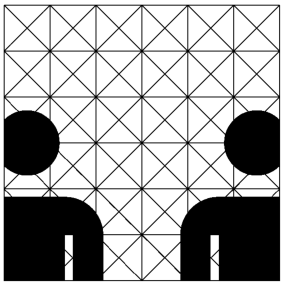

Universität Hamburg
Fachbereich Informatik
Verteilte Systeme und Informationssysteme (VSIS)

Diplomarbeit

Identitäten und ihre Schnittstellen auf Basis von Ontologien in einer dezentralen Umgebung

4. Februar 2005

Gordian Kaulbarsch

Eißendorfer Straße 114
21073 Hamburg

1kaulbar@informatik.uni-hamburg.de
Matrikelnummer: 4340587

Erstbetreuung: Prof. Dr. Winfried Lamersdorf
Zweitbetreuung: Prof. Dr. Klaus von der Heide

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Der Begriff der Identität ist vielschichtig. Die vorliegende Arbeit widmet sich der Identität als der Repräsentation einer Person gegenüber ihrer sozialen Umgebung. Die soziale Umgebung ist dabei im Rahmen dieser Arbeit die virtuelle Welt der globalen Datennetze.

Wenn heute von Identitäten und auch von Identitätsmanagement im Bereich von digitalen Diensten gesprochen wird, so ist damit meist eine eingeschränkte Sichtweise gemeint. Wenn Personen auch im virtuellen Raum mehr sein wollen als Warenempfänger, Zahlende oder "Nicknames", nämlich individuelle und facettenreiche Kommunikationspartner, dann muss sich die Komplexität des digitalen Identitätskonzeptes derjenigen des realen annähern:

> Further on, more than identification and authentication is needed since we want to share *general* information about an identity. (Baier et al., 2003, S. 3)

Im Bereich der realen Identität ist aber weder ein fest abgegrenzter Raum von zu berücksichtigenden Bereichen oder Themen, welche einer Identität zuzuweisen wären, benennbar, noch sind Standards definiert, auf denen der Informationsaustausch zwischen Individuen stattfindet. Dies macht ein umfassendes Konzept erforderlich mit den Möglichkeiten, die notwendige Flexibilität einerseits und eine Vereinheitlichung oder einen Abgleich des Informationsflusses andererseits zu gewährleisten. Dieses Konzept muss dem durch die Individualität der Identitäten gegebenen Mangel an Kompatibilität entgegentreten.

Die Umsetzung des realen Identitätskonzeptes in ein digitales Identitätskonzept muss aber nicht nur den Variantenreichtum des ersteren in eine maschinengestützte Form übertragen. Sie sollte vielmehr auch die zusätzlichen Gegebenheiten digitaler Technik vorteilsnutzend mit einbeziehen. Umfassende Vernetzung mit direkten Verbindungen und die Möglichkeit zur Nutzung von Daten in maschinenlesbarer Form sind für einen vorteilhaften Einsatz beispielsweise denkbar. Um dem Anwender[1] keine Barriere in den Weg zu legen, ist es notwendig, möglichst viele Aspekte – gerade solche von struktureller und organisatorischer Natur – so weit wie möglich transparent zu halten. Wenn der Anwender auf der einen Seite durch keinen oder nur einen minimalen Mehraufwand, aber auf der anderen Seite verschiedene Vorteile, Erleichterungen oder neue Dienste erlangt, die auf dem Konzept digitaler Identitäten aufsetzen, wäre dies eine Bewältigung eines ansonsten sicherlich auftretenden Akzeptanzproblems.

Der in dieser Arbeit verfolgte Ansatz nutzt flexible und modulare Taxonomien, um Identitätsmodelle unterstützen zu können, deren Umfang heutigen Identitätskonzepten

[1] Der Begriff *Anwender* sei in dieser Arbeit im allgemeinen Sinne von "Benutzer" zu verstehen. Insbesondere gelte er nicht als Abgrenzung zu anderen Rollenbegriffen wie Programmierer oder Konzeptioner.

gegenüber stark erweitert sein kann. Mittels der Ontologien werden strukturierte Metadatenmodelle gebildet, deren Einzelelemente den verschiedenen Identitätsdaten zugewiesen werden. Somit kann den Identitätsdaten durch Zuweisung einer Metadaten-Erweiterung und eine umfassende Ordnungsstruktur gegeben werden. Gleichzeitig wird zum Abgleich verschiedener Modelle und Teilmodelle ein Abbildmechanismus verwendet, der ein Identitätsmodell in ein anderes transformieren kann. Eine Kommunikation auf Grundlage dieser Identitätsdaten kann somit trotz unterschiedlicher Modelle auf jeweils ein Modell zurückgreifen und dieses als Basis nutzen. Auf diese Weise wird der hohen Flexibilität ein Verfahren zur Integration beigefügt, um einen einheitlichen Standard für das gesamte Identitätsmodell zu gewährleisten.

1.1. Identitäten und ihre Darstellung bis heute

Identitäten treten in verschiedenen wissenschaftlichen Disziplinen in unterschiedlicher Weise auf. Laut Kunze (2003) benennen die Mathematik und teilweise frühe Philosophien neben der Selbstreferenz das Konzept des Unveränderten als wesentliches Kriterium von Identität und tragen damit insbesondere theoretischen Gegenständen ohne Berücksichtigung des Faktors Zeit Rechnung. Psychologie und Sozialwissenschaften betrachten naturgemäß Lebewesen, meist Menschen, im Kontext ihrer Umgebung und innerhalb der Zeit. Hier können sehr wohl Veränderungen bei einem Menschen geschehen, ohne dass deshalb seine Identität, beziehungsweise die Identität zu sich selbst verloren ginge. Gerade im sozial-psychologischen Kontext weicht der Begriff der Identität einer Person vom ursprünglichen mathematischen Begriff ab. Aspekte wie die Selbstdarstellung und deren Unterschiedlichkeit in verschiedenen sozialen Kontexten fließen in diesen Bereich mit ein, wie auch Identifizierbarkeit und deren Vermeidung. Aus soziologischer Sicht ist der Begriff der Identität durch vielfältige Aspekte geprägt:

> Identität wird heute als komplexe Struktur aufgefasst, die aus einer Vielzahl einzelner Elemente besteht (Multiplizität), von denen in konkreten Situationen jeweils Teilmengen aktiviert sind oder aktiviert werden (Flexibilität). Eine Person hat aus dieser Perspektive nicht nur *eine* "wahre" Identität, sondern verfügt über eine Vielzahl an gruppen-, rollen-, raum-, körper- oder tätigkeitsbezogenen Teil-Identitäten. (Döring, 1999, S. 225)

In der Informatik finden sowohl der rein mathematische Identitätsbegriff Verwendung – ein Standardkonzept in den meisten Programmiersprachen –, als auch der sozialpsychologische Aspekt dieses Begriffs. In dieser Arbeit ist ein Identitätsbegriff der Betrachtungsgegenstand, der von beiden Seiten inspiriert ist. Der mathematische Identitätsbegriff bildet die Grundlage: Anhand eines Identifikators ist eine Identität eindeutig bestimmbar. Dieser Identifikator wird angereichert durch eine beliebige Vielfalt an ergänzenden Attributen und ihre situationsbedingt eingeschränkte Verwendung.

Schon seit langem gibt es im Bereich der Betriebssysteme Konzepte für Identitäten. Möglichkeiten zur Authentifizierung und auch zur Kommunikation über Netzwerke machen solche Konzepte notwendig, sobald ein Mehr-Benutzer-System zum Einsatz kommt.

Der Bedarf nach system-immanenten Informationen zur Identität war dabei der Auslöser. So ist die Anwendung *finger* schon früh fester Bestandteil der Berkeley Software Distribution (ab 3.0 BSD) geworden und somit auch in fast allen Unix-Derivaten enthalten. Mittels *finger* lassen sich diverse Daten über einen Anwender ermitteln: sein Login-Name, sein voller Name, der Zeitpunkt seines Logins, sein Standort und weitere Daten. In Zeiten zunehmender Vernetzung und destruktiver Programme – Viren, Würmer und Trojaner (siehe Spafford (1989) und Bontchev (1998)) – stellt ein solcher Dienst allerdings ein Datenschutz- und damit auch ein Sicherheitsrisiko dar, erleichtern genauere Informationen über die Nutzer eines Systems doch immer auch das Vorgehen von Eindringlingen. Daher ist dieser Dienst mittlerweile zumindest von außerhalb eines lokalen Netzwerkes meist nicht erreichbar. Zudem entspricht die Systemorientierung der finger-Anwendung sicher ihrer ursprünglich zugedachten Funktion, aber die Einschränkung der Erweiterbarkeit auf eine ungeordnete Datei[2] ermöglicht keine Nutzung für ein darüber hinaus gehendes Identitätskonzept.

Auch im Rahmen der Verwendung von e-Mails und Newsgroups hat sich der Bedarf an der gezielten Verbreitung von Eigendarstellung und Identitätsdaten schon früh gezeigt. Die als Signatur bezeichneten abschließenden Zeilen einer E-Mail beinhaltet häufig die Daten, die der Autor dieser e-Mail seinen Empfänger oder Empfängern wissen lassen möchte. Dabei geht es teilweise um Kontaktdaten, aber auch andere Aspekte der Identität wie Wissensgebiete oder Hobbies. Der Vorteil liegt hier in der anfänglichen Kontrolle der ausgegebenen Daten, da dies ausschließlich auf Veranlassung des Autors geschehen kann. Vor einer nicht vorgesehene Weiterleitung sind diese Daten aber nicht geschützt. Zudem gibt es für diese Art der Kommunikation über Identitäten keine Form von Standard und ist in ihrer Art ausschließlich auf den Menschen ausgerichtet. Unter dem Aspekt der Praktikabilität ist der Zahl an Identitätsdaten auch die natürliche Schranke der menschlichen Aufmerksamkeit und des menschlichen Interesses als eine Begrenzung nach oben gegeben.

Diesem Vorbild der Newsgroup-Nutzer folgend unterstützen viele Foren-Systeme im World Wide Web von vornherein das Anlegen eines Identitäts-Profils. Neben diversen Identitäts- und Nutzungsdaten kann hier oft ein Bild als Stellvertreter und Wiedererkennungsmerkmal und emotionale Botschaft eingesetzt werden, für welches der Begriff "Avatar" geprägt wurde[3]. Trotz dieser Erweiterung und der Einheitlichkeit innerhalb eines Foren-Systems, ist auch hier weder ein Standard vorhanden, noch eine beliebige Erweiterbarkeit gegeben. Auch die den Foren verwandten, allerdings synchronen dezentralen Instant Messenger und ihre Möglichkeiten für Identitätsprofile weisen keinen Standard auf und leiden im Zweifelsfall an der eingeschränkten Erweiterbarkeit.

Im Rahmen des World Wide Webs hat sich eine Variante privater Homepages herausgebildet, die als Hauptmerkmal die Darstellung der eigenen Person aufweist. Die starke Verbreitung dieser persönlichen Homepages oder Web-Visitenkarten ist schon in der Form ermittelt worden, dass diese Variante als ein Standard in der Medienanalyse

[2] Jeder Nutzer kann eine Datei namens ".plan" anlegen und dort beliebige Einträge vornehmen.

