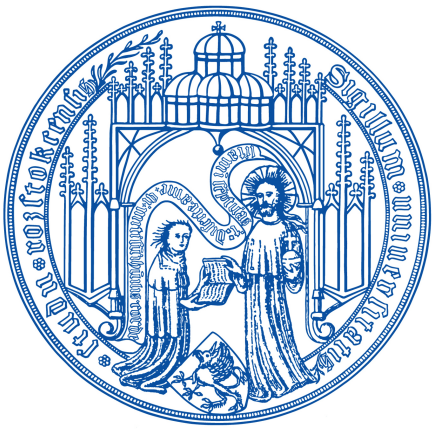
Verfahren zur analogen Langzeitsicherung digitalen Erbes

Masterarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Marc-Eric Meier
Matrikelnummer: 211201312
geboren am: 26.09.1989 in Rostock
Gutachter: Dr.-Ing. Holger Meyer
Zweitgutachter: Prof. Dr. rer.nat.habil. Clemens H. Cap
Abgabedatum: 26. Mai 2017

Inhaltsverzeichnis

Kapitel 1

Einleitung

In Percy Bysshe Shelleys Gedicht **Ozymandias** von 1817 berichtet ein Wanderer von seiner Entdeckung in der Wüste: Er fand in Ruinen ein zerfallenes Monument des gleichnamigen Königs, auf dessen Sockel die Worte *"My name is Ozymandias, king of kings, Look on my works, ye Mighty, and despair!"*[1] eingraviert sind. Mehr ist von seiner Herrschaft nicht übrig geblieben. Im Laufe der Geschichte überdauerte nur ein Teil von Zeitzeugen, sodass wir unsere Vergangenheit nur aus Bruchstücken rekonstruieren können. Wir sind der Ansicht, dass die Vorfahren des *Homo Sapiens* in Höhlen lebten, weil Archäologen entsprechende Spuren, in Form von Feuerstellen, Steinwerkzeugen und Malereien dort entdeckten. Diese Orte stellen häufig besonders gute Bedingungen für den Erhalt von solch stummen Zeitzeugen dar. Anderswo sieht es eventuell schlechter aus: Die Überreste eines toten, frühen Menschen werden im Wald von Tieren in alle Himmelsrichtungen getragen und verrotten restlos, Werkzeuge, Schmuck und Kunstwerke aus Holz oder Leder fallen ebenfalls der Witterung zum Opfer. Diese Informationen sind damit für immer verloren. Anderes Wissen hat bis in die Gegenwart überdauert.
Mithilfe der Schrift war es den Menschen erstmals möglich, komplexe Informationen gezielt festzuhalten. Die ältesten bekannten Schriftzeichen stammen von den Sumerern[23]. Tontafeln, die im als vermutlich älteste Stadt der Welt geltenden *Uruk* im vorderen Orient gefunden wurden, lassen sich auf die Zeit um 3400 Jahre vor Christus datieren. Da es sich hierbei bereits um ein komplexes Schriftsystem, eine **Bilderschrift** mit über 700 Zeichen, handelt, ist davon auszugehen, dass dessen Entwicklung schon viele Jahre zuvor begonnen hat. Ab etwa 2900 v. Chr. entwickelte sich aus dieser Bilderschrift eine **Keilschrift** heraus, welche einfacher wurde und erstmals Zeichen für Silben und Laute beinhaltete. Die Bezeichnung Keilschrift rührt von der Art her, wie sie geschrieben wurde: Mit einem Rohrgriffel wurden die Zeichen in den Ton gedrückt, wobei die namensgebenden keilförmigen Striche entstanden, wie es in den Schriftbeispielen Abbildung 1.1e erkennbar ist. Während die gefundenen Keilschrifttexte bis dahin größtenteils Geschäfte beinhalteten, wurden aus der Zeit ab 2400 v. Chr. zunehmend auch **Gesetzestexte, amtliche Briefe, Chroniken, religiöse Texte und Literatur** gefunden. Interessanterweise beziehen sich außerdem tausende gefundener Schriften auf **Bier**, so wurde etwa der Göttin Ninkasi, welche für das alkoholische Getränk als zuständig angesehen wurde, eine Hymne gewidmet[18], die bis heute überliefert ist. Die Keilschrift verbreitete sich und wurde in viele Sprachen übernommen, etwa in das Akkadische, das Assyrische, das Babylonische, das Hethitische, das Hurritische sowie das Urartäische. Außerdem gelangte sie über das Elamische bis in das Industal. Parallel zur Keilschrift entwickelte sich auf dem afrikanischen Kontinent, wahrscheinlich durch die sumerischen Piktogramme inspiriert, die ägyptische **Hieroglyphenschrift**. Von diesen abstammend entwickelte sich im östlichen Mittelmeerraum das **proto-kanaanitische Alphabet**, welches gerade mal 28 Zeichen beinhaltete, die allesamt für Silben standen, welche sich zu Worten zusammenfügen ließen. Auch diese Schrift wies einige Jahrhunderte später wieder verschiedenste Varianten auf, etwa die nabatäische Schrift, aus der sich später die kufisch-arabische Schrift und damit die heutige **arabische Schrift** entwickelte, die aramäische Schrift, welche ein Vorläufer der südostasiatischen und der modernen indischen Schriften ist, sowie das **phönikische Alphabet**. Letzteres bildete die Grundlage für

[1] Übersetzung nach Adolf Strodtmann: *"Mein Name Ist Osymandias, aller Kön'ge König: Seht meine Werke, Mächt'ge, und erbebt!"*

(a) Griechische Schriftzeichen (b) Chinesische Schrift (c) Tibetische Schrift

(d) Germanische Runen (e) Keilschrift (f) Arabische Schrift

(g) Ägyptische Hieroglyphen (h) Aramäische Schrift (i) Phönizische Schrift

Abbildung 1.1: Auswahl verschiedener, im Unicode verfügbarer Schriftzeichen

die ab dem achten Jahrhundert v. Chr. aufkommende **griechische Schrift**. Hier entsprechen die Buchstaben statt Silben den Konsonanten und Vokalen. Mit dieser Vereinfachung erhöhte sich auch die Zahl derer, die des Lesens und Schreibens mächtig waren, weshalb die Schrift mehr Verwendung fand. Aus dem Griechischen entwickelte sich über das Etruskische das uns bekannte **lateinische Alphabet**, sowie über tausend Jahre später das **kyrillische Alphabet**, welches heute noch beispielsweise in der russischen Sprache Verwendung findet. Unabhängig von den genannten Schriften entstand ab circa 1600 v.Chr die chinesische Bilderschrift, welche sich zu einer Laut-Bild-Schrift weiterentwickelte. Sie ist die Grundlage für das gegenwärtige japanische und koreanische Schriftsystem. In Amerika wurde ab etwa 800 v.Chr. die Hieroglyphenschrift der Zapoteken verwendet. Diese konnte aber nicht alle Aspekte der Sprache ausdrücken, lediglich die sich daraus entwickelnde Schrift der Maya konnte dies. Kulturen, die in den Anden lebten verfügten interessanterweise über keine herkömmliche Schrift, sondern verwendeten stattdessen die sogenannten Quipu-Schnüre, eine Art Knotenschrift. Diese stellte Waren mithilfe verschiedenfarbiger Schnüre und die entsprechenden Quantitäten mittels Knoten dar. Letztendlich ist noch das Rongorongo bekannt, eine hieroglyphen-ähnliche Schrift, die auf den Osterinseln im Pazifik genutzt wurde. Sie fand bei ihrer Entdeckung durch die Europäer allerdings keine Verwendung mehr und konnte bisher nicht entziffert werden.

Es ist also erkenntlich, dass sich im Laufe der Geschichte bereits eine Vielzahl an Sprachen und Schriften gebildet hat. Dabei versteht es sich von selbst, dass viele heutzutage nicht mehr genutzt werden, gar können einige nicht mehr verstanden werden. Wir sind heute nur deshalb imstande, ägyptische Hieroglyphen zu deuten, weil der **Stein von Rosette**[2] deren Entschlüsselung ermöglichte. Dabei handelt es sich um ein 1799 gefundenes Bruchstück einer Steintafel aus dem Jahr 196 v. Chr. Es beinhaltet ein Priesterdekret, welches entsprechend der verschiedenen Bevölkerungsgruppen in drei Sprachen verfasst wurde: Ägyptisch (Hieroglyphen) für die Priester, Demotisch für das Beamtentum und Altgriechisch, weil die damaligen Herrscher über Ägypten griechischer Herkunft waren. Auf diese Weise konnte 1822 der französische Sprachwissenschaftler **Jean-François Champollion** erstmals die ägyptische Sprache entziffern.

Wissen schriftlich festzuhalten ist jedoch keine Garantie, dass dieses die Zeit auch tatsächlich überdauert. Dazu ist es notwendig, dass auch der Träger der Informationen unbeschädigt durch die Jahrhunderte kommt. Die bereits angesprochenen Tontafeln erwiesen sich als äußerst widerstandsfähige Medien hierfür. Erosion, Feuchtigkeit oder organische Gegenspieler, wie Schimmel oder Bakterien beeinflussen den Zustand nur gering, bis gar nicht, die größte Bedrohung stellen mechanische Einflüsse, wie etwa Schläge dar. Jedoch erwiesen sie sich im alltäglichen Gebrauch als umständlich, insbesondere durch Gewicht, Größe und Fertigungsaufwand, weshalb sich andere Möglichkeiten von Schriftträgern etablierten.

Mit dem Aufkommen der Schrift fingen die Menschen an, ihr Leben, wie auch ihre Kultur zu dokumentieren und so für die Nachwelt aufzubewahren. Dass dies nicht immer funktioniert, wissen wir bereits: Texte gehen verloren oder werden zerstört etwa beim Brand der großen Bibliothek von Alexandria, bei Bücherverbrennungen im dritten Reich 1933 oder auch in der Gegenwart, als das Kölner Stadtarchiv im Jahre 2009 einstürzte. Projekte, wie etwa Goo-

[2]Siehe `https://de.wikipedia.org/w/index.php?title=Stein_von_Rosette&oldid=161843771`

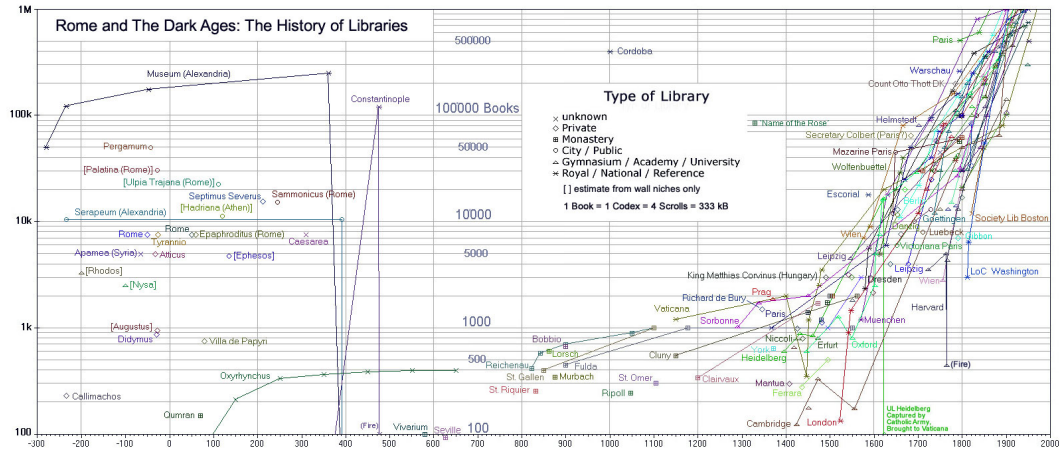


Abbildung 1.2: "Rom und die 'Dunklen Jahrhunderte': die Geschichte der Bibliotheken im Zeitraum von 300 v. Chr. bis ca. 1900 n. Chr."

gle Books[3] versuchen dem Verfall entgegenzuwirken, indem Werke digitalisiert werden und so kopierbar und erhaltbar gemacht werden sollen. Jedoch benötigen gegenwärtige digitale Sammlungen eine aktiven Pflege, welche die stetige Migration auf neuere Medien beinhaltet, da unsere Datenträger bislang nur begrenzt haltbar sind und Lesesysteme verfügbar sein müssen. Dazu kommt ein hoher Aufwand an Dokumentation, damit die Daten auch interpretierbar sind. Es stellt sich also die Frage: Wie sieht es mit unserem Vermächtnis aus? Lässt sich unser digitales Kulturerbe mit einem angemessenen Aufwand und zu vertretbaren Kosten langfristig festhalten?

1.1 Problemstellung und Motivation

Das Ziel dieser Arbeit ist die Entwicklung eines Konzepts zur wartungsarmen und langfristigen Sicherung digitaler Daten auf analogen Datenträgern. Konkret sollen die Metadaten des Wossidlo Digital Archive (WossiDiA) – eine Hypergraphdatenbank, welche Verknüpfungen zwischen über zwei Millionen historischen Belegen des mecklenburgischen Gelehrten und Volkskundlers Richard Wossidlo speichert – für die Nachwelt erhalten bleiben. Dabei stellt diese Arbeit eine Fortsetzung der Digitalisierung und Sicherungsverfilmung auf Mikrofilm dar, welche als Gemeinschaftsprojekt der Universität Rostock, dem Deutsche Forschungsgemeinschaft (DFG) und dem Bundesamt für Bevölkerungsschutz und Katastrophenhilfe (BBK) durchgeführt wurde.

1.2 Gliederung dieser Arbeit

Diese Arbeit ist wie folgt gegliedert: Im Kapitel 2 sollen allgemeine Betrachtungen zur Problematik der Langzeitarchivierung getätigt werden. Es folgt eine Vorstellung möglicher Techniken in Kapitel 3. Anschließend werden das Wossidlo-Archiv, das damit verbundene WossiDiA-Projekt und die darin gespeicherten Metadaten vorgestellt, um deren Archivierung es in dieser Arbeit geht. Kapitel 5 stellt einige Konzepte hierfür vor, welche in Kapitel 6 umgesetzt werden sollen. Die notwendigen Instruktionen, die zur Interpretation der Daten notwendig sind, sollen in Kapitel 7 aufgelistet werden. Abschließend erfolgt eine Zusammenfassung, sowie ein Ausblick in Kapitel 8.

[3] Siehe `https://books.google.de/`

Kapitel 2

Langzeitarchivierung von kulturellem Erbe

Das Problem der Langzeitarchivierung ist kein neues. Daher existiert auf diesem Gebiet bereits einiges an Erfahrung. Dieses Kapitel soll zuerst einige Institutionen und Projekte vorstellen, die sich mit dem Thema befassen. Anschließend werden einige Probleme erläutert und geläufige Lösungsstrategien dargelegt.

2.1 Ausgewählte Institutionen und Projekte

Gegenwärtig existiert eine Anzahl von Institutionen und Projekten, die sich mit der Langzeitarchivierung befassen. Dazu gehören insbesondere auch Museen, Bibliotheken und Archive, welche direkt mit den zu sichernden Objekten in Kontakt stehen. Weitere sollen hier vorgestellt werden, jedoch soll sich zuerst mit der Haager Konvention befasst werden, welche den großen Anstoß auf dem Gebiet veranlasste.

2.1.1 Haager Konvention

Angetrieben durch die massiven Zerstörungen von Kulturgütern während des Zweiten Weltkrieges wurde am 14.Mai 1954 die **Konvention zum Schutz von Kulturgut bei bewaffneten Konflikten** in Den Haag verabschiedet[45, 13]. Diese wurde bisher von mehr als 120 Staaten ratifiziert, darunter auch von der Bundesrepublik Deutschland am 11.April 1967. Sie stellt fest, dass aufgrund der steigenden Zerstörungskraft der modernen Kriegstechnologie die Schäden an Kulturgütern zunehmen. Jede Nation habe einzusehen, dass ein solcher Verlust nicht nur ein Verlust für das betroffene Land, sondern für das kulturelle Erbe der gesamten Menschheit darstellt. Daher sei es die Pflicht der internationalen Gemeinschaft, diese vor Schäden zu bewahren. Kulturgut ist im Sinne des ersten Artikels der Konvention[6]:

a) bewegliches oder unbewegliches Gut, das für das kulturelle Erbe aller Völker von großer Bedeutung ist, wie z. B. Bau-, Kunst- oder geschichtliche Denkmäler religiöser oder weltlicher Art, archäologische Stätten, Gebäudegruppen, die als Ganzes von historischem oder künstlerischem Interesse sind, Kunstwerke, Manuskripte, Bücher und andere Gegenstände von künstlerischem, historischem oder archäologischem Interesse sowie wissenschaftliche Sammlungen und bedeutende Sammlungen von Büchern, Archivalien oder Reproduktionen des oben bezeichneten Kulturguts;

b) Baulichkeiten, die in der Hauptsache und tatsächlich der Erhaltung oder Ausstellung des unter a) bezeichneten beweglichen Gutes dienen, wie z. B. Museen, größere Bibliotheken, Archive sowie Bergungsorte, in denen im Falle bewaffneter Konflikte das unter a) bezeichnete bewegliche Kulturgut in Sicherheit gebracht werden soll;

c) Orte, die in beträchtlichem Umfange Kulturgut im Sinne der Unterabsätze a) und b) aufweisen und als „Denkmalorte“ bezeichnet sind.

Kulturgut ist zu Sichern und zu Respektieren, was bedeutet, dass bereits in Friedenszeiten entsprechende Maßnahmen getroffen werden müssen und weder eigenes, noch fremdes

Kulturgut zu gefährden ist. Es werden Details zum Transport, Sonderschutz und zu militärischen Maßnahmen, sowie zur Kennzeichnung des Kulturgutes und zur Durchführung festgelegt.
Am 26.März 1999 fand eine weitere Konferenz statt, welche anlässlich des hundertjährigen Jubiläums der großen Friedenskonferenz von den Haag im Jahre 1899 durchgeführt wurde. Hierbei wurde ein zweites Protokoll erlassen, welches die ursprüngliche Konvention ergänzt. Zu den Ergänzungen zählt unter anderem die Einrichtung eines Zwischenstaatlichen Komitees, bestehend aus zwölf Regierungen. Dieses führt die Liste der unter Schutz gestellten Güter und überprüft die Umsetzung der Konvention.

2.1.2 Bundesamt für Bevölkerungsschutz und Katastrophenhilfe

Das **BBK**[1] wurde am 1. Mai 2004 errichtet. Die Notwendigkeit dazu ergab sich aus den zu dieser Zeit neuen Bedrohungen, etwa die Anschläge des 11. Septembers 2011 in New York oder die Hochwasserkatastrophe im Jahre 2002. Zuvor hat es diverse Vorgängerorganisationen gegeben, die sich nach dem Abzug der alliierten Luftschutzorganisationen und -einrichtungen 1946 und der anschließenden Gründung der Bundesrepublik Deutschland (BRD) bildeten.
Das BBK übernimmt verschiedene Aufgaben zum Schutz der Zivilbevölkerung bei militärischen Bedrohungen und Katastrophen. Dazu gehört auch der **Kulturgutschutz**. Auf diese Weise entspricht die BRD dem dritten Artikel der Haager Konvention (Abschnitt 2.1.1). Zu den wesentlichen Aufgaben des BBK gehören[2] die Sicherungsverfilmung von national wertvollem Archiv- und Bibliotheksgut, die fotogrammetrische Erfassung unbeweglichen Kulturguts, die Erarbeitung von Richtlinien und Konzepten zum Bau von Bergungsräumen für bewegliches Kulturgut, sowie weitere Konzepte, in Abstimmung mit der United Nations Educational, Scientific and Cultural Organization (UNESCO).
Im Rahmen dieser Arbeit ist vor allem die **Sicherungsverfilmung** von Interesse. Diese wird in der BRD seit 1961 durchgeführt. Archivalien werden hierzu optisch auf **Mikrofilm** gespeichert. Mikrofilm gilt als äußerst beständiges Medium, deren Haltbarkeit mit 500 Jahren angegeben wird[34]. Gespeicherte Informationen können im Notfall annähernd mit bloßem Auge gelesen werden, je nach Art des archivierten Dokuments kann eine optische Lesehilfe, wie etwa eine Lupe oder ein Mikroskop, sowie eine künstliche Lichtquelle nötig sein. Hinzu kommt, dass das Verfahren im Vergleich sehr wirtschaftlich ist. Die **Initiative "Medium Film nutzen! "** veröffentlichte dazu 2013 die **Saarbrücker Erklärung**, welche diese Vorteile zusammenfasst und darauf drängt, nicht ausschließlich digitale Speichermedien zu verwenden, da deren Zuverlässigkeit unter bestimmten Umständen zweifelhaft sein kann. Zur Gewährleistung der qualitativen Anforderungen wurden 1987 die **Grundsätze zur Durchführung der Sicherheitsverfilmung von Archivalien**[19] festgeschrieben. Diese beschreiben im ersten Teil die Grundlagen der Sicherungsverfilmung, dazu gehören der Zweck, die Zuständigkeit, sowie die Auswahl des Verfilmungsgutes. Hervorzuheben ist, dass dabei das Archivgut in drei Dringlichkeitsstufen unterteilt wird. Derzeit werden ausschließlich Archivalien der höchsten Dringlichkeitsstufe 1 verfilmt[7]. Weiterhin sind Sicherungsfilme grundsätzlich von der Benutzung ausgeschlossen. Ausnahmen gelten lediglich zur einmaligen Vervielfältigung. Teil Zwei gibt technische Anweisungen zur Durchführung. Dazu gehören etwa organisatorische Belange, Details zur Aufnahme, wie etwa die Gliederung des Sicherungsfilms, Details der Filmentwicklung, sowie Richtlinien zur Sofortkontrolle, Qualitätsprüfung und vorläufigen Aufbewahrung der Sicherungsfilme.
Als **Zentraler Bergungsort der Bundesrepublik Deutschland**[3] wurde der **Barbarastollen** in Oberried bei Freiburg im Breisgau auserkoren. Beim Barbarastollen handelt es sich um den Untersuchungsstollen eines ehemaligen Silberbergwerkes. Dieser besteht aus Granit und Gneis, wurde zusätzlich mit Schalbeton ausgekleidet und mit Drucktüren abgesichert. Er genießt als einziges Objekt der BRD den Sonderschutz nach den Regeln der Haager Konvention. Alle Sicherungsfilme werden in speziellen Edelstahlbehältern eingelagert. Diese werden luftdicht verschlossen; vor der Versiegelung muss ein staub- und schadstofffreies Mikroklima von 35 % relativer Luftfeuchte geschaffen werden, wodurch die Haltbarkeit des Mikrofilms von 500 Jahren gewährleistet werden soll.

[1] Siehe `http://www.bbk.bund.de`
[2] Siehe `http://www.bbk.bund.de/DE/AufgabenundAusstattung/Kulturgutschutz/kulturgutschutz_node.html` am 24.01.2017
[3] Siehe: `http://www.bbk.bund.de/DE/AufgabenundAusstattung/Kulturgutschutz/ZentralerBergungsort/zentralerbergungsort_node.html`

Eine Sicherungsverfilmung wurde ebenso in der Deutschen Demokratischen Republik durchgeführt. Dies begann in etwa gleichzeitig mit der BRD. Die Sicherungsverfilmung von Kulturgut wurde als Teil der Sicherung staatlicher Archivbestände angesehen und somit größtenteils von der Zentralstelle für Reprografie in Kossenblatt übernommen[22]. Hier wurden die Filme zunächst auch gelagert, ebenso bei den Bezirksverwaltungen der Volkspolizei, beim Zentralen Staatsarchiv in Potsdam, dem Ministerium des Inneren und an weiteren Lagerstellen. Allerdings waren die Lagerbedingungen nicht optimal, sodass man bereits am Ende der sechziger Jahre feststellte, dass die Filmrollen verklebten, was ein Anzeichen auf den Zersetzungsprozess ist. Daher wurden die Bestände ab 1971 nach Babelsberg und Wilhelmshagen umgelagert. Bereits zuvor begann der Bau eines speziellen Einlagerungsortes in der Nähe von Potsdam. Dieser musste allerdings mittels entsprechender Klimaanlagen reguliert werden, um die Lagerräume bei 10°C und 55% Luftfeuchtigkeit zu halten. Das Gebäude war nicht sonderlich *verbunkert*. Nach der Wende wurden die Bestände gesichtet und in den jeweils zuständigen Stellen zugeführt.

2.1.3 Human Document Project

Das **Human Document Project** hat es sich zur Aufgabe gemacht, das Vermächtnis der Menschheit auch nach deren ihrem Ende zu erhalten. Dazu sollen Dokumente, welche die zeitgenössische Kultur repräsentieren, gesammelt und in einer Form präserviert werden, sodass sie noch lange Zeit verfügbar sind. Auf diese Weise sollen sie etwa von einem möglichen Nachfolger des *Homo Sapiens*, einer alternativen, nicht mit uns verwandten dominanten Spezies auf dem Planeten Erde oder gar einer außerirdischen Lebensform gefunden werden können, womit unsere Errungenschaften im Gedächtnis des Universums erhalten blieben würden.
Bisher wurden Konferenzen in Saarbrücken[4], Stanford[5] und Enschede[6] dazu abgehalten. Dass es sich dabei um ein stark interdisziplinäres Projekt handelt, zeigt die Auswahl an Vortragenden und deren Themen. Dazu gehören etwa Überlegungen zu den tatsächlich notwendigen Haltbarkeiten von Datenträgern[14], Technologien für solche potentiellen Langzeitdatenträger[28, 17], hinzu kommen einige philosophische Betrachtungen[35]. Ironischerweise sind viele der Publikationen nicht ohne Weiteres auffindbar, weshalb weitere Details nicht erläutert werden können. Einige interessante Artikel werden im Verlaufe dieser Arbeit jedoch an passender Stelle vorgestellt. Weil auch, bis auf die Homepages zu den einzelnen Konferenzen, mitsamt der dazugehörigen Zeitpläne, die Dokumentation des Projektes eher zu wünschen übrig lässt, können leider auch keine genaueren Aussagen zu den Initiatoren, der Dauer oder den tatsächlich unternommenen Maßnahmen getätigt werden.

2.1.4 Nestor

Das **Kompetenznetzwerk für digitale Langzeitarchivierung in Deutschland** wurde 2003 gegründet und bündelt seitdem einen Großteil aller Projekte zur digitalen Langzeitarchivierung in Deutschland und dem deutschen Sprachraum. Die Bezeichnung *nestor* leitet sich aus der englischen Übersetzung des Projekttitels ab: ***N****etwork of* ***e****xpertise in long-term* ***sto****rage and availability of digital* ***r****esources in germany*. Im Jahr 2007 veröffentlichte das Bündnis erstmals das **nestor Handbuch**[38], welches aktuell in der Fassung 2.3 kostenlos zum Download angeboten wird[7]. Dessen Prämisse ist es, den zum jeweiligen Zeitpunkt aktuellen Wissensstand über die Langzeitarchivierung digitaler Objekte im Überblick festzuhalten und zu vermitteln. Die Version 2.3 des Handbuchs stammt aus dem Jahr 2010.

2.1.5 Google Books

Bei dem Projekt **Google Books**[8] handelt es sich um eine Initiative der amerikanischen Firma Google Inc.] zur Sammlung und Digitalisierung von Büchern, welche 2004 vorgestellt wurde. Ziel ist der Erhalt der Objekte, sowie das Anbieten einer Volltextsuche für die archivierten Werke. Teile der Sammlung werden den Nutzern kostenlos zur Verfügung gestellt.

[4] Siehe `http://smi.ewi.utwente.nl/~abn/hudop/2010/`
[5] Siehe `http://smi.ewi.utwente.nl/~abn/hudop/2012/`
[6] Siehe `http://hudoc2014.manucodiata.org/index.php/history`
[7] Siehe `http://nestor.sub.uni-goettingen.de/handbuch/`
[8] Siehe `https://books.google.com`

2.1.6 Project Gutenberg

Ähnlich, wie Google Books stellt das **Projekt Gutenberg**[9] eine Reihe von Büchern zur Verfügung. Hierbei werden weniger Scans, sondern eher Klartextformate, wie American Standard Code for Information Interchange (ASCII), Portable Document Format (PDF) oder Hypertext Markup Language (HTML) verwendet. Da es sich um urheberrechtsfreie Texte handelt, werden diese kostenlos und frei zur Verfügung gestellt. Während sich das Projekt Gutenberg hauptsächlich mit englischsprachigen Werken befasst, existiert ein deutschsprachiges Schwesterprojekt.

2.2 Probleme der digitalen Langzeitarchivierung

Auf den ersten Blick sind die Vorteile digitaler Dokumente gegenüber den analogen Pendants überwältigend: Sie lassen sich schnell und eindeutig kopieren, ohne dass sie dabei einem verfälschenden Rauschen unterliegen und sind damit an kein Medium gebunden. Die dazu erforderlichen Datenträger sind, verglichen mit Gebäuden von Bibliotheken, Museen und Archiven winzig. Festplatten mit einem Preis von gerade einmal 3 Cent pro Gigabyte sind ohne Probleme für Endverbraucher erhältlich[10]. Geräte dieser Art tragen bei einer Größe von nur dreieinhalb Zoll acht Terabyte an Daten, größere Speicher sind bereits angekündigt[11]. Auf einem solchen Speichermedium könnte eine Privatperson die Bestände mehrerer Bibliotheken, welche Milliarden von Bänden umfassen, zuhause im Klartext speichern[42]. Mit heutigen Verfahren lassen sich diese Datenmengen äußerst effizient durchsuchen, womit ein komfortables Arbeiten ohne Probleme möglich ist.
Jedoch trifft man bei der Speicherung von digitalen Dokumenten, also *Bitströmen* oder allgemeiner *digitalen Zeichenströmen*, auch auf Probleme. Ein Mensch kann digitale Datenträger in der Regel nicht, oder zumindest nicht effizient lesen. Dies ist in den allermeisten Fällen ein sensorisches Problem, da Sinne zum Wahrnehmen von Magnetfeldern – etwa auf Disketten oder Festplatten – beim Menschen nicht vorhanden sind. Auch die Vertiefungen in den geläufigen optischen Datenträgern, wie CDs oder DVDs sind mit bloßem Auge nicht zu erkennen. Einige optoelektronische Schriften sind vom Menschen durchaus *von Hand* zu interpretieren, jedoch bei Weitem nicht in der Geschwindigkeit, wie es mit einem technischen Lesegerät möglich ist. Zum Lesen der Daten wird also ein entsprechendes Werkzeug benötigt, welches auf den Datenträger zugeschnitten ist. Da es aufgrund des Wandels der Technik und der sich damit ständig verändernden Nachfrage keine Garantie gibt, dass solche Geräte langfristig produziert werden und damit verfügbar sind, besteht die Gefahr, dass die Datenträger irgendwann nicht mehr ausgelesen werden können. Dies wird umso deutlicher, wenn man bedenkt, dass ein Abspielsystem häufig nicht nur aus einer Komponente besteht, sondern meist aus einem Lesegerät (etwa ein Diskettenlaufwerk), welches mithilfe einer Schnittstelle, wie P-ATA mit einem Computer verbunden wird. Entsprechend muss das System über die geforderte Schnittstelle verfügen und ein Treiber für das verwendete Betriebssystem existieren. Dass Medien nicht mehr auslesbar sind, kann auch der Fall sein, wenn die Materie einem natürlichen Zerfallsprozess unterliegt oder durch äußere Einwirkung zerstört wird.
Bitströme, also Folgen von Nullen und Einsen sind für Menschen schwer zu interpretieren. Grundsätzlich ist es zwar möglich, dass beispielsweise ein Text mithilfe einer ASCII-Tabelle dekodiert wird, allerdings grenzt dies an eine Sisyphusaufgabe. Spätestens beim Lesen von formatierten Texten, sowie Musik- oder Bilddateien in digitalen Formaten ist das Ende des Möglichen für den Menschen erreicht. Doch auch Computer können Probleme bei der Darstellung der Daten haben. Für die verschiedenen Arten von Dokumenten (beispielsweise Texte, Bilder, Videos, Ton etc.) existiert eine Vielzahl an Formaten. Solche Formate stammen etwa von unterschiedlichen Softwareherstellern, sind gut oder schlecht (oder gar nicht) dokumentiert und werden häufig bis selten genutzt, womit sie möglicherweise kaum bekannt sind. Sie bieten unterschiedliche Leistungsmerkmale und werden im Laufe des technischen Fortschritts erweitert oder durch andere Formate ersetzt. Ist ein Bitstrom gelesen worden, muss bekannt sein, in welchem Format dieser kodiert ist. Entsprechend muss eine Software verfügbar sein, die damit arbeiten kann. Wird sie zusammen mit den Daten ausgeliefert,

[9] Siehe `http://www.gutenberg.org`
[10] Tagesaktueller Preis auf `http://www.alternate.de` am 13.02.2017
[11] Siehe `http://arstechnica.co.uk/gadgets/2017/01/seagate-16tb-hard-drive/` am 13.02.2017

muss sichergestellt sein, dass ein Computersystem verfügbar ist, auf dem sie ausgeführt werden kann. Dieses Problem ist vergleichbar mit der Beschaffung des Abspielgeräts.
In einigen Fällen wird Software zur Konversion von einem Format in ein anderes angeboten. So kann, etwa zum Zweck der langfristigen Archivierung von Dokumenten, die in einem Format vorliegen, welches voraussichtlich in der Zukunft keine gut Unterstützung erhalten wird, eine Überführung in ein besser geeignetes Format vorgenommen werden. Allerdings kann hierbei ein Verlust von Information auftreten. Dies ist zum einen der Fall, wenn das Zielformat ein entsprechendes Merkmal nicht aufnehmen kann. So wird, wenn man ein HTML-Dokument in XHTML überführen möchte, wahrscheinlich ein `footer`-Tag in ein `div` geändert, mit ein wenig Glück sogar in `<div class=‘‘footer’’>`. Die semantische Auszeichnung der Fußzeile wäre damit verloren gegangen. Einen anderen Grund für das Versagen einer Konversion zeigen Borghoff et al. in [4]. Bevor sich Textverarbeitungsprogramme, wie Microsoft Word etablierten, wurden Tabellen in Dokumenten in etwa so dargestellt:

```
 ------------------------------------------------------------
| Team 1                  | Punkte |  Team 2           | Punkte |
 ------------------------------------------------------------
| SV Happy Hour           | 21     | SV 26 Cammin e.V. | 13     |
 ------------------------------------------------------------
| Volley Veterans Rostock | 25     | Wismar Libre      | 20     |
 ------------------------------------------------------------
```

Diese Darstellung ist für einen menschlichen Betrachter ohne weiteres als Tabelle zu erkennen. Unterstützt wird dies, weil zu jener Zeit meist Schriftarten mit fester Zeichenbreite verwendet wurden. Tabellen wurden also eher *gemalt*, als dass man sie, wie heute üblich, semantisch als solche auszeichnet. Überführt man diese reine Textform in LaTeX, so ergibt sich das folgende Bild, bei dem offensichtlich keine Tabelle mehr erkennbar ist:

```
———————————————————————— | Team 1 | Punkte | Team 2 | Punkte |
———————————————————————— | SV Happy Hour | 21 | SV 26 Cammin e.V. | 13 |
———————————————————————— | Volley Veterans Rostock | 25 | Wismar Libre | 20 |
                                ————————————————————————
```

Mit diesem Wirrwarr von Zeichen kann weder ein Computer arbeiten, noch kann ein menschlicher Leser es *intuitiv* deuten.
Insgesamt lassen sich die Probleme der Langzeitarchivierung auf die folgenden Punkte herunterbrechen[42]:

- Das verwendete Medium zur Speicherung der Daten muss unversehrt sein, die Information also in unversehrter Weise über einen längeren Zeitraum erhalten.
- Zum Lesen des jeweiligen Mediums müssen Lesegeräte verfügbar sein, welche die Information auf eine jeweils aktuelle Datenverarbeitungsanlage transportieren können
- Die Daten – also die Bitfolge – müssen ein bekanntes, interpretierbares Datenformat darstellen. Um festzustellen, um was es sich dabei für ein Format handelt, müssen Metadaten Hinweise geben. Diese sind wiederum zu speichern, dabei ergeben sich dieselben Probleme.
- Es müssen Programme existieren, die in der Lage sind, mit den jeweiligen Datenformaten zu arbeiten

2.3 Modelle und Strategien

Dieser Abschnitt soll bekannte Modelle und Verfahren zeigen, die in der Langzeitarchivierung Anwendung finden.

2.3.1 Migration

Ein Ansatz der Langzeitarchivierung ist die Migration. Borghoff et al[4] beschreiben diese als den *periodischen Transfer digitaler Medien von einer Hard-/Softwarekonfiguration zu*

einer anderen Konfiguration, von einer Generation der Computertechnologie zur nachfolgenden Generation. Gesicherte digitale Daten werden also – entsprechend dem technologischen Wandel – stets auf neuere, aktuelle Speichermedien übertragen, wobei auch deren Format an fortschrittliche Techniken überführt werden kann.

Die regelmäßige Überführung der Archivalien auf aktuelle Technologien bringt eine Reihe von Vorteilen mit sich. Zum einen wird der Datenverlust durch altersbedingten Ausfall der Speichermedien durch die regelmäßige *Auffrischung* verringert. Die Vorteile neuer Speichertechnologien – dazu zählt in erster Linie die stetig wachsende Kapazität der Datenträger – kann nach einem Wechsel der Generation ausgenutzt werden. Bei einer periodischen Migration kann außerdem sichergestellt werden, dass stets alle Sicherungen aktuelle Techniken verwenden, im schlimmsten Fall liegen Archivalien der Vorgängergeneration vor. Damit wird die Notwendigkeit der vollständigen Dokumentation zur Interpretation der Daten – sowohl auf Hard-, wie auch auf Softwareebene – deutlich abgeschwächt.

Im Gegenzug kostet diese Variante einen enormen Aufwand. Migrationen müssen langfristig geplant und durchgeführt werden. Dies ist mit Personal- und Datenträgerkosten verbunden. Hinzu kommt, dass bei wachsenden Datenmengen der Zeitfaktor zum Kopieren der Informationen nicht zu vernachlässigen ist. Bei einer schlecht durchgeführten Migration auf ein wenig geeignetes Format besteht die Gefahr einer Datenverfälschung.

2.3.2 Emulation

Beim Ansatz der Emulation soll versucht werden, Information in ihrer Urform aufzubewahren. Daher ist es ebenfalls nötig, eine Einsatzumgebung, welche in der Lage ist, die Daten zu verarbeiten, langfristig zur Verfügung zu stellen. Diese besteht sowohl aus Hard-, wie auch aus Software. Letztere sind für gewöhnlich ebenfalls Daten, weshalb sie grundsätzlich mit denselben Techniken gesichert werden können, wie die Nutzdaten. Da die Langzeitarchivierung von Hardware aufgrund deren natürlichen Zerfall problematisch ist, kann dies in der Praxis nur durch Software bewältigt werden, die deren Funktionsweise nachempfindet.

2.3.3 Das OAIS-Referenzmodell

Beim Open Archival Information System (OAIS) handelt es sich um ein 2002 veröffentlichtes Modell für die Langzeitarchivierung, welches mittlerweile weitläufig als Referenz akzeptiert ist und von der International Organization for Standardization (ISO) zum Standard erhoben wurde. Treibende Kraft zur Entwicklung des OAIS war das Consultative Committee for Space Data Systems (CCSDS) mit Unterstützung der amerikanischen National Archives and Records Administration (NARA) und der Research Libraries Group (RLG) als Partner mit langjähriger Erfahrung im Bereich der Archivierung. Zu den bekanntesten Mitgliedern der CCSDS zählen etwa die National Aeronautics and Space Administration (NASA), die European Space Agency (ESA) und das Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). Weil bei derartigen Organisationen bereits seit den sechziger Jahren des 20. Jahrhunderts große Mengen zu archivierender, digitaler Daten auftreten, war es nötig hierfür einen gemeinsamen Standard zu erarbeiten. Auf diese Weise ist es möglich, dass verschiedene Experten und Organisationen ein gemeinsames Denk- und Referenzmodel haben und sich entsprechend austauschen können. Die Entwicklung begann 1997, federführend war hierbei die NASA. Eine erste vollständige textuelle Fassung des OAIS war das sogenannte **Red Book**, welches 1999 veröffentlicht wurde. Nach einigen wenigen Änderungen und Überarbeitungen wurde es 2001 als Normentwurf bei der ISO angenommen und 2002 als Standard ISO 14721 akzeptiert. Aktuell ist ISO 14721:2012[12] gültig.

Zur Umgebung eines OAIS-Archivs gehören Teilnehmer mit einer von drei Rollen Produzent, Nutzer und die Verwaltung. **Produzenten** dienen als Quellen für Informationen, die es zu archivieren gilt, das können reale Personen oder Client-Systeme sein. Als **Nutzer** (Consumer) werden solche Teilnehmer bezeichnet, die ein Interesse an den gespeicherten Informationen haben und diese anfordern. Auch hier können wieder Menschen, wie auch Maschinen diese Platz einnehmen. Letztendlich existiert eine **Verwaltung** (Management). Deren Aufgabe ist das Umsetzen der OAIS-Richtlinien.

