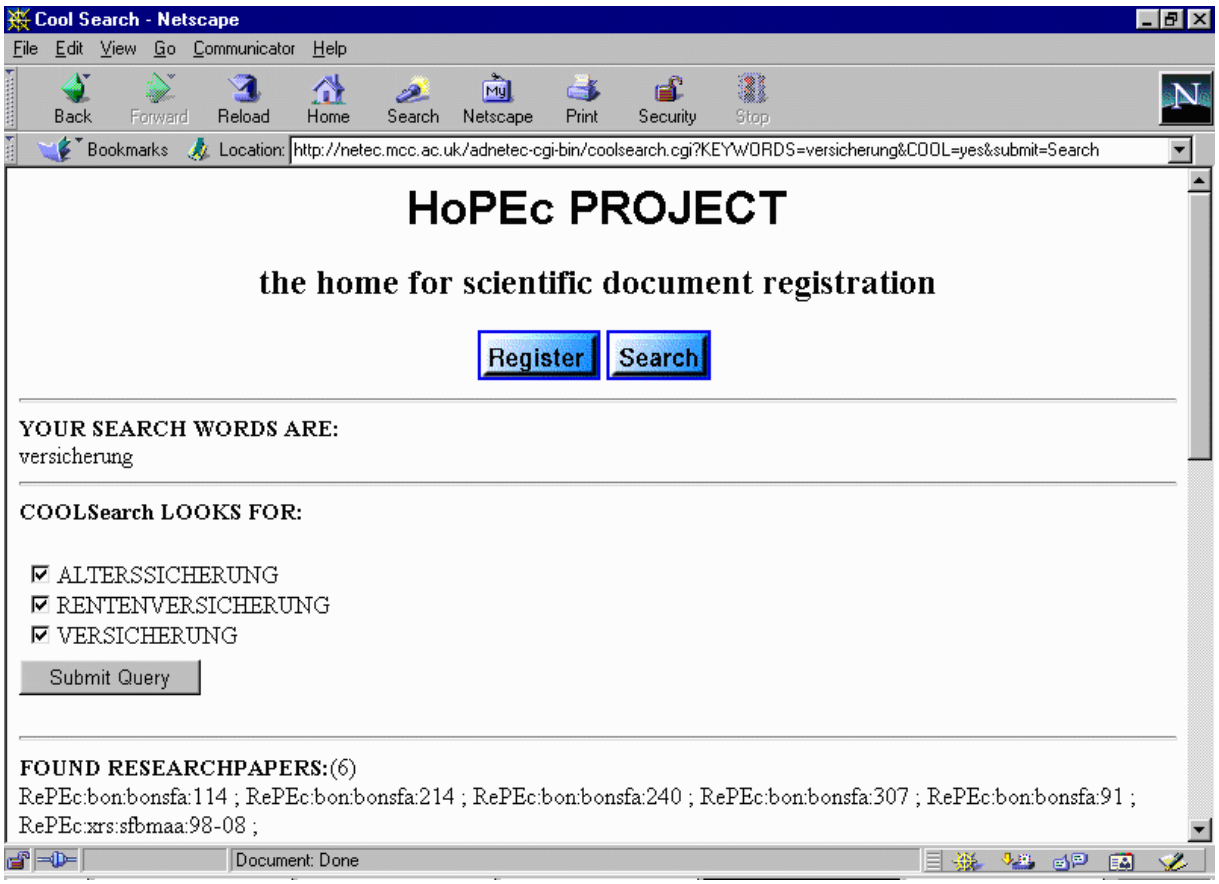


Abbildung 3.15: Bildschirm: coolsearch – Surchergebnisse

3.9.3 Duplettenerkennung

Die Art und Weise, wie das Katalogproblem angegangen wird, um ein möglichst kontrolliertes Vokabular zu erhalten, ist auch hilfreich, um Dupletten zu erkennen. Dupletten sind Dokumente, die im Datensatz mehrfach beschrieben werden. Duplettenerkennung ist insbesondere dann sehr wichtig, wenn die Datenerfassung dezentralisiert ist oder wenn mehrere Datenbestände zusammengefaßt werden müssen. Letzteres Problem tritt in Zukunft zunehmend im Bereich der Multiautoritäten innerhalb von RePEc auf.

Im Bibliographiewesen ist man sich des Problems der Duplettenerkennung schon lange bewußt. Nicht immer können dabei Dupletten anhand eines eindeutigen Identifizierers (z.B. der ISBN-Nummer) erkannt werden. Eine gängige Methode, dennoch doppelte Einträge zu finden, besteht in der Generierung eines künstlichen Identifizierers, der aus ausgewählten Wörtern der Titel und Autoren besteht. Besitzen zwei Dokumente den gleichen Identifizierer, so kann von einer großen Übereinstimmung ausgegangen werden und das Dokumentpaar an einen Bearbeiter weitergeleitet werden.

Die Wahl der Wörter, die aus dem Titel extrahiert werden, kann dabei natürlich wiederum die Struktur der autorisierten Wörter verwenden. Weiterhin können die Namen aus den Tabellen zur Personenregistrierung entnommen werden, weil sie dort schon in recht vereinheitlichter Form vorliegen. Um Dupletten erfolgreich zu erkennen wäre folgende Strategie denkbar: Zunächst werden die Titel aus der Datenbank ausgelesen und die Worte alphabetisch sortiert. Worte, die als Füllwörter bekannt sind („der„die“„das“ ...) werden ignoriert. Aus den Anfangsbuchstaben der Titelwörter, sowie der der Autoren (ebenfalls alphabetisch sortiert) können dann Schlüssel generiert werden. Eine Übereinstimmung der Dokumente liegt dann vor, wenn eine bestimmte „Edit Distance“ der Schlüssel nicht überschritten wird. Aus diesem Grund ist es auch ersichtlich, warum eine Sortierung der Titel und Autoren notwendig ist, da so vermieden wird, daß eine Änderungsoperation (billig) durch eine Lösch- und eine Einfügeoperation (teuer) ersetzt wird. Ähnliche Schlüssel bleiben bei sortierten Einträgen auch für den Algorithmus der „edit distance“-Bestimmung ähnlich.