[3] Neal Stephenson verwendete in seinem Science-Fiction-Roman "Snow Crash" 1992 den eigentlich aus dem Sanskrit stammenden Begriff "Avatar" zuerst in dieser Form. [Quelle: `http://de.wikipedia.org/wiki/Avatar_(Internet)`, Version vom 21.12.2004]

von Dillon und Gushrowsik (2000) genau betrachtet wird:

> Personal home pages on the web seem to have evolved very quickly into a standard form (Dillon und Gushrowsik, 2000, S. 205)

Dies ist ein Indiz für den starken Bedarf nach individueller Darstellung der eigenen Identität im virtuellen Raum. Auch hier gibt es natürlich keine Form semantischer Standards, da Umsetzung und Zielrichtung sehr variabel sind: Diese reicht von der schlichten Darstellung diverser Möglichkeiten zur Kontaktaufnahme, geht über ausführliche Beschreibung der Freizeitbeschäftigung bis hin zur Beschreibung des beruflichen Werdegangs und der eigenen Fähigkeiten. All dies sind Bereiche, von denen verschiedene Nutzer eine Darstellung wünschen. Mangels eines geeigneten Identitätssystems wählten sie als Plattform dieser Darstellungen das World Wide Web. Einen Nachteil neben der mangelnden Standardisierbarkeit und dem demzufolge bestehenden Mangel an automatischer Auswertbarkeit weist die Identitätsdarstellung im World Wide Web ebenfalls noch auf: Es gibt keine praktikable Möglichkeit, zu bestimmen, wer auf diese Daten zugreifen kann und wer nicht. Daten, die sich im World Wide Web befinden, sind im Allgemeinen für jeden einsehbar. Es existiert die Option, Daten im World Wide Web durch ein Passwort zu schützen. Diese stellt sich aber nur dann als sinnvoll dar, wenn nicht nur genau bekannt ist, welche Personen den Zugriff zu ihnen erhalten, sondern diesen auch im Vornherein die Zugriffsmöglichkeit bekannt gemacht wird, was die Verwendungsmöglichkeit dieser Option wiederum beschränkt.

Aufbauend einerseits auf Seiten des World Wide Webs, die sich der Darstellung der eigenen Identität widmen, andererseits aus dem Wunsch nach Optimierung der gebotenen Möglichkeiten, etwa durch Standardisierung in Darstellung und Vernetzung oder Nutzung semantischer Techniken auf Basis der *Web Ontology Language OWL*[4] zur Unterstützung automatischer Prozesse, wurde das Projekt *Friend of a Friend FOAF* [5] begründet – genau beschrieben durch Dumbill (2002). Ziel ist es, die Verbindung von Identitäten untereinander, wie auch die Darstellung ihrer Attribute in Form von Netzen persönlicher Beziehungen abzubilden. Neben der Standardisierung ist durch die dezentrale Vorhaltung der Identitätsdaten und Zugriffskontrollmechanismen auch ein höheres Maß an Datenkontrolle und -schutz möglich. Momentan setzen Programme, die das FOAF Projekt unterstützen, im Bereich der Dateneinbindung, der Darstellung und der Vernetzung auf den Techniken des World Wide Webs auf; die Konzeption ist allerdings nicht daran gebunden, wenngleich die Nutzung von Speicherplatz mit fester Adresse vielen Menschen bisher nur über die Anmiete von Webspace ermöglicht wird.

All dies sind Versuche, der eigenen Identität eine Möglichkeit zur Abbildung im digitalen Raum zu verschaffen. Dass teilweise die ursprüngliche Idee der verwendeten Technik und des umgesetzten Konzeptes nichts oder nur wenig mit dem Aspekt der Identitätsdarstellung zu tun hat, weist nur um so stärker auf den Bedarf vieler Nutzer des Internets nach einem Identitätsmanagement hin. Die Verschiedenheit der Umsetzungen deutet

[4] OWL Web Ontology Language – Reference, `http://www.w3.org/TR/owl-ref/`, Abruf 10.02.2004
[5] FOAF Vocabulary Specification (2004), `http://xmlns.com/foaf/0.1/`, Abruf 10.02.2004

aber auch an, wie unterschiedlich ausgeprägt die verschiedenen Bedürfnisse in diesem Bereich sind.

1.2. Zukünftige Entwicklung: Personalisierung und Individualisierung

Das Ausmaß der Personalisierung des Internets ist im Laufe der Jahre zunehmend angewachsen. Dabei war und ist die Personalisierung in lokalen Netzen – auch wenn sie auf Basis des Internet-Protokolls aufgebaut waren – ein Automatismus: Mittels der meist fest vergebenen IP-Nummer waren sämtliche Aktionen des Nutzers im Netz auf seine IP-Nummer und somit auf den von ihm verwendeten Computer zurückführbar und damit beispielsweise protokollierbar. Durch die zunehmende Öffnung des Internets über Internet-Dienst-Anbieter und die damit einhergehende Technik der dynamischen IP-Vergabe, spätestens aber auch durch Vorschriften des Datenschutzes bezüglich der persönlichen Daten der Kunden solcher Anbieter sind die meisten Nutzer erst einmal weitgehend anonym im Internet.

Für Anbieter von Diensten ist es oftmals notwendig, Teile der Identität eines Kommunikationspartners zu ermitteln: Sei es – im Falle eines Versandhandels – eine Lieferadresse, sei es – im Falle der Absicherung gegenüber Jugendlichen – eine Altersfreigabe oder seien es – im Falle von Entgeltforderungen – Konto- oder Kreditkartendaten. Auch die für das Marketing Verantwortlichen eines solchen Anbieters sind vielmals an einem registrierten und wiedererkennbaren Kunden und an dessen Kaufverhalten interessiert. Im Gegensatz zum Dienstnutzer ist es für den Dienstanbieter und mit ihm verbundene Instanzen meist interessanter, Informationen über die Nutzer an zentraler Stelle vorzuhalten, deren Kontrolle dem Dienstanbieter obliegt. Mit Hilfe der Datenerfassung durch freiwillige oder verpflichtend eingeforderte Dateneingabe, Wiedererkennung und Verfolgung durch Notwendigkeit zur Anmeldung, Speicherung und Auslesen von kleinen identifizierenden Dateien – sogenannten Cookies oder Web-Bugs – und Vergabe von identifizierenden Buchstaben-Zahlen-Kombinationen – den Session-IDs – innerhalb der Adresszeile des Web-Browsers lassen sich heute große Mengen an Daten erfassen, verknüpfen und systematisch auswerten.

Als Reaktion auf solche oftmals ungefragt oder aber vom Anwender ungewünscht erfolgenden Datenerfassungsmethoden werden Gegenmaßnahmen eingesetzt: Bei der Dateneingabe werden bewusst Falschangaben vorgenommen oder Cookies und die Quellen von Web-Bugs werden blockiert. Dies geschieht insbesondere bei Anbietern, bei denen die Daten nicht zwingend benötigt werden oder deren Notwendigkeit zur Erfassung dem Nutzer nicht einsichtig ist. Insgesamt hat das Vorgehen vieler Anbieter zumindest bei kritischen Nutzern des Internets ein starkes Misstrauen gegenüber diesen Techniken geweckt. So finden die Gegenmaßnahmen – zum Beispiel die Blockade von Cookies – auch dann leicht statt, wenn sie unbegründet wäre und vielmehr ein echter Vorteil dadurch ermöglicht würde. Ein Beispiel für einen solchen Vorteil ist die Vereinfachung und Individualisierung von Informationsangeboten durch personalisierte Darstellung.

Weniger kritische Nutzer und solche, die sich ein differenziertes Bild über die Vor- und

Nachteile dieser Techniken verschafft haben, erhalten für sie speziell zusammengestellte Inhalte, bekommen relevante Angebote unterbreitet oder haben die Möglichkeit, mit anderen Nutzern mit ähnlichen Interessen oder mit entsprechend ähnlichen Fähigkeiten in Kontakt zu treten. Dieser Nutzen gilt allerdings immer nur im eingeschränkten Bereich innerhalb eines Angebotes – sei es eine einzelne Website oder ein Verbund beschränkter von Websites.

Aber auch jenseits des World Wide Webs spielen Identitäten eine zunehmende Rolle. Dabei geht es nicht immer um eine der Wirklichkeit entsprechende Darstellung – bei aller Multiplizität sei sie hier als die *reale Identität* bezeichnet –, sondern oftmals auch um spielerische oder die reale Identität verschleiernde Pseudonyme und Rollen-Repräsentationen. Manche Dienste – Online-Spiele beispielsweise – fordern dies sogar explizit ein, während andere – zum Beispiel Instant Messenger – dies problemlos ermöglichen. Wesentlich ist bei beiden die Kontinuität der Identifizierbarkeit. Auch hier gilt die Beschränkung der Nutzbarkeit der Identitätsdaten auf einen Dienst. Dies ist beim Beispiel des Online-Spiels wohl auch grundsätzlich sinnvoll – die erschaffene Spiel-Identität hat schließlich oftmals wenig mit der realen Identität gemein –, beim Instant Messaging aber schon weniger gewünscht.

2. Identitätsdaten

Identitätsdaten sind variantenreich und individuell und beschränken sich nicht auf einen Kundendatensatz oder Anmeldedaten für Online-Dienste. Dies sind allerdings bisher die Hauptbereiche, in denen Identitätsdaten heute zum Einsatz kommen. Die Daten einer Identität müssen aber alle Aspekte einer solchen abbilden können. Diese Vielzahl an persönlichen und auch personengebundenen Daten kann viele Erleichterungen und Automatisierungen mit sich bringen, birgt aber auch Risiken und erschwert die Handhabung. So ist bei einer Betrachtung von Konzepten zu einem Identitätsmanagement immer auch der Blick zu richten auf die Frage nach der Kontrolle der Daten durch den Anwender, nach den Verwendungsmöglichkeiten durch zur Nutzung dieser Daten berechtigte Personen und nach Möglichkeiten des unvorhergesehenen Missbrauchs. Als noch entscheidenderes Kriterium für die Akzeptanz durch die Anwender ist aber sicherlich die Frage nach dem Mehraufwand: Kann ein Konzept, beziehungsweise seine Umsetzung in einer Anwendung gewisse Kriterien erfüllen, dass es der Anwender als vorteilhaft und nicht als belastend wertet? Es muss die Überlegung angestellt werden, nach welchem Verfahren die Daten strukturiert werden sollen. Und in wie weit ist es möglich, die notwendige Individualität von Identitätsdaten und deren Strukturen zu erhalten und trotzdem zu vermeiden, dass jeder Anwender seine eigenen Strukturen entwickeln und aufbauen muss?

2.1. Bisherige Konzepte und Standards

Bis zum heutigen Zeitpunkt gab es zwar vielerlei Bedarf, Identitätsdaten zu nutzen und zu speichern, aber es haben sich nur in geringer Weise Ansätze zur Standardisierung durchsetzen können. In zwei primären Bereichen spielen Identitätsdaten schon seit langer Zeit eine beträchtliche Rolle: Bei der Kontakthaltung zwischen Menschen und bei der Kontrolle von Menschen. Kontaktdaten decken den Bereich der Identitätsdaten ab, der es ermöglicht, mit einer Person über verschiedenste Arten in Kontakt zu treten: Adress- und Telefonbücher stehen hierfür als traditionelles Vorbild. Dabei reicht der Einsatz vom kleinen persönlichen Adressbuch bis zum aufwendigen Datenbank-basierten Kontaktmanagement eines Unternehmens. Datenerfassung und -verwaltung zu Kontrollzwecken basiert ebenfalls auf einer alten Tradition. Das Spektrum des Einsatzes umfasst hier staatliche Kontrollen statistischer Natur, Belege zur Erlangung einer Berechtigung im öffentlichen wie im privaten Sektor, aber auch Identifizierungsmöglichkeiten zu diversen Zwecken.