[12] Siehe `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57284`

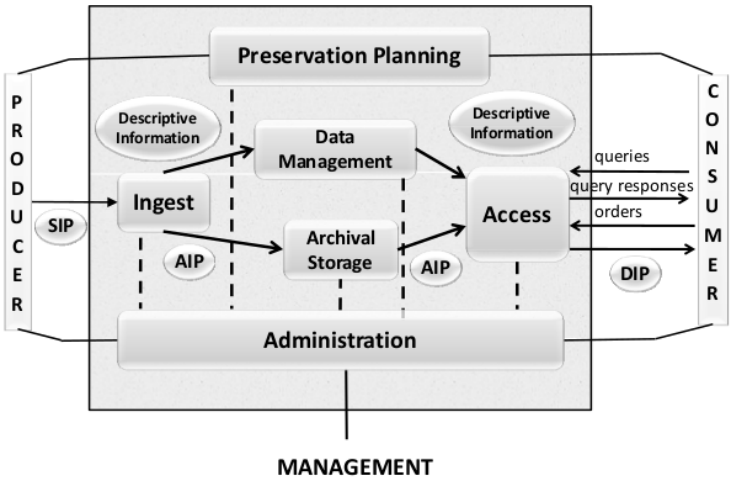


Abbildung 2.1: Funktionales Modell von OAIS[10]

Informationen im Sinne von OAIS werden immer als Daten repräsentiert. Ein Ort auf der Erde wird etwa durch seine Koordinaten im Sexagesimalsystem[13] repräsentiert, welches wiederum in ASCII kodiert ist. Um ein Datenobjekt in ein Informationsobjekt zu überführen, sind also Informationen über die Repräsentation nötig, in diesem Fall also a) die ASCII-Tabelle, b) die Notation der Koordinaten und c) Wissen um die Anwendung, also welche Stelle der Erde mit welcher Koordinate verknüpft ist. Diese Beschreibung der Repräsentation kann mitunter komplex werden, da sie selbst aus nicht-trivial zu interpretierenden Daten bestehen kann. Es ist daher genau zu beachten, welches minimale Verständnis die designierte Nutzergruppe der Information hat.
OAIS führt das Konzept der **Informationspakete** (Information Package) ein. Ein Informationspaket ist dabei ein konzeptueller Container, welcher sowohl Inhalt (Content Information) und Informationen zur Speicherung (Preservation Description Information (PDI)) enthält. Die PDI enthalten dabei Informationen zu Herkunft, kontextuelle Einordnung, weitere bekannte Referenzen auf das Objekt und Zugriffsrechte. Zur Wahrung der Integrität (Fixity) sollen entsprechende Maßnahmen getroffen, werden, etwa Checksummen. Informationspakete werden durch Beschreibungsinformationen (Descriptive Information) referenziert. Diese können, je nach Einsatzumgebung, umfangreicher ausfallen und eine Menge indizierbarer Attribute enthalten.
Konkret gibt es drei Varianten von Informationspaketen. Submission Information Packages (SIPs) werden solche Informationspakete genannt, die durch einen Produzenten für die Archivierung eingereicht werden. Die Inhalte von SIPs existieren stets in ihrer Originalform. Deshalb und weil es ohne Weiteres eine Vielzahl von Produzenten verschiedenster Art gibt, liegen diese mit hoher Wahrscheinlichkeit in heterogener Form vor. So kann ein Produzent A Fotos im JPG-Format einreichen, während ein Produzent B dies im PNG-Format tut. Selbstverständlich kann auch ein einzelner Produzent verschiedene Formate verwenden. Aus einem oder mehreren SIPs, sowie etwaigen Metadaten entsteht ein Archival Information Package (AIP). In dieser Form werden die Informationen dauerhaft systemintern gespeichert. Dabei findet eine Transformation statt, die eingereichten Daten werden in eine homogene Form gebracht. Im Falle von Bilddateien kann dies beispielsweise das Tagged Image File Format (TIFF) gewählt werden. Fordert ein Nutzer Informationen an, so wird ein Dissemination Information Package (DIP) an diesen zurückgegeben. DIPs werden aus einem oder mehreren AIPs generiert. Dabei sind diese immer an die anfordernde Nutzergruppe angepasst, so werden beispielsweise bevorzugte Formate oder Einschränkungen durch Zugangsrechte beachtet.
Das **funktionale Modell** von OAIS umfasst sechs Aufgabenbereiche. Abbildung 2.1 gibt hierzu einem Überblick. Die **Datenübernahme** (Ingest) stellt Maßnahmen zur Aufnahme von Inhalten in das System dar. Dies beginnt mit der Vorbereitung, wobei Schnittstellen definiert, Funktionen geschaffen und akzeptierte Formate festgelegt werden müssen. Einge-

[13] Gemeint ist hier die übliche Angabe geografischer Längen und Breiten in Grad, Minuten und Sekunden. Zum Beispiel: 54°04´39.9″N 12°06´23.5″E

gangene SIPs müssen einer Qualitätssicherung durchzogen werden, so ist sicherzustellen, dass diese lesbar, verständlich und korrekt sind. Es sind entsprechende Metadaten zu sammeln und zusammen mit dem oder den SIPs in ein AIP zu überführen. Der **Archivspeicher** (Archival Storage) befasst sich mit allem, was zur Speicherung der Daten notwendig ist. Dazu gehört die Art der Datenträger, deren Organisation, Prozessbeschreibungen zum Speichern von AIPs nach Übernahme, sowie regelmäßige Wartung und Prüfung. Obwohl es nicht zwingend vorausgesetzt wird, rät das OAIS zur redundanten Lagerung auf verschiedenartigen Datenträgern, sowie – sofern es angebracht ist – zur regelmäßigen Migration auf neue Speichertechnologien. Für die Erhaltung, Wartung und Zugänglichkeit von Verzeichnis- oder Beschreibungsinformationen ist die **Datenverwaltung** (Data Management) zuständig. Dies umfasst vor allem auch das Aktualisieren von Einträgen und die Bearbeitung von Anfragen auf Inhalte. Ebenso ist eine **System-Verwaltung** (Administration) notwendig. Diese kümmert sich um die allgemeine Lauffähigkeit des Archivsystems. Dazu gehören Verhandlungen und Vereinbarungen mit Produzenten, die Festlegung von Zuständigkeiten, Hard- und Softwaremanagement, sowie die Prüfung auf Einhaltung von Archivstandards. Weiterhin bietet diese Funktionseinheit Überwachungsfunktionen, etwa Nutzungsstatistiken, Inventuren, Benachrichtigungen zum Archivbetrieb und -status. Zur Wahrung der dauerhaften Funktionalität wird die **Archivplanung** (Preservation Planning) eingesetzt. Die Archivplanung beobachtet die Umgebung des Systems und gibt anhand dessen Empfehlungen und Pläne zur längerfristigen Informationserhaltung ab. So kann etwa durch externe Stellen ein neues Speichermedium verfügbar gemacht werden, dieses ist durch diese Funktionseinheit zu evaluieren und entsprechende Migrationspläne zu erarbeiten. Auch eine Veränderung der Nutzerschaft kann bedeuten, dass die ausgegebenen DIPs entsprechend angepasst werden müssen. Letztendlich muss der **Zugriff** (Access) geregelt werden. Dazu zählt die Schaffung von Techniken, die den verschiedenen Nutzergruppen ermöglichen, Inhalte zu finden und anzufordern, bei gleichzeitiger Wahrung von Zugriffsrechten.

Kapitel 3

Stand der Technik

3.1 Metadaten und Modelle

Metadaten beschreiben Daten. Wir werden mit ihnen im Alltag häufig konfrontiert. Zu einer Datei auf der Festplatte werden, in Abhängigkeit vom verwendeten Dateisystem, etwa der Dateiname, der Besitzer, Zugriffsrechte, diverse Zeitstempel und die Größe gespeichert. Beschreibende Daten einer E-Mail sind etwa Absender, Empfänger, Versandzeit, Betreff, MIME, der Versandweg und (optional) eine kryptographische Signatur, wobei letztere als S/MIME oder OpenPGP vorliegen kann. Möchte man ein Gemälde beschreiben, so fallen hier schnell der Name, der Künstler, verwendete Techniken, Leinwandgröße und die Schaffenszeit ein, weiterhin kann auch der Verlauf der bisherigen und aktuellen Besitzer und Aussteller von Interesse sein. Über den Künstler kann weit mehr bekannt sein, als der Name, etwa dessen Lebenszeit, Nationalität, sein Lebenslauf und gesprochene Sprachen. Führt man dies weiter, können auch Zeitabschnitte, Nationen, Sprachen und so weiter aufgeführt werden. Den Vorteil von gut verwalteten Metadaten kostet man spätestens dann aus, wenn man etwa *Aquarelle von Künstlern aus sozialistischen Nationen, die nach 1960 verstarben* suchen möchte.
Zur Beschreibung von Metadaten wurden in den letzten Jahrzehnten verschiedenste Techniken entwickelt, einige davon existieren bis heute, andere sind längst wieder in Vergessenheit geraten. Bisher hat sich, wie so häufig, keine Technik als die ewig währende und einzige Lösung erwiesen. Van Hooland et al. geben in [26] einen relativ aktuellen Überblick über Techniken von Metadaten – insbesondere zur Verwendung in Bibliotheken, Archiven und Museen. Der folgende Abschnitt orientiert sich weitestgehend an diesem Buch.
Immer, wenn Daten die erfasst, gespeichert, verarbeitet und ausgegeben werden sollen, muss dies in einer gewissen, zuvor festgelegten Struktur erfolgen. Dies ist auch bei Metadaten der Fall. Wie diese Struktur aussieht, beschreibt das gewählte Metadatenmodell, wobei es sich bei diesem erwartungsgemäß um eine vereinfachte Abbildung der Wirklichkeit handelt. Im Laufe der Zeit haben sich vor allem vier dieser Modelle etabliert: Tabellarische Daten, Relationale Modelle, Meta-Markup-Sprachen und verknüpfte Daten. Diese haben jeweils ihre eigenen Vor- und Nachteile, weshalb man mitnichten behaupten könnte, dass ein Modell das andere ersetzen kann oder dies bereits getan hat.
Als laufendes Beispiel zur Veranschaulichung der Datenmodelle seien in diesem Abschnitt Freizeitvolleyballturniere gewählt. Da der Autor der Arbeit regelmäßig an der Organisation solcher beteiligt ist, kann daher auf eine gewisse Datengrundlage[1] und Domänenwissen zurückgreifen. Turniere werden fortlaufend nummeriert und finden an einem bestimmten Datum statt. Bei der zu betrachtenden Veranstaltung handelt es sich um das 32. Freizeitvolleyballturnier, welches am 14.11.2015 stattfand. Ein Turnier unterteilt sich in mehrere Runden. Für jede Runde werden die teilnehmenden Mannschaften in Gruppen unterteilt. Innerhalb dieser Gruppen begegnen sich die Teams und treten gegeneinander an. Jede Begegnung umfasst in der Regel zwei Sätze, wobei aufgrund einer Hausregel unter Umständen auch drei Sätze gespielt werden können. Die Punktzahlen der Mannschaften im jeweiligen Satz werden gespeichert. Für einen gewonnenen Satz erhält das führende Team zwei Punkte, bei einem Unentschieden bekommen beide Teilnehmer einen Punkt. Die resultierenden Platzie-

[1] Daten für das zu betrachtende Volleyballturnier können unter folgender URL eingesehen werden (Stand 05.02.2017): `http://www.sv-fortuna-rostock.de/volleyballturniere/32/`

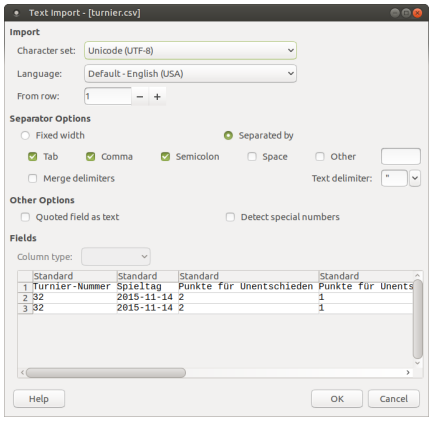


Abbildung 3.1: Importoptionen einer CSV-Datei in LibreOffce Calc

rungen sollen nicht explizit in die Betrachtung aufgenommen werden, es sein angenommen, dass diese aus den Ergebnissen der Begegnungen hergeleitet werden können. Das Turnier wird mithilfe einer Software[2] ausgerichtet, welches einen Export der Turnierergebnisse in HTML erlaubt. Die dabei entstehende Ausgabe ist allerdings nur unzureichend semantisch ausgezeichnet.

3.1.1 Tabellenmodelle

Als einfachste und intuitivste Variante zur strukturierten Darstellung von Daten sind Tabellen zu nennen. Sollen Metadaten auf diese Weise dargestellt werden, definiert man zuerst die notwendigen Felder, den Tabellenkopf, und trägt die Daten, pro Entität eine Zeile, in die Tabelle ein. Dabei ist es ohne Weiteres möglich, den Spalten einen Typ zuzuweisen, etwa eine Zahl, ein Datum oder ein Text. Dieses selbst-erklärende Datenmodell wird bereits seit Jahrhunderten verwendet.
In der modernen Datenverarbeitung ist sind Dateien im CSV-Format beliebt. Dieses wurde im RFC 4180 festgehalten. Da die Standardisierung zu dem Zeitpunkt geschah, als bereits viele Varianten des Formats existierten, hält sich Software nur selten strikt an die Vorgaben des RFC, stattdessen sind häufig Konfigurationsmöglichkeiten gegeben. Abbildung 3.1 zeigt einen Importdialog der Software LibreOffice Calc. Die wichtigsten Gemeinsamkeiten von CSV und seinen Dialekten sei wie folgt zusammengefasst: In einer Textdatei, deren Codierung nicht explizit festgelegt ist, werden zeilenweise Entitäten abgelegt. Durch ein zuvor vereinbartes Trennzeichen, in der Regel ist dies ein Komma (‘,’), werden innerhalb einer Zeile die Felder getrennt. Die Reihenfolge der Felder ist dabei innerhalb einer Datei stets gleich, optional kann die erste Zeile als Kopfzeile betrachtet werden. Einzelne Felder können in Anführungszeichen zusammengefasst werden, auf diese Weise ist es möglich, Zeichenketten zu speichern, die das Trennzeichen enthalten. Außerdem ist es in einigen Dialekten erlaubt, den Dezimaltrenner zu ändern. Populäre Software für Anwender ist neben dem erwähnten LibreOffice auch Microsoft Excel. Viele Programmiersprachen bieten Bibliotheken zur Verwendung von CSV-, etwa Java[3], Python[4] und GNU R[5].
Tabelle 3.1 zeigt einen Modellierungsversuch des Turnier-Beispiels. Aus Platzgründen wurde die Anzahl der Punkte für Siege und Unentschieden ausgelassen. Die Ansicht wurde aus der für den nächsten Abschnitt nötigen relationalen Modellierung generiert. Schnell werden die Kosten sichtbar, mit der die Einfachheit und Intuitivität dieser Methode erkauft werden: Redundanz mit allen ihren Nachteilen. Die Anzahl der Zeilen entspricht den total gespielten Sätzen. Für das 32. Freizeitvolleyballturnier waren das insgesamt über 100. Entsprechend werden in jeder Zeile identische Informationen zum Turnier redundant gespeichert, dazu gehören die laufende Nummer, das Datum, die Punktkonfiguration und – sollte man sich dazu entscheiden, dies mit aufzunehmen – der Name des Veranstalters. Damit einher gehen eine Menge weiterer, allgemein bekannter Probleme. Müssen für jeden Satz die Teamnamen

[2]Siehe `http://www.turnierprogramm.de/`
[3]Apache Commons CSV: `https://commons.apache.org/proper/commons-csv/`
[4]Python Standard Library: `https://docs.python.org/3/library/csv.html`
[5]Siehe: `https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html`

T	Spieltag	R	G	Team 1	P	Team 2	P
32	2015-11-14	1	1	SV Happy Hour I	21	SV 26 Cammin e.V.	13
32	2015-11-14	1	1	SV Happy Hour I	19	SV 26 Cammin e.V.	16
32	2015-11-14	1	1	Zollsportgem. . . Rostock	11	Volley Vet. . . Rostock	21
32	2015-11-14	1	1	Zollsportgem. . . Rostock	6	Volley Vet. . . Rostock	21
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
32	2015-11-14	4	8	Volley Vet. . . Rostock	25	Wismar Libré	20
32	2015-11-14	4	8	Volley Vet. . . Rostock	21	Wismar Libré	25

Tabelle 3.1: Ergebnisse des Volleyballturniers in Tabellenform. Es wurden folgende Abkürzungen verwendet: R - Runde, G - Gruppe, P - Punkte

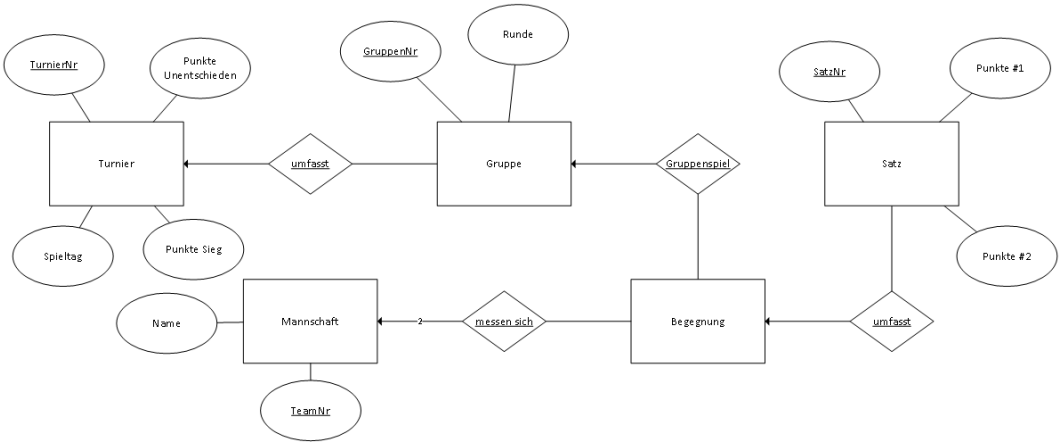


Abbildung 3.2: Modellierung des Volleyballturniers als relationales Modelle mithilfe von ERM

einzeln eingetippt werden, so können Schreibfehler entstehen, die zu Inkonsistenzen führen. Führt man dann eine Abfrage zur Ermittlung der Gesamtpunktzahl der Mannschaft `Wismar Libré` durch, ein Mitarbeiter hat jedoch in einigen Fällen `Wismar Libre` in das System eingepflegt, so werden entsprechende Werte unterschlagen, ohne dass dies für den Nutzer in irgendeiner Weise ersichtlich ist. Änderungen von redundanten Informationen müssen immer in allen ihren Vorkommen verändert werden. Auch beim Suchen nach Informationen muss stets die gesamte Tabelle betrachtet werden, was in schlechter Leistung resultiert. Das Hinzufügen oder Entfernen einer Spalte führt hingegen kaum zu Problemen.

3.1.2 Relationale Modelle

Aufgrund der Unzulänglichkeiten von Tabellenmodellen, die in vielen Einsatzszenarien zutage kommen, wurden gegen Ende der 1970er Jahre relationale Modelle entwickelt. Diese sind bis heute im Datenbankbereich ein etablierter Standard, welcher regelmäßig durch aktuelle Techniken ergänzt wird. Hierbei werden zuerst die vorhandenen Entitätstypen identifiziert, für das Turnierbeispiel ließen sich etwa die Typen `Turnier`, `Mannschaft`, `Begegnung` und `Satz`, aber natürlich auch weitere finden. Für jeden Typen existiert genau eine Tabelle, in der jede Instanz oder Entität genau einmal vorkommt. Zwischen den Tabellen existieren zuvor definierte Relationen. Abbildung 3.2 zeigt einen Vorschlag zur relationalen Modellierung des laufenden Beispiels als Entity Relationship Model (ERM). Ein ERM ist eine verbreitete, grafische Darstellungsform relationaler Modelle und ein häufig verwendetes Entwurfsverfahren. Weiterhin existieren etwa das Dekompoistions- oder das Syntheseverfahren[43].
Die Implementation derlei Modelle ist ein äußerst umfangreiches Gebiet. Relevante Probleme hierbei sind etwa Dateiorganisation & Zugriffsstrukturen, Nutzung von Pufferspeichern, Algorithmen zur Anfragebearbeitung und -optimierung. Weiterhin existieren Transaktionskonzepte zur Integritätssicherung. Einen ausführlichen Überblick zum Thema liefern Saake et al. in [42]. Implementierende Software wird als Relational Database Management Software (RDMS) bezeichnet, häufig wird dabei ein Server-Client-Modell verwendet. Dies hat unter

anderem den Vorteil, dass Datenbestände zentral an einem Ort gehalten werden, während eine potentiell große Anzahl von Nutzern gleichzeitig damit arbeitet und dennoch die ACID-Eigenschaften eingehalten werden. Populäre Vertreter von RDMS sind etwa *Oracle MySQL*, *Microsoft SQL Server* oder *PostgreSQL*. Programmiersprachen umfassen häufig Schnittstellen für die genannten Systeme. Diese ermöglichen es, zumeist über die üblichen Netzwerkprotokolle, Anfragen an die Systeme zu senden um damit Daten abzufragen oder zu manipulieren. Zur lokalen Nutzung existiert etwa die Programmbibliothek *SQLite*, welche unter anderem für C, Java und Python verfügbar ist.
Im gleichen Atemzug, wie relationale Datenmodelle, muss in der heutigen Zeit die Datenbanksprache Structured Query Language (SQL) genannt werden. Diese ist aktuell als **SQL 2011** (ISO/IEC 9075-1:2011) verfügbar, eine Revision ISO/IEC 9075-1:2016 ist in Arbeit. Entsprechend gängiger Datenbanksprachen setzt sich SQL aus den Teilen *Datendefinition* (DDL), *Datenmanipulation* (DML) und *Datensteuerung* (DCL) zusammen. Das bedeutet, mithilfe von SQL ist es möglich Datenbankschemata zu definieren und zu verändern, Daten einzugeben, zu manipulieren und zu löschen. Annähernd die gesamte Datenbanksteuerung ist hiermit möglich. Hervorzuheben ist, dass es dieses Werkzeug ermöglicht, mit wenig Aufwand komplexe Anfragen an das System zu stellen. Aufgrund von Anfragenoptimierungen und Zugriffsstrukturen seitens der RDMS werden diese mit einer hohen Effizienz ausgeführt. Obwohl die Sprache standardisiert ist, verwenden die verschiedenen Implementationen eigene Varianten mit erweitertem Funktionsumfang. Diese sind untereinander nur bedingt kompatibel. Während es in reinem SQL wohl möglich wäre, das gesamte relationale Modell, mit allen darin enthaltenen Daten zu serialisieren[6] und damit für einen Austausch oder eine langfristige Archivierung aufzubereiten, bereiten diese Inkompatibilitäten mitunter Probleme bei der längerfristigen *Offline*-Archivierung oder beim Austausch mit anderen Datenbanken.

3.1.3 Meta-Auszeichnungssprachen

Auszeichnungssprachen (Markup Languages) sind ein mögliches Datenmodell für sogenannte **semistrukturierten Daten**, welche sich insbesondere dadurch auszeichnen, dass sie eine wechselnde oder weniger streng typisierte Struktur aufweisen, als andere Datenmodelle. Sie weisen eines oder mehrere der folgenden Merkmale auf: Semistrukturierte Daten speichern das zugrundeliegende Schema nicht zentral in einem Datenbankkatalog, sondern direkt im vorliegenden Dokument. Zwischen Dokumenten, die gleichartige Objekte beschreiben, können dennoch Strukturunterschiede bestehen, etwa durch den Wegfall von Elementen oder die Veränderung von Datentypen. Die Einhaltung von Datentypen zählt außerdem nicht zwingend als Integritätsbedingung. Weitere Merkmale listet [43].
Datenmodellen von Auszeichnungssprachen liegt zumeist eine Graphstruktur zugrunde, genauer kann man sogar von einer Baumstruktur sprechen. Dabei entsprechen die Knoten den zusammengesetzten Objekten, Blätter Werten und Kanten enthalten Beschreibungen der Knoten-Objekte. Ein Knoten wird als Wurzel des Baumes festgelegt. Mithilfe von Auszeichnungssprachen kann ein solcher Baum modelliert werden. Der bekannteste Vertreter ist wohl HTML[7]. Dabei handelt es sich um eine Beschreibungssprache für semistrukturierte Textdokumente, genauer für Websites. Aktuell liegt diese in der Version 5 vor. Es sei allerdings gesagt, dass sich HTML5 zum einen in ständiger Weiterentwicklung befindet und zum anderen häufig als Sammelbegriff verschiedenster Techniken zur Entwicklung von Websites benutzt wird. Dazu gehören neben HTML in der aktuellen Version auch CSS3 und JavaScript mitsamt AJAX. Alle weiteren Ausführungen beschränken sich ausschließlich auf die Auszeichnungssprache. Diese ermöglicht mithilfe der sogenannten *Tags* eine Auszeichnung von Textteilen zu erwirken, welche den Ansprüchen von Textauszeichnungen genügen und darüber hinausgehen: Überschriften verschiedener Ordnung, Absätze, Tabellen und so weiter. Wurde insbesondere in der Anfangszeit HTML häufig als *Makeup* missbraucht[8], wird das Aussehen heutzutage in der Regel mithilfe von Cascading Style Sheets (CSS) definiert. Gerade mit HTML5 wurden viele neue Tags zur semantischen Auszeichnung eingeführt[24]. Die Platzhirsche im Internet, dazu zählt der Autor etwa Google oder Facebook, regen die

[6] So existiert beispielsweise für Oracle MySQL die Software `mysqldump`, die dies übernimmt.
[7] Aufgrund dieses Bekanntheitsgrades soll im Rahmen dieser Arbeit auf eine genaue Definition der Sprache, sowie auf Codebeispiele verzichtet werden. Der Autor geht davon aus, dass der Leser zumindest mit dem Erscheinungsbild von HTML, wie auch im späteren verlauf von XML vertraut ist.
[8] Hervorzuheben sind hier das `font`-Tag und die Verwendung von Tabellenlayouts

```
<?xml version="1.0" encoding="utf-8"?>
<TOURNAMENTDATABASE>
  <TEAMS>
    <TEAM id="1" name="SV Happy Hour I" />
    <TEAM id="2" name="Zollsportgemeinschaft Rostock" />
    <TEAM id="3" name="SV 26 Cammin e.V." />
    <TEAM id="4" name="Volley Veterans Rostock" />
    <TEAM id="5" name="Wismar Libr\'e" />
  </TEAMS>
  <TOURNAMENTS>
    <TOURNAMENT nr="32" date="2015-11-14" victory_points="2"
      ↪ draw_points="1">
      <ROUND number="1">
        <GROUP number="1">
          <ENCOUNTER team_id1="1" team_id2="3">
            <MATCH nr="1" points1="21" points2="13"/>
            <MATCH nr="2"points1="19" points2="16"/>
          </ENCOUNTER>
          <ENCOUNTER team_id1="2" team_id2="4">
            <MATCH nr="1" points1="11" points2="21"/>
            <MATCH nr="2" points1="6" points2="21"/>
          </ENCOUNTER>
        </GROUP>
      </ROUND>
      <ROUND number="4">
        <GROUP number="8">
          <ENCOUNTER team_id1="4" team_id2="5">
            <MATCH nr="1" points1="25" points2="20"/>
            <MATCH nr="2" points1="21" points2="25"/>
          </ENCOUNTER>
        </GROUP>
      </ROUND>
    </TOURNAMENT>
  </TOURNAMENTS>
</TOURNAMENTDATABASE>
```

Listing 3.1: Die Turnierdatenbank als XML

Ersteller von Webseiten außerdem immer weiter dazu an, ihre Inhalte semantisch auszuzeichnen. Allerdings zählen die dazu verwendeten Techniken, wie das Open Graph Protokoll[9] oder JSON-LD[10] zumindest teilweise besser im nächsten Abschnitt anzusiedeln sind. Der Sprachumfang von HTML ist fest definiert, es können also nicht ohne weiteres eigene Tags hinzugefügt werden.
An dieser Stelle kommen die **Meta-Auszeichnungssprachen** ins Spiel. Diese stellen weniger eigene Sprachen, sondern viel mehr Syntax und Grammatik zum Generieren eigener Auszeichnungssprachen zur Verfügung. Die durch das World Wide Web Consortium (W3C) entwickelte Standard Generalized Markup Language (SGML) wurde 1986 von der ISO standardisiert. SGML war auf lange Sicht nicht erfolgreich. Stattdessen etablierte sich dessen Weiterentwicklung, die eXtensible Markup Language (XML), welche aktuell in der Version 1.1 von 2006 vorliegt. XML ermöglicht die Definition des zuvor beschriebenen hierarchischen und graphenorientierten Datenmodells, wobei jedoch nicht die Kanten, sondern die Knoten bezeichnet werden. Für diese gibt es mehrere Ausführungen, konkret handelt es sich dabei um Elemente, Attribute, Kommentare und Verarbeitungsanweisungen (processing instruction)[43]. Elemente werden durch Tags ausgedrückt, welche wiederum Attribute in Form von *Name-Wert*-Paaren enthalten. Zur Definition einer Ausprägung von XML gibt es zwei Möglichkeiten: Bei der Document Type Definition (DTD) handelt es sich um die ur-

[9] Siehe `http://ogp.me/`
[10] Siehe `http://json-ld.org/`

sprüngliche Variante, welche von XML geerbt wurde. Es handelt sich um eine formale Grammatik zur Festlegung der erlaubten Tags und deren mögliches Auftreten definiert. Allerdings verfügt dieser Ansatz nur über eine begrenzte Ausdruckskraft, die nicht immer ausreichend ist. So ist es etwa nicht möglich, bestimmte Datentypen vorauszusetzen oder weitere Integritätsbedingungen festzulegen. Aus diesem Grund wurde die XML Schema Definition (XSD) entwickelt. Interessanterweise handelt es sich dabei wiederum um eine XML-Sprache. Mithilfe von Schemata lassen sich XML-ausgezeichnete Dokumente validieren. Problematisch ist das Verändern von Schemata; wird dies getan, sind zuvor definierte Dokumente nicht mehr zwangsläufig valide. Daher empfiehlt es sich, Schemata entsprechend zu versionieren. Das Festlegen von eines Schema regt selbstverständlich zum wiederverwenden bereits existierender Definitionen an. Damit dies problemlos möglich ist, werden Namensräume eingeführt, um Mehrdeutigkeiten zu verhindern, beispielsweise wenn das Element `title` mehrfach existiert: Einmal als Name in einem Buch-Schema und einmal als akademischer Grad in einem Personen-Schema. Der benutzte Namensraum wird im Tag vor dessen Namen, getrennt durch einen Doppelpunkt, notiert. So verwendet XSD den Namensraum `xs`.
Zur Verwendung von XML-Dokumenten gibt es grundsätzlich zwei Ansätze. Der erste umfasst, so trivial es klingen mag, das Parsen der Datei und die Verwendung der softwareinternen Repräsentation. Entsprechende Parser sind in vielen Standard- oder Zusatzbibliotheken geläufiger Programmiersprachen vorhanden. Als zweite Variante bietet sich der datenbankbasierte Ansatz, bei dem XML als Datenmodell genutzt wird. Zur Navigation im Datenbaum existieren die Anfragesprachen **XPath** und **XQuery**. Grundsätzlich ist eine Abbildung von XML auf relationale Datenmodelle möglich, 2003 wurde der SQL-Standard um Funktionalitäten zur Arbeit mit XML erweitert.
XML ist so konzipiert, dass es gleichermaßen von Mensch und Maschine gelesen werden kann. Ein großer Vorteil ist die Flexibilität bei der Modellierung der Daten, so können gleiche Umstände in unterschiedlicher Weise dargestellt, nach Belieben erweitert oder gekürzt werden. Dies kann auch gleichzeitig als Nachteil ausgelegt werden. Eine häufige Diskussion ist die Frage, ob Entitäten als Element oder als Attribut modelliert werden sollen. In der Praxis hat sich eine Mischform etabliert, bei der Werte, die nicht weiter zerlegt werden müssen, als Attribut dargestellt werden, alle übrigen als Element. Problematisch ist, dass bei zunehmender Tiefe der Hierarchie die Lesbarkeit für den Menschen leidet.
Mittlerweile hat sich die Java Script Object Notation (JSON) einen festen Platz neben XML erkämpft. Wie der Name bereits impliziert handelt es sich dabei um eine natürliche Notationsweise von Objekten der im Web häufig verwendeten Sprache Java Script. Die Syntax ist einfacher und für den Menschen leichter lesbar, gleichzeitig ist das Format noch flexibler, da es nicht auf feste Schemata angewiesen ist, wodurch allerdings gleichzeitig Probleme im Austausch von Daten mit externen Systemen entstehen können.

3.1.4 Verknüpfte Daten und RDF

Den bisher genannten Ansätzen liegt immer die Annahme zugrunde, dass Datensätze geschlossen an einem Ort abgelegt werden[11]. Verknüpfte Daten (*linked data*) beachten den Umstand, dass Informationen häufig über heterogene Systeme verteilt sind und gleichartige Informationssammlungen nicht zwangsweise denselben Konventionen folgen. Als prädestiniertes Beispiel seien hier Informationen auf Websites genannt. Diese enthalten im Allgemeinen unstrukturierte Daten, sodass Suchmaschinen nur eine Suche nach Schlüsselwörtern durchführen könnten. Eine Suche nach dem Begriff `Turnier` zeigt unweigerlich Ergebnisse für eine Wettkampforientierte Veranstaltung, auch wenn man eigentlich nach einer speziellen Graphklasse sucht. Gleichzeitig werden Seiten nicht angezeigt, die sich thematisch mit der gesuchten Sache beschäftigt, diese aber unter anderem Namen auftreten (etwa in einer anderen Sprache). Letztendlich weisen Datenobjekte für gewöhnlich Verbindungen zu anderen Objekten auf, welche ohne Weiteres an einem völlig anderen Ort lokalisiert sind. Da diese häufig außerhalb des eigenen Wirkungsbereichs liegen, ist es außerdem möglich dass sie sich verändern. Analog dazu haben verschiedene Archive, Museen und Bibliotheken jeweils eigene Datensammlungen, für dessen institutionsübergreifende Verknüpfung durchaus Bedarf besteht.
Aus diesem Grund wurde 2004, nach einem Vorschlag von Tim Berners-Lee vom W3C das Resource Description Framework (RDF) entwickelt. Bei RDF wird ein simples, gerichtetes Graphenmodell verwendet, bei dem *Dinge* als Knoten und Relationen zwischen diesen

[11] Techniken der Replikation zur Erhöhung von Leistung und Ausfallsicherheit seien hier außen vor gelassen.

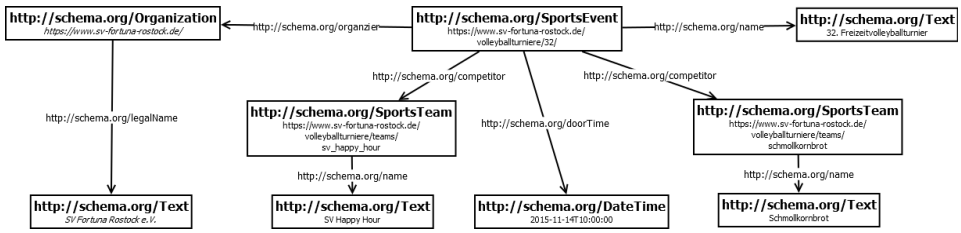


Abbildung 3.3: Mannschaften nehmen an einem Turnier teil, modelliert in RDF mithilfe des `schema.org`-Vokabulars

als Kanten abgebildet werden. Dieser Graph setzt sich aus *Subjekt-Prädikat-Objekt*-Tripeln zusammen.
Abbildung 3.3 zeigt insgesamt acht solcher Tripel: a) Es gibt eine Organisation mit dem Namen "SV Fortuna Rostock e.V." b) Diese Organisation hat ein Turnier organisiert c) Das Turnier trägt den Titel "32. Freizeitvolleyballturnier" d) Team1 heißt "Schmollkornbrot" e) Team2 heißt "SV Happy Hour" f) Team1 hat am Turnier teilgenommen g) Team2 hat am Turnier teilgenommen h) Das Turnier fand am 14.11.2015 um 10:00 Uhr statt.. Auf diese Weise lassen sich komplexe Zusammenhänge ausdrücken. Berners-Lee formulierte zur Verwendung von verknüpften Daten folgende vier Regeln:

1. Zur Bezeichnung von Objekten sollen Unified Resource Identificators (URIs) genutzt werden
2. Genauer sollen HTTP URIs genutzt werden. So ist es möglich, dass ein Objekt direkt aufgerufen werden kann.
3. Ruft man ein Objekt auf, sollen einem nützliche Informationen präsentiert werden.
4. Letztendlich sollen diese Daten auf wiederum andere Objekte verweisen, sodass sie entdeckt werden können.

In unserem Beispiel könnte der Uniform Resource Locator (URL) für das Volleyballturnier beispielsweise `http://www.sv-fortuna-rostock.de/volleyballturniere/32/` lauten. Ruft man diesen auf, so sollte man auf strukturierte Daten treffen, die dieses Turnier beschreiben. Hierfür gibt es mittlerweile eine Vielzahl an Techniken, etwa Terse RDF Triple Language (Turtle), N-Triples oder RDF/XML. Gerade im Webbereich empfiehlt Google die Nutzung von JSON for Linking Data (JSON-LD)[12].

3.2 Speichermedien

3.2.1 Konventionelle digitale Speichermedien

Derzeit existiert ein breites Angebot an konventionellen Speichermedien für digitale Daten. Diese können entsprechend ihres Einsatzgebietes in zwei Gruppen unterteilt werden: Sekundär- und Tertiärspeicher. Primärspeicher finden hier aufgrund ihrer Flüchtigkeit keine Beachtung.
Zu den **Sekundärspeichermedien** zählen gegenwärtig vor allem Festplatten und Solid State Disks (SSDs). Sie sind im Vergleich zum Primärspeicher, also dem Hauptspeicher und Cache, langsam in Bezug auf Lese- und Schreibgeschwindigkeit, bieten aber dafür ungleich höhere Speicherkapazitäten zu einem wirtschaftlichem Preis. Wichtiges Merkmal ist außerdem, dass diese Speicher nicht-flüchtig sind, Informationen bleiben also auch dann erhalten, wenn die Datenträger von ihrer Stromversorgung getrennt oder ausgeschaltet werden. Die langsameren **Festplatten** bestehen aus mehreren magnetischen Platten, sowie der entsprechenden Anzahl von Lese- und Schreibköpfen. Da letztere für die Erfüllung ihres Zwecks bewegt werden müssen, unterliegt die zugrundeliegende Mechanik einem gewissen Verschleiß, welcher zu Defekten und Datenverlust führen kann.
Bewegliche Teile entfallen bei den schnelleren, aber derzeit teureren SSDs. Diese basieren häufig auf Synchronous Dynamic RAM (SDRAM) oder Flash-Bausteinen. Während in ihrer

[12]Siehe `https://developers.google.com/search/docs/guides/intro-structured-data`

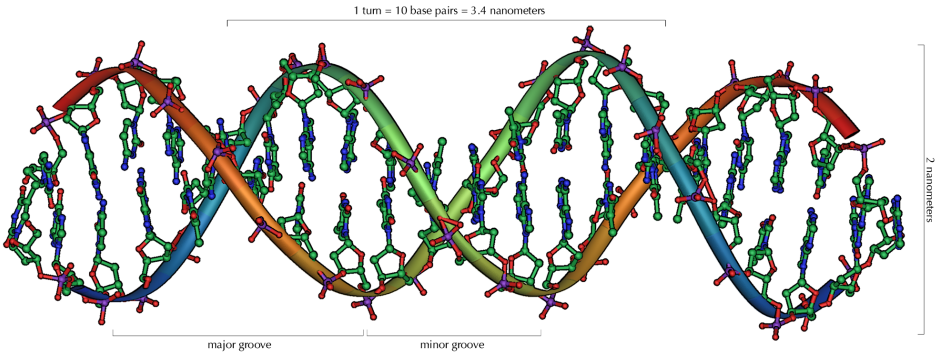


Abbildung 3.4: Schematische Darstellung der Doppelhelixform der DNS

Anfangszeit, besonders im Konsumentenbereich, häufig Bedenken geäußert wurden, dass dieser von Art von Datenträgern aufgrund der begrenzten Anzahl von Schreibzugriffen der Speicherzellen keine hohe Lebensdauer zu erwarten ist, besagt der gegenwärtige Konsens, dass – nicht zuletzt aufgrund gegensteuernder Techniken, wie dem Wear Leveling – innerhalb der normalen Lebensdauer kein nennenswerter Unterschied besteht.
Sekundärspeicher bieten heutzutage bereits Kapazitäten im zweistelligen Terabyte-Bereich. Zur Erhöhung der Lese- und Schreibraten, sowie zur Absicherung gegen Ausfall im Betrieb haben sich verschiedene Varianten von Redundant Arrays of Independend Disks etabliert. Sie eigenen sich allerdings nur bedingt zur längerfristigen Sicherung von Daten, wenn keine regelmäßige Erneuerung des Mediums stattfindet. Festplatten sind empfindlich gegen magnetische Einwirkungen, mechanische Belastungen und ungünstige Umgebungsbedingungen, wie etwa Luftfeuchtigkeit. Aufgrund der relativ kurzen Zeit auf dem Markt gibt es noch wenige aussagekräftige Erfahrungen zur langfristigen Sicherheit von SSDs, jedoch wird hier auch kaum eine Dauer von mehr als zehn Jahren angenommen.
Tertiärspeicher bieten in der Regel mehr günstige Kapazität an, diese wird allerdings mit erhöhten Zugriffszeiten erkauft. Sie werden häufig zum Transport von Daten oder als Backupmedium verwendet. Die bekanntesten, noch gebräuchlichen Vertreter sind optische Speicher, wie Compact Discs (CDs), Digital Versatile Discs (DVDs) und Blu-rays, sowie Magnetbänder.
Bei den optische Datenträgern handelt es sich in der Regel um runde, rotierende Scheiben, welche mit – für das menschliche Auge unsichtbaren – Vertiefungen ausgestattet sind. Rotiert diese Scheibe über einem geeigneten Laser, so können Bitströme dekodiert werden. Die aktuelle Generation dieser Datenträger weist Kapazitäten von bis zu 50 GB[42] auf. Jedoch ist auch hier die Langlebigkeit nicht garantiert. Je nach Quelle wird von einer Lebenszeit von 30 Jahren berichtet, die meisten trauen dem Medium jedoch maximal 10-15 Jahre zu. Dies gilt allerdings nur für industriell gepresste Datenträger. Bei der für Konsumenten verfügbaren Methode des *Brennens* ist das Ende der Lebenszeit häufig schon deutlich früher erreicht. Eine Ausnahme scheint die sogenannte M-Disc darzustellen: Diese bietet laut Herstellerangaben eine Haltbarkeit von circa 1000 Jahren.

3.2.2 Neue digitale Speichermedien

DNS als Langzeitspeicher

Jeder Organismus speichert seine kompletten Erbinformationen in allen seinen Zellen. Der Träger dieser Informationen ist die Desoxyribonukleinsäure, kurz DNS oder DNA, welche sich in den Zellkernen befindet. Es handelt sich hierbei um lange Kettenmoleküle, bestehend aus sogenannten **Nukleotiden**. Diese wiederum bestehen jeweils aus einem *Phosphatrest*, dem Zucker *Desoxyribose* und einer der Basen *Adenin*, *Thymin*, *Guanin* und *Cytosin*, welche häufig durch ihren Anfangsbuchstaben abgekürzt werden. Die DNS bildet normalerweise eine schraubenförmige Doppelhelix. Dabei sind die gegenüberliegenden Einzelstränge durch Basenpaare verbunden. Grundsätzlich ist dabei Adenin mit Thymin, sowie Cytosin mit Guanin gepaart. Die Länge eines DNS-Stranges wird in der Literatur häufig mithilfe der Anzahl der Nukleotide, kurz **nt** angegeben.