3.10 Zusammenfassung

In den Kapiteln 3.8 und 3.9 wurde schon angesprochen, in welche Richtung sich das Projekt RePEc weiter entwickeln wird. Sobald der zu Entwicklungsaufwand sinkt, weil

standardisierte Klassenbibliotheken zur Verwendung von XML und RDF zur Verfügung stehen, werden diese Technologien auch eingesetzt werden. Durch die Diplomarbeit konnten in diesem Gebiet schon erste Erfahrungen gesammelt werden. Zum einen werden bereits XML- und RDF- Dateien im Bereich der Personenidentifizierung generiert, zum anderen wurde erkannt, daß durch den Einsatz von XML viele Eigenentwicklungen durch standardisierte Verfahren abgelöst werden können. Dies betrifft in der Regel die Verwaltung von Konfigurationsdateien, wie sie im Projekt notwendig sind.

Die Entwicklung der Standards von Metadaten wird weiter verfolgt und beobachtet. Insbsondere die weitere Entwicklung des qualifizierten Dublin Core steht noch aus: um tatsächlich qualitativ hochwertige Daten beschreiben zu können, die auch austauschbar sind, wäre diese Entwicklung wünschenswert.

Die Umsetzung der Programme zur Personenregistrierung wurden durchgeführt und werden zukünftig eingesetzt. Dafür ist in nächster Zukunft noch eine Marketingkampagne durchzuführen, um den Bekanntheitsgrad des neuen Dienstes zu erhöhen. Der Dienst ist dabei voll kompatibel zu den alten Datenstrukturen. Mittels der Personentemplates gelingt es allen an RePEc teilhabenden Diensten, diese Information auszunutzen. Weit wichtiger ist allerdings, daß mit der Einführung der relationalen Strukturen alle Dienste einen besseren Service anbieten können, weil die Information insgesamt besser gepflegt und ausgenutzt werden kann. Dies betrifft insbesondere die bessere Ausnutzung der bisher isolierten Institutionsdaten.

Nächste Pläne laufen darauf hinaus, die Personenregistrierung mit der Schaffung eines eindeutigen Archives zu kombinieren, so daß auch individuelle Personen ihre Daten zur Verfügung stellen können. Zur Zeit steht solch ein Service nur Institutionen zur Verfügung, die als eindeutigen Code einen Archivhandle von der Bibliotheksadministration erhalten. Bei computergestützter Erzeugung von Personenhandles, wie dies durch den implementierten Registrierungsdienst geschieht, ist es möglich, dies zu automatisieren. Dadurch werden Wissenschaftler ermutigt, ihre Publikationsdaten der Bibliothek zur Verfügung zu stellen. Mit zunehmender Datenmenge und vor allem -qualität steigt die Attraktivität des Dienstes, so daß sich elektronische preprints tatsächlich zu einer Alternative zum Zeitschriftenwesen entwickeln können.

Literaturverzeichnis

American Library Association, editor (1908). *Cataloging Rules: author and title entries*. American Library Association.

American Library Association, editor (1967). *Anglo–American Cataloging Rules*. American Library Association.

Bray, T., Paoli, J., and Sperberg-McQueen, C. M., editors (1998). *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium. `http://www.w3.org/REC-xml`.

Brickley, D. and Guha, R. V., editors (1999). *Resource Description Framework (RDF) Schema Specification*. World Wide Web Consortium. `http://www.w3.org/TR/PR-rdf-schema/`.

Clark, J., editor (1999). *XSL Transformations (XSLT) Specification*. W3C. `http://www.w3.org/TR/WD-xslt`.

DatakonsultAB, T. (1999). mysql. `http://www.tcx.se`.

Deach, S., editor (1999). *Extensible Stylesheet Language (XSL) Specification*. W3C. `http://www.w3.org/TR/WD-xsl/`.

Deutsch, P., Emtage, A., Koster, M., and Stumpf, M. (1995). Publishing Information on the Internet with Anonymous FTP (IAFA–templates). http://www.roads.lut.ac.uk/System-docs/Internet-drafts/draft-ietf-iiir-publishing-03.txt.

Deutsches Bibliotheksinstitut, editor (1993). *Regeln für die alphabetische Katalogisierung*. Kommission des Deutschen Bibliotheksinstituts für Erschliessung und Katalogmanagement, Berlin.

Deutsches Bibliotheksinstitut, editor (1999). *Deutsche Bibliotheksstatistik*. Deutsches Bibliotheksinstitut, Berlin.

Dublin Core (1998a). *List of Resource Types*. `http://purl.org/DC/documents/working_drafts/wd-typelist.htm`.

Dublin Core (1998b). *Subelement Working Draft*. Dublin Core. http://purl.oclc.org/dc/documents/working_drafts/wd-subelements-current.htm.

Dublin Core (1998c). *User Guide Working Draft*. http://purl.oclc.org/dc/documents/working_drafts/wd-guide-current-version.htm.

Dublin Core (1999a). *Dublin Core Metadata Iniative, Sponsors*. `http://purl.org/DC/sponsors/index.html`.

Dublin Core (1999b). *Dublin Core Metadata Initiative*. `http://purl.oclc.org/dc/`.

French, J. C., Powell, A. L., and Schulman, E. (1997a). Applications of approximate word matching in information retrieval. Technical report, University of Virginia.

French, J. C., Powell, A. L., and Schulman, E. (1997b). Automating the construction of authority files in digital libraries: A case study. Technical report, University of Virginia.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gorman, M. and Winkler, P. W., editors (1998). *Anglo-American Cataloguing Rules*. Joint Steering Committee for Revision of AACR.

Heery, R. (1996). Review of Metadata Formats. *Program*, 30(4). preview available at `http://www.ukoln.ac.uk/metadata/review.html`.