Für beide Zwecke sind bisher in jeder einzelnen Situation proprietäre Entwicklungen getätigt worden, was sich bei simplen Varianten – beispielsweise überschaubare Datensätze ohne Verschachtelung – als unproblematisch erweisen sollte und auch Datenmigrationen oder -übermittlungen ohne größere Komplikationen erlaubt. Einzelne proprietäre

Formate erlangen in ihrem Bereich aufgrund eines verordneten oder vereinbarten weitreichenden Einsatzes immerhin schon den Stand eines Quasi-Standards – sei es die Datenhaltung bei Meldebehörden oder die Erfassung der Daten durch das *Computer Assisted Passenger Prescreening System CAPPS* der Flugverkehrsunternehmen der Vereinigten Staaten von Amerika. Geht es allerdings um komplexere und um nicht von vornherein auf einen speziellen Zweck ausgerichtete Datenhaltung, erweisen sich bisherige Formate, die auch einen Austausch ermöglichen wollen, als ungeeignet: Jede Migration, Abänderung der Datenstruktur, Datenweiterleitung oder -verarbeitung außerhalb des Einsatzbereichs eines bestimmten Formates benötigt einen entsprechenden Transformationsvorgang.

2.1.1. E-Mail – electronic Mail

Der E-Mail Standard ist sicher kein Standard für ein Identitätsmanagement. Er sei hier aber erwähnt, da es sich um den ältesten und am weitesten verbreiteten digitalen Standard handelt, der sich primär auf Individuen und somit Identitäten bezieht. Nach Wikimedia Foundation (2005) wurde die *electronic Mail* im Forschungsunternehmen BBN erfunden. Ihr erster Einsatz datiert auf das Jahr 1971, um schnell eine der wichtigesten Anwendungen des damaligen ARPA-Nets zu werden. Zum ersten Mal standardisiert wurde eine erste Form des Mail Protokolls mit der Veröffentlichung des RFC 524.[1]

Aufbauend auf das E-Mail-Konzept entstanden in den Achtziger Jahren diverse Nachahmer. Zunächst geschlossene Netze wie *Compuserve* oder *America Online* vergaben Kennungen, die einen Nachrichtenaustausch innerhalb dieses Netzes ermöglichten. Dann gab es auch noch alternative Netze – beispielsweise Fido-Net, Bit-Net oder Z-Netz – , die mittels eines Store-and-Forward-Systems die Privatpersonen ansprachen, welche keine Möglichkeit hatten, direkt zum Internet eine Verbindung herzustellen. All diese Netze ermöglichten aber – entweder von Anfang an oder nachdem der entsprechende Bedarf festgestellt worden war –, per Gateway auch E-Mails an andere Teilnehmer des Internet zu senden und von diesen zu empfangen. Auf diese Weise etablierte sich der E-Mail-Standard zunehmend. Heute ist er aus dem Alltag vieler Menschen nicht mehr wegzudenken.

Neben dem reinen Aspekt der gegenseitigen Erreichbarkeit weist eine E-Mail-Adresse nur durch die – heute oft freie – Wahl der Kennung Individualität auf. Die meisten E-Mail-Systeme interpretieren ebenfalls zusätzliche Angaben des vollen Namens und der Organisation. Neben der eher seriösen Variante, den eigenen Namen in voller oder teilweise abgekürzter Form zu verwenden, versuchen viele Personen eine bestimmte Geisteshaltung, Zuneigung oder Gruppenzugehörigkeit durch die Wahl der richtigen Kennung auszudrücken. Genau hier ist aber auch schon die Grenze des E-Mail-Standards als Identitätskonzept erreicht: Weder lässt sich eine Namenwahl klar deuten – es sei denn, man ist mit dem weiteren Kontext des Anwenders vertraut –, noch lässt sich dieses durch automatische Prozesse sinnvoll auswerten. Eine E-Mail-Adresse bleibt als Identitätskonzept das, was sie von Anfang an auch nur sein sollte: Ein eindeutiges Identifizierungszeichen, um der damit verknüpften Identität Daten zukommen lassen zu können.

[1] J. White: Request for Comments 524 – A Proposed Mail Protocol, (1973), `http://www.faqs.org/rfcs/rfc524.html`

2.1.2. Der vCard Standard

Als einziger unabhängiger Standard für Identitätsmanagement im weitesten Sinne wurde bisher *vCard*[2] entwickelt. Dabei handelt es sich ursprünglich um den Teil der *X.500* Spezifikation, der Personen als Ressourcen behandelt. X.500 ist ein vom *CCITT*[3] im Jahre 1988 empfohlenes Rahmenwerk an Konzepten, Prozeduren und Protokollen für die Erfassung und Verwendung der Daten über die Ressourcen und Komponenten eines Netzwerkes mittels eines Verzeichnisses. Es ist von der *International Organization for Standardization* als Standard anerkannt worden[4]. Dabei ist neben den rein technischen Komponenten auch der Mensch als Nutzer mit einbezogen. Zur Erfassung, Vorhaltung oder auch Übermittlung in einem standardisierten Format von Daten über einen Menschen gibt es die beiden im X.500 Standard enthaltenen Spezifikation X.520 und X.521, auf deren Basis unter der Bezeichnung vCard als MIME[5] Typ eine eigene Entwicklung begann:

> The schema is based on the attributes for the person object defined in the X.520 and X.521 directory services recommendations. The schema has augmented the basic attributes defined in the X.500 series recommendation in order to provide for an electronic representation of the information commonly found on a paper business card. (Dawson und Howes, 1998)

Die ursprünglichen Standards X.520 und X.521 versahen einen Menschen als Resource im Netzwerk nur mit den für Netzwerk-Operationen notwendigsten Attributen: Name, Nachname, Telefon und Land. Nur für den an eine Organisation gebundenen Menschen waren auch Attribute wie seine Adresse, Fax, Titel, Postanschrift und Beschreibung vorgesehen.

Ganz klar zielte dieser Standard auf Erreichbarkeit und Kontaktaufnahme ab. Auch der vCard Standard verfolgte dieses Ziel, wenn auch schon weitere Attributkategorien Eingang fanden, darunter auch Geo-Positions-Informationen oder Binärdaten wie Passfoto, Firmenlogo, Namens-Aussprach-Hilfe in einem akustischen Format. Um dem Visitenkarten-Paradigma gerecht zu sein, und um zu den ursprünglichen Standards X.520 und X.521 die bestmögliche Abbildbarkeit gewährleisten zu können, ist der vCard Standard aber auch mit diversen Einschränkungen versehen worden, wie beispielsweise die Beschränkung nach Alden et al. (1996) auf 76 Zeichen pro Zeile.

Neben den an Kommunikations-Aspekten orientierten Attributen sind allerdings auch Attribute des Bereichs der Identifizierung mit berücksichtigt. Dies ist schon aus dem Grunde gegeben, da ja in Netzwerken das Thema Identifizierung und Authentifizierung eine wesentliche Rolle spielt. Daher ist der Aspekt Authentifizierung auch schon im X.500 Standard enthalten. Authentifizierung kann allerdings nicht nur für Menschen, sondern auch für Prozesse und andere Komponenten eines Netzwerkes notwendig sein, und die

[2] *vCard:* kurz für *virtual Card File*

[3] Comité Consultatif International Téléphonique et Télégraphique: ehemaliges Kommitee zur Ausarbeitung von technische Normen und Standards für die Telekommunikation

[4] ISO-Kennnummer 9594-1

[5] Multipurpose Internet Mail Extensions

Spezifizierung von Authentifizierungs-Standards kann eine recht komplexe Angelegenheit sein. Daher ist diesem Bereich eine eigene Teil-Spezifizierung mit der Nummer X.509 zugewiesen worden. In der vCard ist zu diesem Zweck die Einbindung eines öffentlichen Schlüssels (*Public Key*) nach dem PGP[6] Konzept möglich.

Insgesamt erfüllt der vCard Standard recht genau das, was der Name verspricht: Es handelt sich um die virtuelle, respektive elektronische Form einer Visitenkarte. Die fehlende Gebundenheit an ein Material, wie es bei der echten Visitenkarte gegeben ist, ermöglicht Erweiterungen von Film oder Ton; aber die Motivation hinter allen im Standard vorgesehenen Attributkategorien ist ausschließlich die Übertragung des Paradigmas einer Visitenkarte in die virtuelle Welt.

2.2. Identitätsdaten als komplexe Strukturen

Die wenigen bisherigen Standards und auch andere Konzepte ermöglichen nicht mehr als die Speicherung der eigenen Kennung, Kontaktdaten oder Daten zum Bezahlen. Um Identitäten in einer Weise zu unterstützen, die diese bisherigen Standards bei weitem übertrifft und die aufgeführten Mängel überwinden hilft, bedarf es keiner Kennungs- oder Kontaktdatenverwaltung, sondern der von Baier et al. (2003) eingeführten umfassenden *digitalen Identitäten-Infrastruktur*. Sobald aber auf Basis dieser Identitäten-Infrastruktur nicht nur die klassischen Kerndaten einer Identität als wesentlich angesehen werden, sondern sämtliche Daten eines Anwenders seiner Identität zugeordnet werden können sollen, ist auch ein umfassendes Strukturkonzept notwendig.

Viele Anwender verwalten schon heute eine Reihe von Daten, die auf Basis einer digitalen Identitäten-Infrastruktur zusammengefasst betrachtet werden könnten: Dazu zählen solche Dinge wie das digitale Adressbuch, ein Kalender, die Lesezeichen[7] für Web-Seiten, Verwaltungsdaten von Sammlungen (beispielsweise Fotos, Bücher oder Musik), Wunschlisten (beispielsweise bei Onlineshops), Lebensläufe, Ergebnisstände von Computerspielen und vieles mehr. All diese Daten liegen bisher in verschiedenen Strukturen vor, ohne Gesamtstruktur und ohne, dass sich automatisierte Querbezüge bei Bedarf herstellen ließen, obwohl es alles Daten sind, die sich der Identität des Anwenders zuordnen ließen. Diese fehlende Gesamtstruktur kann zur Folge unerwünschte Redundanzen und auch Inkonsistenzen mit sich bringen.

In der Vielfalt und auch der Vielförmigkeit dieser Daten liegt aber auch ein Integrationsproblem. Die aufgezählten Daten lassen sich in der vorliegenden Form nur schwer in einen sinnvollen Zusammenhang bringen. Notwendig ist hierbei eine zusätzliche Struktur, die – ohne die bestehenden Daten und ihren eventuell aktuellen Bezug zueinander zu verändern – diesen Daten eine Gesamtstruktur verleiht: eine *Identitätsdatengesamtstruktur*. Auf diese Weise ließen sich Daten wie bisher speichern. Zusätzlich ließen sich mittels dieser Struktur aber auch Zusammenhänge herstellen, die unabhängig von der ursprünglichen Gebundenheit der Daten bestünden.