Aufgrund ihrer Eigenschaften besteht die begründete Annahme, dass sich die DNS als langfristiges Speichermedium gut eignet. Da sie in allen Lebewesen in gleicher Weise vorkommt und sich dies unter aller Voraussicht nicht ändern wird, könnte man sie als einen **ewigen Standard** bezeichnen. Verfahren zum Synthetisieren und Lesen sollten bis zum Ende jeglicher menschlicher Zivilisation bekannt sein, oder zumindest besteht die Möglichkeit, dass diese wiederentdeckt werden. Darüber hinaus verfügt sie über eine für heutige Maßstäbe gigantische Speicherdichte. Dekodiert man für jedes Nukleotid jeweils zwei Bits, so liegt diese bei einem theoretischen Wert von 455 Exabyte pro Gramm[9]. Funde größtenteils intakter DNS in 300.000 Jahre alten Fossilien, etwa von Menschen und Bären zeigen außerdem, dass es sich dabei um ein relativ stabiles Medium handelt[21]. Insbesondere sind die notwendigen Umgebungsbedingungen relativ einfach und günstig herzustellen, in der Regel reicht eine niedrige Temperatur und ein geeignetes Behältnis dazu, Informationen über Jahrtausende zu präservieren. Schlussendlich existiert mit der **Polymerasekettenreaktion** ein Verfahren, welches eine schnelle und kostengünstige Vervielfältigung von DNS und damit von Informationen ermöglicht.
Bereits 1988 konnte demonstriert werden, dass Informationen in synthetisierter DNS untergebracht werden können. Aufgrund der Schwierigkeit, langkettige Stränge zu erzeugen, verwendenden die meisten Verfahren indizierte, kurzkettige Sequenzen. Ein Vorteil dieses Ansatzes ist, dass eine solche Anhäufung von exakt gleich langen DNS-Sequenzen mit hoher Wahrscheinlichkeit den nicht-biologischen Ursprung erkennen lässt. Goldman et al.[20] kodierten 2013 insgesamt 739 kB. Die Stränge bestanden dabei aus 100 Basenpaaren und überschnitten sich jeweils um 75 Paare, wodurch eine vierfache Redundanz zur Wahrung der Datenintegrität ergab. Aus dem erzeugten Medium konnten die Informationen vollständig und fehlerfrei wieder extrahiert werden. Die Forscher geben an, dass die Anzahl der benötigten Basen annähernd linear mit der zu speichernden Information steigt, dazu kommen die Indizes, deren Länge logarithmisch wächst. Die Speicherdichte des verwendeten Verfahrens wird mit etwa 2,2 Petabyte pro Gramm angegeben. Einen eleganteren Ansatz zur Gewährleistung der Integrität der Informationen fanden Grass et al. 2015[21]: Diese nutzten fehlerkorrigierende Codes, explizit eine Reed-Solomon-Kodierung zu diesem Zwecke. Dabei wurde dieses Verfahren zweimal angewendet, einmal zur Fehlerkorrektur innerhalb eines Stranges, sowie eine Korrekturcode über die Stränge hinweg. Auch hier konnten die Daten fehlerfrei wiederhergestellt werden. Im Experiment wurde mithilfe künstlicher Alterung des Speichermediums bestimmt, dass bei richtiger Lagerung, etwa bei Temperaturen von -18°C im **Global Seed Vault**[15], Informationen auch nach mehr als 2 Millionen Jahren weiterhin lesbar seien.
So vielversprechend die DNS als Erbgut klingen mag, hat sie derzeit noch einen Haken: Die Synthetisierung, sowie die Dekodierung der DNS, also das *Lesen und Schreiben* von Informationen, ist sehr teuer. Goldmann et al. beziffern diese Kosten mit $12.400 pro Megabyte beim Schreiben und $200 pro Megabyte beim Lesen. Dabei soll allerdings beachtet werden, dass sich die Kosten nicht nur aus der initialen Erzeugung des Mediums, sondern auch aus dem Erhalt der Umgebungsbedingungen, sowie dem bei anderen Verfahren eventuell notwendigen regelmäßigem Wechsel der Medien ergibt. Dadurch können DNS-basierte Speichermedien in einigen Fällen bereits jetzt wirtschaftlich vertretbar sein. Allerdings fallen, laut Church et al.[9], die Kosten für Synthetisierungs- und Sequenzierungsverfahren exponentiell, sodass eine ökonomische Nutzung absehbar ist. Ein weiterer Nachteil ist, dass Schreib- und Lesevorgang vergleichsweise langsam sind. Diesem Makel sei mit der Parallelisierung der Vorgänge entgegenzuwirken.
Interessant ist auch der Ansatz, Information im Erbgut von lebenden Organismen unterzubringen. Dies ist bereits mit E. coli Bakterien gelungen[40]. Im Zuge der Fortpflanzung wird auch die Information repliziert und bleibt so erhalten. Allerdings weisen solche Verfahren auch nicht zu vernachlässigende Nachteile auf: Zum einen sind die zu speichernden Daten tatsächlich versteckt. Dass sie eines Tages durch Zufall wiedergefunden werden, wäre schon ein Glücksfall. Außerdem ist nicht gewährleistet, dass die Information tatsächlich unverändert bleibt. Schließlich treten bei der Fortpflanzung immer wieder Mutationen auf, welche eine Evolution erst möglich machten.

Weitere neue Speichermedien

Die Forschungsgruppe um Peter Kazansky an der University of Southampton schlägt vor, Informationen mithilfe eines Femto-Lasers in Quarzglas zu brennen[28]. Neben den drei

räumlichen Dimensionen sollen die dabei entstehenden Markierungen zwei weitere, optische Dimensionen aufweisen, weshalb eine hohe Dichte an Daten ermöglicht wird. Diese soll bei etwa dem 7000-fachen der Datenkapazität von aktuellen Blu-Ray-Disks liegen. Experimente sprechen diesem Medium eine Haltbarkeit von mehr als 13 Millionen Jahren zu, ohne dass dabei außergewöhnliche Umgebungsbedingungen herrschen müssen. Während zur Lese- und Schreibgeschwindigkeit keine Aussagen getätigt werden, wird eine wirtschaftliche Nutzung versprochen, sobald günstige Femto-Laser verfügbar sind.

Im Rahmen des Human Document Project wurde ein Speichermedium basierend auf Siliziumnitrid und Wolfram vorgeschlagen[12]. Diese verfügen sowohl über eine optisch lesbare Komponente, mit deren Hilfe in Experimenten rekursive Quick Response (QR)-Codes auf kleinstem Raum gespeichert wurden, wie auch eine deutlich dichtere Datenschicht, die sich über Elektronenstrahlen auslesen lassen. Diese Kombination ist in sofern von Vorteil, dass die nötigen Informationen zum Lesen der Daten optisch festgehalten werden können. Die Haltbarkeit des Datenträgers liegt laut Angaben der Forschungsgruppe bei mehreren Millionen Jahren.

3.3 Fehlerkorrigierende Codes

Es ist naiv anzunehmen, dass eine auf einem Datenträger gespeicherte Information gegenüber Veränderung immun ist. Alle Medien unterliegen einem natürlichen Zerfall, wobei die Haltbarkeit sich stark unterscheiden kann. Hinzu kommt, dass äußere Einwirkungen diesen Prozess, entweder an einer lokalen Stelle oder auf dem gesamten Medium, beschleunigen können. Gerade bei digitalen Daten wird dies schnell zum Problem. Während sich Buchstaben, etwa die des lateinischen Alphabets, häufig soweit unterscheiden, dass mit ein wenig Glück und *Augenmaß* wieder rekonstruieren lassen, so kann es mitunter schwer sein, zu erkennen, ob ein Bit *gekippt* ist oder nicht. Einige Verfahren, wie etwa gewisse DNS-basierte Ansätze gehen sogar fest von Fehlern während des Schreibvorganges aus[20]. Es ist daher wünschenswert, Informationen in der Art und Weise zu speichern, dass Fehler bis zu einem gewissen Grad erkannt oder gar korrigiert werden können.

3.3.1 Grundlagen der Kodierungstheorie

Gegeben sei das Alphabet $K := \{0, 1\}$. Die Menge der Wörter, die sich aus K erstellen lässt, entspricht $K^* := \bigcup_{t\in\mathbb{N}} K^t = \{(x_1, \ldots, x_t) \in K^t | t \in \mathbb{N}\}$. Nachrichten, wie auch codierte Nachrichten sind die Elemente dieser Menge. Ein Code C ist eine Verschlüsselung der von Nachrichten $A \subseteq K^*$ mithilfe einer injektiven Abbildung $f : A \to K$. Gilt $\emptyset \subset C \subseteq K^m$, so ist C ein Blockcode der Länge m. Sei nun ein solcher Blockcode C mit dazugehörigem Kodierverfahren f gegeben. Wird eine unverfälschte, codierte Nachricht $f(a)$ decodiert, ist die Lage aufgrund der Injektivität von f eindeutig. Hat sich die Nachricht inzwischen verändert, so gestaltet sich dies schwieriger. Da man davon ausgeht, dass Fehler eher unwahrscheinlich sind, ist die codierte Nachricht zu wählen, die der empfangenen Nachricht am *ähnlichsten* ist. Lautet der Code $C := \{000, 111\}$ und die verfälschte Nachricht ist 101, so kann man davon ausgehen, dass das Bit in der Mitte *gekippt* ist und die Nachricht 111 lautete. Um die Ähnlichkeit, bzw Unähnlichkeit zwischen zwei Codewörtern zu quantifizieren ist eine Metrik vonnöten. Diese ist gemeinhin als **Hamming-Abstand** bekannt und wird als Anzahl der sich in den beiden Codewörtern unterscheidenden Stellen definiert. Ermittelt man den minimalen Hamming-Abstand aller Codewort-Paare eines Codes, so kennt man den **Mindestabstand** des Codes. Aus diesem lassen sich zwei wichtige Eigenschaften für Codes ableiten: Hat ein Code C den Mindestabstand D, so ist C $\lfloor \frac{1}{2}(D-1) \rfloor$**-fehlerkorrigierend**. Das heißt, verändert sich eine codierte Nachricht um maximal $\lfloor \frac{1}{2}(D-1) \rfloor$ Stellen, so kann stets noch auf die korrekte ursprüngliche Nachricht geschlossen werden. Da im obigen Beispiel der Mindestabstand drei ist, ist der Code 1-Fehler-korrigierend. Außerdem gilt, dass der Code $\frac{D}{2}$**-Fehler-erkennend** ist, sofern D gerade ist. Diese Eigenschaft besagt, dass wenn die empfangene Nachricht sich um höchstens $\frac{D}{2}$ stellen unterscheidet, feststellbar ist, um wie viele Stellen sie sich genau unterscheidet. Beweise hierzu finden sich in [31].

3.3.2 Redundanzbasierte Fehlerkorrektur

Die offensichtlich einfachste Variante der Fehlererkennung und -korrektur ist das mehrfache Darstellen einer Informationen, oder im Falle einer Nachrichtenübermittlung das wiederholte Versenden der Nachricht. Hompel et al.[25] nennen hier das **Zwei aus Drei**-Verfahren, bei dem das zu übermittelnde Wort in dreifacher Ausführung codiert wird: $c = c_1c_2c_3$. Beim Lesen von c gibt es nun folgende Fälle:

- $c_1 = c_2 = c_3$, es ist also scheinbar kein Fehler aufgetreten und die Nachricht kann als c_i entschlüsselt werden.
- Zwei Symbole c_i, c_j sind identisch, während c_k davon abweicht. Hier ist offensichtlich ein Fehler aufgetreten, der allerdings korrigiert werden kann, das am häufigsten auftretende Symbol $c_i = c_j$ kann ausgegeben werden.
- Alle Symbole unterscheiden sich. Es ist ein Fehler aufgetreten, der nicht korrigiert werden kann.

Man sollte außerdem beachten, dass im ersten Fall drei gleiche Fehler aufgetreten sein können, die damit nicht bemerkt werden. Dabei kann es sich insbesondere um systematische Fehler handeln. Im zweiten Fall können c_i und c_j durch Zufall *gleich falsch* sein, womit eigentlich c_k korrekt wäre und die Information falsch rekonstruiert wird. Auch in diesem Fall ist der Fehler nicht ohne weitere Bemühungen erkennbar. *Zwei aus Drei* lässt sich ohne Weiteres auf *m aus n*, natürlich mit $m \leq n$ verallgemeinern. Dadurch lässt sich auf Kosten der Kapazität weitere Sicherheit erkaufen. Diese Verfahren sind allerdings aus naheliegenden Gründen nur wenig effizient.

3.3.3 Reed-Solomon-Codes

Irving Reed und Gus Solomon entwickelten 1960 eine neue Klasse von fehlerkorrigierenden Codes[11]. Diese **Reed-Solomon-Codes** genannten Verfahren werden bis heute aktiv genutzt, etwa bei Kommunikationsprotokollen für mobile Geräte, Satellitenkommunikation, Digitalfernsehen und Modems. Besonders bekannt sind sie außerdem durch die Verwendung in . Es handelt sich um nicht-binäre, zyklische Blockcodes. Ein Reed-Solomon-Code ist als $RS(n, k)$ mit s-Bit-Symbolen spezifiziert. Dabei entspricht n der Länge von Codewörtern in Symbolen, wovon k Symbole für die Darstellung von Daten genutzt werden. Die restlichen $n - k = 2t$ Symbole werden für Paritäten genutzt. Der Reed-Solomon-Code ist t-Fehlerkorrigierend, sind die betroffenen, fehlerhaften Symbole bekannt, so können gar $2t$ Fehler korrigiert werden. Wird die Symbollänge auf s Bit festgelegt, so ist die Länge des Codeworts auf $2^s - 1$ begrenzt.
Für Verfahren dieser Art wird die Algebra der **Galois-Körpern** (GF) benötigt. Galois-Körper bestehen aus einer endlichen Menge von Elementen, auf denen die Operationen Addition, Subtraktion, Multiplikation und Division definiert sind. Mit der Notation $GF(2^s)$ wird ein Galois-Körper mit 2^s Elementen bezeichnet. Die Elemente sind dabei mit Null beginnend durchnummeriert, wie später gezeigt wird, lassen sie sich auf die binären Symbole des Codes abbilden. Ein Galois-Körper $GF(2^s)$ wird wie folgt generiert: Zuerst wird ein Generatorpolynom benötigt. Dabei handelt es sich um ein irreduzibles Polynom s-ten Grades; diese sind allgemein bekannt. So existiert für $GF(2^2)$ etwa $q(x) = x^2 + x + 1$ oder $q(x) = x^8 + x^4 + x^3 + x^2 + 1$ in $GF(2^8)$[41]. Durch die wiederholte Multiplikation einer Wurzel x für $q(x)$ mit sich selbst ergibt sich dann $GF(2^s) = \{0, x^0, x^1 \bmod q(x), \ldots, x^{2^s-2} \bmod q(x), x^{2^s-1} \bmod q(x)\} = \{0, 1, x, \cdots, x^3 + 1, 1\}$. Auffällig ist, dass $x^0 \equiv x^{2^s-1}$ gilt, wodurch die Generierung offensichtlich zyklisch erfolgt. Damit heute übliche Computersysteme mit diesen Polynomen arbeiten können, ist eine geeignete binäre Darstellung nötig. Dazu existiert die Abbildung $f : GF(2^s) \to \{0, 1\}^s$. Hierbei entspricht das i-te Bit der binären Darstellung $b \in \{0, 1\}^s$ genau dem Koeffizienten des Polynoms $r(x) \in GF(2^s)$. Tabelle 3.2 zeigt den Galois-Körper $GF(2^4)$ in Polynom- und Binärdarstellung.
Operationen sind außerdem wie folgt definiert: Die **Addition** ($+_{GF}$) zweier Elemente eines Galois-Körpers ist als die Exklusiv-Oder-Verknüpfung ($\oplus$) der Koeffizienten festgelegt, das Einselement ist 0. Daraus folgt, dass die Addition mit der **Subtraktion** identisch ist[1]. Entsprechend lassen sich diese beiden Operationen auch in der Binärdarstellung einfach realisieren. Die **Multiplikation** ($\cdot_{GF}$) entspricht der Addition der Exponenten der Generatoren.

Generator	Polynome	Binär	Generator	Polynom	Binär
0	0	0000_2	x^7	x^3+x+1	1011_2
x^0	1	0001_2	x^8	x^2+1	0101_2
x^1	x	0010_2	x^9	x^3+x	1010_2
x^2	x^2	0100_2	x^{10}	x^2+x+1	0111_2
x^3	x^3	1000_2	x^{11}	x^3+x^2+x	1110_2
x^4	$x+1$	0011_2	x^{12}	x^3+x^2+x+1	1111_2
x^5	x^2+x	0110_2	x^{13}	x^3+x^2+1	1101_2
x^6	x^3+x^2	1100_2	x^{14}	x^3+1	1001_2

Tabelle 3.2: Elemente des Galois-Körpers $GF(2^4)$ mit dem Generatorpolynom $q(x) = x^4 + x + 1$ in Polynom- und Binärdarstellung

So ist $(x^3+x^2)\cdot_{GF}(x+1) = x^2+x+1 \equiv x^6 \cdot_{GF} x^4 = x^10$. Ist der Exponent größer, als die Anzahl der Elemente des Galois-Körpers, so muss eine Modulo-Operation durchgeführt werden, dies entspricht dem zyklischen Charakter der Elemente. Das Nullelement ist 1. Schließlich ist die **Division** die Umkehrung der Multiplikation und wird daher durch Subtraktion der Exponenten realisiert, auch hier wieder mit Anwendung der Modulo-Operation bei Werten kleiner als Null. Effiziente Implementationen verwenden Lookup-Tabellen zur Realisierung dieser Funktionen[11]. Außerdem merkt [1] an, dass sich die Multiplikation mit 2 (x oder 0010_2) als linear rückgekoppeltes Schieberegister (LFSR) in Hardware realisieren lässt. Es treffen die Eigenschaften algebraischer Körper zu, dazu zählen unter anderem die Kommutativität, Assoziativität und Distributivität von $+_{GF}$ und $\cdot_{GF}$, die bereits erwähnten Null- und Einselemente, sowie die damit verbundenen multiplikativen bzw. additiven Inversen.
Die Berechnung von Checksummen soll nun anhand eines $RS(n,k)$ mit s-Bit-Codewörtern erläutert werden. Ein Codewort $D_1 \dots D_k C_1 \dots C_{n-k}$ besteht aus den Datensymbolen D_i und den Prüfsymbolen C_i. Sei F_i eine k-stellige Funktion zur Generierung des i-ten Prüfsymbols. Dabei handelt es sich um eine Linearkombination der Datensymbole. Sie ist wie in Gleichung 3.1 definiert.

$$C_i = F_i(D_1, \dots, D_k) = \sum_{j=1}^{n} D_j \cdot_{GF} f_{i,j} \tag{3.1}$$

Bei $f_{i,j}$ handelt es sich um Elemente einer $(n-k) \times n$-Vandermonde-Matrix F. Definiert man D als Vektor der Datensymbole und C als Prüfsymbolvektor des Codeworts, so lässt sich Gleichung 3.1 auch wie folgt darstellen:

$$FD = C \tag{3.2}$$

Die Elemente der Vandermonde-Matrix sind mit $f_{i,j} = j^{j-1}$ definiert. Weil die erste Zeile der Matrix $f_{1,j} = 1$ ist, entspricht die Funktion F_1 genau der Addition ($+_{GF}$) aller Datensymbole. An dieser Stelle folgt nun ein Beispiel zur Berechnung der Prüfsymbole. Dazu wird ein $RS(4,2)$ mit 4-Bit-Symbolen verwendet. Statt die langen Polynom- und Binärschreibweisen zu verwenden, soll die hexadezimale Form genutzt werden. So entspricht exemplarisch $1001 \equiv x^3+1 \equiv a$. Gleichzeitig entsprechen die Operationen + und $\cdot$ ihren Pendants $+_{GF}$ und $\cdot_{GF}$. Es sollen die Prüfsymbole C_1 und C_2 für die Datensymbole $D_1 = a_{16}$ und $D_2 = 2_{16}$ ermittelt werden. Die Berechnung erfolgt nun in Matrixdarstellung:

$$\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}a\\ 2\end{bmatrix} = \begin{bmatrix}2\cdot 1 + a\cdot 1\\ 2\cdot 2 + a\cdot 1\end{bmatrix} = \begin{bmatrix}2+a\\ 4+a\end{bmatrix} = \begin{bmatrix}9\\ f\end{bmatrix} \Rightarrow \underline{\underline{C_1 = 9, C_2 = f}} \tag{3.3}$$

So entsprechen $C_1 = 9$ und $C_2 = f$, das komplette Codewort lautet $a29f$.
Wie Anfangs beschrieben, können bis zu $n-k$ fehlerhafte Symbole korrigiert werden, wenn bekannt ist, um welche es sich handelt. Dazu werden die Matrizen A und E wie folgt definiert, wobei I die $n \times n$-Einheitsmatrix ist:

$$A = \begin{bmatrix}I\\ F\end{bmatrix} E = \begin{bmatrix}D\\ C\end{bmatrix} \tag{3.4}$$

Der Vektor E stellt gewissermaßen einen Codewort-Vektor dar. Es folgt die Gleichung $AD = E$, wobei offensichtlich jede Zeile der Matrix einem Symbol im Codewort E entspricht.

Sind die fehlerhafte Symbole bekannt, genügt es die entsprechenden Zeilen in A und E zu streichen, wodurch die Gleichung $A'D = E'$ folgt. Dieses Gleichungssystem lässt sich mithilfe des *Gaußschen Eliminationsverfahrens* lösen. Im vorhergehenden Beispiel sei bekannt, dass D_2 einen Fehler aufweist. Entsprechend sieht $AD = E$ wie folgt aus:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} a \\ ? \\ 9 \\ f \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \begin{bmatrix} a \\ 9 \\ f \end{bmatrix} \tag{3.5}$$

Das daraus resultierende Gleichungssystem lässt sich durch Subtraktion der ersten von der zweiten Zeile in einem Schritt lösen. Es sei daran erinnert, dass alle Operationen entsprechend ihrer Definition in $GF(2^4)$ verwendet werden müssen.

$$\begin{bmatrix} 1 & 0 & a \\ 1 & 1 & 9 \\ 1 & 2 & f \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & 2 \\ 1 & 2 & f \end{bmatrix} \Rightarrow \underline{\underline{D_2 = 2}} \tag{3.6}$$

Die Rekonstruktion für mehrere fehlerhafte Symbole funktioniert Analog, sofern nicht mehr als $n-k$ bekannte Stellen verfälscht wurden. Existieren nicht mehr als $\frac{n-k}{2}$ Fehler, wobei der jeweilige Ort des Fehlers nicht bekannt ist, kann ebenso eine Korrektur vorgenommen werden. Allerdings übersteigen die entsprechenden Verfahren den Rahmen dieser Arbeit und sollen daher nicht weiter erläutert werden. Hompel et al.[25] schreiben Reed-Solomon-Codes eine hohe Komplexität, sowie eine zeitaufwändige Berechnung zu. Aufgrund der hardwarenahen Realisierungsmöglichkeit des Verfahrens in , kann dem jedoch widersprochen werden.

3.4 Optoelektronische Schriften

Das im Verlaufe dieser Arbeit verwendete Speichermedium wird Mikrofilm sein. Solcher speichert Information nicht etwa magnetisch, mechanisch oder mithilfe anderer, ausgefallener Techniken, sondern optisch durch ein zweidimensionales Bild. Es ist daher notwendig, eine geeignete Darstellung von Information auf diesem Medium zu finden. Auch, wenn die sich im Verlauf noch herausstellen wird, dass die Repräsentation der Metadaten in einem Klartext-Format vorgenommen werden, ist es nicht ratsam, Texte direkt auf dem Mikrofilm auszubeleuchten: Fehlende Fehlerkorrekturmöglichkeiten, sowie die Gefahr von Mehrdeutigkeiten – insbesondere bei ähnlich aussehenden Zeichen (je nach verwendeter Schriftart etwa `l` und `I` oder `O` und `0`) – machen diesen Ansatz unattraktiv.
Stattdessen haben sich bereits optoelektronische Schriften etabliert. Diese ermöglichen es, Information optisch zu kodieren, sodass diese effizient und sicher durch entsprechende Lesegeräte entziffert werden können. Der nachfolgende Abschnitt soll diese Techniken grob umreißen.

3.4.1 Eindimensionale Codes

Eindimensionale Codes bestehen aus parallelen Balken verschiedenster Breite, die durch unterschiedlich große Lücken unterbrochen werden. Sie werden im allgemeinen Sprachgebrauch auch als **Streifen-, Strich- oder Balkencode** bezeichnet. In der Regel sind Balken schwarz, der Untergrund – und damit auch die Lücken – sind weiß. Der generelle Aufbau sieht folgende Teile eines Codes vor: Ein **Startcode** leitet den Beginn der Codes ein, gefolgt von den **Nutzzeichen**, welche die eigentliche Botschaft enthalten. Zur Fehlererkennung folgt eine **Prüfsumme**, die je nach verwendetem Code optional oder verpflichtend ist. Ein **Stopcode** markiert das Ende. Zusätzlich sind Strichcodes auf beiden Seiten von einer **Ruhezone** umgeben. Start- und Stopcodes sind für jedes Kodierverfahren eindeutig, sodass dieses sich nach Lesen der Startsequenz identifizieren lässt. Gleichzeitig geben sie die Leserichtung vor. Die Breite des kleinsten Striches entspricht der Modulbreite. Für alle anderen Elemente des Codes wird die Länge als Vielfaches der Modulbreite angegeben. Man unterscheidet zwischen **Zweibreiten-** und **Mehrbreitencodes**, wobei erstere zwei verschiedene Balken kennen (*dick* und *dünn*) und letztere eine Vielzahl. Weiterhin gibt es **diskrete** und **fortlaufende Codes**. Bei diskreten Codes beginnt jedes codierte Zeichen mit einem Balken, bei fortlaufenden Codes ist dies nicht zwangsläufig der Fall. Wichtige Qualitätskriterien

(a) Code 128 mit Nutzung der Ebene C

(b) Code 128

(c) Code 39

(d) Code 2/5 Interleaved

Abbildung 3.5: Beispiele für Barcodes

von Strichcodes sind unter anderem der verfügbare Zeichenvorrat, die Länge des Codes, die Robustheit bei Fehlern und verfügbare Lesegeräte[25].
Ein einfacher Vertreter der Strichcodes ist **Code 2/5** (*Zwei aus fünf*). Es handelt sich um einen Zweibreitencode, der die Ziffern Null bis Neun, sowie Start- und Stoppzeichen unterstützt. Jeweils Fünf Balken, von denen immer genau zwei breit sein müssen, entsprechen dabei einem der zehn Zeichen, womit alle rechnerisch möglichen Kombinationen ausgeschöpft sind[13]. Durch die Festlegung der Anzahl breiter Streifen wird eine einfache Fehlererkennung implementiert. Ziffern werden einzeln codiert und die resultierenden Balken aneinandergehängt. Die optionale Prüfziffer wird wie folgt berechnet: Aus den Nutzziffern wird eine gewichtete Summe gebildet, wobei mit dem letzten Zeichen beginnend jede ungerade Ziffer den Faktor mit drei multipliziert wird. Die Summe wird durch zehn dividiert und der Rest von zehn subtrahiert. Dieses Verfahren wird auch in weiteren Kodierungen verwendet.
Eine Weiterentwicklung stellt **Code 2/5 interleaved**, auch bekannt als Interleaved Two of Five (ITF) dar. Hierbei werden die Ziffern grundsätzlich auf dieselbe Balkenfolgen abgebildet, jedoch werden dabei abwechselnd schwarze und weiße Balken verwendet. Nach dem Startcode entspricht der erste schwarze Strich dem ersten Strich der ersten Ziffer, der darauffolgende Zwischenraum ist allerdings bereits der zweiten Ziffer zuzuordnen. Es versteht sich von selbst, dass die nun Information tragenden Zwischenräume keine konstante Breite mehr aufweisen. Durch diese Verschachtelung der Zifferndarstellungen gewinnt der Code deutlich an Kompaktheit, jedoch auf Kosten der Korrekturfähigkeit. Code 2/5 Interleaved wurde in DIN EN 801-1995, sowie in ISO/IEC 16390 genormt[33].
Während die bisher genannten Verfahren lediglich die Zahlen von null bis neun darstellen konnten, umfasst **Code 39** zusätzlich die Großbuchstaben A-Z, sowie einige Sonderzeichen, womit insgesamt 43 Zeichen verwendet werden können. Ein einzelnes Zeichen wird durch neun Elemente, dazu gehören sowohl fünf Balken, als auch vier Zwischenräume, dargestellt. Davon müssen genau drei breit sein. Der Code ist diskret. Zwischen zwei Zeichen soll ein kleiner Trennraum gelassen werden. Es wird nicht empfohlen, mehr als 20 Nutzzeichen zu verwenden. Sind mehr Zeichen nötig, können die Daten auf mehrere Codes aufgeteilt werden. Beginnt ein Folgecode mit einem Leerzeichen, signalisiert dies, dass es sich um eine Fortsetzung handelt. Als Normierungen für Code 39 seien DIN EN 800 und ISO/IEC 16388 genannt.
Ein populärer Vertreter der Strichcodes ist der **EAN-Code**. Damit wird ein fortlaufender Mehrbreitencode bezeichnet, der die Ziffern null bis neun darstellen kann. Er wird zur Kennzeichnung von Artikeln im Europäischen Handelsraum verwendet, am häufigsten genutzt wird die Variante EAN-13 zur Speicherung von 13 Ziffern inklusive obligatorischer Prüfsumme. Bei dessen Darstellung gibt es einige Besonderheiten, so gibt es etwa das Mittensymbol, welches aus fünf Balken einer Modulbreite besteht. Es gibt insgesamt drei Zeichensätze A, B und C, wobei deren Verwendung pro Zeichenposition abhängig vom ersten Symbol vorgegeben ist. Dabei entspricht C der dem invertierten Zeichensatz A und der

[13] Es handelt sich um eine Permutation mit Wiederholung: $\frac{5!}{3!\cdot 2!} = 10$

(a) Stufe L - 7% (b) Stufe M - 15% (c) Stufe Q - 25% (d) Stufe H - 30%

Abbildung 3.6: Beispiele für QR-Codes anhand der URL `http://www.wossidia.de`. Zur Erstellung wurde die Software `qrencode` verwendet. Es werden sämtliche Fehlerkorrekturstufen demonstriert.

Spiegelung von B. Optional kann ein weiterer Code mit zwei oder fünf Zeichen an den Code angehängt werden. EAN-13 ist in DIN EN 797, sowie in ISO/IEC 15420 genormt.
Letztendlich existiert noch **Code 128**. Dessen Zeichensatz umfasst die Zeichen 0 bis 128 des ASCII-Codes, die Zifferntupel 00 bis 99, sowie jeweils vier weitere Steuer- und Sonderzeichen. Ein einzelnes Zeichen wird durch je drei Balken und Lücken dargestellt, wobei die Breite immer elf Modulbreiten entspricht. Zusätzlich gibt es für jede Balkenfolge drei Ebenen A, B und C, wodurch jedem der 106 möglichen Muster bis zu drei Bedeutungen zukommen können. So werden derselben Folge beispielsweise die Zeichen '^', '?', wie auch das Zifferntupel '62'zugeordnet. Um zu unterscheiden, welche der Bedeutungen im konkreten Fall relevant ist, existieren verschiedene Start-Symbole, jeweils eines für jede Ebene. Alle Folgen, die nach einem `Start Ebene A` erscheinen, werden mit der entsprechenden Ebene dekodiert. Auch bei Code 128 ist die Prüfziffer verpflichtend. Normen für das Verfahren sind DIN EN 799-1995 und ISO/IEC 15417.

3.4.2 Zweidimensionale Codes

Im Wesentlichen unterscheidet man zwischen Stapel-, Matrix- und zusammengesetzten Codes (Composite Codes). Stapelcodes verwenden häufig die im letzten Abschnitt vorgestellten Techniken, setzen jedoch nach erreichen einer zuvor festgelegten Breite in einer neuen Zeile fort. Matrixcodes stellen Daten, wie der Name bereits impliziert, in einem Raster von Punkten dar. Der Oberbegriff Composite Codes fasst solche Verfahren zusammen, die ein- und zweidimensionale Techniken miteinander verknüpfen[32].
Die einfachste Variante der zweidimensionalen Codes sind wohl gestapelte Barcodes. Ein früher Ansatz war der **Code 49**, welcher entweder 81 Ziffern oder 49 alphanumerische Zeichen auf bis zu acht Zeilen codiert. Zwischen den Zeilen existiert jeweils eine Trennlinie. Der in Deutschland entwickelte **Codablock**-Barcode ist eine Codefamilie mit drei Varianten. Diese fassen die Daten aus mehreren Einzelcodes zu einem Codeblock zusammen. Als Beispiel sei Codablock F sein, dieser basiert auf Code 128,und ermöglicht er die Darstellung von 62 Zeichen in maximal 44 Zeilen, womit bis zu 2728 Zeichen gespeichert werden. Für Codablock F existiert die französische Norm AFNOR NFZ63-322 11/95.
Auch bei **PDF 417** handelt es sich um eine Familie von gestapelten Codes. Dabei lässt die Zahl 417 auf den Aufbau schließen: Vier Elementpaare (repräsentiert durch jeweils vier verschiedene Balken- und Zwischenraumbreiten) werden auf 17 Module aufgeteilt. Im Gegensatz zu Code 49 werden keine Trennlinien verwendet. Es wird zeilenweise zwischen insgesamt drei Codewortzeichensätzen gewechselt, wodurch ein Lesegerät während des Lesevorgangs seine Position ermitteln kann. Das Verfahren zeichnet sich im Vergleich zu den anderen gestapelten Codeverfahren durch eine hohe Kompaktheit und Fehlerkorrektur aus.
Ein Beispiel für zusammengesetzte Codes ist die **RSS-14** Familie. Diese liefert die Grundstruktur für das erweiterte UCC/EAN System, wodurch es den EAN-Code in bestimmten Situationen ersetzen kann. Dem gegenüber können etwa 30% Breite eingespart und noch zusätzliche Daten kodiert werden.

3.4.3 Im Detail vorgestellt: QR-Codes

Der bekannteste Vertreter der Matrix-Codes ist wohl der **QR-Code**. Dieser hat seinen Ursprung im ostasiatischen Raum und hat dort ähnliche Anwendungsgebiete, wie der Data Ma-

trix Code im Westen. Er verdankt seine Popularität vor allem der Anwendung im Endkonsumentenbereich. Aufgrund der Verbreitung von Smartphones und ähnlichen Geräten, tragen viele Menschen bereits ein Lesegerät mit sich, da entsprechende Apps kostenlos verfügbar sind. So ist es möglich, auf Produkten, Werbetafeln, Werbeflyern oder in Zeitschriften QR-Codes mit Nachrichten zu hinterlassen. Diese können URLs, Klartextnachrichten, Telefonnummern, Visitenkarten oder ähnliches enthalten. Neben der sehr hohen Kompaktheit sind schnelle Lesbarkeit (daher *Quick Response*), gute Fehlerkorrektureigenschaften, sowie Korrektur der Verzerrung als Vorteile zu nennen. Das Verfahren wurde 1994 von der japanischen Firma Denso[14] entwickelt. Inzwischen ist eine zweite Version verfügbar, welche die sechsfache Datenkapazität bietet. Sie ist unter ISO/IEC 18004 normiert. Weiterhin bestehen die Varianten *Micro QR-Code* für Kleinstnachrichten auf minimalem Raum, sowie *iQR*. Letzteres ermöglicht die Verwendung von rechteckigen Codes, sowie quadratische Codes bis zu 422×422 Modulen (im Gegensatz zu 177×177 bei QR-Code Version 2).

Ein QR-Code besteht grundsätzlich aus einer quadratischen Matrix schwarzer und weißer Zellen. Dabei ist die quadratische Form der Zellen zwingend vorgegeben, eine Toleranz von $\pm 10\%$ wird dabei gewährt. Die Dimension ist variabel und kann an die notwendige Nachrichtenlänge und Fehlertoleranz angepasst werden. Insgesamt existieren 40 Varianten, beginnend mit 21×21 Zellen in der kleinsten und 177×177 in der größten Variante. Bei jeder Vergrößerung werden vier Felder pro Dimension hinzugefügt. Der Code ist von einer Ruhezone in der Größe von vier Zellen umgeben. In den Ecken unten links, sowie oben links und rechts sind Suchmuster angebracht. Dabei handelt es sich um 3×3-Quadrate, die von einem Rand in der Breite einer Zelle umgeben sind. Zur weiteren Orientierung existieren Taktzellen – dabei handelt es sich um in einer Linie liegende Zellen, die abwechselnd schwarz und weiß gefärbt sind – und Orientierungszellen. Aufgrund dieser Merkmale ist es zum einen möglich, Codes unabhängig von ihrer Orientierung zu lesen und eine perspektivische Verzerrung zu korrigieren.

Im Folgenden soll eine Kodiervorgang detailliert beschrieben werden. Es soll die Zeichenkette `Richard Wossidlo - Рихарт Воссидло` in einem QR-Code mit einer Fehlerkorrekturstufe M gespeichert werden.

Wann immer es nötig ist, wird außerdem auf die Variante 40 mit Fehlerkorrekturstufe H eingegangen. Diese soll später ausschließlich genutzt werden, weil der Vorgang für jede Variante spezifisch ist und somit nur ein einziger Weg beschrieben werden muss. Aus Ermangelung der originalen ISO-Quellen musste hierbei allerdings auf alternative Literatur zurückgegriffen werden[15].

Die Kodierung findet in sieben Schritten statt:

1. **Datenanalyse:** Die zu codierende Zeichenkette wird untersucht und eine entsprechende Wahl der Kodiermethode gewählt.

2. **Datenkodierung:** Abhängig von der im ersten Schritt gewählten Methode wird die Zeichenkette in eine Bitfolge überführt.

3. **Fehlerkorrektur:** QR verwendet zur Fehlerkorrektur den Reed-Solon-Code. Entsprechend der zuvor gewählten Stufe werden die notwendigen Korrekturwörter errechnet.

4. **Nachrichtenstrukturierung:** Die bisher gebildeten Daten- und Fehlerkorrekturwörter werden in einer wohldefinierten Reihenfolge aneinander gehängt. Es entsteht eine lange Zeichenkette.

5. **Modulplatzierung:** In diesem Schritt werden zuerst die *fixen* Bestandteile der grafischen Repräsentation aufgebaut und abschließend das Codewort, welches im vorhergehenden Schritt erstellt wurde, in diese eingesetzt.

6. **Maskierung:** Um eine gute Lesbarkeit des QR-Codes zu gewährleisten, wird anschließend eine Maske über die Matrix gelegt.

7. **Metainformationen:** Den Abschluss macht die Kodierung von Metainformationen, welche das Lesen des Codes ermöglichen.

(a) Nicht-maskierter QR-Code mit farblich markierten Elementen

(b) Das laufende Beispiel als vollständiger QR-Code

Abbildung 3.7: Laufendes Beispiel mit dem Inhalt `Richard Wossidlo - Рихарт Воссидло` zur QR-Code-Generierung

Datenanalyse

QR bietet einige Modi zur Kodierung von Daten an. Dazu zählen ein **numerischer Modus** für reine Ziffernfolgen, der **alphanumerische Modus**, welcher die Darstellung von Großbuchstaben, Ziffern sowie einiger Sonderzeichen erlaubt, ein **Byte**-, wie auch ein **Kanji**-Modus. Die zu codierende Nachricht liegt in 8-Bit Universal Character Set Transformation Format (UTF-8) vor, weshalb der Byte-Modus gewählt wird. Dieser ist zwar standardmäßig Zeichenketten in ISO-8859-1 vorbehalten, eine Nutzung von UTF-8 ist dennoch ohne Weiteres möglich[16]. Die Zeichenkette benötigt 48 Bytes an Speicherplatz. Es sei explizit darauf hingewiesen, dass bei der Verwendung von UTF-8 nicht zwangsweise jedes Zeichen ein Byte belegt. Bei einem Fehlerkorrekturgrad `M` ist die kleinste QR-Version, die diese Kapazität halten kann die 4. Diese Variante kann 62 Codewörter zu je acht Bit aufnehmen. Solche Werte sind Tabellen zu entnehmen und lassen sich nicht trivial berechnen.

Im späteren Verfahren dieser Arbeit soll stets der Byte-Modus mit Korrekturstufe `H` und der maximalen Größe 40 gewählt werden. Auf diese Weise ist es möglich, für die Nachwelt lediglich die Beschreibung einer Untermenge von QR hinterlassen zu müssen, bei gleichzeitiger Gewissheit, dass die Daten problemlos entziffert werden können.

Datenkodierung

Es gilt nun, die Nachricht in einer Bitfolge darzustellen. Diese beginnt stets mit dem Modus-Indikator (Mode Indicator). Dabei handelt es sich um eine Folge von vier Bit, welche den verwendeten Modus bestimmt. Wir wählen `0100` für den Byte-Modus[17]. Es folgt der Längen-Indikator (Character Count Indicator), welcher die Anzahl der nachfolgenden Zeichen bestimmt. Die Länge des Feldes ist abhängig von Version und gewähltem Modus. Im vorliegenden Fall sind dies acht Bit. Entsprechend wird die Länge von 48 Byte mit `0011 0000` angegeben. Anschließend folgt die Nachricht, diese ist eine direkte Kodierung der Zeichenkette in UTF-8. Sie wird mit dem Terminator `0000` abgeschlossen, außerdem werden weitere Nullen hinzugefügt, bis die Länge der Bitfolge ein ganzzahliges Vielfaches von acht ist. Da die Kapazität eines QR-Codes stets ausgereizt werden muss, ist anschließend der Rest aufzufüllen. Im laufenden Beispiel werden 50 Byte belegt[18], um die notwendigen 64 Byte aufzufüllen wird nun mit dem sich wiederholenden Muster `11101100 00010001` ergänzt.

Bei der Kodierung der Metadaten auf Mikrofilm wird bei der Generierung des Version-40-QR-Codes entsprechend der Modus-Identikator `0100` verwendet. Der Längen-Indikator hingegen ist in diesem Falle 16 Bit lang. Die mögliche und geforderte Anzahl der Codewörter ist 1276.

[14] Verschiedene Quellen benennen diese auch mit Nippondenso oder Denso Wave

[15] Siehe `http://www.thonky.com/qr-code-tutorial/`

[16] Einige Lesegeräte sind nicht in der Lage, UTF-8 zu erkennen. Da im Rahmen dieser Arbeit explizit darauf hingewiesen wird, sollte dies jedoch kein Problem darstellen.

[17] Zur Kodierung in UTF-8 gibt es mehrere Ansätze, etwa über die Verwendung des Extended Channel Interpretation (ECI)-Modus. Der Einfachheit halber wird dieser verwendet.

[18] Mode Indicator (4 Bit) + Length Indicator (8 Bit) + Nutzlast (48*8 Bit) + Füllnullen (4 Bit)

Fehlerkorrektur

Vor der Berechnung der Fehlerkorrekturwörter wird die Bitfolge in Codewörter, Blöcke und Gruppen unterteilt. Die Anzahl der Gruppen, Blöcke pro Gruppe und Codewörter je Block ist von der gewählten Version und dem Fehlerkorrekturlevel abhängig. Im vorliegenden Fall wird eine Gruppe mit zwei Blöcken zu je 32 Codewörtern verlangt. Dabei ist ein Codewort immer 8 Bit lang.
Anschließend werden die Korrekturwörter, wie in Abschnitt 3.3.3 beschrieben, berechnet. Diese werden immer innerhalb eines Blockes gebildet, der Anzahl der Fehlerkorrekturcodes ist ebenfalls von Version und Korrekturstufe abhängig und den Tabellen zu entnehmen. Es wird der Galois-Körper $GF(256)$ verwendet, das Generatorpolynom ist ebenso in den Tabellen vermerkt.
Für QR-40-H werden zwei Gruppen zu 20 und 61 Blöcken erstellt. In der ersten Gruppe besteht jeder Block aus 15, in der zweiten aus 16 Codewörtern.