Ianella, R. and Campbell, D. (1998). Admin Core - Administrative Container Metadata. `http://metadata.net/admin/draft-iannella-admin-01.txt`.

International Organization for Standardization (1988). *ISO 8601 – Date and time representation*. International Organization for Standardization.

Krichel, T. (1997). Guildford protocol. `http://netec.mcc.ac.uk/RePEc/GuilP.html`.

Krichel, T. and Cruz, J. M. B. (1997). ReDIF, Research Documents Information Format. `http://netec.mcc.ac.uk/RePEc/ReDIF.html`.

Kunze, J. (1999). Enconding Dublin Core metadata in HTML. `ftp://ftp.ief.org/internet-drafts/draft-kunze-dchtml-01.txt`.

Lassila, O. and Swick, R. R., editors (1999). *Resource Description Framework (RDF) Model and Syntax Specification*. World Wide Web Consortium. `http://www.w3.org/TR/REC-rdf-syntax/`.

Library of Congress (1993). Overview of the CIP–Program. `gopher://marvel.loc.gov:70/00/services/publishers/cip/cipover`.

Martin, S. (1998). Cgi.h. `http://www.marketrends.net/C++/`.

Miller, E., Miller, P., and Brickley, D., editors (1999). *Guidance on expressing the Dublin Core within the Resource Description Framework (RDF)*. Dublin Core. `http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf/`.

Olson, N. B., editor (1997). *Cataloging Internet Resources: a manual and practical guide*. OCLC, 2nd edition. `http://www.purl.org/oclc/cataloging-internet/`.

Pierre, M. S. and LaPlant, W. P. (1998). *Issues in Crosswalking*. National Information Standards Organization.

Piggot, M. (1990). *The cataloguer's way through AACR2: from document receipt to document retrieval*. Library Asscociation Publishing Ltd.

Thomas, C. F. and Griffin, L. S. (1998). Who will create metadata for the internet? *First Monday*. `http://www.firstmonday.dk/issues/issue3_12/thomas/index.html`.

vCard (1996). *vCard - the Electronic Business Card*. Versit Consortium. `http://www.imc.org/pdi/vcard-21.txt`.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173.

Anhang A

Document Type Definition: redif.dtd

```
<!-- redif.dtd BETA 02 June 1999
     Markus Klink
     markus@wotan.econ.surrey.ac.uk

     XML multiplicity cheat sheet
     *     0..n occurences
     ?     0..1
     +     1..n
     none  1

CHANGES:
11 June 1999: added author-software ELEMENT
              added file ELEMENT

21 June 1999: added article template ELEMENT

01 July 1999: improved XLINK structure (author-paper, author-article...)

03 July 1999: improved person structure (introduced person-info)
-->

<!-- One file consists only of one template type -->
<!ELEMENT redif (person*|archive*|series*|paper*|article*|
                 software*|institution*)>

<!-- PERSON TEMPLATE -->
<!ELEMENT person (name, name-first?, name-last?,
                  person-info?, workplace*,
  associations?)>
```

```
<!ATTLIST person
     version (1.0) "1.0"
     handle ID #REQUIRED>

<!-- ARCHIVE TEMPLATE -->
<!ELEMENT archive (name, url, homepage?, description?,
                   maintainer, classification?,
   notification?, accesspolicy?, restriction?)>

<!ATTLIST archive
          handle ID #REQUIRED>

<!-- SERIES TEMPLATE -->
<!ELEMENT series (name, description?, classification?,
                  keywords?, editor?, provider?, publisher?,
  notification?, issn?, maintainer, order?,
  price?, restriction?)>

<!ATTLIST series
          handle ID #REQUIRED
  type (ReDIF-Paper|ReDIF-Software|ReDIF-Article)
        "ReDIF-Paper">

<!-- MIRROR TEMPLATE -->
<!ELEMENT mirror (directory, description?,
                  maintainer, machine, archivesincluded*,
  archivesexluded*, seriesincluded*,
  seriesexcluded*)>

<!ATTLIST mirror
          handle ID #REQUIRED
  redifonly (yes|no) "no"
  user (adnetec) "adnetec"
  group (nwk) "nwk">


<!-- ARTICLE TEMPLATE -->
<!ELEMENT article (author+, contactemail?, title, abstract?,
                 classification?, keywords?, note?,
 number?,  creationdate?,
 publicationstatus?,
 notification?, restriction?, file*,
 journal?, year?, pages?, volume?, month?)>
```

```
<!ATTLIST article
          handle ID #REQUIRED>

<!-- BOOK TEMPLATE -->

<!-- SOFTWARE TEMPLATE -->
<!ELEMENT software (title, programming-language,
                    author+, abstract?, number?,
    keywords?, size?, creationdate?,
    revisiondate+, note?, requires?,file*)>

<!ATTLIST software
          handle ID #REQUIRED>

<!-- INSTITUTION TEMPLATE -->
<!ELEMENT institution (primary, secondary?, tertiary?)>
<!ELEMENT primary (workplace)>
<!ELEMENT secondary (workplace)>
<!ELEMENT tertiary (workplace)>

<!ATTLIST institution
          handle ID #REQUIRED>

<!-- PAPER TEMPLATE -->
<!ELEMENT paper (author+, contactemail?, title, abstract?,
                 classification?, keywords?, note?,
 length?, serie?, number?,
 availability?, creationdate?,
 revisiondate+, price?, publicationstatus?,
 notification?, restriction?, file*)>

<!ATTLIST paper
     version (1.0) "1.0"
     handle ID #REQUIRED>

<!-- CLUSTER START HERE -->

<!-- MAINTAINER CLUSTER: archive, series -->
<!ELEMENT maintainer (email, phone?, fax?, name?)>

<!-- AUTHOR CLUSTER: paper, article, software  -->
<!ELEMENT author (name, email?, fax?, phone?, postal?,
                  homepage?, workplace*)>
```

```
<!-- EDITOR CLUSTER: series-->
<!ELEMENT editor (name, email?, fax?, phone?, postal?,
                  homepage?, workplace*)>

<!-- PUBLISHER CLUSTER: series  -->
<!ELEMENT publisher (name, email?, fax?, phone?, postal?,
                  homepage?, workplace*)>

<!-- WORKPLACE CLUSTER -->
<!ELEMENT workplace (name, email?, phone?, fax?, postal?,
                     homepage?)>

<!-- PERSON-INFO CLUSTER -->
<!ELEMENT person-info (name, email?, phone?, fax?, postal?,
                     homepage?)>

<!-- PERSON-ASSOCIATIONS CLUSTER -->
<!ELEMENT associations (author-paper*| author-article*|
                               author-software*| editor-series)>

<!-- ALL ELEMENTS WHICH ARE LEAVES -->
<!ELEMENT name (#PCDATA)>
<!ELEMENT name-first (#PCDATA)>
<!ELEMENT name-last (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT postal (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT notification (#PCDATA)>
<!ELEMENT accesspolicy (#PCDATA)>
<!ELEMENT restriction (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ELEMENT issn (#PCDATA)>
<!ELEMENT order (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT directory (#PCDATA)>
<!ELEMENT machine (#PCDATA)>
<!ELEMENT archivesincluded (#PCDATA)>
<!ELEMENT archivesexcluded (#PCDATA)>
<!ELEMENT seriesincluded (#PCDATA)>
<!ELEMENT seriesexcluded (#PCDATA)>
```

```
<!ELEMENT contactemail (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT serie (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT availability (#PCDATA)>
<!ELEMENT creationdate (#PCDATA)>
<!ELEMENT revisiondate (#PCDATA)>
<!ELEMENT publicationstatus (#PCDATA)>
<!ELEMENT journal (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT month (#PCDATA)>

<!ELEMENT file (#PCDATA)>
<!ATTLIST file
            format CDATA #IMPLIED>

<!ELEMENT classification (#PCDATA)>
<!ATTLIST classification
            scheme (jel) "jel">


<!ELEMENT homepage (#PCDATA)>
<!ATTLIST homepage
     xml:link CDATA #IMPLIED
     href     CDATA #IMPLIED
     inline   CDATA #IMPLIED>

<!ELEMENT url (#PCDATA)>
<!ATTLIST url
     xml:link CDATA #IMPLIED
     href     CDATA #IMPLIED
     inline   CDATA #IMPLIED>

<!ELEMENT author-paper (#PCDATA)>
<!ATTLIST author-paper
     xml:link NMTOKEN #FIXED "simple"
     inline NMTOKEN #FIXED "false"
     href CDATA #REQUIRED
     handle ID #REQUIRED>

<!ELEMENT author-article (#PCDATA)>
```

```
<!ATTLIST author-article
     xml:link NMTOKEN #FIXED "simple"
     inline NMTOKEN #FIXED "false"
     href CDATA #REQUIRED
     handle ID #REQUIRED>

<!ELEMENT editor-series (#PCDATA)>
<!ATTLIST editor-series
     xml:link NMTOKEN #FIXED "simple"
     inline NMTOKEN #FIXED "false"
     href CDATA #REQUIRED
     handle ID #REQUIRED>

<!ELEMENT author-software (#PCDATA)>
<!ATTLIST author-software
     xml:link NMTOKEN #FIXED "simple"
     inline NMTOKEN #FIXED "false"
     href CDATA #REQUIRED
     handle ID #REQUIRED>

<!-- OMMITTED TAGS, ATTRIBUTES etc

direct-handle SERIES

-->
```