[6] Asymmetrische Verschlüsselung nach Phil Zimmermann, siehe `http://www.pgp.com`
[7] oft als “Bookmarks” oder “Favoriten” bezeichnet

Derart ließen sich auch Identitätsdaten auf automatisierte Weise kontrolliert weitergeben. "Kontrolliert" in diesem Zusammenhang bedeutet die Möglichkeit für den Anwender, selbst zu entscheiden, an wen er welche Daten wann und zu welchen Bedingungen übermittelt. Dies ließe sich leicht bewerkstelligen, indem er Teile der Struktur mit entsprechenden Freigaben oder Einschränkungen versähe. Einen geeigneten Kommunikations- oder Kooperationsdienst vorausgesetzt, könnte der Anwender so bestimmten anderen Anwendern gezielt Daten über sich zukommen lassen, ohne sich selbst um die Zusammenstellung dieser Daten oder deren Übertragung kümmern zu müssen.

2.3. Nutzungsprobleme

Der Ansatz, einen Großteil der persönlichen Daten strukturell der Identität zuzuordnen, bietet ein großes Potenzial für die Personalisierung und Individualisierung in Datennetzen. Es bestehen allerdings auch grundsätzliche Probleme, die ein solches Konzept überwinden muss: Wenn individuelle und umfassende Strukturen die Identitätsdaten in einen Gesamtzusammenhang bringen sollen, so müssen diese Strukturen erstellt werden. Wenn die Strukturen die Kommunikation unterstützen sollen, muss die individuelle Struktur auf Empfängerseite bekannt sein, um dort von Vorteil sein zu können.

2.3.1. Zusatzaufwand für den Anwender

Wenn ein Anwender einen Großteil seiner vorhandenen und zukünftigen Daten in strukturierter Weise seiner Identität unterordnen will, so handelt es sich hier um eine große Vielfalt an Daten, die er zu berücksichtigen hat. Üblicherweise dient eine Struktur der Ordnung der Dinge eines Bereichs und bewirkt auf diese Weise eine Vereinheitlichung. Gerade innerhalb einer Gruppe kooperierender Personen ist eine vereinheitlichte Ordnung in der Kommunikation ein wesentliches Hilfsmittel zum gegenseitigen Verständnis und zur Vermeidung von Missverständnissen. Allerdings erfordert eine solche Ordnung schon selbst eine Verständigung beziehungsweise eine Übereinstimmung: Unabhängig davon, ob diese durch historische Entwicklung, durch gemeinsame Übereinkunft oder per Verordnung umgesetzt wird, übernimmt jeder einzelne einer solchen Gruppe ein strukturelles Konzept, das nicht zwangsläufig seinen eigenen Vorstellungen entspricht. Im Allgemeinen wird die Kompromissbereitschaft der Beteiligten aufgrund des Vorteils der einheitlichen Kommunikationsbasis hierbei keine großen Komplikationen entstehen lassen.

Die Daten der eigenen Identität – gerade wenn mehr als ein paar Merksmals- und Kontaktdaten in Betracht gezogen werden – stellen einen Themenbereich von sehr persönlicher Natur dar. Hier ist zu erwarten, dass die Kompromissbereitschaft, eine vorgegebene Datengesamtstruktur anzunehmen, recht gering sein dürfte. Vielmehr wird ein Anwender – zumindest in bestimmten Bereichen – auch seine eigenen Vorstellungen in der Ordnungsstruktur umgesetzt sehen wollen. Er wird seine Ordnung wünschen und nicht eine, die durch einen Standard festgelegt wurde. Neben der persönlichen Identität spielen hier auch kulturelle und lokale Aspekte eine Rolle. All dies lässt eine Standardisierung in diesem Bereich wenig sinnvoll erscheinen. Der Anwender wäre also selbst

gefordert, seine Struktur aufzubauen.

In der Situation der Erstellung der eigenen Identität in digitaler Form wird der einzelne Anwender allerdings in den wenigsten Fällen geneigt sein, auch noch umfassende abstrakte Strukturen zu definieren. Auch im Falle einer dadurch entstehenden möglichen Erleichterung oder erweiterter Möglichkeiten von Kommunikation und Kooperation bleibt die Problematik des Aufwands für den Einzelnen, alle Aspekte seiner Identität in einer Struktur abzubilden.

Die Wichtigkeit der Individualität der Ordnung unterscheidet sich bei den verschiedenen Bereichen der eigenen Identität. Ebenso unterschiedlich ist auch die Bewertung des Einzelnen, welcher Bereich genau einer individuell festzulegenden Ordnung zu folgen hat und welcher Bereich mit einer generischen, vielfach gebräuchlichen oder auf andere Weise vorgegebenen Struktur auskommt. In solchen Bereichen, deren individuelle Ordnung von geringer Wichtigkeit ist, wäre der Anwender einer schon bestehenden Struktur, die ihm die Aufgabe der eigenen Erstellung abnimmt, viel eher aufgeschlossen. Dies ist sogar in vielen Bereichen der eigenen Identität denkbar, unter der Voraussetzung, dass die vorgegebene Struktur in etwa den eigenen Vorstellungen entspricht. Dies gilt insbesondere in der Vielzahl der Fälle, für die offenbar ist, dass sie eine auf die Nutzerbasis bezogene große Anzahl anderer Anwender ebenfalls auf die gleiche oder sehr ähnliche Weise durchführen müsste – als Beispiel sei hier nur die postalische Adresse genannt. Allerdings verhindern die unterschiedlichen individuellen Einschätzungen die Option für eine festen Standard.

Wenn also auf der einen Seite jeder Anwender jeden einzelnen Bereich mit einer entsprechenden Struktur versehen müsste und es auf der anderen Seite das Ziel ist, möglichst viele und komplexe Daten und Datenstrukturen in die digitale Identität mit einzubeziehen, würde einem solchen Identitätskonzept von vorherein eine Anerkennung durch eine große Anwenderbasis versagt bleiben. Es muss hier die Möglichkeit für eine Hybrid-Lösung gefunden werden, die beide Optionen – die Übernahme von vorgegebenen Standards und die Eigenerstellung – ermöglicht und miteinander integriert.

2.3.2. Kommunikation auf Basis individueller Strukturen

Ein Nachteil der bestehenden minimalen Konzepte für Identitäten, wie sie in Abschnitt 2.1 aufgeführt sind, weist gegenüber den angedachten Erweiterungen auch einen erheblichen Vorteil auf: Die Einheitlichkeit der Struktur eines Standards führt dazu, dass jede Anwendung, die diesen Standard vollständig und korrekt mit einbindet, problemlos auf die entsprechenden Daten zugreifen kann. Beispielsweise kann ein Nutzer die mit seiner Anwendung erstellte vCard einem anderen Nutzer zukommen lassen, der diese dann mit einer völlig anderen Anwendung auslesen oder weiterverarbeiten kann.

Auch bei einer umfangreichen Struktur für Identitätsdaten soll der Zweck über den persönlichen Wunsch nach Ordnung der eigenen Identitätsaspekte hinausgehen und die Möglichkeiten zur Kommunikation und zur Kooperation durch eine solche Ordnung unterstützen und erweitern. Es müssen demnach auch für diesen Bereich Wege gefunden werden, die eine – bei aller Individualität – durch eine solche Struktur unterstützte Kommunikation und Kooperation ermöglichen. Selbst dann, wenn sich in die Struktur

– eventuell auch nur in Teilen – Standards einbinden ließen, wäre die Problematik auch nur dann leicht gelöst, wenn es einen einzigen Standard gäbe. Aufgrund der Überlegungen aus Abschnitt 2.3.1 ist dies aber weder gewünscht noch spiegelt es die Individualität wider. Dementsprechend müsste es mehrere Standards nebeneinander geben. Um hier einen Austausch zu ermöglichen, muss ein Übersetzungsdienst vorhanden sein, der Daten einer Struktur in eine andere Struktur wandelt – vorausgesetzt, sie behandeln das gleiche Thema. Und selbst, wenn ein Anwender seine ganz individuelle Struktur entwickelt, muss ihm die Möglichkeit an die Hand gegeben werden, eine Übersetzung zu erstellen, die ihm von da an ermöglicht, auf Basis seiner individuellen Struktur mit anderen Anwendern basierend auf deren Struktur in Kommunikation zu treten.

Soweit es sich nicht vermeiden lässt, sollte der Anwender dabei selbst sich möglichst nicht mehr oder nur in sehr geringem Umfang um diese Struktur kümmern müssen. Vielmehr ist es wünschenswert, dass alle die Strukturangelegenheiten koordinierenden Vorgänge vor dem Anwender verborgen bleiben. Dies wäre der Rahmen für das oben genannte entscheidende Kriterium der Akzeptanz durch die Anwender: das Verbergen dieser Vorgänge im weitest möglichen Maß und die Belastung des Anwenders durch zusätzlichen Aufwand nur dann, wenn ihm seine individuelle Struktur wichtiger ist, als die Vermeidung eines solchen Mehraufwands.

3. Konzepte zur individuellen Organisation von Identitätsdaten

Als Basis eines Strukturkonzepts für eine digitale Identitäten-Infrastruktur bietet sich das Ontologien-Paradigma an. Es soll zunächst näher untersucht werden, inwieweit dies möglich und angemessen ist. Als zweiter Aspekt des Strukturkonzeptes muss die Möglichkeit gegeben sein, Einzeldaten mittels Datenstrukturspezifizierung zu komplexen Daten zusammen zu führen. Um auf dieser Basis auch Kommunikation und Kooperation zu ermöglichen, ist zudem ein Konzept zur Transformation von diesen – durch die Individualisierung unterschiedliche – Strukturen ineinander unabdingbar. Hierbei ist es wichtig, ein Transformations- beziehungsweise Übersetzungskonzept zur Anwendung zu bringen, welches sich leicht mit dem gewählten Strukturkonzept für die digitalen Identitäten kombinieren lässt.

3.1. Ontologien als semantische Basis

Der Begriff der Ontologie stammt aus der Philosophie. Beizeiten schlicht als *Lehre des Seins* bezeichnet, beschäftigt sich die Ontologie mit dem Sein und den Dingen als den Seienden und deren Merkmalen. Durch Quine (1979) erfuhr der Begriff Ontologie eine Neueinführung insbesondere in der Wendung *ontological commitment*: Eine Aussage, welche durch die Existenz eines Gegenstands die Existenz eines anderen Gegenstands impliziert. Sie verknüpft also zwei Gegenstände mit einer Verbindung der Existenzabhägigkeit. Der Begriff wurde in der Informatik adaptiert und wurde durch Gruber (1993) folgendermaßen definiert:

> A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. ... An ontology is an explicit specification of a conceptualization. (Gruber, 1993, S. 1)

Eine Ontologie stellt demnach eine Struktur dar, welche die Gegenstände eines Betrachtungsgebietes ordnet und in Beziehung zueinander bringt. Zudem wird jeder in diese Struktur eingebundene Gegenstand mit seinen Merkmalen betrachtet und dadurch konkret spezifiziert.

Das Phänomen der meist strikten Begrenzung von Ontologien in ihrem Bezug auf Domänen zur Einschränkung der Komplexität soll zum Anlass genommen werden, die Daten einer Identität ebenfalls zu begrenzen und gegebenenfalls zu gruppieren, um sie nach dem Divide-et-Impera-Prinzip mit einer besseren Handhabbarkeit zu versehen.

Der aus der Linguistik stammende Begriff der *Semantik* hat die Fragen nach Sinn und Bedeutung einfacher und komplexer Begriffe und Zeichen zum Thema. Frege (1892) führt

den Sinn eines komplexen Zeichens auf den Sinn seiner einfachen Bestandteile zurück. Die Bedeutung eines Zeichens lässt sich mittels einer Definition darstellen: Ein Zeichen wird auf eine allgemeinere Form zurückgeführt und durch eingrenzende Attribute spezifiziert.