Nachrichtenstrukturierung

Ziel dieses Schrittes ist die Generierung einer einzelnen Bitsequenz, welche anschließend im Code grafisch dargestellt werden soll. Dazu werden Gruppen, Blöcke und Codewörter, sowie die Fehlerkorrekturwörter miteinander in geordneter Weise *verwoben* (interleaved). Nacheinander werden zuerst das erste Codewort der ersten Gruppe, Block eins, dann dass erste Codewort der ersten Gruppe, Block eins und so weiter aneinander gereiht. Dies wird so lange durchgeführt, bis alle Codewörter verwendet wurden. Anschließend wird derselbe Prozess mit den Fehlerkorrekturwörtern durchgeführt.
In einigen Fällen muss die Bitfolge wiederum mit Nullen aufgefüllt werden. Dies ist wieder von der Version abhängig, ob die Notwendigkeit besteht, muss erneut den Tabellen entnommen werden. Im Laufenden Beispiel werden sieben Nullen angehängt. Für die Version 40 ist dies nicht notwendig.

Modulplatzierung

Nun soll der QR-Code entstehen. Als Orientierung soll Abbildung 3.7a dienen. Zuerst werden die Suchmuster (Finding Pattern) unten links, sowie oben links und rechts gezeichnet. Im Bild sind diese Rot dargestellt. Sie sind bei allen Versionen identisch und werden durch eine Grenze der Breite eines Moduls vom Rest des Codes separiert. Es folgen die Ausrichtungsmuster (Alignment Pattern), welche grün dargestellt sind. Sie werden in regelmäßigen Abständen auf dem Code verteilt, die Positionen sind dabei von der Größe des Codes abhängig und wieder in Tabellen zu finden. Dabei überschneiden sie sich nie mit den Suchmustern.
In der sechsten Zeile und Spalte wird das Taktmuster (Timing Pattern) hinzugefügt. Dabei handelt es sich um eine Folge von alternierenden Modulen, beginnend mit einem dunklen Modul. In der Abbildung ist das Taktmuster blau eingezeichnet. Die violetten Bereiche werden reserviert, diese erhalten zum Abschluss des Vorgangs Metadaten über den Code.
Anschließend werden die Daten selbst hinzugefügt. Dabei wird jede `1` der Bitfolge durch ein dunkles und jede `0` durch ein helles Modul dargestellt. Das Einfügen geschieht in Doppelspalten, wobei unten rechts begonnen wird, die beiden Spalten werden nach oben und anschließend die Doppelspalte links daneben nach unten aufgefüllt. In jeder Doppelspalte wird immer zuerst das rechte, dann das linke Modul gesetzt. Geschützte Module, also die zuvor beschriebenen Muster, werden dabei übersprungen.

Maskierung

Das so entstandene Muster ist häufig nicht optimal für die spätere Bilderkennung. So treten große, gleichfarbige Muster auf oder es besteht eine Verwechslungsgefahr mit den Suchmustern. Aus diesem Grund wird anschließend ein Muster *über den Code gelegt.*
Es stehen insgesamt acht Muster zur Verfügung. Diese werden wie folgt berechnet: Für jedes Modul wird ein Term evaluiert (Siehe Tabelle 3.3). Entspricht dieser dem Wert `1`, so wird dieses dunkel, ansonsten hell gefärbt. Anschließend wird eine Exklusiv-Oder Operation mit dem Code durchgeführt. Die Wahl des Musters ist nicht beliebig. Es werden alle acht Muster erzeugt und unabhängig voneinander auf den Code angewendet. Eine Bewertungsfunktion – Details sollen an dieser Stelle ausgelassen werden – entscheidet anschließend, welche Variante geeignet ist.

Nummer	Muster
0	(row + column) mod 2 == 0
1	(row) mod 2 == 0
2	(column) mod 3 == 0
3	(row + column) mod 3 == 0
4	(floor(row / 2) + floor(column / 3)) mod 2 == 0
5	((row * column) mod 2) + ((row * column) mod 3) == 0
6	(((row * column) mod 2) + ((row * column) mod 3)) mod 2 == 0
7	(((row + column) mod 2) + ((row * column) mod 3)) mod 2 == 0

Tabelle 3.3: Bildungsregeln für QR-Muster

Metainformationen

Abschließend werden noch Metainformationen codiert. Diese beinhalten den verwendeten Fehlerkorrekturmodus, sowie das benutzte Muster. Metadaten werden in einer 15 Zeichen langen Bitfolge dargestellt. Diese wird wie folgt generiert: Bit eins und zwei entsprechen dem Fehlerkorrekturgrad[19]. Entsprechend der Nummerierung in Tabelle 3.3 werden die nächsten drei Bit gewählt. Die restlichen zehn Bit sind Korrekturwörter, die konkrete Berechnung soll an dieser Stelle nicht erläutert werden. Abschließend findet wieder eine Maskierung durch Exklusiv-Oder-Verknüpfung mit dem Bit-Muster `101010000010010` statt. Weil nur die ersten fünf Bit variabel sind, existieren gerade $2^5 = 32$ gültige Zeichenketten. Im Code selbst werden diese Metadaten in den violetten Bereichen (Abbildung 3.7a) doppelt abgelegt.

[19] `L - 01, M - 00,Q - 11, H - 10`

Kapitel 4

Das WossiDiA-Projekt

4.1 Richard Wossidlo

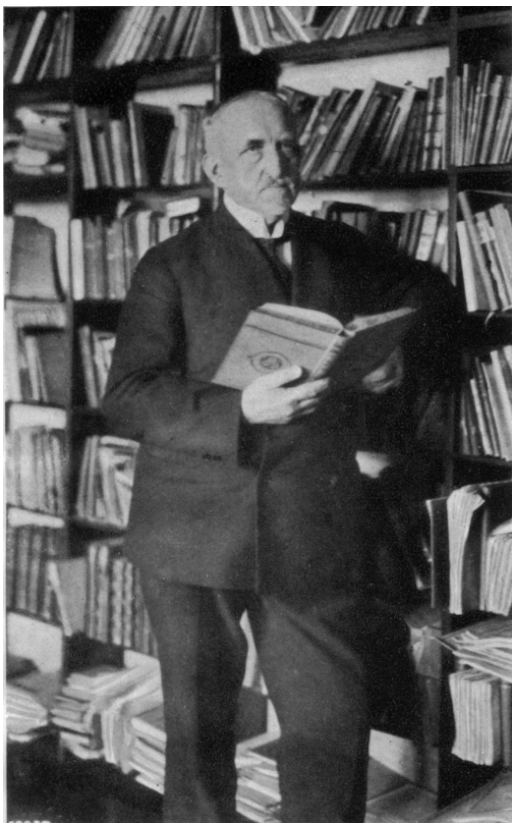


Abbildung 4.1: Richard Wossidlo

Prof. Dr. hc Richard Wossidlo (auch Richard Vosslo[1]) war Gymnasialprofessor in Waren und volkskundlicher Privatgelehrter. In der Zeit von 1884 bis zu seinem Tod 1939 hielt er das damalige Leben der Bewohner von Mecklenburg in allen seinen Facetten auf einer Sammlung von ethnographischen Belegen, den sogenannten *Zetteln* fest, die er entweder durch Korrespondenzen mit Zuträgern erhielt oder oder auf seinen Reisen durch das Land selbst anlegte.

Wossidlo wurde am 26. Januar 1859 auf einem Gut in Friedrichshof bei Tessin, welches seinem Vater gehörte, geboren[30]. Die Eltern waren Alfred Ferdinand und Mathilda Dorothea Wossidlo. Bereits als Richard Wossidlo vier Jahre alt war verstarb der Vater, weshalb er mit seiner Mutter und seinen Geschwistern zunächst nach Bützow und später mit 13 Jahren nach Rostock zog. Dort lernte er am Gymnasium die alten Sprachen, wie Latein und Griechisch kennen und kam das erste mal mit dem Niederdeutschen in Kontakt. Durch seinen Onkel Paul Kohrt, der einen Hof in der Nähe der Stadt besaß, prägte ihn außerdem das ländliche Leben. Von 1876 bis 1882 studierte er in Rostock Klassische Philologie und Archäologie, dabei verbrachte er außerdem einige Semester in Berlin und Leipzig. Schon während des Studiums konnte er sich für die Werke niederdeutscher Autoren, wie etwa Fritz Reuter, John Brinkmann und Klaus Groth begeistern. In dieser Zeit begann er damit, bemerkenswerte Eigenheiten des Plattdeutschen – beispielsweise in Redewendungen – auf Zetteln zu notieren und somit festzuhalten. Nach dem Studium begann er zunächst seine Dissertation, die er jedoch abbrach. Stattdessen erwarb er 1883 die Lehrbefähigung für die Fächer Latein und Griechisch in der Oberstufe. 1884 legte er außerdem das Ergänzungsexamen der allgemeinen Bildung ab. Er arbeitete zunächst an der großen Stadtschule in Wismar, wechselte jedoch 1886 nach Waren an der Müritz, wo er zunächst "Wissenschaftlicher Hülfslehrer" [sic] war. Dort wurde er 1894 zum Oberlehrer und schließlich 1908 zum Gymnasialprofessor ernannt. Im Rahmen seiner Forschungstätigkeit wurde er während der gesamten Zeit als Lehrer lediglich 1891 kurzzeitig von seinen Pflichten entbunden. Die philo-

[1] Siehe `https://nds.wikipedia.org/w/index.php?title=Richard_Vosslo&oldid=775121`

sophische Fakultät der Universität Rostock erteilte Wossidlo 1906 die Ehrendoktorwürde für seine Leistungen. 1919 wurde ihm eine Professur für Niederdeutsche Sprache und Volkskunde angeboten, die er jedoch aufgrund seines Alters von mittlerweile 60 Jahren ablehnte.

Als prägendes Erlebnis nennt Wossidlo Begegnungen mit Tagelöhnern auf dem Gut seines Onkels Hermann Burmeister. Diese lieferten ihm verschiedenste Erklärungssagen für Pflanzenmotive, wodurch ihm der Reichtum der mecklenburgischen Volksüberlieferungen bewusst wurde, was ihn faszinierte. Nachdem er 1884 vorerst alle erforderlichen Examen erhalten hatte, konnte er sich neben seiner Lehrertätigkeit auf seine Sammelleidenschaft für die Eigenheiten des Meckelburgischen Lebens konzentrieren, welche sich bis 1895 vor allem auf Sprachliches fokussierte. Wossidlo ordnete seine gesammelten Redewendungen und Wörter nach einem eigenen System. Bei der Entwicklung dessen half ihm sein Onkel Burmeister, der bereits Erfahrung auf diesem Gebiet hatte, weil er dem Rostocker Germanisten Karl Bartsch, welcher zusammen mit dem Archivar Friedrich Lisch eine systematische Sammlung "Sagen, Märchen und Gebräuche aus Meklenburg " erstellte, zuvor ähnlich zur Hand ging. Wossidlo arbeitete mit einem eigenen Abkürzungssystem und zum dem Ende seiner Schaffenszeit existierten um die 27000 mehrfach gestufte Kategorien. Wossidlo startete mehrere Sammelaufrufe, aus denen sich eine Anzahl von Zuträgern entwickelte, die seine Sammlung in zahlreichen Korrespondenzen erweiterte. Dieser Ansatz war im Grunde nicht neu, etwa die Gebrüder Grimm arbeiteten auf ähnliche Weise, wenn auch auf gesamtdeutscher Ebene. Im Gegensatz dazu reiste jedoch Wossidlo selbstständig durch das Land – Belege dokumentieren über 3000 Orte – und sprach mit mehr als 5000 Leuten. Weil er auch vor Ort stets alle Erzählungen auf Zetteln notierte, erhielt er bald den Spitznamen *Zettelmann*. Gingen ihm die Zettel aus oder hatte er das Gefühl, den Befragten zu verstören, schrieb er häufig heimlich auf seinen Manschetten.

Weil er nahezu auf muttersprachlichem Niveau plattdeutsch sprach und sich sehr gut in die Denk- und Lebensart der Bevölkerung versetzen konnte, erreichte er eine hohe Nähe zu den Befragten. Dies war etwa bei dem bereits erwähnten Karl Bartsch so nicht möglich, da dieser aus Schlesien stammte und zum Verständnis der plattdeutschen Texte auf Helfer angewiesen war. Wossidlo unterschied bewusst zwischen *sue sponte* und *mea sponte*, also ob ein Befragter von sich aus erzählte oder ob dieser erst mithilfe von Suggestivfragen dazu bewegt werden musste. Trotz seiner intensiven Feldforschung schätzte er weiterhin die Beiträge seiner sogenannten *Gewährsleute*. Insgesamt belief sich die Sammlung auf über zwei Millionen ethnographische Belege. Diese beinhalteten 30.000 Sagen, 40.000 Reime und Lieder, 8.000 Schwänke, 2.000 Märchen und 5.000 Rätsel. Weiterhin wurden Erzählüberliefungen, Bräuche, Volksglaube, wenig gebräuchliche Ausdrücke, Redensarten und Sprichwörter, sowie Teile der Arbeits- und Lebenswelt der damaligen Mecklenburger festgehalten, vorwiegend auf Plattdeutsch. Nebenbei sammelte er ab 1900 auch Zeugnisse der Volkskunst, welche im "Bauernmuseum" im Schweriner Schloss ausgestellt wurden. Zum Ende seiner Schaffenszeit wurde seine Arbeit durch das aufstrebende Regime der Nationalsozialisten beeinflusst. Da hier besonders die germanische Herkunft des deutschen Volkes hervorgehoben werden sollte, waren Einflüsse anderer Kulturen unerwünscht. So mussten die Ausführungen über den Schutzgeist des Schweriner Schlosses, welche im zweiten Band seiner Volkssagen zu den Koboldsagen einzuordnen war, weichen, weil das Petermännchen einen slawischen Hintergrund hatte.

Richard Wossidlo war Mitglied in diversen Institutionen. Der *Verein für meklenburgische Geschichte und Alterhumskunde* erteilte ihm 1890 den Vorsitz einer Kommission zur Sammlung von Volksüberlieferungen, für die er bereits seine frühen Sammelerfolge zur Verfügung stellte. Der 1907 in Finnland gegründete *Folklorische Forscherbund* lobte seine Leistungen und ernannte ihn zu einem korrespondierenden Mitglied der *Finnischen Literaturgesellschaft.* Ab 1923 engagierte er sich bei der *Nordischen Rundfunk AG*, wo er im *Kuratorium für niederdeutsches Hörfunkprogramm* saß. So war es ihm möglich, auch über dieses relativ neue Medium seine volkshistorischen Erkenntnisse zu verbreiten. Zu seinem siebzigsten Geburtstag im Jahre 1929 wurde eine *Wossidlo-Stiftung* gegründet, welche sich dafür verantwortlich zeigte, den Nachlass Wossidlos auch nach seinem Tode noch zu erhalten. Er schrieb unter anderem in der *Rostocker Zeitung* und der *Zeitschrift für Volkskunde.* Zu seinen Publikationen gehören Titel, wie *Die Tiere im Munde des Volkes*, *Kinderwartung und Kinderzucht*, *Mecklenburgische Volksüberlieferungen* und diverse Rätselbände. Einige seiner Werke wurden erst posthum veröffentlicht oder gar fertiggestellt. So konnte etwa nur der erste Band der *Mecklenburgischen Sagen* zu Lebzeiten erscheinen, *Reise, Quartier, in Gottesnaam!* wurde durch

Paul Beckmann vollendet und 1940 verlegt. Sein Projekt eines siebenbändigen *Mecklenburgischen Wörterbuches* konnte gar erst 1992 beendet werden. Prof. Dr. hc Richard Wossidlo verstarb am 4. Mai 1939 im Alter von 80 Jahren.

4.2 Das Wossidlo-Archiv

Der Nachlass von Richard Wossidlo umfasst über zwei Millionen Belege. Diese werden in zehn sogenannten Korpussen kategorisiert, welche sich sowohl inhaltlich, als auch in ihrer Struktur unterscheiden.

4.2.1 Das Zettelarchiv Wossidlos (ZAW)

Den größten Korpus stellt das ZAW dar. Es enthält insgesamt über 980.000 Belege, welche in der sogenannten **Zettelwand** systematisch eingeordnet sind und gilt als das *Herzstück* der Sammlung. Sie ist in Abbildung 4.2a abgebildet. Zu den Belegen gehören Wossidlos Feldforschungsbelege und seine fachliterarischen Exzerpte.
Die Zettelsammlung ist in 57 sogenannte Kastengruppen unterteilt, welche im Raster auf der Zettelwand angeordnet sind. Spalten werden mit Großbuchstaben bezeichnet, die Reihen mit römischen Ziffern. Insgesamt beinhaltet das ZAW sechs mal neun Gruppen (`A I`, `A II`, ... `F IX`) sowie `G I`, `G II` und `Z I`. Jede Zettelgruppe beinhaltet vier mal fünf Zettelkästen. Diese Behälter sind für jede Gruppe jeweils mit den Zahlen 01 bis 20 *von oben nach unten, dann von links nach rechts* durchnummeriert. Der in Abbildung 4.2b dargestellte Zettelkasten `C II 19` befindet sich in der dritten Gruppenspalte und der zweiten Gruppenreihe. Innerhalb dieser Gruppe ist es der Kasten in der letzten Spalte, in der vorletzten Reihe. Innerhalb von WossiDiA werden statt der römischen Ziffern gewöhnliche arabische Ziffern verwendet, weshalb der betreffende Kasten dort mit `C-219` bezeichnet wird. Jeder Zettelkasten hat eine Bezeichnung, welche auf den Inhalt schließen lässt, etwa *Haus und Hof*, *Volkserzählungen* oder *Landwirtschaft*. Dabei sind die Bezeichnungen nicht exklusiv, mehrere Kästen können durchaus dieselbe Aufschrift tragen. Ein Zettelkasten beinhaltet eine Vielzahl von Konvoluten, dabei handelt es sich um eine inhaltliche Gruppe von Belegen, die in einem Umschlag zusammengefasst sind. Darauf ist wieder eine Bezeichnung, eine laufende Konvolut-Nummer innerhalb des Kastens und die Anzahl der darin enthaltenen Zettel notiert. Konvolute können thematisch in einem Oberkonvolut zusammengefasst werden. Dabei liegen die Umschläge in einem beschrifteten Faltblatt, dieses ist mit dem Namen des Oberkonvoluts, sowie der Bereich der enthaltenen Konvolute beschriftet. Im Zettelkasten `C II 19` existiert beispielsweise Oberkonvolut *Eheleute*, als Bereich ist `C II-19-10÷19 (80)` angegeben. Dem ist zu entnehmen, dass das Oberkonvolut die Konvolute `C II 19 10` (*Eheleute*) bis `C II 19 19` (*Eheleute varia*) beinhaltet und insgesamt 80 Zettel enthält. Ebenso ist für einzelne Konvolute vermerkt, wie viele Belege diese enthalten. Belege sind – im Konvolut beginnend mit Null – laufend durchnummeriert. Einzelbelege können durchaus mehrseitig sein. So enthält etwa der Beleg `C 219 10 000` zwei Teile: Den Umschlag des Konvoluts *Eheleute*, sowie den des Oberkonvoluts *Eheleute*. Abbildung 4.3 liefert einen Überblick
Einzelne Zettel verfügen in der Regel über drei Bereiche: In der oberen rechten Ecke befindet sich eine Einordnung zur Sachthematik, also eine Art Kategorie. Wossidlo nutzte oft Abkür-

(a) Wossidlos *Zettelwand*

(b) Zettelkasten, Konvolut und Belege

Abbildung 4.2: Impressionen vom ZAW © Universität Rostock, Institut für Volkskunde

zungen, etwa *al weihn* für *Allgemeines zu Weihnachten*. Die Bezeichnung stimmt häufig mit dem Namen des enthaltenden Konvoluts überein. Weiterhin findet sich die eigentliche Information, sowie am Schluss eine Quelle auf dem Beleg. Zudem existieren drei Zetteltypen, die einen unterschiedlichen Aufbau haben: a) Mitteilungen von Erzählerinnen und Erzählern, die Wossidlo im Feld aufgesucht hat, b) Zettel, die auf Befunde seiner mit ihm korrespondierenden Sammelhelfer (BKW) hinweisen und c) Zettel, die Exzerpte publizierter Quellen, zumeist der Fachliteratur, enthalten.

4.2.2 Weitere Korpusse

Weiterhin existieren die folgenden Korpusse, die hier nur kurz erwähnt werden sollen:

- Die Beiträgerkorrespondenz Wossidlos (BKW) beinhaltet sämtliche Korrespondenzen mit Wossidlos Gewährsleuten. Sie umfasst über 54.000 Folioseiten, sowie kleinformatige Anlagen. Belege des ZAW sind häufig mit diesen Beiträgen verknüpft.
- Das Mecklenburgische Wörterbuch (MWW und MWT) umfasst zwei getrennte Bestände mit jeweils etwa 500.000 Seiten. Das MWT wurde aus ZAW und BKW von Wossidlo abgeleitet. Es schafft außerdem die Grundlage für Hermann Teucherts siebenbändiges MWT.
- Das Mecklenburgische Flurnamenarchiv beinhaltet eine Reihe von Namensverzeichnissen und Karten für Landschaften. Es umfasst mehr als 3.000 Seiten.
- Das IBW beinhaltet Seitenscans des Inventarbuch Wossidlos. Dabei handelt es sich um 306 Seiten.
- Für die Orte Neuendorf und Steder-Niendorf wurden Ortschroniken (OCH) festgehalten. Sie umfassen zwei Bücher mit insgesamt 223 Seiten.
- Im Korpus Literaturobjekte (LIT) finden sich zwei von Karl Bartsch herausgegebene Bücher zur Märchen- und Sagenwelt in Mecklenburg. Insgesamt wurden liegen hier 1.086 Seiten vor.
- Abschließend sind außerdem die Publikationen Wossidlo-Archiv (PWA) mit über 2.300 Seiten zu nennen.

4.3 Digitalisierung und Langzeiterhaltung des Archivs

In Kooperation mit der DFG wurde das Wossidlo-Archiv digitalisiert. Aufgrund dessen besonderen Form, die sich durch eine Vielzahl heterogener, untereinander stark vernetzter Dokumente auszeichnet, waren etablierte Arbeitsabläufe nur bedingt anwendbar, beziehungsweise nicht optimal, weshalb ein eigener, semi-automatischer Digitalisierungsprozess angewendet wurde[44].

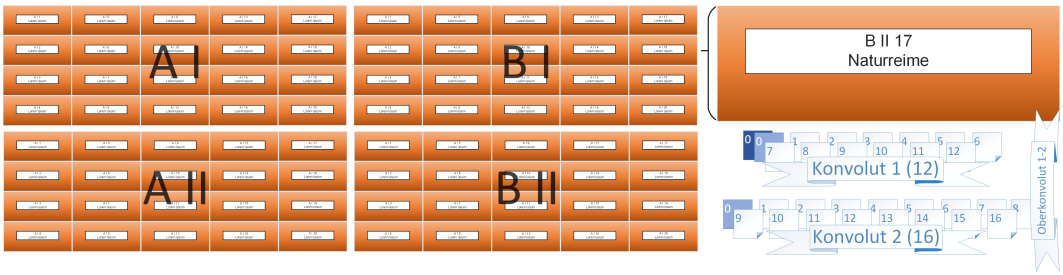


Abbildung 4.3: Schematische Darstellung des ZAW: Im linken Teil die Kastengruppen `A I`, `A II`, `BI` und `B II`. Rechts daneben Kasten `B II 17` mit Beschriftung. Darunter die darin enthaltenen Belege, geordnet in zwei Konvoluten, die in einem Oberkonvolut zusammengefasst sind. Dunkelblaue Belege stellen Umschläge für Oberkonvoluten, hellblaue solche für Konvoluten dar.

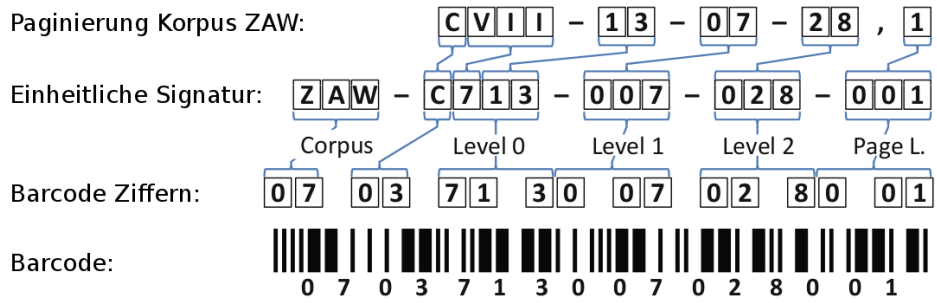


Abbildung 4.4: Beispielhafte Abbildung der Paginierung auf eine einheitliche Signatur, sowie auf einen Barcode, entnommen aus [44]

Wichtig für alle durchzuführenden Schritte war die Kennzeichnung zusammengehöriger Gruppen von Dokumenten, sowie der Dokumente selbst, mithilfe einheitlicher Signaturen. Das Unterfangen war nicht-trivialer Natur, da sich die verschiedenen Korpusse in ihrer Struktur unterscheiden. Die letztendlich gewählte Form der Signaturen bildet die topologische Ordnung (Korpus - Ebene 0 - Ebene 1 - Ebene 2 - Seite) auf eine Folge von Ziffern ab. Abbildung 4.4 verdeutlicht dies in einem Beispiel.
Die Topologieinformationen wurden zusammengetragen und in einer relationalen Datenbank festgehalten[2]. Aufgrund dieser Informationen wurden für Struktureinheiten höherer Ebenen – im ZAW sind dies etwa Zettelkästen und Konvolute – ITF-Barcodes generiert. Zusammen mit weiteren Metainformationen wurden diese auf Etiketten festgehalten und den jeweiligen Dokumentgruppen zugeordnet. Diese wurden etappenweise an einen externen Dienstleister zur Digitalisierung übermittelt.
Sämtliche Digitalisate wurden anschließend einem Evaluationsprozess unterzogen, sodass zu einem späteren Zeitpunkt eine erneute Digitalisierung stattfinden konnte, wenn die Qualität nicht den Anforderungen genügte. Die Scans wurden anschließend in das digitale Archiv überführt, wobei anhand der Barcodes eine Vorsortierung vorgenommen werden konnte. Eine manuelle Einsortierung durch eingewiesene Projektmitarbeiter war jedoch unumgänglich.
Hierbei fand auch der wohl interessanteste Schritt des Digitalisierungsprozesses statt. Das Archiv besteht aus einer enormen Anzahl relativ kurzer Dokumente. Diese beinhalten eine Menge von Referenzen auf verschiedenste Objekte, etwa Personen mit unterschiedlichen Rollen, Orte, Korrespondenzen, andere Objekte und vieles mehr. Speziell geschulte Mitarbeiter extrahierten diese Metadaten und pflegten sie in das digitale Archiv ein. Dabei wurde eine Hypergraphstruktur implementiert, die später noch erläutert werden soll.
Weiterhin wurden die Digitalisate zur Sicherungsverfilmung auf Mikrofilm verwendet. Abhängig von Größe und Format der Dokumente wurden bis zu acht Abbildungen in ein Frame eingebettet und jeweils mit Barcode und Metainformationen, wie etwa den Originalmaßen und der Auflösung versehen. Ebenfalls wurden Trennblätter eingefügt, welche den Beginn einer Struktureinheit signalisieren, etwa ein Kasten oder ein Konvolut. Diese enthalten ebenfalls Metadaten, beispielsweise die Bezeichnung der Einheit und natürlich die Signatur in Klartext und Barcode. Referenzen zwischen den Dokumenten wurden in der Sicherungsverfilmung nicht abgebildet, diese sind Aufgabe dieser Arbeit.

4.4 Gerichtete, typisierte Hypergraphen als Datenmodell

Im Zuge der Digitalisierung des Wossidlo-Archivs wurde außerdem das Projekt *a HYpergraph of Documents in a Relational Archive (HyDRA)* gestartet[3]. Dessen Zielsetzung ist die Entwicklung von Werkzeugen zur effizienten Speicherung und Verarbeitung hochgradig vernetzter Strukturen, insbesondere im Umfeld digitaler Bibliotheken und Archivsysteme. Dabei wurde ein Modell gewählt, welches gerichtete, getypte Hypergraphen verwendet. Es wurde eine zunächst stark mit WossiDiA verknüpfte Implementierung realisiert, welche auf

[2]Nach Angaben der Projektmitarbeiter werden Bestrebungen angestellt, diese Topologieinformationen auf Hypergraphstrukturen abzubilden, um ein einheitlicheres Datenmodell zu erhalten.
[3]Siehe `https://dbis.informatik.uni-rostock.de/forschung/langzeitrahmenprojekte/hydra/`, aufgerufen am 21.03.2017

objekt-relationale Datenbanksysteme, wie etwa PostgreSQL aufsetzt. Diese erfüllte die gestellten Anforderungen und wird derzeit in Form des **Power.Graph**-Systems entkoppelt, um auch anderweitig einsetzbar zu sein.

4.4.1 Kurze Einführung in Hypergraphen

Hypergraphen stellen eine Verallgemeinerung des Graphenkonzepts dar, bei dem insbesondere Kanten mehr als zwei Knoten miteinander verbinden, womit sie über die Ausdruckskraft *gewöhnlicher* Graphen hinausgehen. Die einfachste Variante stellt der ungerichtete Hypergraph dar, wie er durch Berge[3, 5] vorgestellt wurde. Dabei ist ein endlicher Hypergraph $H = (V, \mathcal{E})$ als Tupel aus einer endlichen Kantenmenge V, sowie einer Menge $\mathcal{E} \subseteq \mathfrak{P}(V)$ von Teilmengen der Knoten definiert.
Auch gerichtete Hypergraphen stellen eine Verallgemeinerung ihrer *einfachen* Pendants dar. In Power.Graph werden sie als Tupel $G = (V, A)$ definiert, V ist wieder die Menge der Knoten, A stellt die gerichteten Hyperkanten dar. Letztere erweitern allerdings die Definitionen, welche sich in der Literatur finden lassen[4]. Jede gerichtete Hyperkante ist ein Paar $a = (T, H)$, wobei $T, H \subseteq V \times R$. Dies entspricht einer Menge von eingehenden Knoten (Tail oder b-arc, für backwards-arc) und einer Menge von ausgehenden Knoten (Head oder f-arc, für forward-arc) pro Hyperkante. Dabei ist jeder der Knoten außerdem mit einer Rolle $r \in R$ versehen. Es sei betont, dass die Mengen T und H nicht zwingend disjunkt sein müssen, wie es etwa bei Gallo der Fall ist.
Weiterhin sind Hypergraphen in Power.Graph getypt. Zum Typsystem gehören die Knotentypen Γ^V, Kantentypen Γ^A, die bereits erwähnten Rollen R, sowie eine Menge von Abbildungen $\{\alpha^V, \alpha^A, \rho^V, \rho^A, \delta\}$. Die Funktionen $\alpha^V : V \mapsto \Gamma^V$ und $\alpha^A : A \mapsto \Gamma^A$ weisen Knoten respektive Kanten einen Typen zu. Analog bilden $\rho^V : R \mapsto \Gamma^V$ und $\rho^A : R \mapsto \Gamma^A$ Rollen auf Knoten- und Kantentypen ab. $\delta : R \mapsto \{-1, 0, 1\}$ definiert die Richtung, wobei Rückwärtskanten -1 und Vorwärtskanten 1 entsprechen. Bidirektionale Kanten erhalten den Wert 0. Mit dem Typsystem sind auch einige Integritätsbedingungen verknüpft, die insbesondere voraussetzen, dass Knoten- und Kantentypen den jeweiligen Rollen entsprechen und die Richtung von ein- und ausgehenden Kanten festlegen:

- $\forall (v, r) \in T \cup H : \rho^V(r) = \alpha^V(v)$
- $\forall (v, r) \in T \cup H : \rho^A(r) = \alpha^A((T, H))$
- $\forall (v, r) \in T : \delta(r) \in \{-1, 0\}$
- $\forall (v, r) \in H : \delta(r) \in \{0, 1\}$

Aufgrund dieser Definitionen können nun diverse Sachverhalte modelliert werden. Dabei stehen Knoten für Entitäten der Anwendungsdomäne, etwa Personen, Orte oder Dokumente und Kanten setzen diese in einen Zusammenhang. Mithilfe des Typsystems lassen sich dabei Regeln festlegen die besagen, welche Arten von Knoten, obligatorisch oder optional, mit welcher Kardinalität und Richtung in einem solchen Sachverhalt enthalten sind. Zur Definition der Knotentypen wird eine Power.Graph-eigene Data Definition Language (DDL) verwendet.

4.4.2 Beschreibung des HyDRA-Datenbank-Schema

Es folgt nun eine Übersicht zum aktuellen Stand der HyDRA-Datenbank. Sie ist die Datenquelle für diese Arbeit. Hierbei handelt es sich um ein historisch gewachsenes Projekt, weshalb durchaus Inkonsistenzen auftreten können. Aufgrund der Erfahrungen mit diesem System entsteht Power.Graph.
Informationen über Knoten sind auf mehrere Tabellen verteilt. Dabei entspricht eine Tabelle immer einem Knotentypen, etwa werden in `w.am_person` Daten über Personen gespeichert. In jeder dieser Tabellen fungiert die Spalte `id` als Primärschlüssel, ein konkreter Knoten kann also aus einer Kombination dieser Zahl mit dem Tabellennamen eindeutig identifiziert werden. Insgesamt existieren 53 solcher Knotentypen. Für Power.Graph ist geplant, sämtliche Knoten in einer Tabelle zu verlagern, wodurch der Primärschlüssel für alle Instanzen eindeutig ist. Knotenattribute und deren Typen werden in andere Tabellen ausgelagert, eine Hierarchie von Knotentypen ist ebenfalls vorgesehen.

[4] Genannt seien hier die Arbeiten von Ausiello et al.[2] und Gallo et al.[16], wobei letztere die Grundlage für die Definition in Power.Graph ist.

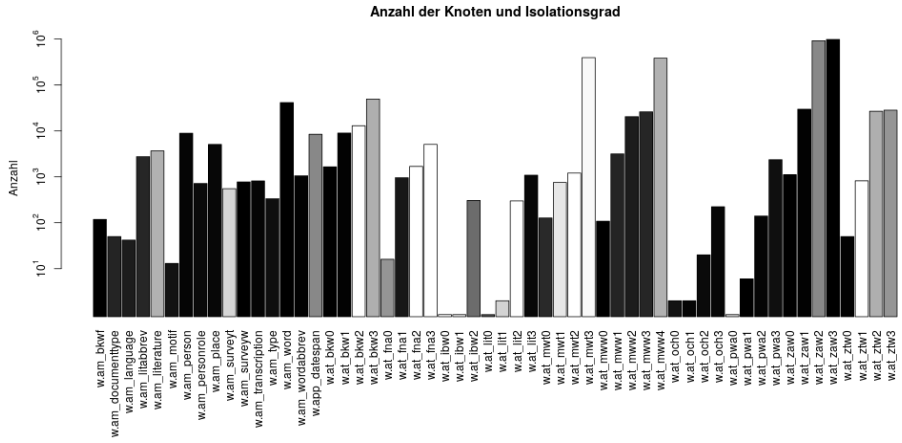


Abbildung 4.5: Anzahl von Knoten pro Knotentyp und deren Isolationsgrad.

Eine große Teilmenge der Knotentypen sind die Topologie-beschreibenden Knoten. Sie beginnen mit dem Präfix `at_`. Es folgt anschließend das Kürzel des dazugehörigen Korpus und die Ebene der enthaltenen Daten. So enthält die Tabelle `w.at_zaw0` Metadaten über die Kästen des ZAW, `w.at_zaw1` beschreibt Konvolute, `w.at_zaw2` die darin enthaltenen belege und abschließend `w.at_zaw3` die Seiten. Als bedeutsame Inkonsistenz im Hypergraphmodell ist hier hervorzuheben, dass die Beziehungen zwischen den einzelnen Ebenen der Topologie nicht mithilfe von Hyperkanten festgehalten wurde. Stattdessen werden Fremdschlüssel verwendet. Dies soll im Power.Graph geändert werden.
Metadaten über die Knoten, wie etwa Namen und Beschreibungen sind in `w.cat_table` zu finden. Speziell für die Topologie übernimmt dies die Tabelle `w.cat_topology`. Informationen zu den Attributen der einzelnen Knoten speichert `w.cat_attribute`. Abschließend liefert die Tabelle `w.cat_metadata` eine Abbildung von Tabellen auf Kürzel, die zur Generierung von Metadaten-Signaturen nötig sind.
Letztendlich sind außerdem Informationen zu den Hyperkanten von Interesse. Diese werden primär in `w.hedge` gespeichert, wobei vor allem der Primärschlüssel `id`, sowie die Referenz auf den Typen wichtig sind. Letztere sind in der Tabelle `w.hedget` gespeichert. Im Kontext der Arbeit liefern insbesondere die Felder `name` und `descr` (Beschreibung) Aufschluss zur Verwendung der jeweiligen Typen. Knoten und Hyperkanten werden über getypte Bögen miteinander verknüpft. Diese sind in der Tabelle `w.hlink` zu finden und referenzieren über Fremdschlüssel die Hyperkante auf der einen und den Knoten, identifiziert durch Tabellenname und `id`, auf der anderen Seite. Informationen zu den Bogentypen sind `w.linkt` zu entnehmen. Das umfasst insbesondere die Beschreibung der Rolle, wie auch die Richtung des Bogens. Bogentypen sind immer direkt mit einem Kantentyp verknüpft.

4.4.3 Weitere Informationen zur Datenbank

Es folgen nun einige informative Feststellungen zum Hypergraphen und dessen Bestandteilen. In der Datenbank sind insgesamt 2.939.731 Knoten in 57 Knotentypen, sowie 339.833 Hyperkanten in 54 Kantentypen hinterlegt. Überraschenderweise ist der Hypergraph – bildet man ihn auf seine Entsprechung eines bipartiten, ungerichteten Graphen ab – nicht nur nicht-zusammenhängend, sondern besteht sogar aus über 7.000 Komponenten, wobei die isolierten Knoten nicht mitgezählt werden. Letztere machen den Großteil des Datensatzes aus: Von den knapp drei Millionen Knoten sind gerade einmal 256.677 mit einer Hyperkante verbunden. Insgesamt 22 Knotentypen umfassen ausschließlich isolierte Knoten. Abbildung 4.5 zeigt die Anzahl der Knoten je Knotentyp, sowie der Anteil der isolierten Knoten je Typ.

Schnell wird ersichtlich, warum dies so ist: Der Löwenanteil der nicht im Hypergraphen verknüpften Knoten ist in den Topologie-Typen zu finden. Wie bereits im letzten Abschnitt beschrieben, handelt es sich hierbei noch um eine Inkonsistenz, die Topologie ist nicht durch den Hypergraphen, sondern durch Fremdschlüsselbeziehungen beschrieben. Wird dies nachgepflegt, so ist davon auszugehen, dass der Anteil der isolierten Knoten, wie auch die Anzahl

der Graph-Komponenten stark reduziert wird. Von einem zusammenhängenden Graphen auszugehen ist nicht abwegig: Da jedes Objekt mit einem Topologieelement verbunden ist und die Topologie eine Baumstruktur hat, ist auch der Hypergraph zusammenhängend.

Kapitel 5

Konzeption einer Strategie zur Langzeitarchivierung von Metadaten

5.1 Vorstellung eines fiktiven Entdeckungsszenarios

Es soll nun exemplarisch ein Entdeckungsszenario vorgestellt werden. Dieses beginnt mit dem Auffinden der verfilmten Materialien, welche die Resultate dieser Arbeit darstellen, führt über eine Einführung und *Bedienungsanleitung* zur hoffentlich vollständigen digitalen Rekonstruktion der Metadaten.
An einem unbestimmten Tag in der Zukunft werden die verfilmten Materialien in einem Bergungsort, wie etwa dem **Barbarastollen** gefunden. Die Umstände dieser Entdeckung können unterschiedlicher Natur sein. So kann etwa eine gezielte Suche nach den Sicherungen veranlasst worden sein, weil die Originale kürzlich zerstört wurden oder eine Forschungsgruppe sich für das Leben im ländlichen Mecklenburg des späten neunzehnten Jahrhunderts interessiert und andernorts Hinweise auf diese Dokumente gefunden haben. Weitaus düstere Szenarien sind nach katastrophalen Ereignissen, wie etwa einem dritten Weltkrieg oder ähnlichem angesiedelt. Die Rechentechnik der auffindenden Zivilisation sollte der gegenwärtigen nicht wesentlich unterlegen sein. Zum einen darf die Implementierung eines effizienten Lesegeräts für QR-Codes kein Hindernis darstellen, wenn die dafür notwendigen Instruktionen geliefert werden. Andererseits müssen größere Datenmengen – also die WossiDiA-Metadaten – effizient handhabbar sein, damit etwa Suchen durchgeführt und die Daten überhaupt genutzt werden können. Es wird angenommen, dass gegenwärtiges Deutsch oder zumindest Englisch, sowie die dazugehörige Schriftsprache weiterhin interpretierbar sind.
Der Inhalt der Filmdosen ist von außen ersichtlich, etwa durch einen Titel und eine kleine Beschreibung. Wurde mehr als eine Filmrolle benutzt, so ist der Umfang ebenfalls notiert. Bereits ohne Hilfsmittel ist erkennbar, dass auf dem Film optische Informationen festgehalten sind. Mithilfe geeigneter, fast trivialer Gerätschaften kann eine Vergrößerung vorgenommen werden, um mehr Details in Erfahrung zu bringen. In einer Einführung werden Sinn und Zweck der in der Sicherung enthaltenen Daten erläutert. Dazu gehört in erster Linie, dass es sich um **ergänzende** Informationen zur Verfilmung von Wossidlos Archich handelt. Entsprechend ist hinterlegt, wo diese zu finden sind oder – für den Fall dass diese Referenz zu dem Zeitpunkt nicht mehr gültig ist – wie die Sicherung des Archivs eindeutig erkennbar ist, wenn man sie findet. Sowohl in natürlicher Sprache, wie auch mithilfe geeigneter bildlicher Mittel, sind einige Beispiele der Metadaten aufgeführt. Diese verdeutlichen, womit der Finder es zu tun hat und schaffen Motivation zur weiteren Erfoschung des Inhalts der Filmrollen.
Für den Fall, dass die Einführung das Interesse der Finder geweckt hat, folgt ein erläuternder Abschnitt, welcher den weiteren Umgang klärt. Sie umfasst zunächst eine erschöpfende Beschreibung des Metadaten-Konzepts und wie diese Daten mit den Elementen des Archivs verknüpft sind. Weiterhin wird die Struktur des Dokuments beschrieben. Mithilfe technischer Anleitungen wird dem Lesenden das Verständnis geschaffen, Werkzeuge zu entwickeln, um die nachfolgenden, optischen Darstellungen digitaler Informationen zu interpretieren und in

eigene datenverarbeitende Anlagen zu überführen. Sofern es passend und möglich ist werden dabei Referenzen auf externe Quellen, etwa nationale und internationale Standards genannt. Auf diese Weise ist das Erkennen und Verwenden eventuell bereits existierender Werkzeuge möglich. Dennoch muss die Rekonstruktion auch ohne externe Dokumente möglich sein.
Die Nutzung ist grundsätzlich in zwei Varianten denkbar. Sofern möglich, ist es zu bevorzugen, dass sämtliche Materialien auf einmal eingelesen werden und so das gesamte Metadatenmodell verfügbar ist. Dabei ist die Reihenfolge des Lesens der einzelnen optischen Codes irrelevant. In diesem Fall kann angenommen werden, dass effiziente Verfahren existieren, um gezielt nach Informationen zu suchen. Weil allerdings nicht davon auszugehen ist, dass alle optischen Codes wiederherstellbar sind, ist ein Verlust von Daten grundsätzlich möglich. Ist es nicht handhabbar, den gesamten Datensatz als Ganzen zu nutzen, etwa weil die Kapazitäten der verfügbaren Computersysteme nicht ausreichend sind oder weil die entsprechende Software noch in der Entwicklung ist und ungeduldige Forscher bereits eine Nutzung erwägen, so soll auch dies möglich sein. Hierfür wird lediglich ein Lesegerät für die optischen Codes benötigt. Möchte jemand den Datensatz manuell erkunden, müssen Hilfen zur Navigation angeboten werden. Ähnlich einer Zugriffsstruktur in den gegenwärtigen Datenbanksystemen, wird der Nutzer Schritt für Schritt zur gesuchten Information geführt.