Anhang B

Konfigurationsdatei: hopec.conf

```
#mySQL database settings
#user and pwd are _not_ UNIX accounts, but myAQL specific!
mysqlhost = localhost
mysqluser = root
mysqlport = 3306
mysqldb= gemini
mysqlpwd=

# PATH and commando of sendmail with options
sendmailcommand=/usr/sbin/sendmail -Uit

# hopecservice mail adress
hopec_mail = markus@wotan.econ.surrey.ac.uk

# the real name of the HopEc Mail
hopec_mail_realname = HoPEc Person Registration Project

# send CC to hopecmail when persons register? (true/false)
hopec_ccmail = true

# full PATH to RePEc-Dir as indicated by $REDIFDIR
redifdir = /home/markus/RePEc/zzz

# Authority
# This is necessary to create new Persons in the correct authority
authority=RePEc

# Homepage of the authority
authority_home=http://netec.mcc.ac.uk
```

```
# complete URL to the DTD which validates the Person XML files
# and to the Stylesheet, looks for redif.dtd and redif.xsl
# e.g. http://www.openlib.org/specs/
DTD_Url = http://wotan.econ.surrey.ac.uk/~markus/
```

Anhang C

Stylesheet: person.xsl

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

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
    xmlns="http://www.w3.org/TR/REC-html40"
    result-ns="">

<xsl:template match = "/">
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="white">
<H1>Experimental Gemini Project Page </H1>
<HR> </HR>
    <xsl:apply-templates/>
</BODY>
</HTML>
</xsl:template>

<xsl:template match ="person">
<TABLE>
<TR BGCOLOR="orange">
<TD><B>Person</B></TD>
<TD BGCOLOR="orange">
<xsl:text>Version:</xsl:text><xsl:value-of select="@version"/>
</TD>
</TR>
<TR><TD><B>Personal Information</B></TD><TD> </TD></TR>
<xsl:apply-templates/>
</TABLE>
```

```
<HR> </HR>
</xsl:template>

<xsl:template match="workplace">
<TR>
<TD><B>Workplace Information</B></TD>
<TD></TD></TR>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="associations">
<TR>
<TD><B>Researchdocument Associations</B></TD>
<TD></TD></TR>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="homepage">
<TR>
<TD> </TD>
<TD><a href="{@href}"><xsl:value-of select="@href"/></a></TD></TR>
</xsl:template>