3.1.1. Spezifizierende Metadaten

Wesentlich für jegliche Handhabung von Daten ist eine Information über ihre Bedeutung. In der gesprochenen Sprache erfahren die einzelnen Worte ihre Bedeutung aus dem Kontext der Sätze, unterstützt durch reale Gegenstände und einen sozialen Kontext, auf welche Bezug genommen werden kann. In anderen Darstellungsformen muss dieser mangelnde Kontext von Daten entsprechend kompensiert werden. In der Schriftform einer natürlichen Sprache werden einzelne Daten in mehreren Ebenen kontextbezogen eingebettet, wobei einzelne Einheiten, zu einem direkten Zusammenhang zusammengefasst, wieder als neue Einheit aufgegriffen in einen größeren Zusammenhang gestellt werden. Dies geschieht auf inhaltlicher Ebene wie auch auf formeller grammatikalischer Ebene. Für eine unproblematische Nutzung von Daten durch Automaten sind entsprechende explizite Einbettungen nötig, die sich automatisch auswerten verarbeiten lassen. Zu diesem Zweck gibt es das Konzept der Metadaten:

> Metadata is structured data about data. (Dublin Core Metadata Initiative, 2004)
>
> Metadata is machine understandable information about web resources or other things. (Berners-Lee, 2000)

Metadaten sind allgemein bekannt, beispielsweise im Bereich von Bibliotheken. Dort lassen sich mittels Katalogdaten Bücher und andere Medien schnell identifizieren und lokalisieren. Die Katalogdaten beinhalten kurze identifizierende Aspekte eines Mediums, darunter standardisierte Daten – wie zum Beispiel den Namen des Erschaffers oder den Titel des Mediums –, darunter ebenso proprietäre Daten – beispielsweise eine intern verwendete Signatur. Welche Katalogdaten erfasst beziehungsweise vergeben und verwendet werden, wird durch einen zuvor festgelegten Satz an Metadaten definiert. Metadaten weisen einem Datum also eine Kategorie oder einen Typ zu. Durch diese Typisierung versehen sie das beschriebene Datum mit einer Reihe obligatorischer und optionaler Attribute. Wenn ein Gegenstand mit dem Metadatum *Buch* versehen wird, so sind beispielsweise Seitenzahl und Verlag obligatorisch, während Autor und ISB-Nummer zwar sehr häufig existent sind, aber nicht in zwangsläufig allen Fällen, die die Kategorie Buch umfasst. Die Nennung des Begriffs Buch erzeugt als gesprochenes oder geschriebenes Wort eine in vielen Aspekten klare Vorstellung beim Rezipienten des Wortes. Zu allen Attributen, die diese Vorstellung ausmachen, können entsprechende Meta-Attribute beziehungsweise Attributkategorien des Metadatums eingesetzt werden. In der Praxis werden nur solche Attribute verwendet, die eine leichte Individuierbarkeit ermöglichen und somit leicht die Identität eines Datums verifizieren lassen. Solche Attributkategorien werden dann explizit der Kategorie des Metadatums beigefügt, alle anderen Attributkategorien – beim Buch wären das zum Beispiel die räumlichen Ausmaße – bleiben als implizit verborgen.

Eine solche Kategorisierung kommt in den natürlichen Sprachen so direkt selten vor. Metadaten erscheinen hier als impliziter Teil der Daten, ohne dass zu den Daten eine sprachliche Trennung vorgenommen wird. So werden Daten und Metadaten durch einfache Satzkonstrukte gebildet: "A ist der Autor von X." verbindet nicht nur *A* mit *X*, sondern versieht *A* auch mit dem Metadatum *Autor*. Das bedeutet allerdings nicht, dass *Autor* selbst nicht mit einem Metadatum versehen werden kann: "Autoren gehören zur Gruppe der Kreativen." stattet das Metadatum *Autor* mit einem Metadatum aus. Selbst der ganze Satz lässt sich in Form der Klasse der Aussagen einem Metadatum zuordnen. Die in der natürlichen Sprache nicht klar vorhandene Trennung von den verschiedenen Formen an Metadaten muss bei der Umsetzung zur automatischen Nutzung berücksichtigt werden.

Auch die allgemein gehaltenen Definitionen zu Metadaten sagen selbst nichts darüber aus, von welcher Art die maschinen-verständlichen Informationen sind. Neben der genannten Typisierung lassen sich zwei Varianten ausmachen: Die erste ist die strukturierte Typisierung und als solche nur eine Variation der genannten Metadaten. Einem Datum werden hierbei nicht nur ungeordnet Metadaten zugewiesen, sondern es setzt sich zusammen aus Teildaten, deren Struktur auf Metadatenebene vorgegeben wird. Eine Postadresse setzt sich beispielsweise aus Name, Straße und Ort zusammen; diese wiederum aus Vorname und Nachname, Straßenname und Hausnummer, Postleitzahl und Ortsname. Erst die letztgenannten sind dann mit Werten zu belegen. Um hervorzuheben, dass in diesem Fall nicht nur ein einzelnes Metadatum gemeint ist, sondern eine komplexe Struktur, sei für diese Variante der Begriff *Metadatenkomplex* im weiteren Verlauf verwendet.

Die zweite Variante nimmt auf ein anderes Konzept Bezug. Es handelt sich um die Einbettung in einen größeren Zusammenhang. Es handelt sich dabei um strukturelle Generalisierungen in die eine Richtung und Spezialisierungen in die andere.

3.1.2. Semantische Strukturen

> One requirement for a linguistic representation formalism is that the information can be structured in a way that captures generalizations over linguistic objects in order to minimize redundancy (Barg, 1996, S. 3)

Wie in obigem Beipspiel schon angedeutet, kann ein Metadatum selbst mit einem Metadatum versehen werden. Dies ändert aber nicht den Status als Metadatum. Vielmehr besteht die Funktion des übergeordneten Metadatums darin, eine Kategorisierung vorzunehmen. Mit diesem übergeordneten Metadatum verhält es sich aber gleichfalls so, dass es sich kategorisieren lässt und somit einem Metadatum höherer Ordnung angehört. Dies lässt sich nicht unendlich fortführen, da mehrfach aufeinander folgende Generalisierungen letztendlich zu einem Metadatum führt, dass als einzige Attributkategorie den Bezeichner aufweist: Das Metadatum selbst lässt sich als "Ding" oder "Objekt" bezeichnen und es umfasst alle Dinge beziehungsweise alle Daten[1]. Ein Datum ist somit seinen eigentlichen

[1] "Datum" entsprechend zu "Metadatum"

oder direkten Metadaten zugewiesen wie auch denen sämtlicher übergeordneter Metadaten. Dabei ist es ausreichend, nur die Zugehörigkeit zum direkten Metadatum explizit darzustellen, wenn die Verbindung der Metadaten untereinander ebenfalls gegeben ist.

Eine Generalisierung bedeutet immer eine Verallgemeinerung von zumindest potenziell existenten Spezialfällen. Demnach kann es in einem komplexen Metadatensystem in jeder Kategorie mehrere Vertreter als Spezialisierung der Generalisierung geben. Bei einem solchen System von mehreren kaskadierenden Ebenen von jeweils einer Reihe von Spezialfällen handelt es sich um eine eine Taxonomie. Der Begriff *Taxonomie* beschreibt die Einteilung von Dingen in Taxa[2]. (Wikimedia Foundation, 2004a)

Taxonomien sind – wenn auch noch nicht unter dieser Bezeichnung – schon in der Antike von Aristoteles beschrieben worden. In seiner Kategorienschrift (Aristoteles, 1998) legt er die Konzepte dar, die einen Begriffsraum durch Kategorisierung strukturieren können. Ab dem 18. Jahrhundert, mit Einführung durch den schwedischen Wissenschaftler Linnaeus (1735) in seinem Werk *Systema Naturae* erfuhr das Konzept der Taxonomien eine Verbreitung zuerst in der Biologie, später auch in der Linguistik. Gerade der Bezug zur Sprache und damit auch zu Denkstrukturen über die Linguistik machte taxonomische Systematiken zu einem passenden Konzept zum möglichen Einsatz in Automaten-Systemen.

Das Konzept der Taxonomie ist daher auch in der Informatik vertraut: Unter anderem baut darauf die Objektorientierte Programmierung auf. Neben der Strukturierung tritt als wesentlicher Effekt die Vermeidung von Redundanzen auf: Ein spezialisiertes Metadatum wird gegenüber seiner generalisierten Form ausschließlich durch hinzukommende Attributkategorien – also assoziierte Metadaten für beschreibende Attribute – hervorgehoben.

Taxonomische und andere klassifizierende Ordnungssysteme erfuhren in der Vergangenheit auch einige Kritik: Ludwig Wittgenstein kritisiert das hierarchische Konzept, denen solche Systemen unterworfen sind, welches der Komplexität nicht entsprechen kann. Jede Eingrenzung oder taxonomische Festlegung sei damit Willkür und von der jeweiligen Perspektive abhängig:

> Wir sehen ein kompliziertes Netz aus Ähnlichkeiten, die einander übergreifen und kreuzen. ... Kannst Du die Grenze angeben? Nein. Du kannst welche ziehen: denn es sind noch keine gezogen. (Wittgenstein, 1953, S. 278)

Michel Foucault wirft den Taxonomien die grundsätzliche Raum-Zeit-Gebundenheit hierarchischer Denkmuster vor. Zu diesem Zweck stellt er eine imaginäre Taxonomie eines anderen Kulturkreises vor und stellt folgendes fest:

> Bei dem Erstaunen über diese Taxonomie erreicht man mit einem Sprung, was in dieser Aufzählung uns als der exotische Zauber eines anderen Denkens bezeichnet wird - die Grenze unseres Denkens: die schiere Unmöglichkeit, *das* zu denken.

[2]Taxa, Singular: Taxon, alt-griechisch für: Gruppen

> Was ist eigentlich für uns unmöglich zu denken? Um welche Unmöglichkeit handelt es sich? Jeder dieser eigenartigen Rubriken kann man einen präzisen Sinn und einen bestimmbaren Inhalt geben. (Foucault, 1971, S. 17)

Am Beispiel historischer Zeitepochen macht er die Schwierigkeit für viele Ordnungssysteme deutlich, geeignete Kategorien zur Klassifizierung zu finden:

> Der Status der Diskontinuitäten ist für die Geschichte im allgemeinen nicht leicht herzustellen, wahrscheinlich noch schwieriger ist das jedoch für die Geschichte des Denkens möglich. Wollen wir eine Trennungslinie ziehen? Jede Grenze ist vielleicht nur ein willkürlicher Einschnitt in ein unendlich bewegtes Ganzes. (Foucault, 1971, S. 82)

Unter diesen Gesichtspunkten muss die Verwendung von Taxonomien als die Verwendung eines Instruments mit eingeschränkten und auch einschränkenden Möglichkeiten betrachtet werden. Innerhalb dieser gesetzten Grenzen und Einschränkungen und mit Hinnahme eines Verlustes impliziter Zusammenhänge kann ihr Einsatz aber trotz dieser Kritik sehr nützlich sein. Taxonomien ermöglichen doch – in Bezug auf jegliche Form von Daten – Wege zur Automatisierung und Speicherung. Mit ihrer Hilfe lässt sich für die Gegenstände eines Betrachtungsbereichs eine semantische Struktur zur Verfügung stellen.