5.2 Konzeption der Metadatenkodierung

Wie ist die Speicherung der digitalen Metadaten auf Film nun zu realisieren? Dazu folgen nun einige Ausführungen, welche die zu unternehmenden Schritte grob zusammenfassen. Dabei werden zwei unterschiedliche Ansätze vorgestellt.
In der ersten Methode soll der Hypergraph direkt in RDF dargestellt werden. Dies stellt scheinbar die einfachste Methode dar, da – wie es sich noch zeigen wird – generische Möglichkeiten existieren, um dies zu bewältigen. Dazu wird der Hypergraph in einen bipartiten gerichteten Graph überführt.
Um das **Paradox of Digital Preservation**[8] zu bestätigen, sollen in der zweiten Variante der Metadatenkodierung die Daten nicht als Hypergraph in eine geeignetere Darstellung überführt werden. Die ursprüngliche Form eignet sich nur bedingt als Austauschformat, da es sich hierbei eher um eine Insellösung handelt. Werden die Metadaten eines Tages rekonstruiert, so müssen Werkzeuge speziell für diese Form neu geschaffen werden. Insbesondere die Verknüpfung mit anderen Modellen wird erschwert, da Schnittstellen geschaffen, beziehungsweise diese in ein gemeinsames Modell überführt werden müssen. Erstrebenswert ist daher die Verwendung von geläufigeren Modellen, wie dem CIDOC Conceptual Reference Model (CIDOC CRM). Dabei handelt es sich um eine standardisierte Ontologie, welche speziell die Beschreibung von Konzepten und Beziehungen in der Archivierung kulturellen Erbes umfasst. Sie wurde in langjähriger Arbeit von Experten entwickelt und wird von diversen Organisationen verwendet, etwa der National Library in London[1]. Somit ist die Grundlage dafür geschaffen, dass die Sachverhalte in einem für Experten einheitlich und verständlich formuliert werden. Die grundsätzliche Machbarkeit dieser Überführung demonstrieren Kiesendahl und Meyer et al. in [29, 36]. Es wird sich zeigen, dass dies die zu bevorzugende Variante darstellt.
Das in CIDOC CRM formulierte Modell muss in einem maschinenlesbaren Format vorliegen. Hier bietet sich ebenfalls das RDF an. Dieses formuliert Sachverhalte in Subjekt-Prädikat-Objekt-Tripeln. Jeder dieser drei Bestandteile der Tripel wird durch einen Typ festgelegt, wobei diese im Wesentlichen durch das CIDOC CRM bereitgestellt werden. Entsprechend der späteren Speicherung auf Mikrofilm muss das Modell in einer geeigneten Form zerlegt werden, sodass sich dessen Bestandteile mithilfe optischer Codes darstellen lassen. Hierbei ist es wünschenswert, diese Zerlegung so vorzunehmen, dass die daraus resultierenden Einheiten systematisch auf den Filmen und Frames verteilt werden kann. So wäre es beispielsweise möglich, einen manuellen Suchvorgang zu erleichtern, in dem Elemente nicht wahllos, sondern gut auffindbar platziert sind.
Letztendlich sollen noch Optimierungsmöglichkeiten untersucht werden, etwa die Möglichkeiten der platzsparenden Darstellung, wie auch verbesserte Redundanz, damit im Falle des Verlustes einzelner Codes oder eventuell gar ganzer Mikrofilmrollen der Verlust an Daten minimiert wird.

[1] Siehe `http://research.ng-london.org.uk/wiki/index.php/National_Gallery_API`

5.2.1 Direkte Darstellung des Hypergraphen in RDF

An dieser Stelle soll noch einmal eine kurze Zusammenfassung des Modells folgen, um festzustellen, welche Informationen gesichert werden müssen. Die eigentlichen Datenobjekte stellen die **Knoten** dar. Sie sind außerdem mit **Attributen** ausgezeichnet. Die Verbindung der Knoten wird durch `Hyperkanten` realisiert. In der Datenbank selbst wird der Hypergraph als bipartiter Graph dargestellt. Das bedeutet, dass Hyperkanten wiederum als Knoten dargestellt werden, welche mit den Hypergraph-Knoten über Kanten (Bögen) verbunden sind. Diese **Bögen** verfügen insbesondere über eine Richtung und eine Rolle.
Als Grundlage für diesen Ansatz dient die Tatsache, dass sich der Hypergraph als gerichteter, bipartiter Graph darstellen lässt. Dabei wird zwischen den beiden Knotentypen `HyDRA-Knoten` und `Hyperkante` unterschieden. Entsprechend der vorhandenen HyDRA-Knoten werden Unterklassen erstellt, etwa `Person`, `Wort` oder `Ort`. Selbiges geschieht für die Hyperkanten. Klassen, wie `Inhalt` oder `Synonym` werden zu Stellvertretern der jeweiligen Kantentypen.
Attribute werden als Prädikate realisiert, entsprechend wird ein neues Prädikat festgelegt. Anschließend werden Sub-Prädikate für jedes bekannte Attribut gebildet. Deren Definitionsbereich ist immer der konkrete Knotentyp, als Wertebereich genügt im Rahmen dieser Arbeit die Literalklasse. Entsprechend der tatsächlichen Typen kann selbstverständlich eine spezifischere Unterklasse verwendet werden.
Als Verbindung zwischen Knoten und Hyperkanten dienen die Bögen. Für eine Definition wird jeweils eine Klasse für vorwärts, wie auch rückwärts gerichtete Bögen benötigt, da die Werte- und Defintionsbereiche jeweils vertauscht sind. Entsprechend werden auch hier wieder spezifische Subproperties gebildet. Eine Übersicht über das Schema gibt Abbildung 5.1.

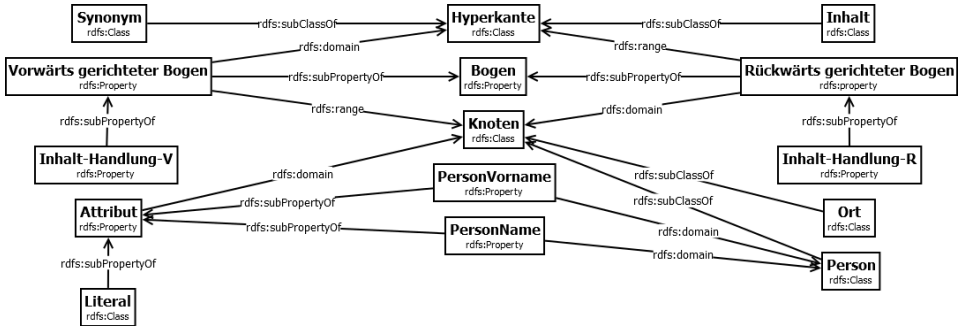


Abbildung 5.1: Schematische Darstellung des RDF-Schema (RDFS) zur Darstellung des Hypergraphen. Zur besseren Lesbarkeit werden Prädikate der Spezialisierungen ausgelassen.

Ein alternatives Modell für Hypergraphen in RDF schlägt [37] vor. Dieses ist allerdings nicht anwendbar, da hier keine Entsprechung der Rollen (Typen der Bögen) existiert. Aus diesem Grund wird die eigene Lösung bevorzugt.

5.2.2 Überführung des Hypergraphenmodells in CIDOC CRM

Das CIDOC CRM stellt eine standardisierte Ontologie für den Austausch von Daten, insbesondere im Bereich des kulturellen Erbes dar. Aufgrund dieser Standardisierung und der Verbreitung ist davon auszugehen, dass – zumindest gegenwärtig – Domänenexperten Informationen, welche mithilfe des Referenzmodells dargestellt wurden, verstehen. Weiterhin besteht die Möglichkeit, existierende Systeme, welche dieselbe Modellierung verwenden, miteinander zu verknüpfen, ohne diese zuvor zu transformieren. Daher ist es erstrebenswert, den Hypergraphen vor der Langzeitarchivierung in das CIDOC CRM zu überführen.
Dass die Überführung des Hypergraphenmodells in eine CIDOC CRM-kompatible Form möglich ist, zeigt Kiesendahl in seiner Bachelorarbeit[29]. Darüber hinaus erläutert er konkrete Regeln zur Überführung einer Auswahl an Hyperkantentypen und gibt eine Übersicht über die dazu notwendigen Entitäten und Eigenschaften. Allerdings ist der von ihm gewählte Ansatz nicht immer optimal, wie die folgende Veranschaulichung zeigen soll.
Als Beispiel soll der Hyperkantentyp **Inhalt** dienen. Dieser wird in der Datenbank durch die `id = 1019` identifiziert. Zum Zeitpunkt der Arbeit existieren 118.912 Kanten diesen Typs.

Quelle				Handlung		Kontext	
0	ZAW0	13	BKW2	3	Worte	17	Worte
1	ZAW1	14	BKW1	4	Orte		
2	ZAW2	15	MWT2	5	Rollen		
6	MWW1	16	MWT3	19	Personen		
7	MWW2	20	ZTW2	21	Motive		
8	MWW3	23	Transkriptionen	22	Typen		
9	W-Fragen	24	OCH1				
10	T-Fragen	25	OCH2				
11	MWW4	26	OCH3				
12	BKW3						

Tabelle 5.1: Erlaubte Bogen- und Knotentypen der Hyperkante *Inhalt*

Das Modell[2] ist wie folgt definiert:

```
seq(alt(9, 10, 0, 1, 2, 14, 13, 12, 6, 7, 8, 11, 15, 16, 20, 23, 24, 25, 26),
    rep(alt(3, 4, 19, 5, 18, 21, 22), 1,100),
    rep(17, 0, 100))
```

Bei den Ziffern in der ersten Zeile handelt es ausschließlich sich um Bögen des Typs `Quelle`. Zeile zwei beinhaltet `Handlungs`-Bögen, ein Bogen vom Typ `Kontext` schließt die Definition ab. Dies ist wie folgt zu interpretieren: Eine Hyperkante des Typs `Inhalt` umfasst genau eine Quelle, mindestens einen, jedoch maximal einhundert Handlungen, sowie bis zu einhundert Kontexte, wobei letzterer optional ist. In Tabelle 5.1 sind die drei Knoten-, sowie die dazugehörigen Knotentypen mitsamt der entsprechenden Ziffern zur Übersicht abgebildet. Alle Bögen sind bidirektional. Kiesendahl formuliert zur Überführung die folgende, intuitiv verständliche Regel zur Abbildung auf CIDOC CRM:

```
[Inhalt: Quelle, Handlung*]
        Inhalt = E7 Handlung (von Typ Inhalt)
        Inhalt -> Quelle = E7 : P70 wird belegt in: E31 Dokument
        Inhalt -> Handlung = E7 : P15 wird beeinflusst durch: E55 Typus
```

Vergleicht man diese mit der Hyperkantendefinition, so stellt man fest, dass der Kontext nicht beachtet wird. Die Verwendung dieser Abbildungsregeln würde also einen Informationsverlust bedeuten, was selbstverständlich nicht gewünscht ist. Weiterhin beachtet diese Regel nicht ausreichend, wie die referenzierten Knotentypen konkret aussehen. So haben Personen einen Namen, Vornamen, ein Geschlecht, einen Titel, eine Rolle und weitere Attribute. Eine Person als `E55 Type`-Objekt darzustellen würde demnach der Ausdrucksfähigkeit des CIDOC CRM nicht gerecht werden. Je nach Bedeutung sollte ein Knotentyp auf eine entsprechende Klasse des Referenzmodells abgebildet werden. Weiterhin führt diese Herangehensweise zu Problemen, wenn derselbe Knoten in unterschiedlichen Regeln auf verschiedene CIDOC CRM-Objekte abgebildet wird. Führen wir einen Hyperkantentyp *Familie* ein und bilden diesen auf das Referenzmodell ab, so benötigen wir offensichtlich die Eigenschaft `P152 ist Elternteil von`. Diese ist nur auf Objekte vom Typ `E21 Person` anwendbar, nicht aber auf `E55 Typ`, weshalb ein zusätzliches Objekt erzeugt werden müsste. Es sei daher vorgeschlagen, den Überführungsvorgang mit einer Abbildung der Knoten zu beginnen, bevor diese anschließend aufgrund der Hyperkanteninformationen in Relation gesetzt werden.

Überführung der Knoten in CIDOC CRM

Die folgenden Ausführungen sollen zuerst auf allgemeine Probleme hinweisen und dann beispielhaft einige Überführungen von Hypergraphknoten in einer CIDOC CRM-Darstellung erläutern. Um unnötige Verwirrungen zu vermeiden, werden ab sofort ausschließlich die englischen Bezeichnungen von CIDOC CRM-Bestandteilen verwendet. Als Grundlage wird die aktuelle Version 5.0.4 des Referenzmodell angewendet.

[2] `seq(a,b,c)` - Sequenz `a b c`, `alt(a,b,c)` - Auswahl `a`, `b` oder `c`, `rep(a,m,n) - a tritt m bis n mal auf. Bogentypen a, b, c, m, n` $\in \mathbb{N}$

Allgemein Bei der Überführung der Knotentypen in das CIDOC CRM handelt es sich um einen nicht-trivialen Prozess, der sowohl ein Verständnis der zugrundeliegenden Daten, wie auch des Referenzmodells voraussetzt. Im Wesentlichen wird dabei zuerst eine Klasse gesucht, die dem Knotentypen am ehesten entspricht. Anschließend ist eine Möglichkeit zu suchen, die gegebenen Attribute ebenfalls mithilfe des Referenzmodells darzustellen und mit dem *Hauptobjekt* zu verknüpfen. Weil das CIDOC CRM in jahrelanger Arbeit erfahrener Fachleute entstanden ist, und sich dabei stets an realen Problemen orientiert wurde, ist davon auszugehen, dass im Regelfall entsprechende Klassen zur Verfügung stehen. Für konkrete Fälle wurden außerdem gar Entwurfsmuster ausgearbeitet, an denen man sich beim Modelliervorgang orientieren sollte.

Häufig möchte man im Modell Informationen über eine Klasse von Objekten festhalten. Beispielsweise kann bei der Überführung des Knotentypen `Erzählmotiv` gewünscht werden, dass das entsprechende CIDOC-Objekt, welches eine Instanz der Klasse `E90 Symbolic Object` sein kann, auch als *Erzählmotiv* ausgezeichnet wird. Zusätzlich ist es sinnvoll, eine Beschreibung dieser Klasse von Objekten zu modellieren. Eine solche ist häufig im Hypergraphenmodell vorhanden. Hierfür gibt es zwei mögliche Ansätze:

Die erste Möglichkeit ist es, dem Objekt einen Typen zuzuordnen. Das ist mithilfe der Eigenschaft `P2 has type`, sowie dem Typen `E55 Type` problemlos realisierbar. Die anschließende Beschreibung kann durch `P3 has note` und einem `E62 String` vorgenommen werden. Als Vorteil dieser Herangehensweise ist zu nennen, dass ausschließlich im Standard festgehaltene Klassen verwendet werden, was der Verständlichkeit der Modellierung zugute kommt. Abbildung 5.2 stellt diesen Ansatz in einem Beispiel dar.

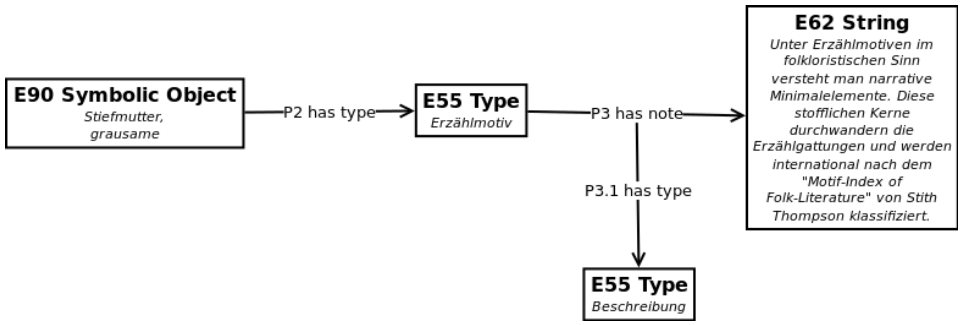


Abbildung 5.2: Typisierung eines Objektes und Beschreibung der Klasse

Die alternative und wahrscheinlich elegantere Variante ist es, das Referenzmodell zu erweitern. Diese Möglichkeit wird explizit angeboten und sollte in einem gesunden Maße auch genutzt werden. Auf diese Weise könnte eine Unterklasse von `E90 Symbolic Object` dem Modell hinzugefügt und verwendet werden. Diese könnte etwa `E90-1-Erzählmotiv` heißen. Aufgrund der Substitutionseigenschaften kann es dann überall dort erscheinen, wo ein Objekt der Elternklasse gefordert ist. In der Schemadefinition, beziehungsweise in der Erweiterung der Ontologie kann dann die Beschreibung der Klasse angeführt werden. Durch die nun obsoleten `E55 Type` Objekte kann im weiteren Verlauf Platz eingespart werden.

An dieser Stelle soll außerdem ein weiterer Vorteil genannt werden, der durch die Erweiterung des Schemas entsteht. Wie sich später noch herausstellen wird, ist die Abbildung von CIDOC CRM auf RDF im Wesentlichen problemlos möglich, in einem Aspekt jedoch nicht: Eigenschaften von Eigenschaften – wie etwa `P3.1 has type` in Abbildung 5.2 zu sehen – sind nicht vorgesehen. Die Methode der Reifikation kann sich als unzureichend herausstellen, da diese Eigenschaften von Tripeln beschreiben. Eine saubere Definition des Schemas ist daher nicht möglich. Stattdessen kann eine Unterklasse der Eigenschaft `P3 has note` erstellt werden, welche dem jeweils gewählten Typen entspricht. Somit wäre derselbe Sachverhalt in einer anderen Variante des Models beschrieben.

Die Kehrseite des Ansatzes ist, dass der wohldefinierte und allgemein bekannte Teil der Ontologie verlassen wird. Alle neuen Klassen für Objekte und Eigenschaften müssen wohl dokumentiert werden und diese Beschreibung zusammen mit dem Modell gespeichert werden. Ein Verlust dieser Erweiterung ist immer ein potentieller Informationsverlust, weil dargestellte Sachverhalte möglicherweise nicht mehr eindeutig interpretiert werden können.

Person Für Personen existiert der bereits angesprochene Objekttyp `E21 Person`, welcher daher Verwendung finden soll. Im Hypergraphmodell sind für Knoten des Typs *Person* die folgenden Attribute gespeichert:

Attribut	Beschreibung
Version	Versions-ID (zur Vermeidung von lost updates, etc).
Anrede	Herr oder Frau (optional).
Titel	Titel (optional).
Beiträger	Ob diese Person ein Beiträger ist.
Gelehrter	Ob diese Person ein Gelehrter ist.
Erzähler	Ob diese Person ein Erzähler ist.
Info	Zusatzinformationen und Anmerkungen (optional).
ID	ID von Person (null ... noch nicht in der DB).
Name	Nachname (optional).
Vorname	Vorname (optional).
BKW-Signatur	Nummer der Person, Teil z.B. der Signatur der Beitraeger.

Tabelle 5.2: Attribute des Knotentyps *Person*

Vor- und Nachname können und sollten als `E82 Actor Appellation` festgehalten werden. Person und Bezeichnung werden mithilfe der Eigenschaft `P131 is identified by` verknüpft. Weiterhin können die Namen durch einen `E55 Type` spezifiziert werden, damit Vor- und Nachname ersichtlich sind. Zur Speicherung eines Titels existiert im Referenzmodell keine konkrete Klasse. Daher muss nach einer Alternative gesucht werden. Die Klasse `E35 Title` ist als Bezeichnung eines Werkes vorgesehen und daher nicht anwendbar, es muss also wieder auf einen Typen ausgewichen werden. Die Anrede einer Person wird intern als Geschlecht gespeichert. Dementsprechend soll hier eine Zuweisung von *männlich* oder *weiblich* geschehen. Eine dafür zuständige CIDOC CRM-Klasse `E76 Gender` wurde mit Version 3.3.1 entfernt. Daher wird auf `E55 Type` ausgewichen. Einigen Personen wird eine Signatur im BKW zugewiesen. Auch hierfür kann `E82 Actor Appellation` verwendet werden, das entsprechende Objekt wird mit einem `E55 Type` als *BKW-Signatur* ausgezeichnet. Bei einigen Figuren enthält das Feld *Info* noch Zusatzinformationen. Diese enthalten eine Liste von Zusatzinformationen in XML. Dabei sind Listenelemente mit einem Typ versehen, wie etwa *Quelle* oder *Anmerkung*. Hier eröffnen sich zwei Möglichkeiten: Entweder wird der XML Klartext übernommen, in einem `E62 String` Objekt gespeichert und via `P3 has note` mit der Person verknüpft, wobei diese Property außerdem durch `P3.1 has type` als *Personeninformation* ausgewiesen wird. Oder die einzelnen Listenelemente werden extrahiert und einzeln aufgeführt. Die letztere Variante ist in Abbildung 5.3 dargestellt. Abschließend wird in den Personenattributen festgehalten, ob und wie diese Person zu Wossidlos Arbeit beigetragen hat. Dazu wird ihm die entsprechende Rolle – also *Gelehrter*, *Beiträger* oder *Erzähler* – bei Vorhandensein als `E55 Type` zugewiesen. Diese können jeweils wieder durch einen Typ ausgezeichnet werden, um zu kennzeichnen dass sich diese Rolle von den nachfolgenden Personenrollen unterscheidet. Eine Anmerkung in Form eines `P3 has note` zur Beschreibung dieser Rollentypen ist angemessen, in der Abbildung wird allerdings aus Platzgründen darauf verzichtet. Alternativ ist es – wie im letzten Abschnitt bereits erläutert – auch möglich, eigene Klassen zu definieren. Dazu wurden im Beispiel Vor- und Nachname und die BKW-Signatur als Unterklassen von `E82 Actor Appellation`, sowie Personentitel, Kooperationstyp und Geschlecht als Abkömmlinge von `E55 Type` gewählt. Dieser Ansatz wird in Abbildung 5.4 dargestellt, aus Gründen der Übersicht wurden die Objekte zur Beschreibung des Datenbankfelds `Info`[3] ausgelassen.

Es sollen nun noch einige Anmerkungen zu den Knoten vom Typ *Person* folgen. Nach Ansicht des Autors dieser Arbeit ist es anzuraten, eine Bereinigung der Daten durchzuführen und eventuell eine geringfügige Neukonzeption vorzunehmen, bevor eine Langzeitarchivierung in Angriff genommen werden kann. Dafür sprechen unter anderem folgende Gründe: Die Tabelle enthält Einträge, die keine Personen darstellen. Hier ist unter anderem die Zeile mit dem Namen *Deutsche Akademie der Wissenschaften zu Berlin* zu nennen. Die Spalte

[3]Die hierbei verwendete Eigenschaft `3.1 has type` ließe sich durch Definition von Unterklassen von P3 has note obsolet machen. Dieser Gedankengang würde allerdings eine dynamische Generierung von Klassen aus der Datenbank bedeuten.

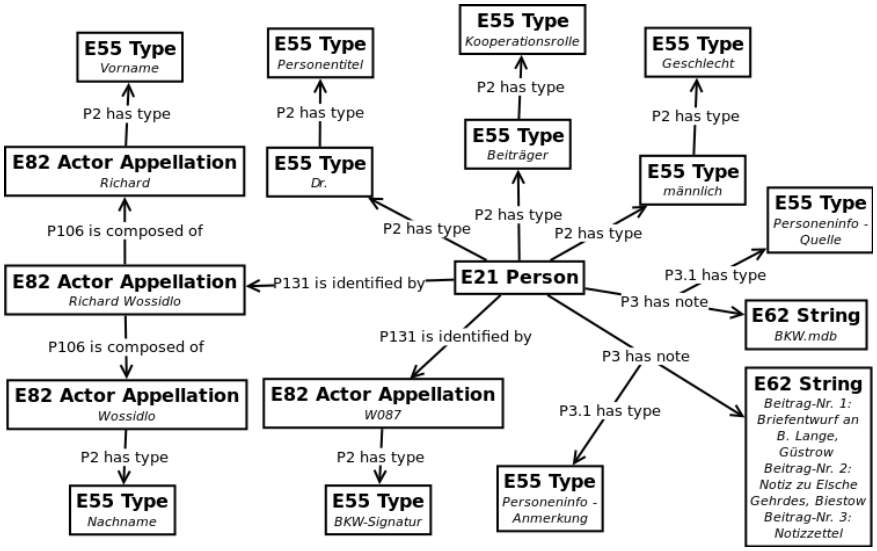


Abbildung 5.3: Abbildung eines Knotens vom Typ *Person* in CIDOC CRM

`title` enthält diverse Personentitel. Dies können sowohl akademische, wie auch Adelstitel, aber auch Berufsbezeichnungen sein. Es ist auffällig, dass einige Einträge mit derselben Bedeutung in unterschiedlicher Schreibweise auftreten. Dazu zählen etwa *Freiherr*, *Freiherr von*, sowie die Abkürzung *Frh.*, *Dr. phil* und *Dr. Phil*, sowie *senior* und *sen.* Leere Felder wurden unterschiedlich bezeichnet, etwa mit *nicht spezifiziert* und *(nicht spezifiziert)*, sowie vermutlich auch *x*. In einigen Fällen wurden Daten in das falsche Feld eingetragen, etwa ist in dieser Spalte der Vorname *Arnold* und die Anrede *Herr* zu finden. Besonders heimtückisch sind Werte, die sich nur durch nicht-druckbare Zeichen unterscheiden. So existieren vier unterschiedliche Schreibweisen für *Prof. Dr.*, welche sich nur schwer unterscheiden lassen. Eine Möglichkeit zur Vermeidung dieser Problematik wäre die Einführung eines neuen Knotentyps *Titel*, welcher sich über einen neuen Hyperkantentyp mit einer Person verknüpfen lässt. So können einer Person auch problemlos mehrere Titel zugewiesen werden, sofern es nötig ist. Es existieren 179 isolierte Personenknote, die in keiner Hyperkante vorkommen. Dies sollte ebenso bereinigt werden.

Orte Auch für Orte stellt das Referenzmodell eine eigene Objektklasse zur Verfügung, nämlich `E53 Place`. Darunter fallen alle Arten von Orten, weshalb es hier verwendet wird. Die Benennung erfolgt über die Eigenschaft `P87 is identified by` und `E44 Place Name`. Letztere hat außerdem auch die Eigenschaft `P139 has alternate form` zur Darstellung der Alternativbenennungen. Die Ortskategorie kann als E55 Type modelliert werden, diese wird wiederum durch einen weiteren Typ ausgezeichnet, der verdeutlicht, dass es sich um eine Ortskategorie handelt. Region und Überregion werden ebenfalls als `E53 Place` dargestellt, die entsprechenden Beziehungen mithilfe von `P89 falls within` erschaffen. Beide Objekte erhalten den Typ *Region*. Der Infotext kann wie bei Personen als `E62 String` dargestellt werden.

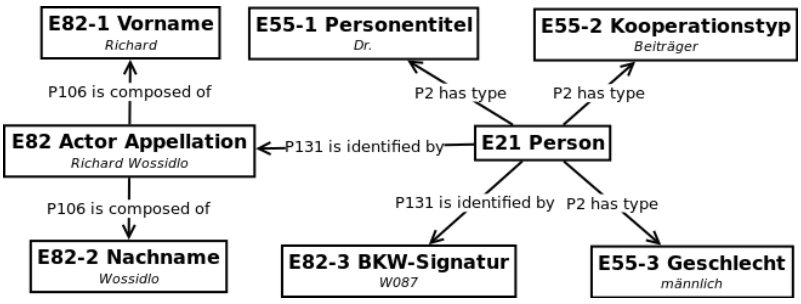


Abbildung 5.4: Abbildung eines Knotens vom Typ *Person* in CIDOC CRM unter Verwendung selbst-definierter Klassen

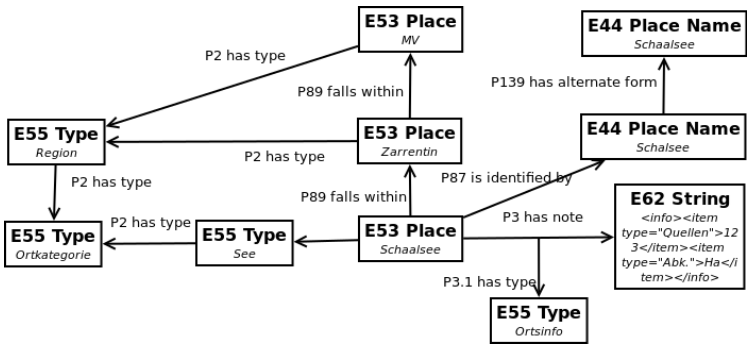


Abbildung 5.5: Abbildung eines Knotens vom Typ *Ort* in CIDOC CRM. Im Fokus steht der Ort *Schaalsee*

Auch hier sollten einige Aufräumarbeiten durchgeführt werden, bevor an eine Langzeitarchivierung gedacht wird. Es treten wieder verschiedene Schreibweisen für dieselben Orte auf. Zum Teil enthalten Ortsnamen Sonderzeichen, in einigen Fällen lässt sich – aufgrund der Nutzung des Fragezeichens – Unsicherheit der Bezeichung vermuten. Da es sich bei Regionen wiederum um Orte handelt, bietet es sich an, diese Beziehung mit einem anderen Ort-Knoten über eine Hyperkante darzustellen. Dabei würde außerdem die Überregion entfallen würde, beziehungsweise implizit modelliert werden. Dies würde unter anderem folgende Probleme lösen: Häufig treten unterschiedliche Schreibweisen auf, etwa *MV* und *Mecklenburg-Vorpommern*. *Dorf Mecklenburg* und *Mecklenburg B. Wismar* sind derselbe Ort, jedoch existieren unterschiedliche Einträge. In einigen Fällen wurde statt einer Region im herkömmlichen Sinne das *Ortsverzeichnis Sagen* angegeben. Da die Überregion immer von der Region abhängig ist, sollten diese Paare immer gleich sein, dies ist aber nicht immer der Fall. So existieren Einträge, welche die Region *Ganzlin* enthalten, darunter einige, für welche die Überregion *MV* angegeben wurde. Bei anderen Einträgen fehlt diese Angabe. Als weiterer Vorteil würde der Typ des übergeordenten Ortes genauer bezeichnet werden können, etwa *Bundesland*, *Land* etc. Die Ortstypen benötigen eine Sichtung. So sind hier konkrete Orte angegeben, etwa *Gemeinde Daberkow, Wietzow, Hedwigshof*. Weiterhin existieren Tippfehler, etwa *Halbinsel* und *Halbsinsel*. Auch hier empfielt sich das Auslagern der Typen in eigene Knoten. 196 Knoten diesen Typs sind im Hypergraphen isoliert.

Personenrollen Personen können verschiedene Rollen einnehmen. Diese ermöglichen eine Einordnung etwa auf sozialer,familiärer oder beruflicher Ebene. So kann einerseits eine konkrete Person genauer beschrieben werden, indem ihr – etwa mithilfe einer Hyperkante – eine Rolle zugewiesen wird. Andererseits ist es so möglich, unbestimmte Personen zu referenzieren, etwa wenn *ein Bauer* in einer Erzählung genannt wird. Als Objekttyp eignet sich die Klasse `E82 Actor Appellation`. Diese kann und sollte durch einen Typ *Personenrolle* spezifiziert werden. Neben der Bezeichnung ist eine Beschreibung, sowie ein XML-Infofeld zu finden. Beide Attribute werden nur spärlich verwendet, letzteres in nur einem Fall. Da es sich dabei offensichtlich nur um eine interne Notiz handelt kann das Feld nach Ansicht des Autors in der CIDOC CRM-Modellierung ausgelassen werden. Eine Beschreibung der Rollen hingegen wird als wichtig angesehen und kann als `E62 String` dargestellt werden. Die vorgeschlagenen Änderungen für diesen Knotentypen halten sich in Grenzen. Bei den Rollenbezeichnungen tauchen wieder einige ähnliche Schreibweisen auf, etwa *Hofphotograph* und *Hoffotograf* oder *Musikant* und *Musiker*. Hier böte es sich an, sich für eine Schreibweise zu entscheiden und Alternativen über eine *Synonym*-Hyperkante einzupflegen. In diesem Zug können auch in Einträgen, wie *Füerböter [Heizer]* und *Kiepenmaker [Kiepenmacher]* Bezeichnung und alternative Schreibweise entkoppelt werden. Wieder enthalten einige Bezeichnungen ein Fragezeichen, welches wohl Unsicherheit spezifizieren soll. Dieses Konzept sollte anders modelliert werden. Außerdem treten Bezeichnungen auf, die keine Personenrolle beschreiben, so etwa *Kriegskasse*, *[693] ?*, *Liberaler Ortsverein* oder *Lübtheen*. In einem Fall (*Großvater*) enthält der Beschreibungstext Informationen, welche sich nicht auf die Personenrolle selbst, sondern auf einen Bogentyp bezieht. Es existieren 30 Knoten diesen Typs, die in keiner Hyperkante enthalten sind.

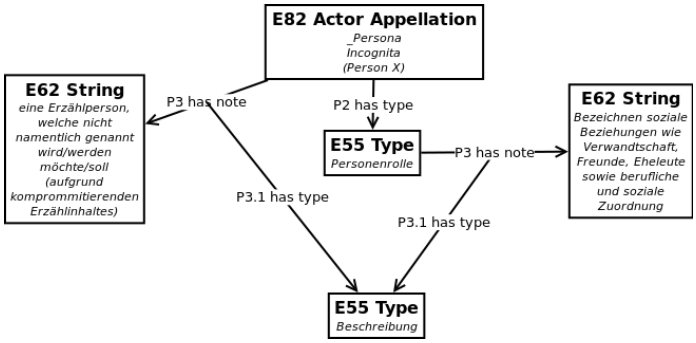


Abbildung 5.6: Abbildung eines Knotens vom Typ *Personenrolle* in CIDOC CRM

Erzähltypen und Erzählmotive Erzähltypen und -motive beschreiben wiederkehrende Inhalte der Erzählungen, etwa verschiedene Varianten des Märchens *Dornröschen* oder Überlieferungen, in denen eine *böse Stiefmutter* eine Rolle spielt. Diese Knotentypen ähneln sich stark, weshalb sie hier zusammen beschrieben werden sollen. Tatsächlich verwenden beide Knotentypen dasselbe Schema. Jeder Knoten hat eine Beschreibung des Motivs, etwa *Dornröschen*. Diese können als `E90 Symbolic Object` modelliert werden. Weiterhin existieren systematische Bezeichnungen, etwa der *Aarne-Thompson-Uther-Index* oder die *Stith-Thompson-Klassifikation*. Diese setzen sich aus einem Typen und einer Nummer zusammen. Intuitiv wird die Nummer als `E41 Appellation` dargestellt, der Typ wird als `E55 Type` beschrieben. Es ist anzunehmen, dass Typ und Nummer aufgrund ihrer Bedeutung

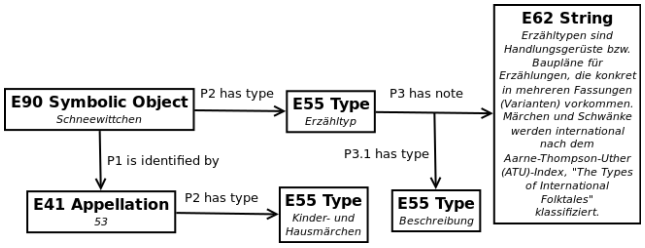


Abbildung 5.7: Abbildung eines Knotens vom Typ *Erzähltyp* in CIDOC CRM

Kandidaten für zusammengesetzte Schlüssel sind. Diese Eigenschaft wird jedoch verletzt, so treten gleiche Kombinationen dieser Attribute mitunter mehrfach auf. Häufig ist dies der Fall, wenn die deutsche und die englische Bezeichnung verwendet werden. In einigen Fällen werden die Felder vertauscht oder nur ein einziges verwendet. Sehr häufig taucht die Systematik zusätzlich in der Bezeichnung auf, dies ist nicht notwendig. Vereinzelt sind Knoten isoliert. Eine neue Art der Problematik eröffnet sich, wenn man diese Knoten inhaltlich betrachtet. So stellen in einigen Fällen mehrere Knoten dasselbe Erzählmotiv dar, jedoch in einer anderen Systematik und/oder einer Übersetzung. Im Falle von *Schneewittchen* mit der Kinder- und Hausmärchen (KHM)-Nummer 15 existiert außerdem ein Knoten *Snow White* mit dem Aarne-Thompson-Uther (ATU)-Index 709. Einzeln betrachtet ergäben sich zwei Knoten, wie sie in Abbildung 5.7 dargestellt sind. Erstrebenswert wäre es jedoch, diese beiden zusammenzuführen, ähnlich Abbildung 5.8. Tatsächlich sind beide Knoten über eine Hyperkante der Art *Übersetzung* miteinander verbunden, durch eine weitere Kante *Sprachspezifisch* lässt sich außerdem die jeweils konkrete Sprache feststellen. Daher muss für jeden Knoten und jeden Hyperkantentypen, der diesen Knotentyp enthält erwogen werden, ob eine Zusammenführung von Knoten möglich und sinnvoll ist. Verstärkt wird dieses Problem dadurch, dass die Daten nicht zwangsläufig konsistent sind: Eine *Übersetzung* von *Hänsel und Gretel (KHM 15)* zu *Hansel and Gretel (ATU 327A)* existiert nicht. Allerdings sind beide Knoten über eine Kante *Gutartige Referenz*[4] miteinander verbunden. Es muss also auch eine Zusammenführung mit Knoten anderen Typs in Erwägung gezogen werden, was den manuellen Arbeitsaufwand weiter steigen lässt.

[4]Dieser Hyperkantentyp verbindet laut internem Model ein Wort mit mehreren Typen oder mehreren Motiven. Im gegebenen Fall existiert interessanterweise kein Wort, weshalb diese Bedingung verletzt ist.

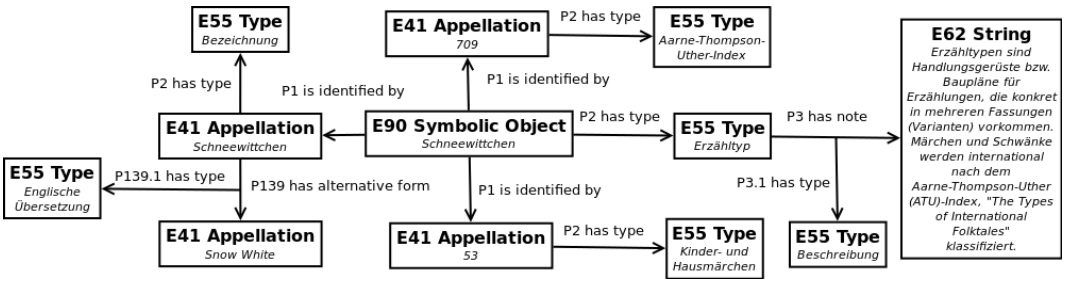


Abbildung 5.8: Abbildung eines Knotens vom Typ *Erzählmotiv* in CIDOC CRM

Überführung der Hyperkanten in CIDOC CRM

Im Wesentlichen unterscheidet sich die Überführung von Hyperkanten in CIDOC CRM nur geringfügig von der Vorgehensweise bei Knoten. Zuerst ist eine Klasse zu finden, die dem gewählten Kantentypen am ehesten entspricht. So überführt Kiesendahl den Hyperkantentyp Inhalt beispielsweise in `E7 Activity` mit dem `E55 Type` Inhalt. Wie es auch bei den Knoten der Fall war, können hier wieder eigene Unterklassen erzeugt werden, solange diese ausreichend dokumentiert werden. Das ist insbesondere auch für Properties der Fall. Mit dem die Kante repräsentierenden Objekt muss nun eine Menge von Knotenrepräsentationen verbunden werden. Diese unterscheiden sich in ihrem Typ und in dem Typ des Bogens der Hyperkante, in dem sie auftreten. Entsprechend dieser Gegebenheiten ist die Modellierung individuell durchzuführen, gegebenenfalls lassen sich einige Fälle zusammenfassen. Beim Modelliervorgang ist auf Konsistenz zu achten, das beinhaltet neben dem erlaubten Definitions- und Wertebreich von Properties auch die Einhaltung der vorgegebenen Kardinalitäten.