<xsl:template match="email">
<TR>
<TD> </TD>
<TD><a href="mailto://{.}"><xsl:value-of select="."/></a></TD></TR>
</xsl:template>

<xsl:template match="fax|phone|postal|name">
<TR><TD> </TD><TD><xsl:apply-templates/></TD></TR>
</xsl:template>

<xsl:template match="author-paper">
<TR><TD>Paper: </TD><TD><a href="{@href}"><xsl:value-of select="@href"/>
</a>:<xsl:value-of select="."/></TD></TR>
</xsl:template>

<xsl:template match="author-article">
<TR><TD>Article: </TD><TD><a href="{@href}"><xsl:value-of select="@href"/>
</a>:<xsl:value-of select="."/></TD></TR>
</xsl:template>

<xsl:template match="author-software">
```

```
<TR><TD>Software: </TD><TD><a href="{@href}"><xsl:value-of select="@href"/>
</a>:<xsl:value-of select="."/></TD></TR>
</xsl:template>

<xsl:template match="editor-series">
<TR><TD>Series: </TD><TD><a href="{@href}"><xsl:value-of select="@href"/>
</a>:<xsl:value-of select="."/></TD></TR>
</xsl:template>

</xsl:stylesheet>
```

Anhang D

Dokumentation

Die Dokumentation enthält nur die wichtigsten Klassen aus den Basisklassenmodell. Es fehlen alle Hilfsklassen für die Verwaltung von Daten, Arbeitsplätzen, Institutionen sowie alle Klassen die für die Dienstprogramme zusätzlich erstellt worden sind. Weiterhin fehlen die Klassen zur Fehlerbhandlung, Systemkonfiguration und zum Versenden von emails. Die Klassen sind nur aus Platzgründen nicht aufgeführt, sind aber dennoch dokumentiert und stehen zur Verfügung.

D.1 Dokumentation

1

class **SQLdatabase**

Public Members

	SQLresult ()	
	SQLresult (const SQLTMPresult& res)	*constructor.*
iterator	**begin** ()	*begin of rows.*
iterator	**end** ()	*end of rows.*
bool	**empty** ()	*empty.*
vector <MYSQL_ROW>	::size_type	
	size ()	*size.*
virtual	**~SQLresult** ()	*desctrutor. Free the result if necessary.*
	SQLdatabase (const string& db=standarddb, const string& user=standarduser, const string& passwd=standardpwd, const string& host=standardhost, int port=standardport)	*Connect to SQL server using these settings*
char*	**status** ()	*Returns a string of the server status.*
char*	**client_info** ()	*Returns version number of the client library.*
char*	**host_info** ()	*Returns hostname of the server host.*
int	**proto_info** ()	*Returns protocol version used by connection.*
char*	**server_info** ()	*Returns the version number of the server.*
unsigned long int		

	insert_id ()	*return the value of the AUTO_INCREMENT variable*
SQLTMPresult		
	query (const char *querystring, ...)	*start a query on the database. querystring and arguments have the format of a ̈printf ̈-string. The result can be stored in a SQLresult object.*
virtual	**˜SQLdatabase** ()	*destructor.*

2

class **perlvar**

Implementation of a perl variable

Public Members

SQLerror (const char *errmsg)
SQLerror(errmsg). Provide custom error message.

SQLerror (const MYSQL* mysql)
SQLerror(mysql). Provide an error message like MYSQL would.

const string
what () *return error message as a string.*

perlvar () *Standard constructor. Creates a Scalar with undefined value.*

perlvar (IV i) *Integer constructor. Create a new IV variable and load it with the value of i.*

perlvar (double d)
Double constructor. Create a new double variable and load it with the value of d.

perlvar (const char *c)
String constructor. Create a new string variable and load it with the value of d.

perlvar (SV* sv_var)

SV constructor. Create a new variable of the same type as an existing one. Takes a pointer to a Scalar Value. You should use the copy constructor instead if you do not intend to create a reference.

perlvar (const perlvar& p)

Copy constructor.

perlvar (char *module, bool mode)

Module constructor. Parameter module takes the form package::varname. Mode can be TRUE or FALSE. TRUE indicates that the variable already exists in the given perl package, FALSE will create it in this package.

operator SV* () *convert perlvar to a Scalar Value pointer (SV*).*

operator IV () *convert perlvar to a integer value (IV).*

operator double ()

convert perlvar to a double value (double)

operator const char * ()

*convert perlvar to a string (const char *).*

bool **OK** () *am I OK? (defined !eq undef)*

bool **IOK** () *am I an Integer?*

bool **NOK** () *am I a double?*

bool **POK** () *am I a string?*

bool **ROK** () *am I a reference?*

long unsigned int

	TYPE ()	*return my type. See perls sv.h for values of svtype. perlvar must be a ref!*
perlvar&	**operator =** (const perlvar& p)	*assignment operator.*
perlvar&	**operator=** (SV* sv)	*assignment operator.*
SV*	**RV** ()	*dereference. Use only on references.*
SV*	**CreateRef** ()	*create a reference.*
int	**RefCount** ()	*return the value of the reference count.*
void	**undef** ()	*decrease the reference count of the variable.*