Ordnungssysteme weisen aber keinesfalls immer zwangsläufig eine reine Baumstruktur auf: Je größer der betrachtete Gegenstands- oder Datenbereich ist, desto eher kann das Phänomen einer Mehrfach-Kategorisierung aufkommen. Das Entscheidende ist dabei die Näherungsweise des Betrachters gegenüber einem Gegenstand. Diese kann für jeden Betrachter unterschiedlich sein, weshalb ein Ordnungssystem mit einem Anspruch auf Allgemeingültigkeit diese verschiedenen Betrachtungen wiederspiegeln muss. Desweiteren kann dieser – unter Berücksichtigung der Multiplizität der Identität eines einzelnen Betrachters – sich schon auf verschiedene Weisen einem Gegenstand nähern. Schon das würde die Möglichkeit einer Mehrfach-Kategorisierung erfordern. Beispielsweise kann ein einzelner Anwender ein großes Sportereignis einerseits einer Kategorie “Sport” zuordnen, andererseits gleichzeitig als Medienphänomen einer Kategorie des Medienbereichs. Auch innerhalb der Programmierparadigmen ist das Konzept der Mehrfachvererbung bekannt. Durch Mehrfach-Kategorisierungen ist es möglich, die Attribute zweier (oder noch mehr) Kategorien in eine Spezialisierung einfließen zu lassen, während bei den generalisierten Kategorien weiterhin nur die jeweiligen Attributkategorien mit expliziten Attributen Eingang finden.

Für die weitere Betrachtung soll die Mehrfachvererbung an dieser Stelle keine weitere Beachtung finden, da sie am konzeptionellen Vorgehen keinen wesentlichen Unterschied herbeiführt, während sie auf der anderen Seite eine Struktur leicht unüberschaubar werden lässt. Aus diesem Grund sei der Begriff Taxonomie beibehalten.

3.1.3. Flexibilität durch Modularisierung

Nun sind geeignete Konzepte für die Strukturierung von Identitätsdaten innerhalb einer digitalen Identitäten-Infrastruktur gegeben. Es bleibt aber ein Spagat zu bewältigen:

Zu einem Teil wird ein Anwender geeignet vorgegebene Strukturen – Datenstrukturen und semantische Strukturen – akzeptieren; zu einem anderen Anteil jedoch wird er aber viel Wert auf eine individuelle Ordnung legen. Da die entsprechende Gewichtung und die Ausrichtung – je nach Interessenlage – bei jedem Anwender unterschiedlich ausfällt, muss ein Ordnungskonzept die verschiedenen Bereiche der Ordnungsstruktur getrennt betrachten. Für jeden einzelnen Bereich sollte es dem Anwender nun frei stehen, auf eine vorhandene Ordnung zurückzugreifen und diese zu seiner eigenen zu machen, oder auf seiner individuellen Ordnung zu beharren, diese selbst zu formulieren und für seine Zwecke zu nutzen.

Während die Erstellung einer Struktur, egal ob sie sich als beliebte und vielgenutzte oder als sehr individuelle Struktur erweist, konzeptionell in den beiden vorhergehenden Abschnitten behandelt wurde und nur die technische Umsetzung noch eine genaue Betrachtung erfordert, muss für die Integration schon vorhandener Strukturen ein geeigneter Weg gefunden werden. Wiederverwendbarkeit ist ein aus dem Bereich der Programmiersprachen sehr wohl bekanntes Konzept. Dieses Software-Paradigma, in dem es die Ausnahme und nicht die Regel darstellt, Software von Grund auf neu zu entwickeln, stellte schon McIllroy (1969) vor. Er bezeichnete das damals bisherige Vorgehen der Software-Entwicklung als überholt und forderte eine für sonstige industrielle Massenproduktion übliche Standardisierung, die den mehrfachen Einsatz von derart standardisierten Komponenten in verschiedenen Zusammenhängen ohne weitere Anpassung ermöglichen sollte:

> The idea of subassemblies carries over directly and is well exploited. The idea of interchangeable parts corresponds roughly to our term modularity, and is fitfully respected. The idea of machine tools has an analogue in assembly programs and compilers. Yet this fragile analogy is belied when seek for analogues of other tangible symbols of mass production. There do not exist manufacturers of standard parts, much less catalogues of standard parts. One may not order parts to individual specifications of size, ruggedness, speed. (McIllroy, 1969, S. 139)

Von Anfang an wurde dabei der Fokus nicht nur auf den Programm-Code gerichtet, sondern ebenso Prozesse und Strukturen berücksichtigt – das schließt natürlich Daten- und Metadatenstrukturen mit ein.

In dieser zunächst widersprüchlichen Situation – Standardisierung und Individualität – kann der Ansatz der Wiederverwendbarkeit weiterhelfen, indem die eigenhändige Erstellung eigener Metadatenstrukturen zur Ausnahme gemacht wird. Für all die Bereiche, bei denen ein Anwender mit einer generischen, vielfach gebräuchlichen oder auf andere Weise schon vorhandenen Struktur nach eigenem Ermessen auskommt, sollte es ihm erspart bleiben, sich um eine eigene Erstellung kümmern zu müssen. Voraussetzung dafür ist natürlich, dass sich jemand anders schon um die Erstellung dieser Struktur gekümmert hat – sei dies ein Anwender oder ein Dienstleister. Für den Fall, dass für einen Bereich sogar mehrere Alternativen an Datenstrukturen existieren, sollte es einem Anwender auch möglich sein, aus diesen die eine auszuwählen, die seinen Vorstellungen am ehesten entspricht.

Um die Flexibilität für den Anwender zu erhalten, gelte dies alles aber immer nur für Teilbereiche einer Taxonomie, nicht für die gesamte Taxonomie. Genau wie in der Software-Entwicklung muss die Struktur für Identitätsdaten also in einzelne Komponenten aufgeteilt sein: die *Metadatenstrukturkomponenten*, bei der es sich jeweils um eine Teiltaxonomie – also eine semantische Teilstruktur – oder einen Metadatenkomplex oder eine Verbindung von beiden handelt. Um eine Weitergabe und damit auch die Wiederverwendbarkeit an anderer Stelle zu ermöglichen, müssen die Teiltaxonomien von der Gesamttaxonomie als Kopie abtrennbar sein. Allerdings gibt es semantische Verbünde innerhalb einer solchen Gesamttaxonomie, die auch in der kopierten Teiltaxonomie enthalten sein müssen oder verloren gehen. Ebenso untrennbar sind die Metadatenkomplexe. Eine Hausnummer beispielsweise wäre getrennt vom Straßennamen wenig sinnvoll und sollte daher mit jenem immer verbunden bleiben.

Neben der Modularisierung der Identitätsdatengesamtstruktur in Metadatenstrukturkomponenten muss eine Möglichkeit gegeben sein, eine Metadatenstrukturkomponente öffentlich bereitzustellen. Eine solche öffentliche Metadatenstrukturkomponente müsste einem anderen Anwender die Möglichkeit geben, sie zu untersuchen – und zwar dahingehend, ob sie seinen Vorstellungen entspricht. Bei Bedarf sollte sie sich in die bestehende Datengesamtstruktur seiner Identität integrieren lassen.

Da jeder Anwender seine eigene erstellen kann, unterliegt die Erstellung der Metadatenstrukturkomponenten grundsätzlich keiner zentralen Kontrolle. Es kann bei der Integration einer fremden Metadatenstrukturkomponente also auch nicht ausgeschlossen werden, dass es zu einer Überlagerung mit der schon bestehenden Identitätsdatenstruktur kommt – sei sie semantischer oder syntaktischer Natur. Dabei geht es nicht so sehr darum, dass ein Anwender für einen Bereich zwei verschiedene Metadatenstrukturkomponenten einsetzt, sondern um den Fall, dass es – aufgrund verschiedener Sichtweisen auf Aspekte einer Identität – in einem oder mehreren Elementen einer Metadatenstrukturkomponente zu einem redundanten Aufkommen von Metadaten gegenüber den Metadaten einer anderen Metadatenstrukturkomponente kommt, also zu Überschneidungen. Ein solches Nebeneinander ist allerdings nicht problematisch; vielmehr bietet es eine echte Erweiterung für ein Datum der Identität, wenn ein zusätzlicher Metadatenkomplex mit eingebunden ist. Es muss aber von vornherein erkannt werden, dass dieses Datum bzw. sein Metadatum schon vorliegt. Im anderen Fall – nämlich dass aufgrund eines redundant vorhandenen Metadatums auch die Daten doppelt vorhanden sind – besteht im Falle einer Modifizierung nur eines der beiden Daten die Gefahr von Inkonsistenz. Um den Fall einer zufälligen Übereinstimmung abzuwenden, ist aber eine Rückfrage an den Anwender unabdingbar.

Im Verbund mit den Daten bildet die über die Metadatenkomplexe so verbundene Identitätsdatengesamtstruktur aufgrund dieser Überlagerungen in ihrer integrierten Form nicht mehr zwangsläufig eine Baumstruktur, auch wenn Mehrfachvererbung innerhalb der Struktur vermieden wird. Durch dieses Vorgehen wird Wittgensteins Kritik aus Abschnitt 3.1.2 entsprochen und eine mehrdimensionale Sichtweise auf die Daten ermöglicht.

3.2. Das Verständigungsproblem

Im Falle der Nutzung einer vorhandenen Metadatenstrukturkomponente sollte es dem Anwender auch möglich sein, aus verschiedenen Alternativen zu wählen – so diese denn vorhanden sind. Das ermöglicht die Nutzung derjenigen, die seinen Vorstellungen am ehesten entspricht. Im Falle der individuellen Formulierung der eigenen Struktur kann aber auch in vielen Fällen davon ausgegangen werden, dass die so erstellte Struktur auch bei anderen Anwendern Zustimmung findet – sei es, weil für diesen Bereich bisher noch keine oder noch keine so detailierte Struktur vorhanden war, sei es, dass die bisher vorfindbaren Strukturen den Vorstellungen auch weiterer Anwender nicht entsprechen. Es bleibt nun noch der geeignete Ort für das Speichern der Metadatenstrukturkomponenten zu finden. Zudem entstehen aus diesem Konzept zwei Probleme: Zuerst stellt sich die Frage, wie und an welcher Stelle eine neu hinzukommende Metadatenstrukturkomponente in die bisher vorliegende Identitätsdatengesamtstruktur eines Anwenders integriert werden kann. Die zweite Frage ergibt sich im Zusammenhang mit Kommunikation und Kooperation auf Basis integrierter Metadatenstrukturkomponenten: Nutzen zwei Kommunikationspartner zu einem Thema zwei verschiedene Metadatenstrukturkomponenten, wie sollen dann auf deren Basis die Kommunikationsmöglichkeiten verbessert werden können?

3.2.1. Standard contra Individualität

Weder Teiltaxonomien – eingesetzt als semantisches, offenes Ordnungskonzept für die Metadaten – noch Metadatenkomplexe – die syntaktischen Verknüpfungen einzelner Metadaten – dürften also einheitlich sein, wenn sie die Individualität der Identitäten und ihrer Daten berücksichtigen wollen. Auch der persönlichen Entwicklung des einzelnen Anwenders muss insofern Rechnung getragen werden, dass zu keinem Zeitpunkt mit Sicherheit behauptet werden kann, dass eine vorliegende Identitätsdatengesamtstruktur alle erforderlichen Aspekte auch für die zukünftige Entwicklung des Anwenders abbildet.