Sonderfall Topologie

Wie bereits an anderer Stelle erläutert, wird die Topologie des Archivs aktuell nicht mithilfe der Hypergraphen beschrieben. Stattdessen sind die Knoten durch Fremdschlüsselbeziehungen miteinander verknüpft. Selbstverständlich ist es nicht hinnehmbar, diese Information nicht zu sichern, nur weil sie nicht dem Standardkonzept entsprechen. Daher muss ein alternativer Weg gefunden werden, dies zu erreichen.
Tatsächlich fällt die Lösung grundsätzlich einfach aus, bedarf allerdings etwas Aufwand, da ein wenig *von Hand* modelliert werden muss. Weiterhin müssen die gespeicherten und in die RDF-Modelle überführten Fremdschlüssel in korrekte Referenzen auf Objekte der nächsthöheren Stufe umgeformt werden. Außerdem sollten Signaturen, welche nicht direkt in der Datenbank hinterlegt, sich aber mit den verfügbaren Informationen generieren lassen, nachgepflegt werden. Die Topologie selbst ist durch eine Baumstruktur beschrieben. Dabei ist zu jedem Knoten der Elternknoten bekannt. Dieser kann mit den mitteln von CIDOC CRM direkt referenziert werden. Anhand es ZAW soll dieser Vorgang nun exemplarisch beschrieben werden.
Das Wurzelelement der Topologie ist das Wossidlo-Archiv selbst. Man kann es als eine Komposition der Korpusse betrachten, ohne diese existiert es nicht. Eine passende Klasse des

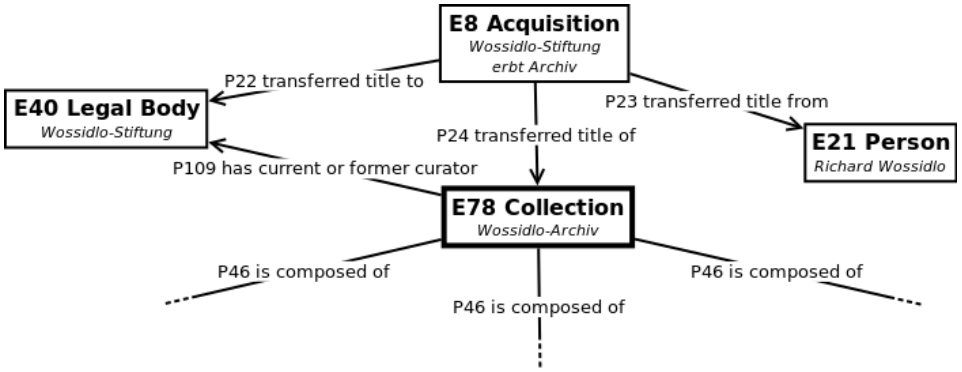


Abbildung 5.9: Modellierung des Archivs

Ebene	Inhalte	Tabelle
0	Kästen	`w.at_zaw0`
1	Konvolute	`w.at_zaw1`
2	Belege	`w.at_zaw2`
3	Seiten	`w.at_zaw3`

Abbildung 5.10: Ebenen des ZAW

Referenzmodells stellt `E78 Collection` dar. Sie kann durch einen Typen *Archiv* weiter spezifiziert werden, die Einführung ein neuen Datentyps ist aufgrund der Einmaligkeit nicht notwendig. Das Archiv selbst kann ebenfalls mithilfe des CIDOC CRM beschrieben werden, von Interesse wäre etwa eine Übersicht der bisherigen Besitzer.
Auf der nächsten Ebene folgen die verschiedenen Korpusse. Hierbei ist die Einführung einer eigenen Klasse durchaus denkbar, wobei diese ebenso zu `E78 Collection` gehören sollte. Auch auf dieser Ebene bietet es sich wieder an, weitere Hintergründe im Referenzmodell zu beschreiben. Dazu zählen etwa die Umzüge der einzelnen Korpusse, oder der Digitalisierungsvorgang. Sowohl das Archiv selbst, wie auch die Korpusse müssen von Hand modelliert werden. Die folgenden Ebenen werden aus der Datenbank generiert.
Tabelle 5.10 gibt einen Übersicht über die Ebenen des ZAW Innerhalb des ZAW sind Zettelkästen die oberste Einheit. Informationen finden sich in Datenbank-Tabelle `w.cat_zaw0`. Kiesendahl modelliert einen Kasten als Informationsträger (`E84 Information Carrier`). Dieser Ansatz kann diskutiert werden, da der Kasten an sich keine Information enthält, sondern lediglich ein Aufbewahrungs Ort für Informationsträger ist. Die Definition als Ort wäre insofern von Vorteil, dass sich das darüberliegende ZAW als aus verschiedenen Sektionen bestehendes Objekt beschreiben ließe. Dafür kann die Eigenschaft `P59 has section` verwendet werden. Es sei betont, dass neue Klassen durchaus mehrere Oberklassen haben können. Zettelkästen speichern eine Signatur und einen Namen. Weiterhin wird außerdem die alte, von Wossidlo festgelegte Signatur gespeichert. Diese kann von Nutzen sein, wenn ein Vergleich mit Abbildungen von Zettelkästen gezogen werden muss, etwa wie in Abbildung 4.2b. Die Anzahl der enthaltenen Zettel kann ebenfalls mit ins Modell übernommen werden, so kann eine eventuelle Unvollständigkeit erkannt werden.
Es folgen die Konvolute. Diese können grundsätzlich wie Zettelkästen modelliert werden. Dabei sollte allerdings wieder eine eigene Klasse definiert werden, welche sie als Konvolute auszeichnet und diese Einheit beschreibt. Einige Konvolute werden in Oberkonvolute zusammengefasst. Diese werden – aus datenbanktechnischer Perspektive ungünstig – lediglich im XML-Info-Feld namentlich referenziert. Oberkonvolute können analog als weitere Schicht eingefügt werden.
Auf der untersten Ebene befinden sich die Seiten. Die in `w.cat_zaw3` hinterlegten Informationen umfassen die Signatur, das beinhaltende Konvolut als Fremdschlüssel, sowie eine Referenz auf die Bilddatei in der Datenbank. In Kiesendahls Arbeit werden die Zettel in zwei Objekten festgehalten: Zum einen wird der physische Gegenstand modelliert, dazu wird ein `E84 Information Carrier` verwendet. Weiterhin wird die Information, welche auf dem Zettel enthalten ist, durch ein `E31 Document`-Objekt repräsentiert. Diese wird mit der Signatur (`E75 Conceptual Object Appellation`) bezeichnet. Die Referenz auf die Bilddatei ist aufgrund der Angabe der Signatur obsolet. Es sei jedoch an dieser Stelle nochmal erwähnt, dass selbst das Digitalisat des Zettels mithilfe eines `E62 String` im Referenzmodell festgehalten werden könnte. Dies ist jedoch durch die begrenzte Kapazität der später verwendeten QR-Codes nicht möglich. In anderen Tabellen der Datenbank, die dem Autor der Arbeit jedoch aufgrund von Zugriffsrechten nicht zugänglich sind, sind außerdem auch Maße der Zettel gespeichert. Diese können ebenfalls modelliert werden, die dazu geeignete Klasse ist `E62 Measurement`.

5.3 Beschreibung des CIDOC CRM in RDF

Die Darstellung von CIDOC CRM-Modellen in RDF wird durch die CIDOC ausdrücklich angeraten, so wird etwa eine vollständige Schemadefinition (RDFS) zur Verfügung gestellt[5]. Es enthält fast sämtliche Klassen und Eigenschaften, die Verebungsbeziehungen und bis auf die Kardinalitäten auch die Konsistenzbedingungen des Modells. Zum besseren Verständnis sind die Bezeichnungen aller Klassen in verschiedenen Sprachen, unter anderem Englisch, Deutsch, Russisch, Französisch und Spanisch, angegeben, sowie eine Erläuterung des jeweiligen Elements in englischer Sprache. Listing 5.1 zeigt verkürzt einen exemplarischen Eintrag:

```
<rdfs:Class rdf:about="E30_Right">
  <rdfs:label xml:lang="fr">Droit</rdfs:label>
  <rdfs:label xml:lang="en">Right</rdfs:label>
  <rdfs:label xml:lang="de">Recht</rdfs:label>
  <rdfs:label xml:lang="pt">Direitos</rdfs:label>
  <rdfs:comment>This class comprises legal privileges concerning
    ↪ material and immaterial things or their derivatives.
  These include reproduction and property rights</rdfs:comment>
  <rdfs:subClassOf rdf:resource="E89_Propositional_Object"/>
</rdfs:Class>
```

Listing 5.1: Auszug aus dem CIDOC-RDF-Schema

Mithilfe der auf diese Weise zur Verfügung gestellten Klassen und Prädikate ist kann das Model auf fast triviale Weise überführt werden. Aus jedem Objekt entsteht ein stellvertretender RDF-Knoten. Dieser erhält einen passenden URI, welcher in Abschnitt 5.3.2 erläutert werden soll; einige Ressourcen hingegen bleiben anonym, da auf sie *von außen* nicht mehr zugegriffen werden muss. Mithilfe der Prädikate `rdf:type` und `rdf:value` werden der Ressource Typ und Inhalt zugewiesen. Abbildung 5.11 veranschaulicht dies bildlich.

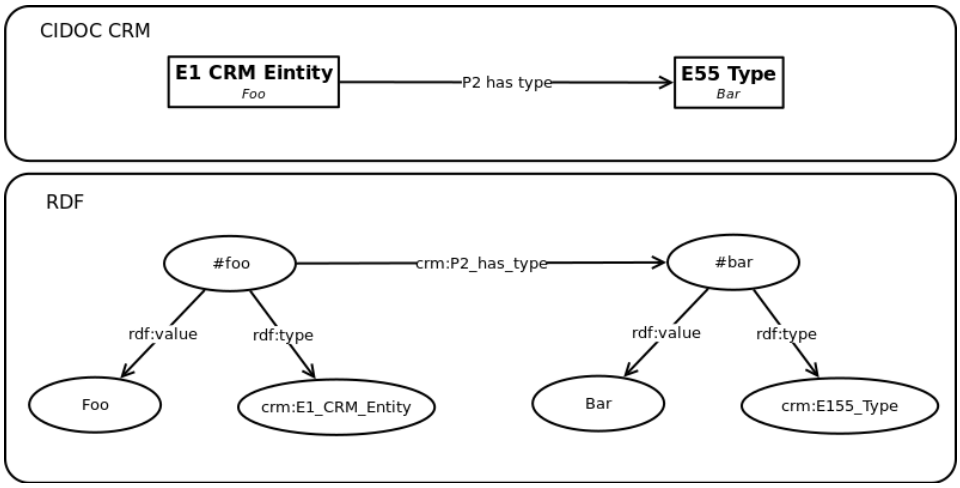


Abbildung 5.11: Allgemeine Überführung von CIDOC CRM in RDF

5.3.1 Primitive Datentypen und Eigenschaften von Eigenschaften

Zu den nicht aufgeführten Bestandteilen des Schemas gehören die Klasse `E59 Primitive Values` mitsamt ihren Unterklassen, sowie Eigenschaften von Eigenschaften, etwa `P3.1 has type`. Primitive Datentypen werden in RDF als Literal dargestellt, daher entfällt die Notwendigkeit einer CRM-Klasse. Das Konzept, Eigenschaften von Eigenschaften zu beschreiben,

[5]Siehe `http://www.cidoc-crm.org/sites/default/files/cidoc_crm_v5.0.4_official_release.rdfs.xml`

ist in RDF nicht vorgesehen. Zwar könnte man meinen, das Problem mittels Reifikation lösen zu können, hier steckt der Teufel jedoch im Detail: Die genannte Eigenschaft `P3.1 has type` etwa hat den Wertebereich `P3 has note`. Möchte man diese in RDFS definieren, kann man allerhöchstens ein Statement, also ein Tripel als Wertebereich angeben. Da man hierbei keine Anforderungen an das dabei verwendete Prädikat stellen kann, ist eine saubere Schemadefinition nicht möglich. Als Ausweg aus dieser Situation wurde bereis in Abschnitt 5.2.2 angesprochen. So werden, statt `P3 has note`-Prädikate mit einem Typen auszuzeichen, Unterklassen dieser Eigenschaft gebildet, die der Bedeutung des Typs entsprechen, etwa `P3.1 describes`.

5.3.2 Adressierung der Ressourcen

RDF schreibt vor, dass Ressourcen mit einem URI ausgezeichnet werden, um sie zu identifizieren und referenzieren. Eine Ausnahme bilden hier anonyme Knoten (*blank nodes*). Im Rahmen des Semantischen Webs wird diese Anforderung sogar soweit konkretisiert, dass URLs verwendet werden sollen, die das Hypertext Transfer Protocol (HTTP) nutzen. Um die im Rahmen dieser Arbeit generierten RDF-Ressourcen auch in einem *Online*-System zur Verfügung zu stellen, lohnt es sich daher, diesen Ansatz zu verfolgen. Es müssen später lediglich Hilfestellungen zur *Auflösung* der URLs der auf Mikrofilm gespeicherten Ressourcen zur Verfügung gestellt werden.
Im Laufe der Arbeit offenbarten sich zwei Ansätze für diese Aufgabe. Beim ersten Vorschlag handelt es sich um einen für den Autor der Arbeit intuitiven Lösungsweg, der sich ergab, weil benötigtes Hintergrundwissen für die Alternative noch fehlte. Hierbei setzt sich die URL einer Ressource aus den folgenden Bestandteilen zusammen:

1. Eine Basis-URL, wie etwa `http://www.wossidia.de/wossidia/`. Diese wurde vom Autor willkürlich bestimmt und ist austauschbar.
2. Eine Unterscheidung zwischen Knoten und Hyperkanten, was durch eine Fortsetzung der Adresse mit `nodes/` oder `edges/` gekennzeichnet wird.
3. Ein Label für den verwendeten Typen, etwa `person`, für Knoten aus der Tabelle `w.am_person/` oder `inhalt/` für Hyperkanten mit dem Typ der ID 1019. Diese Labels lassen sich später in der Konfiguration einstellen.
4. Abschließend folgt die `id` des jeweiligen Knotens, beziehungsweise der Hyperkante. Weil diese in HyDRA als Primärschlüssel dient, ist sie eindeutig. Wiederverwendbare Typendefinitionen erhalten hier die Zeichenkette `types`.

Es folgen nun einige Beispiele für Ressourcen-URLs:

- `http://www.wossidia.de/wossidia/nodes/person/types`
- `http://www.wossidia.de/wossidia/nodes/person/42`
- `http://www.wossidia.de/wossidia/edges/inhalt/230152`

Auf die alternative Variante haben die Mitarbeiter der Forschungsgruppe hingewiesen. Demnach können Knoten mithilfe einer Metadaten-Signatur identifiziert werden. Diese böte sich zur Verwendung in einer URL an. Metadaten-Signaturen passen sich nahtlos in die in Abbildung 4.4 dargestellten Signaturen des Digitalisierungsprozesses ein. Die Korpus-Bezeichnung für Metadaten lautet `XMD`. Auf Ebene Null erscheint eine Kennung der Tabelle, in welcher der Knoten gespeichert wird. Eine entsprechende Abbildung ist der Tabelle `w.cat_medatadata` zu entnehmen. So werden Knoten, die Personen beschreiben – also in der Datenbanktabelle `w.am_person` hinterlegt sind – hier durch die Zeichenkette `M010` ausgezeichnet. Es folgt die `id` der Instanz, welche auf den Ebenen eins bis drei dargestellt werden. Dabei werden führende Nullen aufgefüllt, sodass alle neun Ziffern belegt sind. Die Person *Richard Wossidlo*, welche in der Datenbank unter dem Schlüssel 7728 abgelegt ist, hätte demnach die folgende Signatur: `XMD-M010-000-007-728`.
An der Universität Rostock wird ein Dienst für persistente URLs bereitgestellt. Dieser ist unter `http://purl.uni-rostock.de` erreichbar. Hieran wird ein Dokumenten-Identifikator (record-identifier) angehängt, welcher die Ressource eindeutig identifiziert. Alle dem Wossidia-Projekt zugehörigen Ressourcen beginnen mit `wossidia/`. Anschließend folgt eine sich aus der Signatur ableitbare Zeichenkette. Diese wird wie folgt gebildet:

1. Die Bindestriche werden entfernt.

2. Das Korpus-Kürzel (`XMD`) wird in Kleinbuchstabe (`xmd`) überführt.

3. Der Buchstabe in der Ebene-Null-Angabe wird als Zahl kodiert. Dieser wird auf seine Position im Alphabet abgebildet, wobei `A` der Zahl `1` entspricht.

Auf diese Weise wird der Person Richard Wossidlo entsprechende Knoten auf den Dokumenten-Identifikator `wossidia/xmd13010000007728` abgebildet, die vollständige URL, wie sie auch im RDF verwendet werden kann, lautet dann `http://purl.uni-rostock.de/wossidia/xmd13010000007728`.
Knoten, welche die Topologie wiederspiegeln, verwenden ihre eigene Signatur. Diese ist jeweils vom Verwendeten Korpus abhängig. Eine Entsprechung für Hyperkanten und wiederverwendbare Ressourcen steht noch aus.

5.4 Aufteilung des Modells zur Speicherung auf Mikrofilm

Mit dem jetzigen Stand liegt ein RDF-Modell vor, welches entweder den Hypergraphen direkt oder formuliert als CIDOC CRM darstellt. Dieses soll nun auf QR-Codes gespeichert und auf Mikrofilm ausbeleuchtet werden. Dabei können verschiedene, möglicherweise sich widersprechende Anforderungen gelten:

Kompaktheit Die Verteilung der Daten soll so vorgenommen werden, dass insgesamt so wenig Filme wie möglich verwendet werden. Zusätzlich kann auch gefordert sein, dass auf dem *letzten* Film so wenige Frames wie möglich in Anspruch genommen werden, da deren Ausbeleuchtung zusätzliche Kosten verursacht.

Redundanz QR-Codes verwenden bereits einen Fehlerkorrekturmechanismus, welcher die Rekonstruktion der Daten bei beschädigten Codes bis zu einem gewissen Grad erlauben. Werden einer oder mehrere Filme zum Teil oder vollständig zerstört oder gehen verloren, so ist es wünschenswert, dass deren Inhalte bei annehmbarem Mehraufwand wiederherstellbar sind.

Navigierbarkeit Unter Umständen wird eines Tages jemand auf die Idee kommen, den Datensatz nicht als Gesamtes in eine Datenverarbeitungsanlage zu überführen, sondern mit einem Lesegerät manuell Suchen im Datensatz anstellen. Das kann etwa der Fall sein, wenn sich ein Gerät zum automatischen Einlesen des Modells noch in der Entwicklung befindet oder gerade anderweitig nicht verfügbar ist und der Datensatz in der Überbrückungszeit untersucht werden soll. Dies ist nicht möglich, wenn die potentiell hunderttausenden Codes keiner Ordnung unterliegen, die diesen Vorgang unterstützen.

Weiterhin soll gelten, dass – bis auf wenige Ausnahmen, etwa bei gesondert ausgezeichneten Paritäten – jeder einzelne QR-Code in sich eine korrekte RDF-Repräsentation darstellt. Das heißt, dass das Lesen eines solchen Codes immer eine Menge intakter Tripel liefert. Dabei ist es irrelevant, ob diese dieselbe Ressource als Subjekt behandeln oder alle eine Ressource betreffenden Tripel vollständig enthalten sind. Ein Vorteil dieser Herangehensweise liegt auf der Hand: die Reihenfolge des Lesens der Codes – so man das gesamte Modell einlesen möchte – ist irrelevant und ein Code kann nicht durch den Wegfall eines anderen wertlos gemacht werden. Allerdings wird gleichzeitig eine künstliche Einschränkung eingeführt: Die maximale Größe eines einzelnen Tripels wird auf die Größe des speichernden Codes, abzüglich der Namensraum-Definition limitiert. Sonderregelungen, für den Fall dass dies nicht ausreicht, sollen evaluiert werden.
Für sämtliche der folgenden Betrachtungen wird angenommen, dass der Hypergraph direkt in RDF vorliegt. Dies hat den Grund, dass diese Variante es erlaubt, annähernd den gesamten Umfang der Daten zu konvertieren und zu betrachten, wodurch eine realistischere Betrachtung möglich ist. Für die CIDOC CRM-Ausführung wäre es nötig, sämtliche Knoten- und Kantentypen zu modellieren und zu überführen, was im Rahmen dieser Arbeit nicht realistisch ist.

5.4.1 Kompaktheit

Es soll versucht werden, den notwendigen Aufwand an Filmmaterial zur Speicherung weitgehend zu minimieren.
Die erste, fast schon triviale Maßnahme ist das Reduzieren der Länge selbst-definierter Schlüsselworte auf ein Minimum, etwa die Verwendung von `e`, statt `HyperEdge`. Derlei Schritte wurden in der Implementation des letzten Abschnitts vorgenommen, dabei konnten bis zu 16% Speicherplatz eingespart werden.
Weiterhin ist die Wahl der RDF-Serialisierung ein entscheidender Faktor. Unter den verfügbaren Varianten hat sich Turtle als die kompakteste herausgestellt. Hierbei gibt es zwei Darstellungsweisen[6]: In der *Flat*-Variante wird jedes Tripel einzeln ausgegeben:

```
w:n510-554 ws:a510text "Wie heisst das Kind, das zuletzt ..." .
w:n510-554 ws:a510id "554" .
w:n510-554 ws:a510snum "wz1" .
w:n510-554 ws:a510vid "1" .
w:n510-554 rdf:type ws:n510 .
```

Pretty hingegen fasst mehrere Statements mit demselben Subjekt zusammen, es wird nur ein einziges mal ausgeschrieben, in nachfolgenden Zeilen signalisiert ein Einschub die Fortsetzung.

```
w:n510-554  a              ws:n510 .
            ws:a510text     "Wie heisst das Kind, das zuletzt ..." .
            ws:a510id       "554" .
            ws:a510snum     "wz1" .
            ws:a510vid      "1" .
```

Die *Pretty*-Variante hat sich als kompakter herausgestellt. Das gilt allerdings nur solange, wie keine oder nur wenige anonyme Knoten genutzt werden. In diesem Falle wird der genannte Vorteil durch eine Vielzahl von nicht-druckbaren Zeichen zunichte gemacht. Konkret ist nach Erfahrungen des Autors *Pretty* die zu bevorzugende Variante bei der direkten Überführung des Hypergraphen in RDF, bei CIDOC CRM *Flat* zu bevorzugen. Freilich sind auch Mischformen möglich, diese werden allerdings von Jena nicht angeboten, weiterhin erschwert dies die nachfolgenden Schritte.
Es sollte möglichst eine QR-Code Variante verwendet werden, die eine hohe Datendichte aufzeigt. Mithilfe des Verhältnisses der Kapazität zu den hierfür notwendigen Modulen lässt sich leicht zeigen, dass hohe QR-Versionen hierbei effizienter sind, als niedrige. Weiterhin wäre die Wahl einer niedirgen Fehlerkorrekturstufe möglich, aber aufgrund der Tatsache, dass das oberste Ziel der Unternehmung der Erhalt von Daten ist, wird eine möglichst hohe Stufe ausdrücklich angeraten.

Optimale Anordnung der Informationen in Codes

Es versteht sich von selbst, dass die im letzte Schritt entstandenen Teilmodelle nicht die vollständige Kapazität eines QR-Codes ausschöpfen. Würde man in jedem Code nur genau eines dieser Teilmodelle speichern, so würde ein Großteil der Kapazität ungenutzt bleiben. Aus diesem Grund ist es angeraten, in einem Code soviel Information, wie möglich unterzubringen. Definiert man Informationseinheiten, welche eine Größe haben und versucht diese, in so vielen Codes derselben Größe unterzubekommen, so haben wir eine Instanz der Optimierungsvariante des Behälterproblems (Bin Packing). Es ist bekannt, dass dieses NP-schwer ist, eine optimale Lösung in polynomieller Zeit ist also nach derzeitigem Stand nicht zu erwarten. Jedoch ist ebenso der **First-Fit-Decreasing**-Algorithmus als Approximation mit der Güte $\frac{3}{2}$ bekannt[46]. Dieser hat eine Laufzeit von $\mathcal{O}(n \cdot \log n)$.
Eine wichtige Rolle spielt die Definition der atomaren Informationseinheiten. Wird das Modell in der Variante *Pretty* formatiert, so ist immer eine Ressource mitsamt aller Statements, in denen diese das Subjekt ist, als nicht-teilbare Einheit zu betrachten. Bei Modellen, die eine Hyperkante symbolisieren, sollten außerdem die Rückwärts-Bögen hinzugenommen werden. Die *Flat*-Variante ist in diesem Fall flexibler, hier stellen Tripel die kleinsten Elemente dar.

[6] Die Darstellungsvarianten werden vom Jena Framework bereitgestellt. Es ist unabhängig von Jena grundsätzlich möglich, diese zu kombinieren.

```
Data : N ;                                                    // Elemente
Result : B ;                                          // Menge der Behälter
sort(N) ;                                        // Sortiere Element absteigend
foreach n ∈ N do
    if ∃b ∈ B : n.fitsIn(b) ;     // Existiert Behälter, der n aufnehmen kann?
     then
        b.add(n) ;                                  // Füge n dem Behälter hinzu
    end
    else
        b ← B.newBin();                        // Erzeuge einen neuen Behälter
        b.add(n) ;                            // Und füge n dem Behälter hinzu
    end
end
return B
```

Algorithmus 1 : First Fit Decreasing als Approximation des Behälter-Problems

Vermutlich wird dieser Ansatz der effizientere sein, jedoch ist er mit dem Ziel der *Navigierbarkeit* nicht vereinbar. Da Tripel verschiedenster Ressourcen über den gesamten Datensatz verteilt sind, ist selbst bei Anwendung eines geeigneten Indexes ein komfortables Finden von Daten per Hand nicht realistisch.

Kompression

Es soll noch kurz die Möglichkeit beleuchtet werden, die benötigten Kapazitäten mithilfe von Kompressionsverfahren weiter zu minimieren. Dabei versteht es sich von selbst, dass ausschließlich verlustfreie Techniken in Betracht gezogen werden können. Erfahrungsgemäß lassen sich Klartextdateien mit heute bekannten Verfahren gut auf einen Bruchteil ihrer ursprünglichen Größe verkleinern. Dies wird dann noch weiter verstärkt, wenn die Inhalte gewissen Regelmäßigkeiten unterliegen und häufige Redundanzen auftreten. Vor allem aber profitieren diese von einer großen Menge an Daten, die dann verhältnismäßig stark verkleinert werden.
Gebräuchliche Verfahren, wie etwa GNU Zip (GZip), fassen Daten in Blöcken zusammen und komprimieren diese dann. Es wird sich schnell zeigen, dass diese Blöcke bei weitem größer sind, als die Kapazität eines QR-Codes. Daher ist es nicht effizient, diese einzeln zu komprimieren. Stattdessen böte es sich an, große Teile des Modells oder gar das Gesamte zu verkleinern. Vermutlich übertreffen die Einsparungen an Kapazität die der vorhergehenden Methoden bei weitem. Teilt man die komprimierte Datei entsprechend der Kapazität von QR-Codes in kleinere Einheiten auf, so lassen sie sich ebenfalls mit durch diese Speichern. Jedoch bringt sich auch Nachteile mit sich: Die Datenkompression ist ein weiteres Verfahren, was dokumentiert werden muss. QR-Codes verlieren ihre Eigenschaft, als eigenständige Objekte. Sie müssen in fester Reihenfolge gelesen und interpretiert werden. Abhängig vom verwendeten Kompressionsverfahren bedeutet der Verlust eines Codes auch den Verlust von Daten, die in anderen Codes gespeichert werden.

Optimierung der QR-Codes

Insgesamt sind drei Optimierungen bei der Verwendung von QR-Codes denkbar: Im Modus *alpha-numerisch* kann ein QR-40-H insgesamt 3.057 Zeichen aufnehmen. Das ist ein Zuwachs von 45% gegenüber der im Byte-Modus maximal möglichen Anzahl von Zeichen, bei der Verwendung von UTF-8. Allerdings bietet dieser Modus nicht den notwendigen Zeichenumfang. Jedoch kann zwischen verschiedenen Modi gewechselt werden. Es sollte möglich sein, einen Großteil der zur Definition von nicht-Literalen notwendigen Zeichen in dieser speichersparenden Kodierung darzustellen und dann entsprechend in den Byte-Modus zu wechseln.
Weiterhin kennt QR den **Structured Append Modus**. Hierbei werden bis zu 16 Codes als eine Einheit definiert, wodurch die Behältergröße deutlich vergrößert wird. Hierdurch würde voraussichtlich weniger Speicher verloren gehen, da Dateneinheiten in einem verminderten Maße Code-Grenzen überschreiten dürfen und weniger Behälter zu befüllen wären, die über ungenutzten Raum verfügen.

Abschließend existiert die QR-Variante iQR. Dieser ermöglicht noch größere Matrizen (422 × 422 Module), die eine erhöhte Datendichte aufweisen. Allerdings ist diese nicht weit verbreitet. Zum Zeitpunk dieser Arbeit konnte kein Standard gefunden werden.

5.4.2 Navigierbarkeit

Es muss möglich sein, dass ein gewünschter Inhalt in so wenigen Schritten, wie möglich gefunden werden kann, wobei ein Schritt immer das Einlesen eines Codes darstellt. Der Nutzer muss weiterhin in der Lage sein, den RDF Graphen zu erkunden und in diesem zu navigieren. Weil außerdem das *Blättern* im Datensatz und insbesondere zwischen den Filmen viel Zeit in Anspruch nimmt, sollten zusammengehörende Inhalte auch immer nahe beieinander liegen.

Datenorgansiation mit Hashes

Unter den Verfahren für Indizes und Datenorgansiationen heben sich Hashfunktionen durch – im besten Fall – konstante Zugriffszeiten hervor. Ein identifizierendes Attribut stellt die Eingabe einer Hashfunktion dar, deren Ausgabe ist die Seite, oder in unserem Fall der Frame, welcher das gewünschte Objekt enthalten soll. Ist das gesuchte Objekt hier nicht zu finden, so existieren verschiedene Strategien der Überlaufbehandlung.
Im gegebenen Kontext handelt es sich beim identifizierenden Attribut um die URL der jeweiligen Ressource. Dies lässt sich auf Zeichenketten verallgemeinern. Der Wertebereich kann auf eine Anzahl von QR-Codes oder Frames festgelegt werden, wobei diese sich ohne weiteres auf mehreren Filmen befinden können. Die Anzahl der benötigten Frames, beziehungsweise QR-Codes ist – wie in den vorhergehenden Schritten ersichtlich – abschätzbar, weshalb eine passende Hashfunktion mit entsprechendem Wertebereich wählbar oder generierbar ist. Notfalls sind unter Umständen dynamische Hashfunktionen anwendbar, welche den Wertebereich nachträglich erhöhen können. Für die folgenden Betrachtungen soll eine Festlegung auf eine feste Anzahl von Frames getroffen werden.
Ein erprobtes Verfahren zum Berechnen von Hash-Werten ist der Rabin-Fingerabdruck[27]. Dieser wird, wie folgt berechnet: $h_1(c_1c_2 \dots c_n) = c_1 \cdot a^{n-1} + c_2 \cdot a^{n-2} + \dots^+ a_n \cdot a^0$. Das Verfahren wird standardmäßig zur Berechnung in der Methode `.hashCode()` der Java-Klasse `String` verwendet. Dabei wird $a = 31$ festgelegt. Allerdings entspricht der Wertebereich in diesem Falle dem des `int`-Datentyps, welcher auf $\{-2^{31}, \dots, 2^{31} - 1\}$[7] festgelegt ist. Damit ist der Wertebereich nicht optimal.
Abhilfe schaffen kann eine einfache Hashfunktion mit der *Divisions-Rest-Methode*: $h_2(k) = k \mod m$. Der Wertebereich dieser Funktion ist $\{0 \dots m-1\}$. Bei der Wahl einer Primzahl für m kommt diese Hashfunktion einer Gleichverteilung nahe. Es folgt also zum Abbilden der URLs aus Frames die Funktion: $H(url) = h_2(h_1(url))$.
Ein Frame ist immer dann als voll zu betrachten, wenn es nicht mehr möglich ist, alle enthaltenen Datenobjekte vollständig auf eine zugelassene Anzahl von QR-Codes zu verteilen, dabei wird nicht außer Acht gelassen, dass dies nur durch eine Approximation bestimmt werden kann. Ist dies der Fall, so muss das zuletzt hinzugefügte Objekt auf einem anderen Frame untergebracht werden, es bietet sich beispielsweise das quadratische Sondieren an.
Das Ziel, die Suche durch möglichst wenige Überläufe kurz zu halten steht stark in Konkurrenz zu einer platzsparenden Speicherung. Tatsächlich häufen sich Kollisionen mit steigender Befüllung der Struktur. Es kann ein Ziel sein, die maximale Anzahl der Überläufe für jedes Objekt zu begrenzen. Für einen Menschen, der Daten per Hand sucht, und dabei tatsächlich jeden Code einzeln lesen muss, kann bereits eine Begrenzung auf zwei Überläufe zur Tortur werden, müssen hier doch bei einem hypothetischen Wert von 70 Codes pro Frame bis zu 210 Codes gescannt werden. Dass es hierbei keinerlei Garantien gibt, liegt in der Natur der Sache. Bei einer sehr ungünstigen Verteilung der URLs kann ein mehrfacher Überlauf vorkommen, während andere Frames noch vollkommen leer sind. Dies ist allerdings eher als Ausnahme zu sehen.

Bereitstellung eines einfachen Verzeichnisses

Alternativ zum Hashing kann auch ein einfaches Verzeichnis im Stil eines Telefonbuchs angeboten werden. Dieses listet URLs von Ressourcen – oder besser die dazugehörigen Signa-

[7] Siehe `https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html`

turen[8] – in alphabetischer Reihenfolge auf und verknüpft diese jeweils mit einer Koordinate. Diese besteht aus einer Filmnummer, eine Framenummer und optional dem dazugehörigen QR-Code auf der Seite. Bei einer überschaubaren Anzahl von Codes, die für dieses Verzeichnis benötigt werden und einer äußerlich sichtbaren Kennzeichnung des Inhalts kann ein Mensch Einträge durch *Überblättern* schnell finden.
Signaturen bestehen konstant aus 16 Zeichen, sofern man die Bindestriche weglässt. Ein mögliches Format für Koordinaten wäre `<Filmnummer>-<Framenummer>-<Codenummer>`. Geht man von einer einstelligen Filmnummer, vier Ziffern für den Frame und zwei weiteren für den Code aus und lässt die Bindestriche weg, so ist eine Koordinate durch sieben Ziffern darstellbar. Eine einzelne Zeile würde dementsprechend 25 Zeichen (inklusive Leerzeichen und Zeilenumbruch) in Anspruch nehmen. Im Byte-Modus könnte einer der verwendeten QR-Codes bis zu 50 solcher Einträge fassen. Daraus resultiert ein Bedarf von über 66.000 Codes, was bei 70 Codes pro Seite etwa 950 zusätzliche Frames bedeuten würde. Dies scheint nicht zweckmäßig.
Abhilfe kann die Verwendung des alphanumerischen Modus für QR-Codes, sowie die Verringerung der Fehlertoleranz auf die niedrigste Stufe schaffen. Diese würden jeweils 171 Einträge aufnehmen können, wodurch insgesamt über 19.000 Codes (276 Frames) benötigt würden.

Erhöhung des Verknüpfungsgrades

Betrachtet man die Verknüpfungen der verschiedenen Objekte im Modell, wird man schnell feststellen, dass Verbindungen von Hyperkanten zu Knoten stets Sackgassen darstellen. Die jeweiligen Knoten-Teilmodelle speichern keinerlei Verbindungen dieser Art. Dies ist auch in der CIDOC CRM-Variante der Fall. Fügt man den Knotenmodellen jedoch auch sämtliche Bögen hinzu, die sie mit einer Hyperkante verbinden, so ist es möglich, aus dieser lokalen Information über den gesamten Hypergraphen zu navigieren. Erkauft wird dieser Vorteil durch einen Mehraufwand an Kapazität, sowie durch eine erhöhte Anzahl an übergroßer Modelle, die in einem einzelnen QR-Code keinen Platz finden.

5.4.3 Redundanz

Es ist nicht ausgeschlossen, dass Teile des Filmmaterials zerstört werden oder verloren geht. Für diesen Fall sollten Maßnahmen getroffen werden, sodass Daten aus den verbleibenden Datenträgern wiederhergestellt werden können, oder falls dies nicht mehr möglich ist, der Verlust minimiert wird.

Übergreifende Fehlercodes

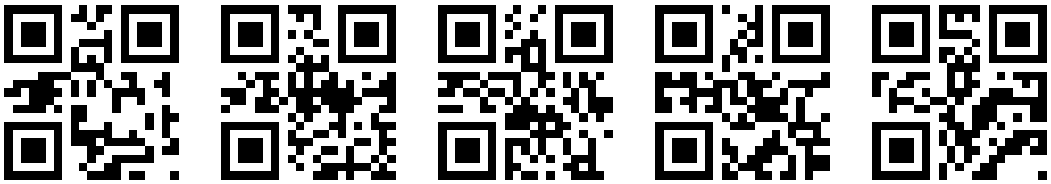
Einzelne QR-Codes werden durch einen Reed-Solomon-Code (Abschnitt 3.3.3 gegen Datenverlust geschützt. Ist ein einzelner Code jedoch zu stark beschädigt, so sind die darin enthaltenen Informationen unwiederbringlich verloren. Weil dies absolut zu vermeiden ist, sollte auch ein Code-übergreifender Mechanismus zur Fehlerkorrektur implementiert werden. Es soll daher ein Vorschlag zur Generierung von Codes zu Fehlerkorrektur aus mehreren QR-Codes folgen.

Generieren einer Code-übergreifenden Fehlerkorrektur Im Allgemeinen ist dieser Vorgang nur dann möglich, wenn alle Codes dieselbe Konfiguration verwenden. Dies umfasst die Version, den Fehlerkorrekturlevel und den verwendeten Modus. Dass dies eventuell auch bei Mischformen möglich ist, sei für den Moment außer Acht gelassen. Zur Demonstration soll ein Fehlerkorrekturcode aus zwei QR-1-H generiert werden. Diese haben die Inhalte `Hallo` und Welt. In beiden Fällen wird UTF-8, also der Byte-Modus verwendet.
Die Erstellung des Fehlerkorrekturcodes setzt mit der Generierung der Bitfolge und der Aufteilung in Codewörter ein. Je nach Konfiguration ist eine bestimmte Anzahl der ersten Bits dieser Folge vorgegeben. Diese definieren den verwendeten Modus und die Länge der Nachricht. Im vorliegenden Fall handelt es sich dabei um die ersten zwölf Bits. Anschließend folgt die tatsächliche Nachricht, sowie der Terminator (mindestens vier Nullen) und das Füllmuster. Diese werden in Codewörter einer Länge von acht Bits unterteilt. Dabei wird

[8] Hier würde es sich anbieten, die URLs der Ressourcen so zu wählen, dass sie sich auf Signaturen direkt abbilden lassen oder umgekehrt. Siehe dazu Abschnitt 5.3.2

Codewort	1	2	3	4	5	6	7	8	9
QR Code #1		H	a	l	l	o			
	40	54	86	16	C6	C6	F0	EC	11
QR Code #2		W	e	l	t				
	40	45	76	56	C7	40	EC	11	EC
XOR		1F	04	00	18	61	CF	DF	D
Reed-Solomon #1		1C	95	AD	85	02	74	5F	0
Reed-Solomon #2		03	91	AD	9D	63	BB	80	0

Tabelle 5.3: Beispiel zur Berechnung von Code-Übergreifenden Korrekturwörtern. Für die Reed-Solomon-Korrekturwörter wird das Generatorpolynom $x^8+x^5+x^3+x^2+1$ verwendet.



(a) QR-Code mit Inhalt `Hallo` (b) QR-Code mit Inhalt `Welt` (c) Parität mit Exklusiv-Oder (d) Parität Reed-Solomon #1 (e) Parität Reed-Solomon #2

Abbildung 5.12: Äußere Fehlerkorrektur für QR-Codes

ein eventuell am Ende begonnenes, nicht vollständiges Codewort wieder entfernt, es enthält keine verwendbare Nutzlast und ein daraus durch Reed-Solomon generiertes (ganzes) Fehlerkorrekturwort kann nicht mehr durch einen QR-Code aufgenommen werden. Es lässt sich leicht beweisen, dass hierbei kein Datenverlust auftritt, wenn die Länge der Nachricht ein ganzzahliges Vielfaches von 8 Bits umfasst, was bei der Verwendung von UTF-8 der Fall ist. Anschließend lässt sich durch Anwendung von Fehlerkorrekturverfahren auf Codewörter mit gleichem Index eine neue Bitfolge generieren. Dabei ist Reed-Solomon genauso anwendbar, wie eine einfache Bildung von Paritäten durch `xor`. Tablelle 5.3 zeigt dies am laufenden Beispiel, die dazugehörigen QR-Codes sind in den Abbildungen unter 5.12 zu finden.

Verteilung der Fehlerkorrekturcodes Abschließend muss entschieden werden, nach welchen Kriterien QR-Codes zusammengefasst werden und wo die Korrekturcodes auftauchen sollen. Der Autor sieht hier grundsätzlich drei Möglichkeiten. Zur Verbildlichung der Bezeichnungen kann man sich vorstellen, dass ein einzelner Film von oben nach unten abgerollt und mehrere Filme nebeneinander gehalten werden, siehe Abbildung 5.13.

Lokale Fehlerkorrektur Aus allen sich auf demselben Frame befindlichen Codes wird eine zuvor festgelegte Anzahl von Korrekturcodes generiert. Dabei kann optional eine weitere Gruppierung in Spalten und Zeilen vorgenommen werden.

Vertikale Fehlerkorrektur Aus jedem Frame wird jeder QR-Code einer Gruppe zugeordnet, in jeder Gruppe ist ein Code von jedem Frame vorhanden. Die Paritäten werden über den Film verteilt, etwa auf gesonderten Frames, welche nur derartige Codes enthalten.

Horizontale Fehlerkorrektur In diesem Ansatz stammt jeder QR-Code von einem anderen Mikrofilm, wodurch eine Rekonstruktion auch bei Verlust eines vollständigen Filmes noch möglich ist.