	~perlvar ()	*destructor.*

3

class **perlarray**

implements a perl array.

Public Members

	perlarray ()	*Constructor. Constructs a new empty array.*
	perlarray (perlvar pv, string name)	*When an array is returned as a reference you need this conversion constructor.*
	perlarray (char *module, bool mode)	*Module constructor. see description at perlvar.*
void	**push** (SV *pscalar)	*push given scalar on top of the array.*
SV*	**pop** ()	*get the scalar on top of the array.*
unsigned int long		
	TYPE ()	*return the type of the variable.*
void	**undef** ()	*explicitly free the array.*
void	**decRef** ()	*decrease the reference count of the array.*
int	**RefCount** ()	*return the value of the reference count.*
perlvar	**operator** [] (I32 key)	*read only index operator. If a nonexisting key is requested a undef-Value will be inserted at that position. I32 is an integer type.*
I32	**len** ()	*returns the size of the array.*

~perlarray () *destructor.*

4

class **perlhash**

implements a perl hash table.

Public Members

	perlhash ()	*constructor.*
	perlhash (perlvar pv, string name)	*Conversion constructor. Is needed when the Hashtable must be accessed through a reference.*
	perlhash (char *module, bool mode, string name)	*Module constructor. see description at perlvar.*
perlvar	**operator[]** (const perlvar& key)	*read only index operator. Missing key values will return an undefined value.*
	operator HV* ()	*explictly convert to a Hashtable.*
void	**store** (const perlvar& key, const perlvar& value)	*store a variable in the hashtable.*

unsigned int long		
	TYPE ()	*returns the type of the variable.*
void	**undef** ()	*explicitly free the hashtable.*
void	**decRef** ()	*decrease the reference count of the hashtable.*
int	**RefCount** ()	*return the value of the reference count.*
bool	**exists** (const perlvar& key)	

	checks whether a given key exists.
~perlhash ()	*destructor.*

5

class **perlfunc**

implements calls to perlfunctions with no parameters and no return values or wit one parameter and no return values.

Public Members

perlfunc (const perlvar name)
: *Constructor. See the perlguts and perlcall manpages. Include the package name if desired. @param name is the name of the perlfunction.*

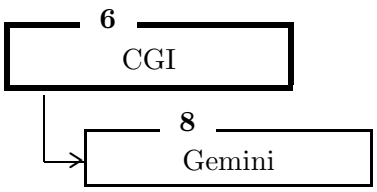
void **operator)** ()
: *call the function (function without parameter).*

void **operator)** (perlvar& parameter)
: *call the funtion (function with one return parameter as call by reference).*

6

class **CGI**

Inheritance



Public Members

ostrstream **cgios** — *the stream for normal output. Will be printed inbetween the <BODY>...</BODY> tags.*

ostrstream **meta** — *the stream for metadata output. Will be printed inbetween the <HEAD>...<HEAD> tags.*

CGI (ostream& o=cout)
Constructor. Takes an ostream as argument. cgios and meta will be printed in this stream.

CGI (const char *content_type,
ostream& o=cout)
Constructor. You can specify the content_type for the CGI output.

void **CGI::form_start** (char action[], char method[])
start the CGI form.

void **CGI::form_end** ()
end the CGI form.

void **CGI::submit** (char Value[], char Name[])
create a SUBMIT button.

void **CGI::hidden** (const char *Name,
const string& Value)

		create a HIDDEN variable.
void	**CGI::textarea** (char Name[], char Cols[], char Rows[])	*create a TEXTAREA.*
string	**CGI::get_value** (char arg[])	*get a CGI value.*
virtual	**CGI::~CGI** ()	*Destructor. On destruction of an object the contents of the streams meta and cgios are written to the outputstream as specified by the constructor.*

7

class **GeminiPage**

GeminiPage is an abstract class which is responsible for displaying the body of a CGI – page.

Public Members

GeminiPage () *Constructor.*

virtual void

display (Gemini* cgi)
display the page. It gets a pointer to Gemini in order to use the correct output stream.

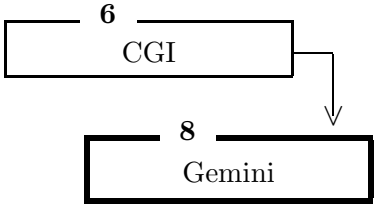
virtual **~GeminiPage** ()

8

class **Gemini** : public CGI

Class Gemini structures HTML pages into pages which consist of constant headers and footers with variable body content.

Inheritance



Public Members

virtual void
setTitle (const string& t)
set the title of the page.

virtual void
display_header ()
displays the header of a page.

virtual void
display_footer () *displays the footer of the page.*

void **display_page** (GeminiPage& page)
displays the body of a page.

Gemini (ostream& os=cout)
Constructor. Sets output stream.

Gemini (const string& t, ostream& os=cout)
Constructor. Sets title and output stream.

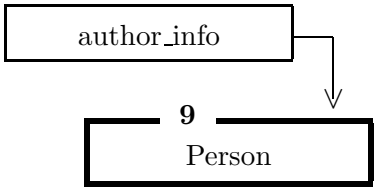
virtual **~Gemini** ()

9

class **Person** : public author_info