Ohne Einheitlichkeit und mit dieser Möglichkeit zu anhaltender Veränderung wären an eine zentrale Verwaltungs- oder Ordnungsinstanz für diese Identitätsstrukturen recht hohe Ansprüche zu stellen: Unter der Voraussetzung, dass die umrissene digitale Identitäten-Infrastruktur eine gewisse Verbreitung findet, sollten nicht nur eine ausreichende Berücksichtigung des Datenschutzes und der Datensicherheit gewährleistet sein, sondern auch die Fragen nach Zugriffszeiten, Speicherkapazität und Ausfallsicherheit gestellt werden.

Die Alternative ist eine lokale dezentrale Vorhaltung. Sie erfordert entsprechend lokal die Möglichkeiten zur Verwaltung und Berarbeitung der Identitätsdatenstrukturen. Datenschutz und -sicherheit wären Aufgabe der lokalen Anwendung und des Anwenders. Zugriffszeiten und Speicherkapazitäten sind keine Probleme mehr.

Auf Basis von Dezentralität kann es auf der Ebene einer semantischen und syntaktischen Ordnung keinen Standard geben, da hier die standardisierende Instanz fehlt. Es kann aber sehr wohl solche Bereiche – innerhalb der Vielzahl der Bereiche, die eine Identität ausmacht – geben, in denen eine bestimmte Ordnungsstruktur sich einer gewissen Verbreitung erfreut. Wesentlich für diese Nutzung durch andere Anwender wäre natür-

lich der freie Zugriff auf die bestehende Metadatenstrukturkomponente und Möglichkeit, diese losgelöst von den anderen sie umgebenden Bereichen der Identitätsdatengesamtstruktur aufzugreifen und in anderem Zusammenhang verwenden zu können. Zu diesem Zweck müssen die Metadatenstrukturkomponenten so gestaltet werden, dass sie die in Abschnitt 3.1.3 erläuterte Wiederverwendbarkeit für andere Identitäten ermöglichen.

Neben standardisierten Schnittstellen, welche die Wiederverwendbarkeit verlangt, ist ebenso die Übermittlung inhaltlicher Informationen notwendig: Wenn eine Anwendung eine fremde Metadatenstrukturkomponente in eine bestehende lokale Identitätsdatengesamtstruktur integrieren soll, so ist dies von der technischen Seite her an jeder Stelle möglich; aber auch bei der Betrachtung der Semantik bieten sich diverse Möglichkeiten an. Es wäre denkbar, auf Basis von bestimmten Parametern – im einfachsten Falle des Namens des Wurzelelements beziehungsweise des obersten Elements der Metadatenstrukturkomponente – einen geeigneten Integrationspunkt in der bisherigen Identitätsdatengesamtstruktur zu finden und dem Anwender als beste Möglichkeit vorzuschlagen. Die Entscheidung, welches die passende Möglichkeit wäre, ist aber wieder Teil der Entscheidungen, die zur individuellen Zusammenstellung der lokalen Ordnungsstruktur gehören, die jeder Anwender selbst fällen muss.

3.2.2. Die Wandlung vom Individuellen zum Allgemeingut

Wenn zwei Personen auf Basis der digitalen Identitäten-Infrastruktur eine Kommunikation aufbauen wollen, sie aber in einem zu behandelnden Themenbereich unterschiedliche Metadatenstrukturkomponenten in ihrer jeweiligen Identitätsdatengesamtstruktur besitzen, so ist eine Unterstützung durch die bisher dargestellte Identitäten-Infrastruktur nicht möglich.

Ein Mechanismus, der sich auf gleiche oder ähnliche Bezeichnungen innerhalb der beiden Metadatenstrukturkomponenten stützt, lieferte schon gewisse Indizien für eine entsprechende Abbildung: Die Möglichkeiten, Homonyme[3] und Synonyme[4] aufzuspüren, ist aufgrund der semantischen Strukturen denkbar. Wenn aber verschiedene Identitäten auch unterschiedliche strukturelle Zusammenhänge nutzten oder wenn Sprachunterschiede die ganze Identitätsdatengesamtstruktur durchzögen und somit Felder mit gleicher Bedeutung der Erkennung durch einen solchen Mechanismus entzögen, würde dieser von vornherein unbrauchbar sein. Es gibt für diese Angelegenheit Mapping-Algorithmen, welche mit probalistischen Methoden auf Struktur-, Syntax- und Semantik-Ebene den größeren Teil solcher Abbildungen korrekt bewältigen könnten. Laut den Erfindern des sehr wirkungsvollen Abbild-Algorithmus *Cupid*, Madhavan et al. (2001), kann auf die Intervention eines Menschen aber auch in einem solchen Fall nicht verzichtet werden. Halevy (2003) äußert sich wie folgt dazu:

> Unfortunately, complete automation of the generation of semantic mappings is unlikely to be possible. The crux of the problem is that writing a correct mapping requires an understanding of the underlying semantics of the

[3] Bezeichnungen mit mehreren unterschiedlichen Bedeutungen
[4] Mehrere unterschiedliche Bezeichnungen mit der gleichen Bedeutung

> schemas being matched. While the schema and data instances provide clues on their intended semantics, they do not fully capture the full semantics. (Halevy, 2003, S. 3)

Daher seien hier die Möglichkeiten zum teilautomatisierten Abbildverfahren ganz außer Acht gelassen.

Wenn nun also ein Mensch diese Abbildung von Hand erstellen muss, so sollte dies nur einmal geschehen müssen. Für zwei sich entsprechende Metadatenstrukturkomponenten muss eine Regel oder vielmehr ein Satz an Regeln aufgestellt werden, die alle Abbild-Möglichkeiten von der ursprünglichen zur Ziel-Metadatenstrukturkomponente beinhaltet. Für die Gegenrichtung ist bei Bedarf das gleiche zu tun. Diese Regeln müssen nach einmaligem Erstellen gespeichert werden und immer dann aufgerufen werden, wenn dieses Paar von Metadatenstrukturkomponenten erneut in Verbindung steht.

Für eine nähere Untersuchung der Erfordernisse, die an ein solches Abbild gestellt werden ist ein genauer Blick auf eine minimale Form der Metadatenstrukturkomponente notwendig: Eine solche bestehe aus einer Reihe einzelner Felder, welche zwei Angaben aufweisen: Das erste – das Namens-Element – versieht ein Feld mit einem innerhalb der Metadatenstrukturkomponente eindeutigen Kennzeichner. Eindeutigkeit über die Grenzen der Metadatenstrukturkomponente sei an dieser Stelle noch nicht weiter beleuchtet. Dies sei später mit einem ebenfalls eindeutigen Schlüssel zu ermöglichen, der der Metadatenstrukturkomponente insgesamt zugewiesen werden muss. Das zweite Element sei das Verweis-Element, welches die Verbindung zum Eltern-Feld darstellt. Eine Ausnahme bildet das oberste Eltern-Feld. Sein Verweis stellt eine Verbindung zu einem Feld außerhalb der Metadatenstrukturkomponente dar. Dieses Feld trage den Namen “Extern”. Mit Hilfe dieser beiden Elemente – Namens-Element und Verweis-Element – lässt sich ein solches Feld als Teil einer Metadatenstrukturkomponente darstellen.

Diese beiden Elemente sind es auch, die ein Abbild-Mechanismus betrachten muss. Das Abbild zwischen zwei Namens-Elementen entspricht einem Eintrag in ein Wörterbuch – Voraussetzung bei diesem Vergleich sei, dass das Wörterbuch keine Mehrdeutigkeiten enthält: Zu einem Wort auf der Ursprungs-Seite findet sich ein Wort auf der Ziel-Seite. Worte auf der Ursprungs-Seite, für die es keine Entsprechung auf der Ziel-Seite gibt, werden ignoriert. Wenn für zwei oder mehrere Namens-Elemente auf der Ursprungs-Seite nur ein Namens-Element auf der Ziel-Seite existiert, so werde eine geeignete Konjunktion der beiden Elemente gefunden: Das konjugierte Namens-Element bilde sich beispielsweise aus dem ursprünglich zweiten Namens-Element, einem Freizeichen als Trenner und dem ursprünglich ersten Namens-Element. Reihenfolge und die Art der als Trenner fungierenden zusätzlichen Elemente müssen in den Regeln eindeutig beschrieben sein.

Das Abbild zwischen zwei Verweis-Elementen wird ganz ähnlich gebildet. Voraussetzung ist die schon zuvor durchgeführte Abbildung der Namens-Elemente; dann kann jedes Verweis-Element, welches auch immer nur ein Verweis auf ein Namens-Element ist, auf einen Verweis zu einen beliebigen Namen, der aus der Abbildung kommen muss, abgebildet werden. Zeigt der ursprüngliche Verweis eines Feldes auf ein Feld, für das kein Abbild existiert, weil es keine Entsprechung bei der Abbildung der Namens-Elemente gibt, so wird dieser umgelenkt auf das entsprechende Elternelement. Existiert auch dieses nicht,

so wird diese Umlenkung rekursiv so oft durchgeführt, bis der Verweis ein existierendes Feld zum Ziel hat. Im äußersten Fall handelt es sich dabei um das oben eingeführte Feld "Extern". Durch diese Abbildung ist eine komplette Transformation einer Metadatenstrukturkomponente möglich, ohne dass die Gefahr aufkommt, die Anbindung eines Feldes zu verlieren.

Mittels eines solchen Regelsatzes lässt sich nun die ursprüngliche Metadatenstrukturkomponente kapseln, so dass jeder Zugriff von der anderen Seite auf die transformierte Metadatenstrukturkomponente erfolgt. Diese andere Seite müsste nun nicht berücksichtigen, dass hier eine andersartige Metadatenstrukturkomponente vorliegt. Der Zugriff könnte genauso erfolgen, als handele es sich um zwei gleiche Metadatenstrukturkomponenten. Da die Abbildung aber nicht bijektiv ist – das Ignorieren von Feldern ohne Entsprechung und die Konjugation detailierterer Felder zu umfassenden Feldern sprechen dagegen –, lässt sie sich nur in eine Richtung durchführen. Eine Abbildung in die Gegenrichtung muss ebenfalls durch geeignete Regeln beschrieben werden. Sind auf diese Weise beide Metadatenstrukturkomponenten gegenüber der jeweils anderen im Zugriff gekapselt, so ist die Kommunikation auf Basis dieser beiden eigentlich verschiedenen Metadatenstrukturkomponenten durch Transparenz gekennzeichnet. Der Anwender und seine auf diese Komponenten und die zugehörigen Transformationen aufbauende Kommunikation nutzten diese Strukturen, als folge sie einem einheitlichen Standard.

3.2.3. Flexibilisierung der Transformationen

Auf der einen Seite lässt sich beim Erstellen die Intervention durch den Anwender nicht vermeiden. Auf der anderen Seite lässt sich aber annehmen, dass bestimmte Metadatenstrukturkomponenten, die sich ein Anwender ausgedacht hat, auch durch andere Anwender Zuspruch erführen. Viele Anwender wären froh, wenn ihnen schon eine oder mehrere Metadatenstrukturkomponenten zur Auswahl ständen. Soweit ein Anwender nicht sehr auf seine individuellen Ordnungskriterien fixiert ist, sollte die Wahrscheinlicheit recht groß sein, dass er eine vorgegebene, auf einen Bereich ausgerichtete Komponente für sich annimmt. Die endgültige Entscheidung wird aber von keinem System gefällt, sondern vom Anwender selbst. Er wählt den Weg der Übernahme einer schon vorhandenen Metadatenstrukturkomponente oder er entscheidet sich für die Eigenerstellung.