Zwar besteht der Vorteil der lokalen Paritäten darin, dass zur Korrektur notwendige Elemente schnell verfügbar sind, jedoch versagt dieser Mechanismus wahrscheinlich in der Praxis, weil Schäden am Filmmaterial voraussichtlich durch lokale Einwirkungen, wie Risse, Knicke oder Fingerabdrücke entstehen. Dabei werden nahe beieinanderliegende Codes in Mitleidenschaft gezogen, also viele Elemente derselben Seite. Daher ist diese Variante zu verwerfen. Die vertikale Paritätsbildung verringert derartige Fehler durch Verteilung der Wörter auf die gesamte Länge des Films. In diesem Fall ist allerdings ein einzelnes Korrekturwort pro Spalte

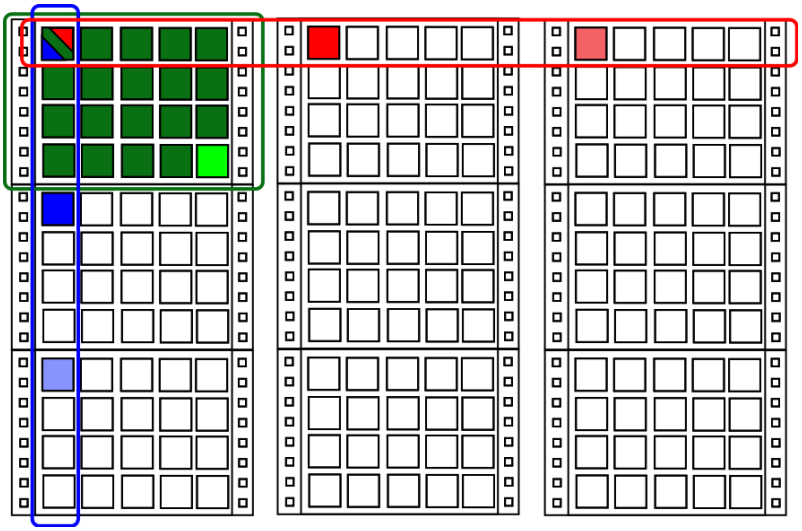
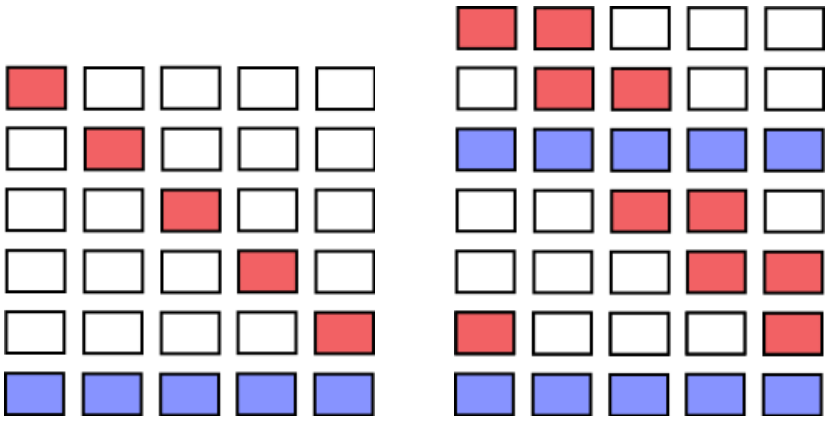


Abbildung 5.13: Gruppierung von QR-Codes zur Bildung von Paritäten. Grün: Lokale Fehlerkorrektur, Blau: Vertikale Fehlerkorrektur, Rot: Horizontale Fehlerkorrektur. Die Paritäten sind jeweils in einem helleren Farbton dargestellt.

nicht ausreichend, da bei einer Anzahl von bis zu 1.440 zugehöriger Codes für eine Fehlerkorrekturgruppe das Risiko für den Ausfall von mehr als einem Code als zu hoch eingeschätzt wird. Daher wird empfohlen, in regelmäßigen Abständen Frames mit Paritäten einzufügen. Entsprechend wächst der Platzbedarf. Dies ist gleichzeitig ein Ausschlusskriterium für die Paritätsbildung mithilfe von Exklusiv-Oder, da diese nur eine einzelne Parität unterstützen. Jedoch ist hiermit die Wiederherstellung ohne Zuhilfenahme weiterer Mikrofilmrollen weitestgehend gesichert.
Zu guter Letzt ermöglicht die horizontale Fehlerkorrektur die Rekonstruktion mithilfe anderer Rollen. Dies kann notwendig sein, wenn Filmrollen verloren gehen, weil sie etwa an verschiedenen Orten gelagert wurden oder während der Sichtung beschädigt werden. Außerdem kann diese Methode der letzte Rettungsanker sein, wenn eine Rolle zu stark beschädigt ist. Aus technischer Sicht spricht nichts dagegen, sämtliche Paritäten auf einer Filmrolle festzuhalten, die Verteilung auf verschiedene Bänder ist genauso möglich.
Es versteht sich von selbst, dass Paritäten enthaltende QR-Codes gesondert gekennzeichnet werden sollten, um diese von den mit RDF gefüllten Codes zu unterscheiden. Deshalb empfiehlt sich das Zusammenfassen von Paritäten auf gesonderten Frames, mit zusätzlicher Kennzeichnung, etwa ein zuvor definiertes Symbol oder einfach durch einen Schriftzug am Rande. Weiterhin verlässt sich diese Methode darauf, dass bekannt ist, wo ein Fehler lokalisiert ist. Dies sollte im vorliegenden Fall stets möglich sein: Existiert ein QR-Code nicht mehr, so ist an dieser Stelle ein Fehler aufgetreten. Kann ein vorhandener Code nicht rekonstruiert werden, so wird dies durch die *innere* Fehlerkorrektur des Codes signalisiert. Es werden keine Paritäten von Paritäten gebildet.

Beispiel Es soll nun ein kurzes Beispiel durchgerechnet werden, welches die kombinierte Nutzung von horizontaler und vertikaler Fehlerkorrektur demonstriert und den Mehraufwand aufzeigt. Dabei soll angenommen werden, dass die Nutzdaten auf 9461 Frames verteilt werden können, was sieben Rollen entspräche. Weiterhin werden auf jedem Film 40 Seiten anderweitig in Anspruch genommen, wodurch pro Rolle 1400 Frames verfügbar sind. Um die Anzahl der horizontalen und vertikalen Paritäten gleich zu halten, wird die in Abbildung 5.14a dargestellte Anordnung vorgeschlagen. Es gilt nun, die zur Speicherung notwendige Anzahl von Filmrollen n zu ermitteln.
Dies gestaltet sich tatsächlich als recht einfach. Abhängig von n passen *untereinander* bis zu $\left\lfloor \frac{1400}{n+1} \right\rfloor$ der dargestellten Anordnungen. Davon fasst jede $n^2 - n$ Frames mit Daten. Gesucht ist also ein minimales, ganzzahliges n, sodass gilt:

$$\left\lfloor \frac{1400}{n+1} \right\rfloor \cdot (n^2 - n) \geq 9461 \tag{5.1}$$



(a) Einfache Anordnung (b) Dopplte Annordnung

Abbildung 5.14: Anordnung von Frames mit Paritätsinformationen. Spalten stellen Filmrollen dar, Rechtecke symbolisieren einzelne Frames. Weiße Frames enthalten Daten, rote horizontale und blaue vertikale Paritäten

Die Gleichung wird durch $n = 9$ erfüllt. Damit werden zwei zusätzliche Filmrollen benötigt. Insgesamt werden 2.800 Frames mit Paritäten enthaltene QR-Codes befüllt. Bei 9 Filmen ist noch Platz für weitere 619 Frames, welche Nutzinformationen enthalten. Nicht belegte QR-Codes sollen immer mit einem Standard-Inhalt gefüllt werden, welcher zur Berechnung der Paritäten herangezogen wird.

Es sei darauf hingewiesen, dass mit steigendem n das Verhältnis von Nutzdaten zu Redundanz gegen Null geht. Wird ein gewünschter Wert unterschritten, so muss diese in dem Fall durch das Hinzufügen weiterer Frames mit Redundanzinformation erhöht werden. Dies wird in Abbildung 5.14b verdeutlicht. Der Bedarf an Filmrollen lässt sich analog berechnen.

Partitionierung des Graphen

Ist ein Datenverlust unvermeidbar, etwa weil mehr Filmrollen fehlen, als zur Wiederherstellung mithilfe von Paritäten notwendig sind, ist dies ein herber Rückschlag. Bei einer ungünstigen Verteilung der Knoten und Hyperkanten über die Filmrollen können die dennoch erhaltenen Daten nutzlos sein: Was ist das Wissen über einen Knoten wert, wenn sämtliche damit verbundenen Hyperkanten auf den verlorenen Filmrollen gespeichert wurden?

Aus diesem Grund bietet es sich an, die Ressourcen des RDF-Modells so auf die Mirkofilme zu verteilen, dass miteinander verknüpfte Elemente bei Möglichkeit auf demselben Datenträger vorhanden sind. Im Umkehrschluss kann man fordern, dass die Anzahl der Kanten zwischen den Filmen möglichst minimal ist, wobei auf jedem die selbe Anzahl von Elementen vorhanden sein soll. Es handelt sich hierbei also um ein Graphpartitionierungsproblem. Dieses ist bekanntermaßen NP-vollständig, allerdings existieren Heuristiken zum Finden einer annähernd optimalen Lösung, darunter auch einige, speziell für bipartite Graphen[39, 47]

Wie in Abschnitt 4.4.3 bereits angemerkt, ist der Hypergraph derzeitig nicht zusammenhängend. Es würde sich anbieten, die insgesamt über 7000 Komponenten bei der Partitionierung des Graphen zu verwenden, da die Anzahl der Verknüpfungen zwischen den Komponenten auch nach Implementierung der Topologie nicht sehr groß sein wird. Tatsächlich verspricht dieser Ansatz allerdings nur wenig Erfolg, da bei genauerer Betrachtung eine sehr große Komponente und sehr viele sehr kleine Komponenten erkennbar sind.

Wird der Graph partitioniert und die Partitionen entsprechend auf die Filme aufgeteilt und soll gleichzeitig eine gute Navigierbarkeit möglich sein, so ist es notwendig, dass die Zuordnung von Ressourcen zur Partition gesondert festgehalten wird. Verwendet man einen Hash-Index (Abschnitt 5.4.2), so muss dieser bei der Berechnung der Position mit einfließen. Dazu wäre ein Verzeichnis notwendig. Aufgrund des nicht geringen Platzaufwandes empfiehlt es sich nicht, das gesamte Verzeichnis auf jeder Filmrolle zu speichern.

5.4.4 Behandlung übergroßer Datenelemente

Es kann ohne Weiteres auftreten, dass Datenelemente, welche in der Datenbank auftauchen zu groß sind, als dass sie in einem einzigen QR-Code platziert werden können. Da diese selbstverständlich dennoch gespeichert werden sollen, muss eine Ausweichstrategie gefunden werden. Eine kleine Auswahl soll vorgestellt werden.
Im einfachsten Fall handelt es sich bei dem Objekt um ein Teilmodell in *Pretty* Turtle Notation, also um eine Ressource mit sämtlichen Tripeln, welche diese im Subjekt stehen haben. Da Statements nicht zwingend zusammen notiert müssen, kann hier ohne Weiteres eine Teilung vorgenommen werden. Diese ist möglichst so durchzuführen, dass ein Teilmodell anschließend annähernd einen eigenen QR-Code ausfüllt. Sofern der Bedarf besteht, können weitere Teilungen vorgenommen werden. Bei der anschließenden Platzierung ist darauf zu achten, dass die betreffenden QR-Codes aufeinander folgen.
Etwas schwieriger ist es, wenn bereits ein einzelnes Statement die Kapazität eines QR-Codes erschöpft. Dies ist insbesondere dann der Fall, wenn im Objekt lange Literale enthalten sind, etwa Texte oder XML. Hier stehen folgende Möglichkeiten zur Verfügung:

- XML-Strukturen sollten im Allgemeinen in eine geeignete, RDF-kompatible Darstellung überführt werden.
- Ist die Verkleinerung des Statements nicht möglich, so muss versucht werden, die Kapazität des speichernden Codes erhöht werden. Die Möglichkeiten hierzu wurden in Abschnitt 5.4.1 erläutert.

Ist dies nicht möglich, so muss eine komplett andere Methode evaluiert werden. Im gegebenen Kontext sollte allerdings kein Problem bestehen.

Kapitel 6

Technische Umsetzung

Die in dieser Masterarbeit vorgestellten Konzepte sollen prototypischen Implementierungen Anwendung finden. Den Präferenzen des Autors entsprechend werden Applikationen in der Programmiersprache Java entwickelt.
Zur Realisierung des Prototypen werden einige Softwaremodule von Dritten eingebunden. Dabei handelt es sich, sofern nicht anders deklariert, um frei verfügbare Bibliotheken, welche mithilfe des Tools Maven[1] organisiert werden. Zur granularen Steuerung der Programmausgaben wird die Logging-Bibliothek *log4j2*[2] verwendet. Sie ermöglicht es, vom Programm erzeugte Ausgaben unterschiedlicher Bedeutung semantisch auszuzeichnen. So existieren unter anderem die Stufen `INFO`, `DEBUG`, `WARN` und `ERROR`, die einzeln gefiltert werden. Die Verwendung mehrerer Instanzen von Loggern ermöglicht es, dass die Ausgaben von Objekten verschiedener Klassen unterschiedlich gefiltert werden. Ein wichtiges Werkzeug für die Arbeit mit RDF-Modellen stellt Apache Jena dar[3]. Es bietet neben dem Erstellen und Modifizieren von Modellen auch die Möglichkeiten zur Arbeit mit Ontologien. Als Template Engine wurde Apache Velocity[4] gewählt. Es ermöglicht das effiziente Zusammenführen von Java-Strukturen und Textvorlagen unter der Verwendung zahlreicher Steuermechanismen. Zur Verarbeitung von XML-Dateien wird JDOM[5] verwendet. Schließlich finden diverse Bibliotheken der Apache Commons[6] Verwendung, etwa die Projekte IO, CSV und Compress. Für die Verbindung mit der Datenbank wird der PostgreSQL-Connector benötigt.

6.1 Überführung des Hypergraphen in RDF

In diesem Abschnitt wird die Implementierung des in 5.2.1 vorgestellten Konzepts beschrieben. Ziel war die generische Überführung des Hypergraphen in eine geeignete RDF-Repräsentation. Dieses wurde in zwei Teilziele zerlegt:

1. Aus den in der Datenbank hinterlegten Informationen über Knotentypen, Attributen, Hyperkantentypen und Bogentypen sollte zunächst ein RDF-Schema generiert werden.

2. Anschließend wird der Hypergraph abgefragt und entsprechend in RDF überführt, wobei das im ersten Schritt generierte Schema verwendet wird.

Hierbei wurde das bereits genannte Framework Apache Jena verwendet. Es erlaubt die komfortable Erstellung von RDF-Modellen, sowie die Ausgabe in einer gewünschten Form.

6.1.1 Schemagenerierung

Die Generierung des Schemas funktionierte entsprechend dem vorgestellten Konzept. Als Namensraum wurde vorerst `http://www.wossidia.de/schema#` verwendet. Es wurden die Klassen `Node` und `HyperEdge`, sowie die Properties `Attribute`, `Link`, `ForwardLink` und

[1] Siehe `https://maven.apache.org/`
[2] Siehe `https://logging.apache.org/log4j/2.x/`
[3] Siehe `https://jena.apache.org/`
[4] Siehe `http://velocity.apache.org/`
[5] Siehe `http://jdom.org`
[6] Siehe `https://commons.apache.org/`

`BackwardLink` erzeugt. Sämtliche darunter liegenden Klassen und Eigenschaften stammen aus der Datenbank.
Informationen zu den Knoten sind der Tabelle `w.cat_table` zu entnehmen. Das gewählte Namensschema war `Node<id>`. Verwendet wurden die Felder `id` für den Namen der RDF-Klasse, `label_long` als `rdfs:label`, `description` als `rdfs:description`. Im Property `rdfs:subClassOf` erscheint stets `Node`.
Auch Hyperkanten stellen keine Herausforderung dar. Die Wurzelklasse ist `HyperEdge`, das Namensschema `HyperEdge<id>` mit dem Feld `id` aus der Tabelle `w.hedget`. Der einzige Unterschied im weiteren Verlauf ist die Benennung der Datenbankfelder: Statt `label_long` wird `name` für das `rdfs:label` verwendet, `rdfs:description` stammt aus `descr`.
Die nachfolgenden Elemente werden nicht als Ressourcen, sondern als Properties modelliert. Bekannte Attribute sind in `w.cat_attribute` hinterlegt. Das Feld `parent`, welches eine Referenz auf den zugehörigen Knotentypen bildet und `name` wurden zur Benennung der Subproperties genutzt: `Attribute<parent>-<name>`. Wieder wurden die Felder `label_long` und `description` für die beschreibenden Eigenschaften `rdfs:label` und `rdfs:description` genutzt. Das übergeordnete Property wird mittels `rdfs:subPropertyOf` stets als `Attribute` festgelegt.
Abschließend folgen nun noch die Bogentypen. Das verwendete Namensschema für Vorwärtsbögen ist `ForwardLink<eid>-<idx>`, entgegengesetzte Bögen werden stattdessen mit `BackwardLink<eid>-<idx>` bezeichnet. Dabei ist `eid` der Primärschlüssel der referenzierten Hyperkante und `id` der Index des Bogentyps. Wie bei den Hyperkanten fließen die Felder `name` und `descr` in das Schema ein. Als Besonderheit stellt sich die Richtung des Bogens heraus. Das Feld `fwd` hat drei mögliche Werte: `true`, `false` und `null`, wobei ersteres einem vorwärts und der zweite Fall einem rückwärts gerichteten Bogen entspricht. Ist keine Richtung angegeben, ist die Kante bidirektional, es sind also beide Möglichkeiten erlaubt. Entsprechend dem Wert werden Bogentypen dem Schema hinzugefügt.
Bei der Anwendung auf die HyDRA-Datenbank wurden insgesamt 56 Knotentypen, 65 Hyperkantentypen und 437 Bogentypen (unabhängig von der Richtung) erstellt. Dabei waren 366 Bögen nach vorne gerichtet, 324 Bogentypen rückwärts. Das Schema wird als Turtle-formatiertes RDFS-Modell ausgegeben.

```
ws:ForwardLink1017-12   a                  rdf:Property ;
                        rdfs:domain        ws:HyperEdgeType1017 ;
                        rdfs:label         "Bedeutung"@de ;
                        rdfs:range         ws:NodeType502 ;
                        rdfs:subPropertyOf ws:ForwardLink .
```

6.1.2 Überführung von Knoten und Hyperkanten

Auch die Überführung der eigentlichen Daten ist nicht weiter spektakulär. In der Vorbereitung der Knotenüberführung werden zuerst für jeden Knotentyp die zugehörige Tabelle, wie auch die Attribute ermittelt. Dabei war eine geringfügige Modifikation der Namen der Tabellenspalten notwendig: Während in der Tabelle `w.cat_attribute` die Namen der Attribute häufig in Großbuchstaben notiert waren, war die Spaltenbezeichnung in den Knotentabellen stets in Kleinbuchstaben vorgenommen. Für jede einzelne Knoten-Instanz wurde ein RDF-Modell in Jena erstellt und die Daten eingepflegt. Dabei wurde die Namenskonvention `<tableID>-<knotenID>` verwendet. Für Knoten wurde ein eigener Namensraum `http://www.wossidia.de/nodes#` verwendet. Das Modell für jeden einzelnen Knoten wurde in Turtle-Schreibweise ausgegeben.
Das Abfragen der Hyperkanten verlief leicht anders. Im ersten Versuch wurde zuerst die Tabelle `w.hedge` abgefragt, welche die einzelnen Kanten enthielt. Für jede enthaltene Hyperkante wurden aus `w.hlink` die betreffenden Bögen abgefragt. Aufgrund der daraus resultierenden, großen Anzahl von Anfragen an die Datenbank stellte sich diese Methode als sehr ineffizient dar. Daher wurde die Tabelle `w.hlink` direkt und vollständig angefragt, wobei eine Sortierung nach Kantentyp und Kante vorgenommen wurde, wie Listing 6.1 zeigt Sämtliche Hyperkanten, sowie die dazugehörigen Bögen wurden in ein einzelnes RDF-Modell überführt. Anschließend wird das Modell in kleinere Modelle zerlegt und in einer Datei gespeichert. Jedes Teilmodell enthält dabei alle für genau eine Hyperkante relevanten Statements, dies umfasst die Kante selbst, sowie alle ein- und ausgehenden Bögen.

```
SELECT  fwd,
    w.hedget.id as hyperEdgeTypeID,
    w.hlink.ntype as nodeTableName,
    w.hlink.eid as hyperEdgeID,
    w.hlink.type as linkTypeID,
    w.cat_table.id as nodeTypeID,
    w.hlink.nid as nodeID
FROM  w.hlink
JOIN  w.hedge ON w.hlink.eid = w.hedge.id
JOIN  w.hedget ON w.hedget.id = w.hedge.type
LEFT JOIN w.cat_table ON w.cat_table.name = w.hlink.ntype
ORDER BY w.hedget.id, w.hlink.eid
```

Abbildung 6.1: SQL zur Abfrage der Hyperkanten

	Minimal	Maximal	Durchschnitt	Summe	> QR-40-H
Pretty	212	19.364	360	1.194.273.637	188
Flat	233	18330	376	1.246.906.825	364

Tabelle 6.1: Informationen zu den generierten Modellen. Alle Größen sind in Bytes angegeben. Die letzte Spalte beschreibt die Anzahl der Modelle, die nicht in einen QR-Code 40 mit Fehlerkorrekturstufe H passen.

6.1.3 Ergebnisse und Probleme

Insgesamt wurden 3.310.393 einzelne Modelle generiert, wobei jedes einen Knoten oder eine Hyperkante vollständig abdeckt. Zur Vergleichbarkeit wurde der Datensatz sowohl im in der *Pretty*-Variante von Turtle, wie auch *Flat* generiert. Tabelle 6.1 liefert eine Übersicht zu den Größen der dabei entstandenen Dateien.

Der benötigte Speicherplatz ließ sich leicht durch Verkürzung der Benennungen von Klassen und Properties verringern. So wurde etwa `HyperEdge` schlicht in `e` umbenannt, selbiges gilt für alle Unterklassen. Auf diese Weise konnten 16% der Gesamtdatenmenge eingespart werden.

Die kleine Dateigröße stellt in Verbindung mit modernen Dateisystemen ein Problem dar. Da häufig – wie auch im vom Autor verwendeten Dateisystem – eine standardmäßige Blockgröße von 4K konfiguriert ist und ein Großteil der erzeugten Dateien weitaus kleiner ist, bleibt hier ein nicht geringer Teil an Speicher ungenutzt. In Zeiten von günstigen Datenträgern ist dies eigentlich kein Problem, reizte aber die Speicherkapazitäten der verwendeten Hardware des Autors aus, weshalb Lösungen zu suchen waren. Vorschläge, dieses Problem zu umgehen sind die Nutzung von Dateisystemen mit geringerer Blockgröße, das Zusammenfassen der Modelle in größere Modelle[7], das Packen der Dateien in Archive oder die Speicherung in einer Datenbank. Für der Implementierung wurden sämtliche Dateien in ein Tape Archiver (TAR)-Archiv zusammengefasst und darüber hinaus mit GZip komprimiert.

Ein weiteres Problem stellt eine Inkonsistenz der Schemainformationen dar: So ist etwa in der Tabelle `w.cat_table` kein Eintrag für den Knotentyp `w.app_datespan` hinterlegt. Daher werden weder RDFS-Informationen für diesen erzeugt, noch die Knoten selbst abgefragt. Es gehen also knapp 8.500 Knoten verloren.

XML-Datenfelder wurden lediglich als Zeichenkette in das RDF-Modell übernommen. Hier sollte eine geeignetere Variante gefunden werden. Darüber hinaus kann es vorkommen, dass bestimmte Attribute in `w.cat_attribute` nicht notiert sind und daher nicht abgefragt werden. Entsprechend fehlen sie im Modell. Eine Überprüfung wurde nicht vorgenommen, widerspricht aber auch dem generischen Charakter dieses Ansatzes. Bei Anwendung auf eine Power.Graph-Datenbank sollten diese Probleme nicht bestehen, wenn die entsprechenden Konsistenzbedingungen zwingend eingehalten werden.

[7] Diese Methode wurde vom Autor bewusst vermieden, da das verwendete System nicht über genügend Arbeitsspeicher verfügte, um große Modelle zu erzeugen.

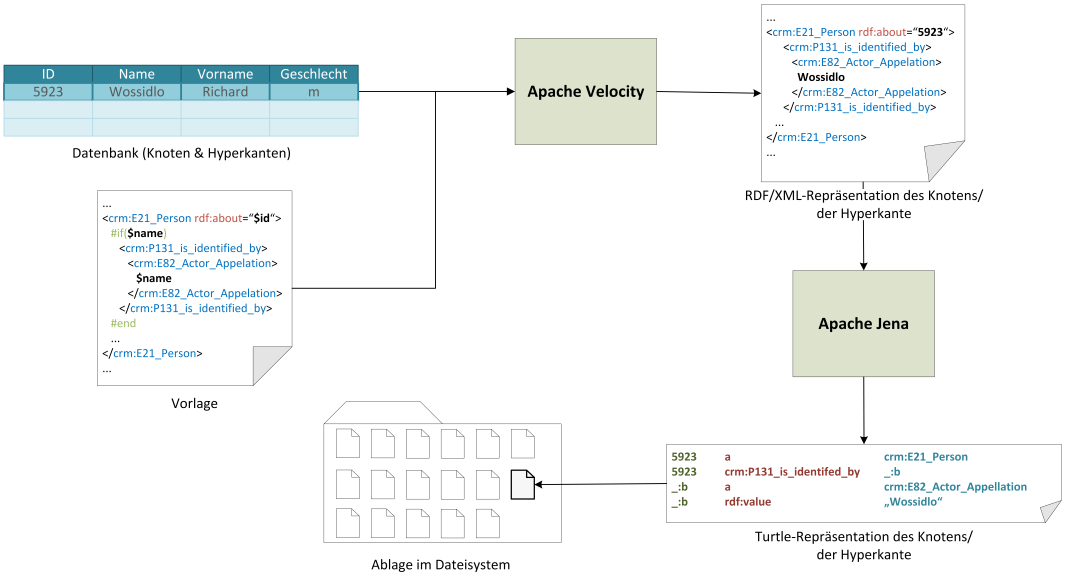


Abbildung 6.2: Abbildung eines Knotens vom Typ *Person* in CIDOC CRM

Nachtrag Im Verlaufe der Arbeit haben sich an dieser Stelle noch einige Änderungen der Implementation ergeben. Grund hierfür war die spontane Verfügbarkeit besserer Hardware, welche es ermöglichte, auch größere RDF-Modelle im Speicher zu halten. So war es möglich, sämtliche Daten aus der Datenbank abzufragen, ein vollständiges Modell im Speicher zu halten und als Ganzes auf Festplatte zu speichern. Dabei wurde außerdem eine GZip-Komprimierung vorgenommen.

6.2 Überführung des Hypergraphen in CIDOC

Die Hypergraphdatenbank soll in das CIDOC Referenzmodell überführt werden. Es sollen Knoten und Kanten voneinander getrennt betrachtet werden, wobei die genannte Reihenfolge bei der Abbildung unerheblich ist. In Absprache mit Mitgliedern der Forschungsgruppe, die derzeit an der Entwicklung von Power.Graph arbeiten, soll es möglich sein, neue Knoten- und Kantentypen einzuführen und abzubilden, ohne die Implementierung der Software zu ändern, beziehungsweise zu erweitern. Vorgeschlagen wurde daher eine generische Methode, welche notwendige Informationen im Datenbankschema speichert. Als Gegenvorschlag wurde eine Vorlagen-basierte Lösung implementiert, die im Folgenden beschrieben werden soll.
Die prototypische Umsetzung umfasst die Klassen `Configuration`, welche die in Abschnitt 6.2.2 beschriebene Konfiguration ausliest und zur Verfügung stellt, sowie der Klasse `Converter`, welche alle restlichen Aufgaben übernimmt.

6.2.1 Allgemeiner Ansatz

Ein fachkundiger Nutzer der Power.Graph-, beziehungsweise HyDRA-Datenbank kann zur Abbildung von Knoten und Kanten Vorlagen definieren, welche die Darstellung eines einzelnen solchen Elements in CIDOC CRM festlegt. Als geeignete Darstellungsform wurde hierbei RDF/XML gewählt. Es sei angemerkt, das *wahrscheinlich* ohne eine Veränderung des Quellcodes auch andere RDF-Serialisierungen, verwendet werden können, solange diese von Jena unterstützt werden. Außerdem besteht grundsätzlich die Möglichkeit, andere Eingabeformate zu nutzen. Dieser Ansatz setzt allerdings einen Parser der Sprache und eine manuelle Überführung in RDF voraus. Eine mögliche Alternative ist der von Lida Harami vorgeschlagene XML-Dialekt, wie er von der CIDOC verwendet[8] wird. Ebenfalls von der CIDOC kommt der Vorschlag zur Verwendung des Tools `XML2RDF`[9] für die Überführung in RDF zur Abbildung beliebiger XML-Dateien in das CIDOC CRM. Mit diesem ist der Au-

[8] Siehe `http://old.cidoc-crm.org/docs/epitafios1.xml`
[9] Siehe `https://github.com/isl/XML2RDF-DataTransformation-MappingTool`

tor allerdings nicht weiter vertraut, weshalb zur Anwendbarkeit keine Aussagen getroffen werden können.
Bei der Gestaltung der Vorlagen können sämtliche von Velocity zur Verfügung gestellten Mechanismen verwendet werden. Als Variablen werden bei der Knotenüberführung sämtliche Datenbankfelder, sowie URLs und Namensräume übergeben.
Im Falle der Hyperkanten erhält das Template Informationen zu den damit verbundenen Bogentypen, den involvierten Knoten und deren Identitäten. Listing **??** und **??** enthalten Beispiele für Vorlagen.
Die definierte Vorlage wird von der Velocity Template-Engine gelesen und mit den Daten der jeweiligen Instanzen zusammengeführt. Das resultierende XML-Dokument wird anschließend mithilfe des Jena-Frameworks gelesen und in die Turtle-Darstellungsform überführt. Jede einzelne Instanz wird in einer eigenen Datei abgespeichert. Abbildung 6.2 gibt eine grafische Übersicht über den Prozess.
Häufig ist es sinnvoll, bestimmte Objekte nur ein einziges mal zu erstellen und abzuspeichern. Dazu zählen etwa Beschreibungen von Typen, einige bestimmte Typen (etwa die Kooperationsrollen *Gelehrter*, *Erzähler* und *Beiträger*, welche im Knotentyp Person verwendet werden) oder – sofern man sich dazu entscheidet, eigene Klassen zu definieren – deren CIDOC-RDFS-Definitionen. Aus diesem Grund kann für jeden Knoten-, beziehungsweise Hyperkantentypen eine Vorlage definiert werden, welche solch wiederverwendbare Ressourcen enthält. Diese können dann, entsprechend ihres URI, beziehungsweise URL von anderen Orten referenziert werden. In diese Vorlage werden keine Daten aus den Instanzen eingefügt, stattdessen wird lediglich der Namensraum zur Verfügung gestellt, unter dem die Elemente der Datei abrufbar sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:lang="en"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/" xml:base="
    ↪ $crm_base">
  <crm:E90_Symbolic_Object rdf:about="$crm_base" rdf:value="$name"
    ↪ >
    #if($number)
    <crm:P1_is_identified_by>
      <crm:E41_Appellation rdf:value="$number">
      #if($type)
        <crm:P2_has_type>
          <crm:E55_Type rdf:value="$type"/>
        </crm:P2_has_type>
      #end
      </crm:E41_Appellation>
    </crm:P1_is_identified_by>
    #elseif($type)
    <crm:P2_has_type>
      <crm:E55_Type rdf:value="$type"/>
    </crm:P2_has_type>
    #end
    <crm:P2_has_type rdf:resource="$type_typErzaehlmotiv" />
  </crm:E90_Symbolic_Object>
</rdf:RDF>
```

Listing 6.1: CIDOC-Vorlage für den Knotentyp Erzählmotiv

6.2.2 Konfiguration

Die Konfiguration des Prozesses wird in einer XML-Datei festgehalten. Ein Beispiel hierfür zeigt Listing 6.3. Sie wird vor dem Konvertierungsvorgang einmalig durch die Klasse

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:lang="de"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/" xml:base="
    ↪ $crm_base">
    <crm:E7_Activity rdf:about="$crm_base">
      <crm:P2_has_type rdf:resource="$type_handlung" />
      #foreach($link in $links)
        #if($link["type"] == "Handlung")
          <crm:P15_was_influenced_by rdf:resource="$link["
            ↪ resource_url"]" />
      #elseif($link["type"] == "Quelle")
        <crm:P70i_is_documented_in rdf:resource="$link["
          ↪ resource_url"]" />
      #end
    #end
  </crm:E7_Activity>
</rdf:RDF>
```

Listing 6.2: CIDOC-Vorlage für dem Hyperkantentyp Inhalt

`Configuration` gelesen. Es findet eine Unterteilung der Konfiguration in vier Abschnitte statt:

- Der erste Abschnitt beinhaltet die zur Verbindung mit der Datenbank notwendigen Informationen. Dazu zählen Server und Port, Datenbankname, wie auch Login-Informationen.
- Es folgt die Konfiguration der Knoten. Allgemein wird zunächst das Ausgabeverzeichnis (`dir`) für Knoten, relativ zum später gewählten allgemeinen Ausgabeverzeichnis (`outdir`) festgelegt, außerdem noch die Fundorte der Abbildungsvorlagen für Knoten (`mappings`), sowie der wiederverwendbaren Definitionen (`reusables`). Letztere liegen relativ zur Konfigurationsdatei selbst. Es folgt eine Liste definierter Knoten (`nodes`, beziehungsweise `node`, für einen einzelnen Eintrag). Diese beinhalten den Namen der Tabelle, welche die Knoten des jeweiligen Typs enthält, sowie eine für den Menschen lesbare Benennung (`label`). Weil letztere sowohl als Name für das Ausgabeverzeichnis der Knoten fungiert, wie auch später in den URLs der Ressourcen auftaucht, sollte eine Benennung vorgenommen werden, die den jeweiligen Kriterien entspricht. Wird für einen hier definierten Knotentypen keine Vorlage im angegebenen Verzeichnis gefunden, so wird eine Warnung ausgegeben und der jeweilige Eintag übersprungen.
- Analog zum zweiten Abschnitt werden im dritten Teil die vorhandenen Hyperkantentypen definiert. Sie unterscheiden sich lediglich in einem Punkt: Statt des Tabellennamens wird für Hyperkantentypen die `id` definiert. Diese entspricht dem Datenbankfeld `w.hedget.id`.
- Abschließend folgen noch einige unsortierte Konfigurationen. Als Präfix für die URL der generierten RDF-Ressourcen dient das Feld `namespaceBase`. Für den Fall, dass die Funktion noch realisiert wird, kann mittels `schema` die Schemadefinition für das CIDOC CRM angegeben werden. Es handelt sich um einen Dateipfad, relativ zur Konfigurationsdatei. Abschließend enthält `outdir` den Ausgabeordner.

Das XML-Dokument wird mithilfe der Java-Bibliothek JDOM gelesen und anschließend die Datenfelder manuell ausgelesen und im `Configuration`-Objekt abgelegt. Die elegantere Variante wäre die Verwendung von Java Architecture for XML Binding (JAXB). Hierbei wird zunächst eine Java-Klasse definiert und anschließend mit Annotationen ausgezeichnet. Mithilfe von JAXB ist des dann zum einen möglich, den Objektzustand in XML zu überführen (*Marshalling*), wie auch die entgegengesetzte Richtung, wobei ein Java-Objekt aus einem XML-Dokument erzeugt wird (*Unmarshalling*).