The Person class provides basic functionality to handle ReDIF-Persons. It basic functionality provides methods to retrieve and store data in the database, to establish connections to workplace and researchpapers via the role mechanism. This implies that the role of a Person must me known before establishing Person-Researchpaper associations.

Inheritance



Public Members

Person ()

Person (const string& xlname,
const string& xfname=,
const string& xemail=,
const string& xpostal=,
const string& xphone=,
const string& xfax=,
const string& xhpage=,
const string& xbdate=,
const string& xhandle=)
Constructor.

Person (SQLdatabase& db, const string& id,
person_load loadworkingpa-
pers=YES_RESEARCHPAPERS,
const string& mtable=äuthor_map")
Constructor. Loads information from database.

Person (const Person& p)

Copy constructor.

static void
setOutput (PersonOutput* o)
sets the output mode for all Persons.

static void
parsename (string fullname, string& fname, string& lname)
splits a fullname string in firstname and lastname parts.

Person& **operator=** (const Person& x)
Assignment operator.

Person& **merge** (const Person& x)
merge tries to merge two Persons, especially the workingplaces.

bool **operator ==** (const Person& p)
Comparison. Two Persons are equal if there handle is equal.

string **archive** ()
returns the archive identifier of the Person.

void **archive** (const string& f)
set archive identifier.

string **content** ()

void **content** (const string& c)

void **id** (const string& i)
returns numerical id.

string **id** ()
set numerical id.

void **addWorkplace** (Workplace w)
add a Workplace.

void **addWorkplace** (const Institution& i)
add a Institution

void **addResearchpaper** (Researchpaper *w, const string& role)

add a Researchpaper with a certain role.

void **removeResearchpaper** (Researchpaper *w)

remove a Researchpaper.

const_workplace_iterator **workplace_begin** ()

const_workplace_iterator **workplace_end** ()

workplace_iterator **workplace_begin** ()

workplace_iterator **workplace_end** ()

const_paper_iterator **paper_begin** ()

const_paper_iterator **paper_end** ()

paper_iterator **paper_begin** ()

paper_iterator **paper_end** ()

const_role_iterator **role_begin** ()

const_role_iterator **role_end** ()

role_iterator **role_begin** ()

role_iterator **role_end** ()

const PersonRole*

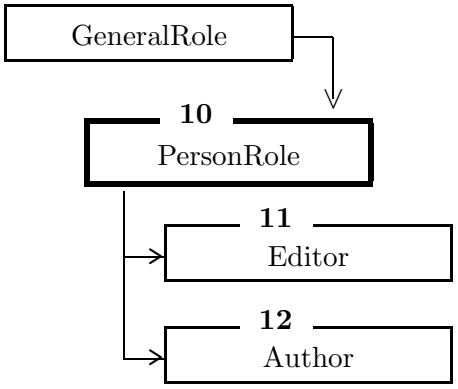
	findRole (const SmartPointer<const Researchpaper> w)	*find the role which is associated with this Person and the given Researchpaper.*
bool	**valid** ()	*returns true if the Person is not in process of updating information.*
string	**validationnumber** ()	*returns the validationnumber which was assigned as a passwd to validate the update process.*
unsigned long int	**store** (SQLdatabase& db, unsigned long int id=0)	*stores the Person in the database*
void	**erase** (SQLdatabase& db)	*erase Person from the database.*
unsigned long int	**update** (SQLdatabase& db)	*updates the Person information in the database.*
friend ostream&	**operator <<** (ostream& os, const Person& auth)	*output operator.*
virtual	**~Person** ()	*destructor.*

10

class **PersonRole** : public GeneralRole

Abstract PersonRole class. A PersonRole gives an explanation of the role a given Person has in the process of creating a given Researchpaper. It is an associated class for the association between Person and Researchpapers. Editor and Author are implementations of this class. So far they do not have additional functionality.

Inheritance



Public Members

PersonRole (string i, const Person *p,
const Researchpaper *r)
Constructor. Parameters are the Roletype, the person and the researchpaper involved in the relationship.

virtual PersonRole*
clone (const Person *p, const Researchpaper* r)
provides clone mechanism to copy PersonRole-Objects.

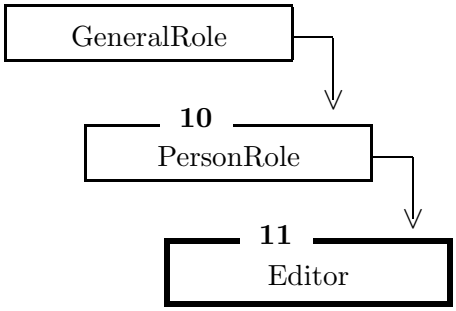
static PersonRole*

	create (const string& whichRole, Person *p, Researchpaper *r)	*constructs a role correponding to the string and connects it to the Person in p.*
void	**store** (SQLdatabase& db)	*stores the Role in the database Tableauthor_map.*
const	SmartPointer <const Researchpaper> **getResearchpaper** ()	*returns a pointer to the Researchpaper involved in the role.*
const	SmartPointer <const Person> **getPerson** ()	*returns a pointer to the Person involved in the role.*
bool	**operator ==** (const PersonRole& r)	*Comparison.*
bool	**operator <** (const PersonRole& r)	
virtual	**~PersonRole** ()	

11

class **Editor** : public PersonRole

Inheritance



Public Members

Editor (const Person *p, const Researchpaper *r)

constructs an Editor role.

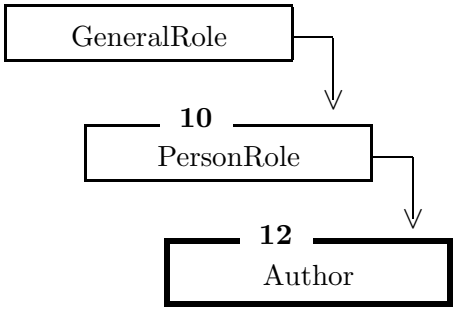
PersonRole*

clone (const Person *p, const Researchpaper *r)

12

class **Author** : public PersonRole

Inheritance



Public Members

Author (const Person *p,
const Researchpaper *r)
constructs an Author role.

PersonRole*
clone (const Person *p, const Researchpaper *r)

13

class **Researchpaper**

Implements common functionality for Researchpapers. Each Researchpaper must assume a role (Article, Paper, Series, Software) which specifies the type of the Researchpaper.

Public Members

Researchpaper (const string i=, const string t=, const string o=)
Constructor.

Researchpaper (SQLdatabase& db, const string& handle, researchpaper_load loadpersons=NO_PERSONS)
Constructor. Gets information from the database.

Researchpaper (const Researchpaper& p)
Copy constructor

Researchpaper&
operator= (const Researchpaper& x)
Assignment operator.

bool **operator <** (const Researchpaper& x)

bool **operator ==** (const Researchpaper& x)
Comparison operator. Two Researchpapers are equal if their id is equal.

void **store** (SQLdatabase& db, researchpaper_load store_person=NO_PERSONS)
store researchpaper in database.

string **id** ()
return id/handle of the Researchpaper.

void **id** (const string& i)

		set id/handle of the Researchpaper.
string	**title** ()	*return title of the researchpaper.*
void	**title** (const string& t)	*set title of the researchpaper.*
string	**online** ()	*returns yes if the Researchpaper if available online.*
void	**online** (const string& o)	*set to yes if the Researchpaper is available online.*
void	**abstract** (const string a)	*set abstract of the Researchpaper.*
string	**abstract** (bool force=false)	*read the abstract from the database. If force equals true read even if the abstract is already set.*
void	**addPerson** (Person *a, const string& role)	*adds a Person to be associated with this Researchpaper.*
void	**removePerson** (Person *a)	*remove a certain person.*
vector <Person>	**co_Persons** (SQLdatabase& db)	*returns every Person involved with this Researchpaper. It needs access to the database in order to read the relations stored in author_map.*
void	**role** (ResearchRole *myRole)	*sets the role of the Researchpaper.*
ResearchRole*		

role () *returns the role of the Researchpaper.*

static void
setOutput (RpOutput* o)
sets outputmode.

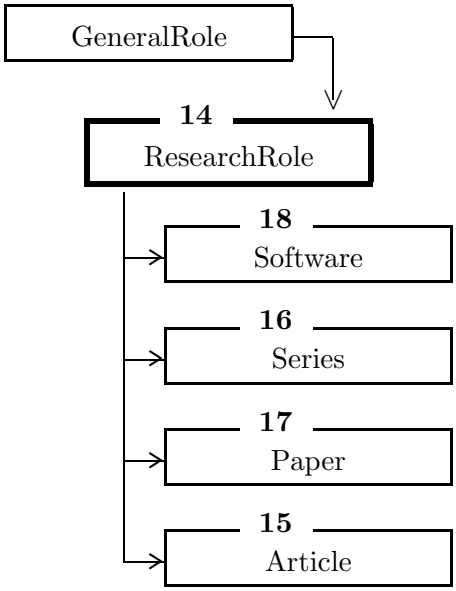
friend ostream&
operator << (ostream& os,
const Researchpaper& rp)
Output operator

virtual **~Researchpaper** ()
Destructor.

14

class **ResearchRole** : public GeneralRole

Inheritance



Public Members

ResearchRole (string i, Researchpaper* p)

virtual string
handle2url (const string& handle,
const string& online)

static ResearchRole*
create (const string& whichRole,
Researchpaper* p)

virtual ResearchRole*
clone (const Researchpaper *p)

virtual void
store (SQLdatabase& db)

virtual string

jelclassification ()

virtual void

jelclassification (const string& c)

virtual string

abstract ()

virtual void

abstract (const string& a)

virtual string

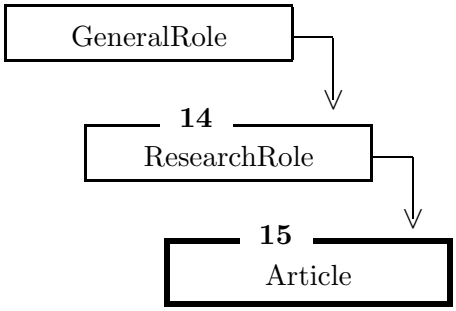
possible_roles ()

virtual **~ResearchRole** ()

15

class **Article** : public ResearchRole

Inheritance



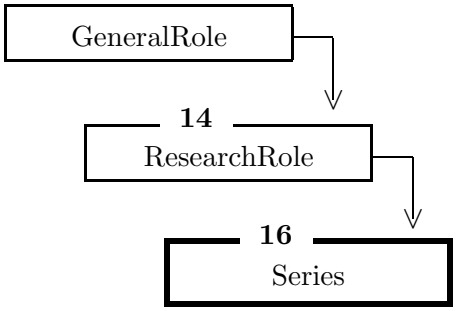
Public Members

	Article (Researchpaper* p)
void	**store** (SQLdatabase& db)
string	**handle2url** (const string& handle, const string& online)
ResearchRole*	**clone** (const Researchpaper *p)
static ResearchRole*	**create** (Researchpaper *p)
string	**abstract** ()
string	**possible_roles** ()

16

class **Series** : public ResearchRole

Inheritance



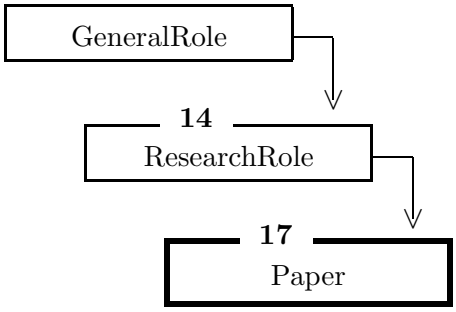
Public Members

	Series (Researchpaper *p)
void	**store** (SQLdatabase& db)
ResearchRole*	**clone** (const Researchpaper *p)
static ResearchRole*	**create** (Researchpaper *p)
string	**possible_roles** ()
string	**handle2url** (const string& handle, const string& online)

17

class **Paper** : public ResearchRole

Inheritance



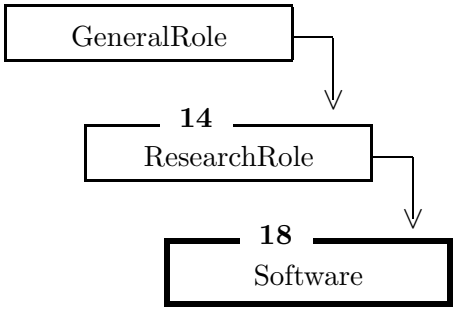
Public Members

	Paper (Researchpaper *p)
void	**store** (SQLdatabase& db)
string	**handle2url** (const string& handle, const string& online)
ResearchRole*	**clone** (const Researchpaper *p)
static ResearchRole*	**create** (Researchpaper *p)
string	**abstract** ()
void	**abstract** (const string& a)
string	**possible_roles** ()

18

class **Software** : public ResearchRole

Inheritance



Public Members

	Software (Researchpaper* p)
void	**store** (SQLdatabase& db)
ResearchRole*	**clone** (const Researchpaper *p)
static ResearchRole*	**create** (Researchpaper *p)
string	**possible_roles** ()
string	**handle2url** (const string& handle, const string& online)

Ehrenwörtliche Erklärung

Ich versichere, daß ich die beiliegende Diplomarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ich bin mir bewußt, daß eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum Unterschrift