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <database>
    <host>localhost</host>
    <port>26989</port>
    <name>wossidia</name>
    <user>mm543</user>
    <password>password</password>
  </database>
  <nodes dir="nodes" mappings="node_mappings" reusables="
      ↪ node_reusables">
    <node name="w.am_person" label="person" />
    <node name="w.am_motif" label="erzaehlmotiv" />
    <node name="w.am_type" label="erzaehltypen" />
  </nodes>
  <edges dir="edges" mappings="edge_mappings" reusables="
      ↪ edge_reusables">
    <edge id="1019" label="inhalt" />
  </edges>
  <misc>
    <namespaceBase>http://www.wossidia.de/wossidia/</namespaceBase
        ↪ >
    <schema>cidoc-crm.rdf</schema>
    <outdir>out</outdir>
  </misc>
</config>
```

Listing 6.3: Konfigurationsdatei für Hyperkanten und Knoten

Neben den Einstellungen aus der Konfiguration bietet die `Converter`-Klasse außerdem zahlreiche Methoden zur Generierung von Namensräumen und Dateipfaden an. Diese können in den Vorlagen teilweise benutzt werden. Darüber hinaus stellt sie Namensräume und URLs für die wiederverwertbaren Ressourcen zur Verfügung.

6.3 Aufteilung des Modells

In Abschnitt 5.4 wurde die Aufteilung des RDF-Modells diskutiert. Es wurden mögliche Anforderungen besprochen und Konzepte erläutert, diese zu realisieren. Im Folgenden sollen daher die Ausführungen der Konzepte vorgestellt werden.

6.3.1 Kompaktheit

Es sollen zwei Konzepte evaluiert werden: Der Versuch einer optimalen Verteilung des Modells, sowie die Kompression. Zugrunde liegt ein vollständiges Modell des Hypergraphen in RDF. Die gewählte Serialisierung ist *Flat*, welche etwas mehr, als 622 MB (652.969.243 Bytes) in Anspruch nimmt. Würde man diese Datei entsprechend der Grenzen eines QR-40-H-Codes in 1273 Byte große Einheiten aufteilen, so benötigte man 542.938 solcher Codes. Unter der Annahme, dass auf einem Frame $10 \cdot 7 = 70$ dieser Codes Platz finden, werden 7.757 Frames oder sechs Filme zu je 1440 Frames benötigt.

Platzoptimierte Verteilung

In der ersten Variante wurde versucht, einzelne Tripel des Modells so zu verteilen, dass möglichst wenige Codes benötigt werden. Die Java-Klasse `FlatFit` liest die oben genannte Datei ein und bestimmt die Länge der Serialisierungen der einzelnen Tripel. Ein Jena-Modell wird dabei nicht generiert. Anschließend approximiert die Klasse `FirstFitDecreasing` eine optimale Verteilung und gibt die Anzahl der benötigten QR-Codes zurück. Bei der Kapazität

eines jeden Behälters wird die für jeden Code notwendige Definition der Namensraum-Präfixe berücksichtigt.
Die *Flat*-Variante ermöglicht die Verteilung auf 569.467 Behälter, wobei jedoch 76.182 Byte ($< 1\%$) Daten nicht berücksichtigt werden, da es sich um Tripel handelt, die aufgrund ihrer Größe nicht in einem einzelnen QR-Code Platz finden. Diese müssen gesondert behandelt werden. In dieser Anordnung würden die Daten ebenso auf sechs Rollen Mikrofilm passen.
Der zweite Versuch beschäftigte sich mit der Verteilung der Ressourcen mitsamt aller Statements, die diese als Subjekt enthalten. Bei den Hyperkanten kommen außerdem noch die rückwärts gerichteten Bögen hinzu. Die Klasse `PrettyFit` liest das Modell und baut es im Speicher auf. Aufgrund des erhöhten Speicherbedarfs ist es in diesem Fall notwendig, entsprechende Instruktionen an die Java Virtual Machine (JVM) weiterzugeben, um dem Heap mehr Arbeitsspeicher zuzuteilen. Es werden Submodelle generiert und deren Größe in ihrer Pretty-Turtle-Entsprechung ermittelt. Anschließend wird wieder die Ermittlung der notwendigen Codes durchgeführt.
Wie erwartet wird in diesem Fall mehr Speicher in Anspruch genommen. Insgesamt werden 622.993 Codes benötigt, dies entspricht 8900 Frames, welche bereits einen siebten Film benötigen. Hinzu kommen knapp 2 MB, welche in übergroßen Modellen zu finden sind, die gesondert behandelt werden müssen.

Kompression

Das betreffende, vollständige Model wurde, zusammen mit dem Schema (295.856 Bytes) mittels TAR zusammengefasst und mit GZip komprimiert. Das resultierende Archiv hatte eine Gesamtgröße von 113.073.271 Bytes (107 MB). Nimmt man wieder 70 Codes pro Frame an, so passt dieses Variante auf 1267 Frames, ist also auf einer Filmrolle vollständig enthalten. Mithilfe besserer Kompressionsverfahren ist dieses Ergebnis sogar noch deutlich zu unterbieten: Der Lempel-Ziv-Markow-Algorithmus 2 (LZMA2) in der Implementierung von 7-Zip[10] erreichte gar eine Größe von 36.245.504 Bytes. Dies entspricht knapp über 400 Frames. Eine Nutzung von Kompression unter Aufgabe der Wahlfreiheit bei der Reihenfolge der QR-Codes scheint also eine ernstzunehmende Alternative darzustellen.

6.3.2 Navigierbarkeit

Zur Verbesserung der Navigierbarkeit wurde die Verwendung eines Hash-Indexes evaluiert. Ziel der Evaluation war es, das RDF-Modell auf so wenigen Frames, wie möglich unterzubringen und dabei die Anzahl der Überläufe auf zwei zu begrenzen. Es wurde dasselbe Modell, wie in den vorangegangenen Abschnitten verwendet, die *Pretty*-Darstellungsweise wurde gewählt. Wieder wird eine Kapazität von 70 QR-Codes pro Frame angenommen.
Als Hashfunktion wurde die `.hashCode()` Methode der Java-Klasse `String` verwendet, mit anschließender Modulo-Operation durch die maximale Anzahl der Frames. Weil Java keine `unsigned`-Datentypen kennt, wurde mithilfe von `Math.abs()` der Betrag erzeugt. Bei einem Überlauf – das bedeutet, ein Frame kann das zuletzt hinzugefügte Modell nicht mehr aufnehmen – soll sondiert werden. Dazu wird der letzte Hashwert quadriert und anschließend Modulo gerechnet.
Im ersten Versuch wurde als mögliche Anzahl von Frames die notwendige Kapazität von 8.900 Frames auf 10.080 erhöht, was sieben vollständigen Filmrollen entspricht. Hierbei jedoch schnell ersichtlich, dass die Verwendung einer Primzahl als Anzahl der Speicherplätze unumgänglich ist, da ansonsten Kollisionen in einer nicht mehr handhabbaren Größenordnung auftreten. Aus den Primzahlen zwischen 8.900 und 10.080 wurde eine Anzahl von 9.461 als kleinste Anzahl von Frames ermittelt, bei der das gegebene Modell mithilfe des Hashverfahrens gespeichert werden konnte, ohne dass zur Suche irgendeines Teilmodells mehr als zwei mal sondiert werden musste. Dabei waren pro Frame im Durchschnitt gerade einmal 66 QR-Codes gefüllt.

6.3.3 Redundanz

Zur Redundanz wurden keine gesonderten Versuche unternommen. Dies ist insbesondere der geringen verbleibenden Restzeit zur Bearbeitung dieser Abschlussarbeit geschuldet. Aller-

[10] Siehe `http://www.7-zip.de/`

dings ist bei der Betrachtung des Problems der Graphparitionierung (Abschnitt 5.4.3) und den damit verbundenen Komponenten der bipartiten Form des Hypergraphen (Abschnitt 4.4.3) etwas Erwähnenswertes aufgefallen:
Wie bereits erläutert, ist der Graph aufgrund des Fehlens der Topologie-Verknüpfungen in über 7.000 Komponenten zerlegt. Die größte Komponente enthält 553.200 Knoten (Knoten und Hyperkanten). Diese lassen sich ohne Probleme mithilfe des First-Fit-Decreasing-Algorithmus auf 1.437 Frames, also einer Filmrolle unterbringen. In den weiteren Komponenten kommen insgesamt 43.367 Knoten zusammen, die sich – offensichtlich – auf einer weiteren Rolle unterbringen lassen. Die restlichen 2.713.826 Knoten sind – im Sinne des Hypergraphen – isoliert. Diese können annähernd beliebig auf die weiteren Filme verteilt werden.

6.4 Experimentelle Bestimmung der Größe von QR-Codes

Zur Speicherung digitaler Daten auf Film werden im Rahmen dieser Arbeit QR-Codes in Betracht gezogen. Diese in ISO/IEC 18004 standardisierten, optoelektronisch lesbaren Schriften bestehen aus Matrizen quadratischer Zellen in schwarzer und weißer Farbe. Sie ermöglichen die Speicherung weitaus größerer Datenmengen, als die bisher im WossiDiA-Projekt verwendeten ITF-Barcodes, welche zur Codierung von Signaturen genutzt wurde[44]. Daher ist die Anwendbarkeit zur Konservierung von umfassenden Hypergraphdaten denkbar.
Im Laufe des Lesevorgangs eines solchen QR-Codes wird dieser von der verwendeten Software über ein Referenzraster gelegt und die Färbung der einzelnen Zellen erkannt. Dies kann jedoch nur dann erfolgreich geschehen, wenn die Zellen sich ausreichend voneinander unterscheiden. Da man zumeist bestrebt ist, viel Information auf wenig Raum unterzubringen, bietet es sich an, eine Codedarstellung auf das Mindeste zu verringern. Allerdings ist dies nicht unbegrenzt möglich. Ab einem bestimmten Grad verschwimmen die Details und können nicht mehr differenziert werden, wodurch auch die Farben der einzelnen Zellen nicht mehr unterschieden werden können. Aus diesem Grund soll auf experimentellem Wege ermittelt werden, mit welcher Größe ein auf 35mm-Film gespeicherter QR-Code ohne Probleme wieder rekonstruiert werden kann. Neben der Dimensionierung soll außerdem untersucht werden, wie sich eine künstliche Verschlechterung des Kontrastes auf die Dekodierbarkeit auswirken. Da es nicht im Kompetenzbereich des Verfassers dieser Arbeit liegt und die Entwicklung des Films an einen externen Dienstleister delegiert werden muss, sowie wirtschaftliche Einschränkungen existieren, können die Einflüsse der Verwendung unterschiedlicher Filmmaterialien, sowie der Entwicklungsprozesse nicht untersucht werden. Es kann lediglich angenommen werden, dass den Anforderungen des BBK[19] zur Genüge getan wird.

6.4.1 Vorausgehende Versuche

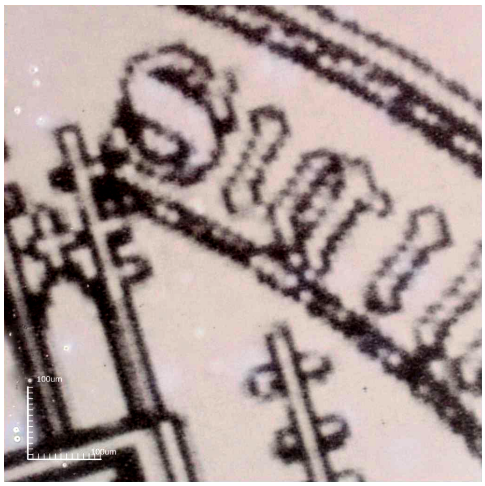
Im Vorfeld des eigentlichen Experiments eröffnete sich die Gelegenheit für einen spontanen Vorversuch. Mithilfe eines Laser-Scanning-Mikroskops[11] konnte ein bereits vorhandener, belichteter Mikrofilm untersucht werden. Ziel der Unternehmung war es festzustellen, ob bei der verwendeten Auflösung der Quellbilder einzelne Pixel auf dem Film zuverlässig differenzierbar sind. Wäre dies der Fall, so könnte in Erwägung gezogen werden, die Auflösung zu vergrößern und damit höhere Datendichten zu untersuchen.
Abbildung 6.3b zeigt eine Detailansicht des untersuchten Frames. Hier sind einzelne Pixel in verschiedenen Abstufungen von Grautönen gut erkennbar, weshalb diesem Bereich besondere Aufmerksamkeit geschenkt wurde. Mit einer Höhe von 69 Pixeln hätte hier ein QR-Code der Version 13 ohne Ruhezone Platz, wenn eine Zelle einem Pixel entspräche. Bei 10-facher Vergrößerung der Mikroverfilmung wird sehr schnell eine Problematik ersichtlich: Über das gesamte Bild sind schwarze Punkte verteilt (Abbildung 6.3c). Mithilfe der Oberflächenvermessung konnte festgestellt werden, dass es sich hierbei um Vertiefungen handelt. Da der untersuchte Mikrofilm zu Demonstrationszwecken mehrfach ab- und wieder aufgerollt wurde ist anzunehmen, dass diese durch Staub und andere Verunreinigungen entstanden sind, die sich in die Oberfläche gedrückt haben. Es ist außerdem nicht auszuschließen, dass sie ihren Ursprung im Produktionsprozess haben. Aufgrund der Größe dieser Erscheinungen stellen sie ein Problem bei der Abbildung von QR-Codes dar, sie verdecken mehrere Pixel. Aus

[11] Siehe `http://www.lfm.uni-rostock.de/ausstattung/analyse/mikroskop/`

(a) Gesamtansicht des BKW-Frames

(b) Detailansicht des Universitätslogos

(c) Logo mit Vertiefungen (hell) bei 10-facher Vergrößerung, Farben invertiert



(d) Detailaufnahme des Logos bei 20-facher Vergrößerung, Farben invertiert

Abbildung 6.3: Oben: Ansichten der Originaldatei. Die Lage der Detailansicht ist im linken Bild rot skizziert. Die Größe eines einzelnen Pixels ist rechts grün dargestellt. Unten: Mikroskopische Aufnahmen der Verfilmungen. Zur besseren Darstellung wurden die Farben invertiert und Kontraste verstärkt.

diesem Grund ist die Verwendung hoher Fehlerkorrekturstufen unabdingbar. Weiterhin verdeutlichen sie die Wichtigkeit der Regelung, dass Mikroverfilmungen nur zum Zwecke des Anlegens weiterer Kopien gebracht werden dürfen[19].
Der weitere Verlauf der Untersuchung gestaltete sich als schwierig. Bei 20-facher Vergrößerung konnte das Bild nicht ausreichend scharf geschaltet werden. Grund hierfür ist, dass der Mikrofilm nicht korrekt auf dem Objektträger fixiert werden konnte, ohne den Film zu beschädigen. Daraus resultierte ein ungünstiger Abstand zum Objektiv. Nach Einschätzung des bedienenden Mitarbeiters ist der schwarze Hintergrund des Mikrofilms beim verwendeten Auflicht nicht optimal, was dem Kontrast der Aufnahme nicht zugute kommt. Weiterhin geriet der Film bereits durch geringsten Luftzug in Schwingung, was dies weiter erschwerte. Abbildung 6.3d zeigt eine Detailaufnahme in dieser Vergrößerungsstufe. Einzelne Pixel sind hier nicht erkennbar. Nach zuversichtlicher Einschätzung des Autors können bei guten Bedingungen der Aufnahme Gebilde in der Größe von drei mal drei Pixeln bei der verwendeten Auflösung erkannt werden. Damit hätte der in Abbildung 6.3d gezeigte Bereich immerhin noch Platz für einen QR-Code in der Version 1. Anders ausgedrückt hätten in dieser Auflösung $10 \times 14 = 140$ QR-Codes maximaler Größe auf einem einzigen Frame Platz, womit alle bisherigen Platzanforderungen halbiert wären.

6.4.2 Grundlegender Aufbau des Experiments

Der Ablauf des Experiments sei wie folgt skizziert: Für jede Variante (1-40) wird eine Reihe von QR-Code-Anordnungen verschiedener Größe erzeugt. Ziel ist es, die Größe so zu wählen, dass an der kurzen Seite des Frames vier bis zwanzig Codes randlos nebeneinander platziert

Platz finden. In jeder Reihe wird von links nach rechts die Qualität des erzeugten Codes künstlich verschlechtert, indem der Kontrast verringert wird. Für jede Variante und Größe werden vier Reihen erzeugt, die jeweils den unterschiedlichen Fehlerkorrekturmodi entsprechen. Jede dieser Anordnungen wird regulär (schwarze Punkte auf weißem Hintergrund) und noch einmal invertiert (weiße Punkte auf schwarzem Grund) erzeugt. Der Inhalt jedes Codes entspricht vier Ziffern, gefolgt von einem Buchstaben. Dabei entsprechen die ersten beiden Ziffern der Anzahl der nebeneinanderliegenden Codes, danach folgt der Grad der Kontrastverringerung in Prozent. Den Abschluss macht ein B (Black) oder ein W (White), dies stellt die Punktfarbe auf dem Film dar. Version und Fehlerkorrekturmodus sind den Metadaten des Codes zu entnehmen. Mithilfe des First-Fit-Decreasing-Algorithmus werden die Anordnungen platzsparend auf den Frames verteilt. Abbildung 6.4 zeigt exemplarisch ein Frame.
Es sei angemerkt, dass nicht jede Anordnung für jede QR-Version abgebildet wurde. Außerdem wurden zum Teil mehr als 20 Codes nebeneinander angeordnet. Dies ist mit der Notwendigkeit begründet, dass eine Zellgröße immer aus einer ganzzahligen Anzahl von Pixeln bestehen sollen. Folgende Beispiele sollen dies verdeutlichen, dabei soll angenommen werden, dass die kurze Seite des Frames aus 5600 Pixeln besteht und jeder Code von einer Ruhezone aus zwei Zellen umgeben ist. Ein QR-Code der Version 1 hat eine Breite von 25 Zellen. Um 20 dieser Codes auf dem Frame nebeneinander zu platzieren, muss eine Zellengröße von elf Pixeln gewählt werden. Möchte man selbiges mit 19 Codes tun, so muss dies mit derselben Zellgröße getan werden. Da dies keinen Vorteil bringt, wird die Anordnung ausgelassen. Sollen 20 QR-40-Codes nebeneinander im Frame erscheinen, ergibt sich eine rechnerische Zellgröße von 1,55 Pixeln, es wird entsprechend auf einen Pixel abgerundet. Damit wäre allerdings fast ein Drittel der Seite ungenutzt, weshalb die maximale Anzahl von Codes platziert wird, in diesem Fall 30.

6.4.3 Erzeugung der Frames

Zur Erzeugung der Framedateien wurde eine Handvoll frei verfügbarer Software genutzt, deren Durchführung mithilfe eines Bash-Skriptes organisiert wurde. Die Generierung der QR-Codes wird durch `qrencode`[12] vollzogen. Für die Komposition, sowie Nachbearbeitung, etwa Invertierung und Kontrastverringerung, wurde die Software `ImageMagick`[13] verwendet. Im gesamten Prozess wurde darauf geachtet, dass Bilddateien so generiert werden, dass sie in ihrer Größe nicht mehr verändert werden müssen Außerdem wurde auf verlustbehaftete Kompression verzichtet.
Bei der Dimensionierung der Frames wurde eine Auflösung von 7920 mal 5600 Pixel gewählt, dies entspricht Bilddaten der bereits archivierten Dokumente. Für alle 40 Versionen wurden insgesamt 391 Anordnungen erzeugt, die jeweils regulär und invertiert abgebildet wurden. Diese wurden auf 303 Frames verteilt, hinzu kommen 40 Frames als Trenner, sie enthalten die Versionen der nachfolgenden Anordnungen.

6.4.4 Stand zum Ende der Arbeit

Aufgrund von organisatorischen Gründen konnte eine Ausbeleuchtung der Filme nicht stattfinden. Daher kann auch die Auswertung des Experiments nicht durchgeführt werden.

[12] Siehe `https://fukuchi.org/works/qrencode/`
[13] Siehe `http://www.imagemagick.org/`

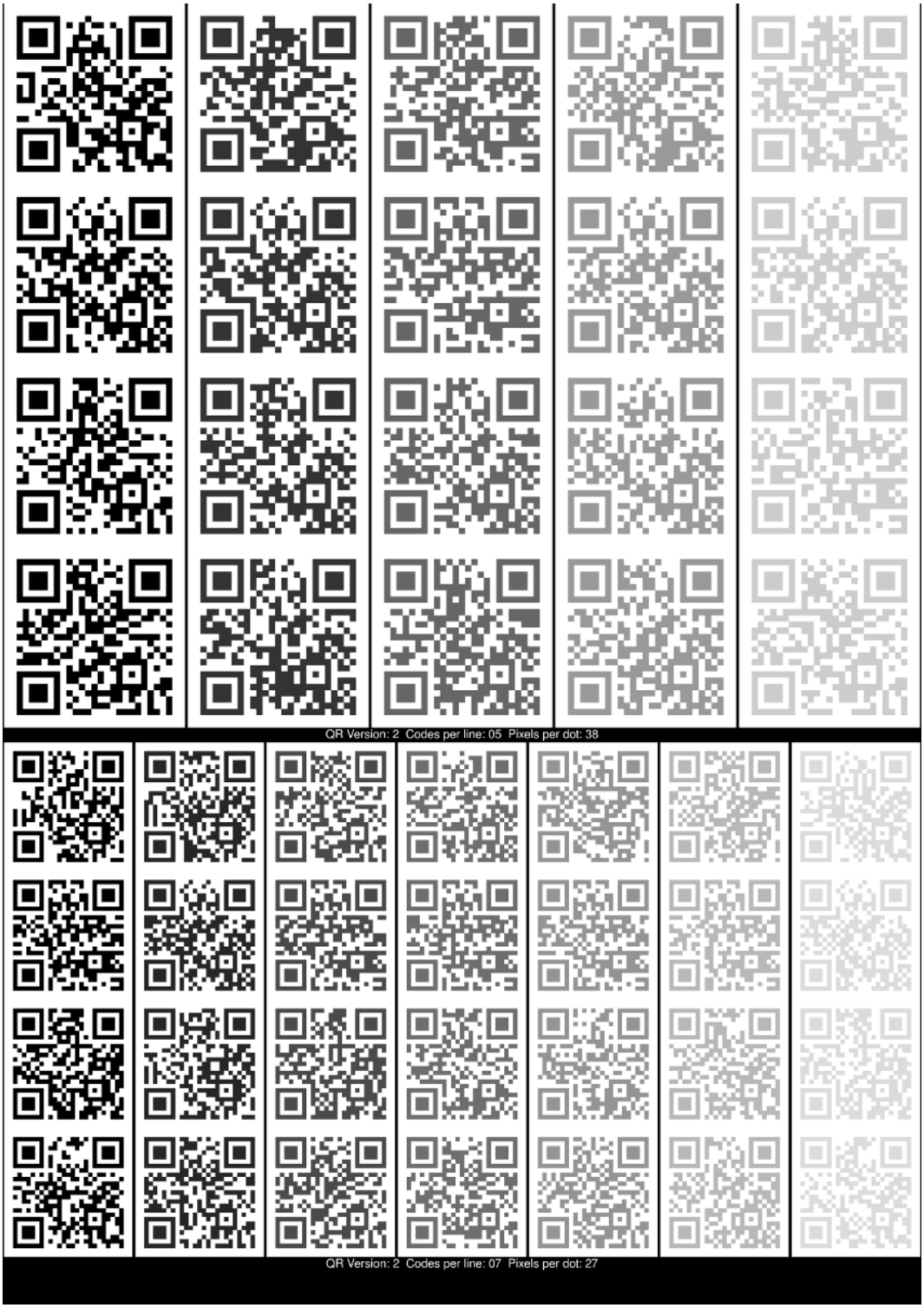


Abbildung 6.4: Anordnungen von fünf und sieben QR-Codes in einem Frame. Die Abbildung wurde bereits invertiert, sie erscheint in dieser Form auf dem Film.

Kapitel 7

Instruktionen für die Nachwelt

Angenommen, im Jahr 2018 – also ein Jahr nach Entstehung dieser Arbeit – gelangt ein durchschnittlich technik-affiner Mensch in den Besitz der Mikrofilmrollen, welche die Metadaten der WossiDiA-Datenbank enthalten. Mithilfe eines geeigneten Lesegerätes ist er in der Lage, diesen zu vergrößern und erkennt ihm bekannte QR-Codes. Da auf seinem Smartphone eine entsprechende App installiert ist, kann er wahllos Codes scannen und erhält "seltsam formatierte" Texte. In diesen kommt häufiger die Abkürzung RDF vor, welche er bei Google eintippt. Hierbei erfährt er, dass es sich dabei um ein Konzept für verknüpfte Daten handelt. Mithilfe verfügbarer Software ist er in der Lage, die Teilmodelle zu einem großen zusammenzufügen. Dabei stößt er auch auf die Schemainformationen und kann sich einen Reim, auf die Bedeutung der von ihm gefundenen Filmrollen machen.
Wie könnte diese Situation in zehn Jahren ab heute aussehen? Die findende Person sieht wieder zuerst die QR-Codes. Diese sind ihr noch von "damals" bekannt, daher glaubt sie, dass diese irgendwelche Daten enthalten könnten. Nach einiger Recherche wird eine Software zum Lesen dieser Codes gefunden, jedoch ist sie mit den aktuellen *Ho-Lenses*[1] nicht kompatibel. Mit etwas mehr Aufwand wird eine entsprechende Software gefunden, die den Inhalt der eingescannten Symbole in separate Dateien überführt. Beim Betrachten mit einem aktuellen Editor macht sich erst Ernüchterung breit, weil die Codes offenbar beschädigt waren: Der Inhalt ist unleserlich. Erst, als auf Anraten eines Kollegen ein veraltetes UTF-8-Plugin installiert wird, offenbart sich der Inhalt...
Darüber, wie das Szenario im Jahre 2117 oder gar noch später aussehen könnte, kann nur spekuliert werden. Offensichtlich ist aber, dass nicht erwartet werden kann, dass zukünftige Generationen die aktuell geläufigen Techniken noch kennen oder gar Werkzeuge haben, diese zu verwenden. Aus diesem Grund ist es unerlässlich, entsprechende Instruktionen zusammen mit den Metadaten zu hinterlassen, weil ansonsten alle Mühen vergeblich waren.
Instruktionen müssen auf mehreren Filmen vorhanden sein, falls es zu Verlusten kommt. Ob sie auf allen Trägern zu finden sein sollten, ist diskutierbar und sollte dem tatsächlich anfallenden Aufwand angepasst werden. Über die Entwicklung der natürlichen Sprache kann nur gemutmaßt werden, daher sollen sämtliche Beschreibungen in deutscher Sprache formuliert sein. Diese wird schließlich auch für das Verständnis des Wossidlo-Archivs benötigt. Aufgrund der Verbreitung kann außerdem eine englischsprachige Variante hinzugefügt werden. Weiterhin sollen grundlegende mathematische Konzepte als gegeben gelten, jedoch nicht unbedingt deren Bezeichnungen. So kann etwa das abstrakte Konzept von Galois-Körpern und deren Algebra bekannt sein, wer Évariste Galois war und womit er sich beschäftigte jedoch nicht zwangsläufig. Es müssen also entsprechende Hinweise gegeben werden.

7.1 Interpretation des Modells

Dem ahnungslosen Finder werden sich Rollen voller "Mosaikkunstwerke"[2] offenbaren. Weil deren Inhalt nicht offensichtlich ist, muss zuerst eine Motivation geschaffen werden, den notwendigen Aufwand zur Interpretation in Angriff zu nehmen. Dazu zählt in allererster Linie die Verknüpfung der vorliegenden Inhalte mit der ursprünglichen Sicherungsverfilmung des

[1] Ein fiktives Gerät, welches in etwa den selben Stellenwert hat, wie das gegenwärtige Smartphone
[2] Zitat eines Gastes der Zwischenverteidigung dieser Arbeit.

Wossidlo-Archivs oder – sollte der unglückliche Fall des Verlusts noch nicht eingetreten sein – gar mit dem originalen Archiv. Es ist nötig, die Motivation zur ursprünglichen Schaffung dieser Daten zu erklären, nämlich die Verknüpfung der Millionen von *Zetteln* und anderer Archivalien untereinander, die in mühevoller Arbeit erreicht wurde. Dabei kann es nicht schaden, deren ursprüngliche Form – nämlich die eines Hypergraphen – zu erwähnen, da diese als Grundlage für die dann vorliegenden Daten gedient haben werden.
Anschließend ist das überlieferte Datenmodell zu klären. Das sollte, wenn sich an den Ergebnissen dieser Arbeit orientiert wird, das CIDOC CRM sein. Dessen allgemeine Funktionsweise ist zu erläutern, ebenso ist die gegenwärtig große Verbreitung und Anerkennung im Bereich ist anzubringen. Dabei sollen gerne Verweise auf andere Institutionen auftauchen, welche das Modell verwenden – eventuell hat man dort ebenfalls eine analoge Langzeitarchivierung in Angriff genommen. Ein Verweis auf die Schemadefinition (RDFS) stellt eine gute Überleitung auf das verwendete Trägerformat des Referenzmodells dar: Das RDF.

7.2 Das RDF-Modell

Dem Findenden muss verdeutlicht werden, dass das es sich bei dem Resource Description Framework nur um eine von vielen Möglichkeiten handelt, das CIDOC CRM zu beschreiben. Gleichzeitig stellt es einen Mechanismus dar, der flexibel genug ist, um annähernd beliebige, selbst definierte Aussagen zu tätigen. Der Nutzer muss in der Lage sein zu verstehen, dass ein RDF-Modell eine Menge von Subjekt-Prädikat-Objekt-Tripeln ist, die als gerichteter und beschrifteter Graph interpretierbar ist. Die Mechanismen zur Beschreibung – das umfasst sowohl die RDF-eigenen Elemente, als auch die von RDFS – sollen erläutert werden. Unabhängig von der gegebenen Schemadefinition sollte die Abbildung von RDF auf CIDOC CRM beispielhaft erläutert werden.
Weiterhin sollen die Serialisierungen Erwähnung finden. In erster Linie umfasst dies die verwendete Variante `Turtle`. Aufgrund der gegenwärtigen Verbreitung ist es denkbar, sowohl RDF/XML, wie auch JSON-LD aufzuführen und deren semantische Gleichheit betont werden. Dies ist damit zu begründen, dass Varianten ohne Weiteres in anderen Projekten auftauchen können.
Ein Mechanismus zur Suche im Modell, wie etwa die SPARQL Protocol and RDF Query Language (SPARQL) müssen nicht zwangsläufig beschrieben werden. Da sich das RDF-Modell als gerichteter Graph interpretieren lässt, sind sämtliche Graphenalgorithmen auf diesen anwendbar. Ebenso können Techniken zur effizienten Speicherung und Abfrage solcher Modelle als gegeben betrachtet werden. Die Unabhängigkeit der Reihenfolge beim Einlesen des Modells, beziehungsweise der Teilmodelle sollte betont werden.
Es existiert eine Vielzahl von Dokumentationsmaterial zum RDF. Diese sind vollständig auf den Seiten des W3C gelistet, siehe `https://www.w3.org/standards/techs/rdf#w3c_all`. Deren Erwähnung wird ausdrücklich empfohlen.

7.3 Vom Bild zum Klartext

Abschließend muss erklärt werden, die im vorangegangenen Schritt erwähnten RDF-Teilmodelle aus den QR-Codes gelesen werden können. Dies umfasst zuerst die Generierung einer Folge von Bits, mitsamt Fehlerkorrektur, aus einem QR-Code und anschließend deren Interpretation als Zeichenfolge. Es wurde angeraten, sich bei der Verwendung dieser Verfahren auf eine oder zumindest wenige Varianten zu beschränken. Diese Maßnahme kommt dem Durchführenden spätestens hier zugute. Zwar sind die Prinzipien der Kodierung und Dekodierung von QR-Codes bei allen Versionen gleich, jedoch unterscheiden sie sich in nicht-trivialer Weise in den Details, etwa bei der Anzahl der Gruppen und Blöcke pro Gruppe, der konkreten Anzahl von Fehlerkorrekturwörtern, der Platzierung von Such- und Ausrichtungsmustern und so weiter. Die Beschränkung auf wenige Varianten erspart die unnötige Auflistung. Ebenfalls notwendig ist die Klärung der Fehlerkorrektur, konkret der Reed-Solomon-Code oder – falls diese Verwendung fand – die Methode der Paritätenbildung mithilfe von Exklusiv-Oder. In der Hoffnung, dass entsprechende Dokumente noch auffindbar sind, kann natürlich auch ein Verweis auf den Standard ISO/IEC 18004 angeführt werden. Dies ermöglicht die Verwendung bereits – oder eventuell wieder – vorhandener Werkzeuge.

Anschließend ist die Übersetzung der Bitfolge in Klartext zu erläutern. Dies beginnt zuerst mit der Festlegung der Bytereihenfolge(Big Endian oder Little Endian). Darauf folgt eine Klärung des Konzepts von UTF-8, insbesondere die Variable Länge der einzelnen Zeichen sollte Erwähnung finden. Abgeschlossen wird der Abschnitt dann mit den konkreten Zeichentabellen – selbstverständlich ebenso in menschenlesbarer Form. Dass dabei nicht der vollständige Unicode dokumentiert werden kann ist naheliegend. Zumindest sollten aber die verwendeten Teile gelistet werden. Dazu zählt definitiv der Block `Basic Latin`, aber auch `Greek and Coptic`, sowie `Cyrillic`. Während griechische Buchstaben in der WossiDiA-Datenbank Verwendung finden, taucht das russische Alphabet spätestens im CIDOC CRM-RDFS in den mehrsprachigen Bezeichnungen auf. Referenzierbare Normen sind Request for Comments (RFC) 3629 und ISO/IEC 10646-1. UTF-8 ist in den ersten 128 Zeichen identisch zum ASCII-Code.
Wurde eine Teilmenge der QR-Codes genutzt, um Paritäten zur Korrektur von Fehlern zu erzeugen, so muss außerdem eine Unterscheidung möglich sein. Weiterhin muss die Funktionalität des Korrekturmechanismus verdeutlicht werden.

Kapitel 8

Abschluss

8.1 Zusammenfassung

Diese Arbeit beschäftigte sich mit den Methoden zur Langzeitarchivierung digitaler Daten auf analogen, optischen Medien. Als Ergänzung der Sicherungsverfilmung des Wossidlo-Archivs sollten Techniken untersucht werden, welche die unterstützenden Metadaten dauerhaft auf Mikrofilm festhalten können. Dabei wurde zuerst eine Überführung des zugrundeliegenden Hypergraphenmodells in eine geeignetere, bekannte Form – konkret handelt es sich um das CIDOC CRM – überführt. Dieses wiederum wurde mithilfe des RDF in eine transportfähige und vor allem teilbare Form gebracht, dessen Bestandteile sich in optoelektronischen Codes speichern lassen.
Alle diese Teilschritte basieren auf gegenwärtig wohl-dokumentierten, standardisierten und verbreiteten Technologien. Daher ist die Verwendung in anderen Projekten durchaus denkbar und empfehlenswert: Das RDF ermöglicht die problemlose Zerlegung großer Modelle in kleinere Teilmodelle, welche sich auf Speichermedien mit geringer Kapazität verteilen lässt. Mithilfe von RDFS ist es außerdem selbst-beschreibend, weshalb eigene Schemata definiert und den Daten ohne nennenswerten Mehraufwand beigefügt werden können. Gleichzeitig existieren Regeln für etablierte Ontologien, etwa das genannte CIDOC CRM, welches im Bereich der Archivierung von Kulturerbe ein de facto-Standard ist, oder der allgemein weitverbreitete Dublin Core.
Es wurden mögliche weitere Ziele vorgeschlagen, für die Konzepte erarbeitet und zum Teil umgesetzt wurden. Mithilfe einer Approximation des Behälterproblems konnte eine annähernd optimale Zusammenlegung von Teilmodellen in QR-Codes ermittelt werden. Bei einer geschätzten möglichen Speicherdichte dieser Codes und Verwendung eines generischen Modells des Hypergraphen konnte die benötigte Anzahl auf sieben Filmrollen festgelegt werden. Zur Unterstützung von Nutzern bei der manuellen Suche von Datensätzen auf den Rollen wurde ein Hashing-Verfahren evaluiert. Dieses zeigte sich bei vertretbarem Mehraufwand an Platz als anwendbar, wobei eine gewünschte Ressource – sofern existent – spätestens nach dem Scannen aller QR-Codes auf maximal zwei Frames zu finden war. Allerdings zeigte sich diese Variante als wenig Speichereffizient und ermöglichte keine Zerlegung des Graphen in Partitionen. Hiermit wäre der Schaden beim endgültigen Verlust einer Filmrolle minimiert, da Referenzen vorrangig innerhalb eines Mikrofilms geschehen würden. Weiterhin wurde die Möglichkeit einer verbesserten Fehlerkorrektur untersucht, die es ermöglichen, nicht mehr rekonstruierbare QR-Codes oder gar vollständig fehlende Filmrollen bis zu einem gewissen Grad auszugleichen. Auch hier war der notwendige Mehraufwand vertretbar.
Insgesamt bewertet der Autor die untersuchten Methoden grundsätzlich als anwendbar. Der notwendige Bedarf von etwa zehn Mikrofilmrollen zum Fassen der gesamten Metadaten scheint auf den ersten Blick sehr hoch. Allerdings ist dieser, selbst bei einer zu erwartenden Verdopplung, aufgrund einer Nachpflege der Datensätze, der nachträglichen Implementierung der Topologie und der Verwendung des CIDOC CRM im Vergleich zum Gesamtaufwand der Sicherung noch gering, so hat die Sicherungsverfilmung des Archivs knapp 180 Rollen Film in Anspruch genommen.
Auf Kosten eines erhöhten nötigen Dokumentationsgrades zur Interpretation können die Anforderungen an den Speicher weiter verringert werden. Dieser Effekt kann durch die Verwendung von Kompressionsverfahren nochmal deutlich verstärkt werden, allerdings dann

auf Kosten der Egalität der Lesereihenfolge der Codes. An dieser Stelle, wie bei allen vorgestellten Verfahren müssen die tatsächlichen Bedürfnisse abgewogen werden, bevor sich für eine endgültige Lösung entschieden werden kann, welche dann im Rahmen einer Sicherungsverfilmung Verwendung findet.

8.2 Ausblick

Bevor tatsächlich an eine Langzeitsicherung der Metadaten gedacht werden kann, ist es notwendig, dass diese bereinigt werden. Zu Inkonsistenzen und möglichen Neukonzeptionen wurden in den entsprechenden Kapiteln Hinweise gegeben. Anschließend muss eine endgültige Abbildung des Hypergraphen auf das CIDOC CRM erarbeitet werden.
Im Rahmen der Arbeit war es zeitlich nicht mehr möglich eine zuverlässige minimale Größe für QR-Codes auf Mikrofilmen zu ermitteln. Dieser Schritt ist ebenfalls unumgänglich und sollte vollzogen werden. Die dazu notwendigen vorbereitenden Schritte konnten während der Laufzeit der Arbeit durchgeführt werden, lediglich die Belichtung der Filme und deren Auswertung sind noch nötig.
Es müssen konkrete Anforderungen formuliert und die vorgestellten Techniken angewendet werden. Dabei ist anzuraten, sich mit anderen Projekten in Verbindung zu setzen, die ebenfalls eine Langzeitarchivierung anstreben und gemeinsame Anforderungen und Umsetzungen zu erarbeiten. Mit steigender Verbreitung und Akzeptanz einer homogenen Lösung steigt der Nutzen, wenn eines Tages die Daten rekonstruiert werden oder im Rahmen eines Generationenwechsels der Datenträger nach 500 Jahren neu verfilmt werden müssen.
Für zukünftige Projekte sollten Konzepte erarbeitet werden, welche die Speicherung der Metadaten von Anfang an in Betracht ziehen. Während es sich bei WossiDiA um eine nachträgliche Archivierung der reinen digitalen Daten handelt, kann es empfehlenswert sein, die Metadaten direkt an Ort und Stelle mit den betreffenden Archivalien zu lagern.

Anhang A

Literaturverzeichnis

[1] H Peter Anvin. The mathematics of raid-6, 05 2009.

[2] Giorgio Ausiello. *Directed hypergraphs: Data structures and applications*, pages 295–303. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[3] C Berge and North-Holland Hypergraphs. Amsterdam. *Combinatorics of finite sets, Translated from the French*, 1989.

[4] Uwe M Borghoff, Peter Rödig, Jan Scheffczyk, and Lothar Schmitz. *Langzeitarchivierung - Methoden zur Erhaltung digitaler Dokumente*. dpunkt, Heidelberg, 1. aufl. edition, 2003.

[5] Andreas Brandstädt. Graphen und algorithmen. stuttgart: Bg teubner, 1994. Technical report, ISBN 3-519-02131-5, 1994.

[6] Bundesamt für Bevölkerungsschutz und Katastrophenhilfe. Schutz von kulturgut bei bewaffneten konflikten, 2012.

[7] Bundesamt für Bevölkerungsschutz und Katastrophenhilfe. Sicherungsverfilmung. `http://www.bbk.bund.de/DE/AufgabenundAusstattung/Kulturgutschutz/Sicherungsverfilmung/sicherungsverfilmung_node.html` zuletzt abgerufen am 11.01.2017, 2017.

[8] Su-Shing Chen. The paradox of digital preservation. *Computer*, 34(3):24–28, 2001.

[9] George M. Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628, 2012.

[10] Consultative Committee for Space Data Systems. Reference model for an open archival information system (oais), recommended practice. Technical report, CCSDS 650.0-M-2 (Magenta Book) Issue 2, 2012.

[11] Intel Coropration. Intelligent raid 6 theory, overview and implementation. *Intel Storage Building Blocks*, 2005.

[12] Jeroen de Vries, Dimitri Schellenberg, Leon Abelmann, Andreas Manz, and Miko Elwenspoek. Towards gigayear storage using a silicon-nitride/tungsten based medium. *arXiv preprint arXiv:1310.2961*, 2013.

[13] Deutsche UNESCO-Kommision. Haager konvention. `https://www.unesco.de/kultur/haager-konvention.html` zuletzt abgerufen am 18.01.2017, 2017.

[14] Miko C. Elwenspoek. Long-time data storage: Relevant time scales. *Challenges*, 2(1):19–36, 2011.

[15] Cary Fowler. The svalbard seed vault and crop security. *BioScience*, 58(3):190, 2008.

[16] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2-3):177–201, 1993.

[17] Tom Geller. The forever disc. *Commun. ACM*, 57(5):24–26, May 2014.

[18] Florian Gless. Ein Bier für die Götter. *National Geographic Deutschland*, page 28, Februar 2017.

[19] GMBl Sicherheitsverfilmung. Grundsätze zur durchführung der sicherheitsverfilmung von archivalien. `http://www.bbk.bund.de/SharedDocs/Downloads/BBK/DE/Gesetzestexte/Grundsaetze_Durchfuehrung-Sicherheitsverfilmung-Archivalien.html` zuletzt abgerufen am 11.01.2017, 03 2011.

[20] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature*, 494(7435):77–80, 2013.

[21] Robert N Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J Stark. Robust chemical preservation of digital information on dna in silica with error-correcting codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, 2015.

[22] Michael Grube. Kulturgutschutz und bergungsorte in deutschland ost & west. `https://www.geschichtsspuren.de/artikel/58-ausweichsitze-regierungsbunker/83-barbarastollen-kulturgutschutz.html` zuletzt abgerufen am 24.01.2017, 2017.

[23] JOHN HAYWOOD. Völker, staaten und kulturen. *Ein universal-historischer Atlas. Westermann, Braunschweig*, 1998.

[24] Brian P. Hogan. *HTML 5 & CSS 3* -. O'Reilly Germany, 1. aufl. edition, 2011.

[25] Michael Hompel, Hubert Büchter, and Ulrich Franzke. *Identifikationssysteme und Automatisierung* -. Springer Berlin Heidelberg, Wiesbaden, 1. aufl. edition, 2007.

[26] Seth van Hooland and Ruben Verborgh. *Linked data for libraries, archives and museums: how to clean, link and publish your metadata.* Facet Publ., London, 2014.

[27] Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[28] Peter Kazansky, Ausra Cerkauskaite, Martynas Beresna, Rokas Drevinskas, Aabid Patel, Jingyu Zhang, and Mindaugas Gecevicius. Eternal 5d data storage via ultrafast-laser writing in glass. *SPIE Optoelectronics & Communications*, 2016.

[29] Roladn Kiesendahl. Konzeptuelle modellimodel historischer daten in digitalen, volkskundlichen archive. Master's thesis, Universität Rostock, 2014.

[30] Susan Lambrecht, Gerd Richard, and Christoph Schmitt. *Das große WOSSIDLO-Lesebuch.* Hinstorff-Verlag, 2009.

[31] Dietlinde Lau. *Algebra und Diskrete Mathematik 2.* Springer-Verlag Berlin Heidelberg, 2004.

[32] Bernhard Lenk. *Handbuch der automatischen Identifikation. 2. 2D-Codes: Matrixcodes, Stapelcodes, Composite-Codes, Dotcodes.* Lenk Fachbuchverl., 2002.

[33] Bernhard Lenk. *Optische Identifikation - Schwerpunkt Lesetechnik.* Lenk, Kirchheim, 1. aufl. edition, 2005.

[34] LVR-Archivberatungs- und Fortbildungszentrum. Bundessicherungsverfilmung. `http://www.afz.lvr.de/de/bestandserhaltung_2/bundessicherungsverfilmung/bundessicherungsverfilmung_1.html` zuletzt abgerufen am 24.01.2017, 2017.

[35] Andreas Manz. The human document project and challenges. *Challenges*, 1(1):3–4, 2010.

[36] Holger Meyer, Alf-Christian Schering, and Christoph Schmitt. Wossidia - the digital wossidlo archive. *Corpora Ethnographica Online*, 2014.

[37] Amadıs Antonio Martınez Morales and Marıa Esther Vidal Serodio. A directed hypergraph model for rdf. In *Proc. of Knowledge Web PhD Symposium*, 2007.

[38] Heike Neuroth, Achim Oßwald, Regine Scheffel, Stefan Strathmann, and Karsten Huth. *nestor-Handbuch: Eine kleine Enzyklopädie der digitalen Langzeitarchivierung.* vwh Verlag, 2010.

[39] N. B. Omeroglu, I. H. Toroslu, S. Gokalp, and H. Davulcu. K-partitioning of signed or weighted bipartite graphs. In *2013 International Conference on Social Computing*, pages 815–820, Sept 2013.

[40] Jan Petr, Václav Ranc, Vítězslav Maier, Pavlína Ginterová, Joanna Znaleziona, Radim Knob, and Juraj Ševčík. How to preserve documents: A short meditation on three themes. *Challenges*, 2(1):37–42, 2011.

[41] James S Plank et al. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Softw., Pract. Exper.*, 27(9):995–1012, 1997.

[42] Gunter Saake, Kai-Uwe Sattler, and Andreas Heuer. *Datenbanken - Implementierungstechniken (3. Aufl.).* MITP, 2011.

[43] Günter Saake, Kai-Uwe Sattler, and Andreas Heuer. *Datenbanken - Konzepte und Sprachen.* MITP-Verlags GmbH & Co. KG, Heidelberg, 5. aufl. edition, 2013.

[44] Alf-Christian Schering, Ilvio Bruder, Susanne Jürgensmann, Holger Meyer, and Christoph Schmitt. From box to bin–semi-automatic digitization of a huge collection of ethnological documents. In *International Conference on Asian Digital Libraries*, pages 168–171. Springer, 2011.

[45] UNESCO. Convention for the protection of cultural property in the event of armed conflict with regulations for the execution of the convention 1954, May 1954. `http://portal.unesco.org/en/ev.php-URL_ID=13637&URL_DO=DO_TOPIC&URL_SECTION=201.html` zuletzt abgerufen am 18.01.2017.

[46] Ingo Wegner. Theoretische informatik-eine algorithmische einführung. *B. G. Teubner Stuttgart* Leipzig*, 1999.

[47] Hongyuan Zha, Xiaofeng He, Chris Ding, Horst Simon, and Ming Gu. Bipartite graph partitioning and data clustering. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 25–32. ACM, 2001.

Anhang B

Abbildungsverzeichnis

Anhang C

Tabellenverzeichnis

Anhang D

Glossar

ACID Atomicity, Consistency, Isolation, Durability

AFNOR Association française de normalisation

AIP Archival Information Package

AJAX Asynchronous JavaScript and XML

ASCII American Standard Code for Information Interchange

ATA AT Attachment

ATU Aarne-Thompson-Uther

BBK Bundesamt für Bevölkerungsschutz und Katastrophenhilfe

BKW Beiträgerkorrespondenz Wossidlos

BMBF Bundesministerium für Bildung und Forschung

BRD Bundesrepublik Deutschland

CCSDS Consultative Committee for Space Data Systems

CD Compact Disc

CIDOC Internationales Komitee zur Dokumentation / Comité international pour la documentation

CIDOC CRM CIDOC Conceptual Reference Model

CSS Cascading Style Sheets

CSV Comma-separated values

DCL Data Control Language

DDL Data Definition Language

DDR Deutsche Demokratische Republik

DFG Deutsche Forschungsgemeinschaft

DIN Deutsches Institut für Normung

DIN EN DIN Europäische Norm

DIP Dissemination Information Package

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.

DML Data Manipulation Language

DNS Desoxyribonukleinsäure

DTD Document Type Definition

DVD Digital Versatile Disc

EAN Europäische Artikel Nummer

ECI Extended Channel Interpretation

ERM Entity Relationship Model

ESA European Space Agency

FNA Mecklenburgisches Flurnamenarchiv

GF Galois Körper / Galois Field

GNU GNU's Not Unix

GZip GNU Zip

HECM Hyperedge Content Model

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HyDRA a HYpergraph of Documents in a Relational Archive

IBW Inventarbuch Wossidlos

IEC International Electrotechnical Commission

Inc. Incorporated

ISO International Organization for Standardization

ITF Interleaved Two of Five

JAXB Java Architecture for XML Binding

JPG Joint Photographic Experts Group

JSON Java Script Object Notation

JSON-LD JSON for Linking Data

JVM Java Virtual Machine

KHM Kinder- und Hausmärchen

LFSR Linear-Feedback Shift Register

LIT Literaturobjekte

LZMA2 Lempel-Ziv-Markow-Algorithmus 2

MDS Maximum Distance Separable

MIME Multipurpose Internet Mail Extensions

MWT Alphabetische Zettelsammlung für das Mecklenburgische Wörterbuch, Bestand Teuchert

MWW Alphabetische Zettelsammlung für das Mecklenburgische Wörterbuch, Bestand Wossidlo

NARA National Archives and Records Administration

NASA National Aeronautics and Space Administration

nestor Kompetenznetzwerk für digitale Langzeitarchivierung in Deutschland / Network of Expertise in long-term STOrage and availability of digital Resources in Germany

NRW Nachlass Richard Wossidlos

OAIS Open Archival Information System

OCH Ortschroniken

OpenPGP Open Pretty Good Privacy

P-ATA Parallel ATA

PDF Portable Document Format

PDF 417 Portable Data File 417

PDI Preservation Description Information

PNG Portable Network Graphics

PWA Publikationen Wossidlo-Archiv

QR Quick Response

RAID Redundant Array of Independend Disks

RAM Random Access Memory

RDF Resource Description Framework

RDFS RDF-Schema

RDMS Relational Database Management Software

RFC Request for Comments

RLG Research Libraries Group

RM4SCC Royal Mail 4 State Customer Code

RSS-14 Reduced Space Symbology 14

S/MIME Secure / MIME

SDRAM Synchronous Dynamic RAM

SGML Standard Generalized Markup Language

SIP Submission Information Package

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

SSD Solid State Disk

TAR Tape Archiver

TIFF Tagged Image File Format

Turtle Terse RDF Triple Language

UCC Uniform Code Council

UCS Universal Character Set

UNESCO United Nations Educational, Scientific and Cultural Organization

URI Unified Resource Identificator

URL Uniform Resource Locator

UTF-8 8-Bit Universal Character Set Transformation Format

W3C World Wide Web Consortium

WossiDiA Wossidlo Digital Archive

XHTML Extensible Hypertext Markup Language

XML eXtensible Markup Language

XOR Exclusiv-Oder

XSD XML Schema Definition

ZAW Zettelarchiv Wossidlos

ZTW Zetteltranskriptionen Wossidlos

Anhang E

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Die Stellen der Masterarbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Rostock, den 26. Mai 2017

Marc-Eric Meier