Wenn ein Anwender den Vorteil einer vorgegebenen Metadatenstrukturkomponente nutzt, möchte er aber kaum mit der Frage belangt werden, wie seine neu integrierte Komponente sich zu einer thematisch entsprechenden, aber strukturell unterschiedlichen Komponente eines Kommunikationspartners transformieren lässt. Mit der Annahme, dass es zu einem Themenbereich jeweils nur eine relativ – zur Zahl der Anwender – geringe Anzahl an verschiedenen Metadatenstrukturkomponenten gibt, lässt sich zum Ausdruck bringen, dass der überwiegende Teil der Anwender von der Erstellung der Metadatenstrukturkomponenten durch andere profitiert. Trotzdem wäre eine – wenn auch überschaubare – Vielfalt an Metadatenstrukturkomponenten gegeben. Die Zahl der notwendigen Transformationsregelsätze – auch unter der Annahme, dass diese nicht

symmetrisch seien – bliebe ebenfalls überschaubar[5].

Die Transformationsregelsätze müssen also – genau wie die Metadatenstrukturkomponenten selbst – im öffentlichen Zugriff stehen, um die einmal erstellten Transformationsregeln für das Paar zweier Metadatenstrukturkomponenten für jede Kommunikation, die auf diesen beiden Komponenten basiert, zugänglich zu machen. Wünschenswert wäre es, wenn die Verbreitung von Metadatenstrukturkomponenten – soweit sie ein Thema behandelt, welches bekanntermaßen schon durch eine andere Komponente bedient wird – sogar gekoppelt ist an die Verbreitung von entsprechenden Transformationsregelsätzen: Eine – als Alternative zu einer oder mehreren bestehenden – neu erstellte Metadatenstrukturkomponente sollte demnach mindestens einen Transformationsregelsatz zu einer schon vorhandenen liefern. Durch den Mechanismus der Kapselung ist ein weiterer Transformationsregelsatz zu einer weiteren alternativen Komponente nicht unbedingt nötig. Aus den schon existierenden Transformationsregelsätzen wird ein solcher dann automatisch erzeugt. Sollte es aber einen relevanten Informationsverlust geben – etwa durch die Auswirkung der in Abschnitt 3.2.2 erwähnten ignorierten Felder –, so könnte sich ein weiterer Transformationsregelsatz zu einer zusätzlichen alternativen Metadatenstrukturkomponente als notwendig erweisen.

Auch hier erweist sich die Teilung der Identitätsdatengesamtstruktur in kleine Metadatenstrukturkomponenten als sehr nützliches Konzept, hält es den Aufwand zur Erstellung selbst mehrerer Transformationsregelsätze verhältnismäßig gering. Trotz allem bedeutet die Entscheidung, seine eigene Metadatenstrukturkomponente zu erstellen, hierdurch einen zusätzlichen Mehraufwand. Die private Metadatenstrukturkomponente in einem durch schon weit verbreitete Komponenten versehenen Themenbereich wird so eher zum Randphänomen. Metadatenstrukturkomponenten werden daher mehr durch Notwendigkeit – zum Beispiel lokale Strukturen und Sprachen oder noch nicht mit Komponenten abgedeckte Spezialgebiete – erstellt werden, als denn durch individuellen Eigensinn. Laut Marshall und Shipman (2003) ist eine Beschränkung in dieser Hinsicht auch durchaus zu begrüßen:

> Users tend to catalog information resources with an implicit (and often inconsistent) sense of how they themselves intend to use the resource. They are neither trained to apply authority control, nor do they have a larger sense of the institutional purpose of a given resource. (Marshall und Shipman, 2003, S. 6)

Dieses grundsätzliche Problem für *ungeschulte*[6] Anwender ist aber gerade in der Domäne der eigenen Identität am wenigsten gegeben, ist doch der Anwender mit seiner Identität am ehesten vertraut. Wenn der Domäne der Identität aber ein Großteil der durch den Anwender verwalteten Daten untergeordnet ist, sind auch andere Domänen betroffen, für die es wiederum durchaus geeignetere Domänenexperten gibt. In diesem Fall ist die Verbreitung einer durch einen solchen Domänenexperten entwickelten Metadatenstrukturkomponente wünschenswert.

[5] Sie ist potenziell steigend, nämlich $x(x-1)$.

[6] hier als Übersetzung von "neither trained" gemeint

4. Techniken für eine durch Teilontologien unterstützte Identitäten-Infrastruktur

In diesem Kapitel werden Techniken zur Umsetzung der in Kapitel 3 entwickelten Konzepte erörtert und entsprechend ausgewählt. In vielen Bereichen der Informatik gibt es eine unüberschaubare Vielzahl an Ideen, Konzepten und Implementierungen. Die Frage nach der Entscheidung für eine bestimmte Technik ist zuerst einmal geprägt durch das Wissen und die Recherchemöglichkeiten des Entscheiders. Wesentliche Aspekte, welche die Recherche einschränken helfen und damit den Aufwand sinnvoll reduzieren können, sollten Dinge wie Offenheit, Vereinheitlichung, Verbreitung und Unabhängigkeit sein. Offenheit – im Sinne offen oder frei in Zugriff, Erreichbarkeit und Einsehbarkeit – ist eine für die Wissenschaft sehr wesentliche Angelegenheit, da nur so neue Konzepte, die auf bestehenden Techniken und Konzepten fußen, für jeden vollständig nachvollziehbar sein können. Diesem Konzept – im Bereich wissenschaftlichen Publikationswesens als grundlegend vorhanden – muss auch eine eingesetzte Technik oder Implementierung soweit möglich folgen.

Verfahren der Vereinheitlichung weisen im Allgemeinen eine Verknüpfung vorhandener Ideen und strenge Prüfungskriterien auf. Dies gilt zumindest für anerkannte Standardisierungseinrichtungen – beispielsweise die *ISO*[1] und die *IETF*[2] –, aber auch für Gremien von weniger offiziellem Charakter: Das *World Wide Web Consortium W3C* publiziert nur Empfehlungen, welche allerdings wie Standards gehandhabt werden. Die Standardisierungsverfahren sorgen für eine detailierte Spezifikation und umfassende Dokumentation.

Eine Technik, welche einen Standardisierungsprozess durchlaufen hat, ist aufgrund der Zustimmung der Beteiligten an diesem Prozess – zum großen Teil Vertreter von Wissenschaftseinrichtungen und Wirtschaftsunternehmen – sicher ein guter Kandidat für eine Technik mit hoher Anerkennung und weiter Verbreitung. Allerdings ist dies nicht zwangsläufig gegeben, was den weiteren Aspekt der Beachtung hinzufügt: eine vorhandene oder – bei aktuellen Themen – absehbar ausreichende Verbreitung, zumindest bei Personen und Gruppen, die in den Bereich der betrachteten Technik involviert sind. Neben dem leichteren Verständnis bei Verwendung verbreiteter Techniken und dem daraus resultierenden erleichterten Zugang zum behandelten Thema profitiert eine Arbeit bei Verwendung einer verbreiteten Technik auch durch die Vielzahl an unterstützenden Ergebnissen und Entwicklungen, die für diese Technik existieren.

Als letzter Aspekt für eine geeignete Technik sei die Unabhägigkeit als für die Praxis

[1] Zur Standardentwicklung bei der ISO siehe:
`http://www.iso.org/iso/en/aboutiso/introduction/index.html`, Version vom 26.01.2005

[2] Die Spezifikation des Standardisierungsprozesses der Internet Engineering Task Force (Oktober 1996) findet sich unter `http://www.ietf.org/rfc/rfc2026.txt`

wünschenswert bezeichnet. Für Ideen und Konzepte theoretischer oder allgemeiner Natur ist dies schon von vornherein gegeben. Umgesetzte Techniken und Implementierungen sind aber oftmals an eine technische Plattform gebunden, was die Nachvollziehbarkeit wie auch einen testweisen Einsatz erschwert. Plattformunabhägigkeit ist daher ein wichtiger Aspekt.

Die gleichzeitige Forderung aller vier genannten Aspekte ist vermutlich nicht in allen Entscheidungsgebieten zu erreichen. Für diese Bereiche muss daher eine Abwägung durchgeführt werden, welche den Grad der Erfüllung der einzelnen Aspekte in Betracht zieht und gleichzeitig die Zielsetzung des Einsatzes einer Technik berücksichtigt.

4.1. Metadaten und Strukturierung

Während es sich beim Bereich der Metadaten schon um ein wesentliches Thema der Informatik handelt, sind die Aspekte Struktur und Strukturierung eines ihrer Kernthemen. Für beide Bereich existiert eine Fülle an Techniken. Der in dieser Arbeit formulierte Ansatz zur digitalen Identitäten-Infrastruktur fordert eine freie Auswahl durch den Anwender bezüglich der Entscheidung, was er zu seinen Identitätsdaten zurechnen möchte und was nicht. Um von Vornherein eine Eingrenzung bei den zu untersuchenden Techniken zu ermöglichen, seien nur solche Techniken betrachtet, die die entsprechende Flexibilität unterstützen, zumindest aber nicht behindern.

4.1.1. Metadaten-Standards

Metadaten sind zwar Daten über alle möglichen Ressourcen, aber sie differenzieren sich auf zweierlei Weise: Sie können simpel sein oder Metadatenkomplexe, sie können Allgemeingut sein – im Sinne von "allgemein bekannt und genutzt" – oder die private Eigenentwicklung eines Einzelnen, die zwar öffentlich zugänglich, aber für nur wenige interessant ist. In beiden Differenzierungen existieren Zwischengrade: Die Komplexität kann in der Zahl der Metadaten und deren Verschachtelung verschiedene Grade aufweisen; der Spielraum zwischen Allgemeingut und Eigenentwicklung ist fließend. Im Bereich komplexer Metadaten, welche gleichzeitig Allgemeingut sind, ist es geboten, möglicherweise existierende Standards daraufhin zu untersuchen, ob sie geeignet sind, als Grundlage für Metadatenkomplexe von Ontologien genutzt zu werden. Für die anderen Formen von Metadaten ist ein vorliegender Standard entweder – aufgrund mangelnder Verbreitung – unwahrscheinlich oder – aufgrund mangelnder Komplexität – nicht notwendig. In diesen Fällen muss mittels der Ontologiesprache selbst der Metadatenkomplex modelliert werden.

Viele der Metadatenkonzepte sind auf bestimmte Gebiete spezialisiert, insbesondere angesiedelt im Bereich von Bibliotheken und Medien. Dies begrenzt die Auswahlmöglichkeiten. Hier bleibt zu untersuchen, ob geeignete Standards existieren, die entweder für sich flexibel genug für den gesamten Attributsbereich eines Identitätsmanagements sind oder sich entsprechend erweitern lassen – möglichst so, dass der Standard-konforme Anteil weiterhin den Standard einhalten kann. Neben der Wiederverwendung bestehender

Konzepte und Standards, würde deren Nutzung auch die Interoperabilität mit anderen Anwendungen fördern, welche ebenfalls einen dieser Standards nutzen.

Einen Überblick über die gegebenen Metadaten-Konzepte vermittelt Jeffery (1998). Der bei Bibliotheken recht verbreitete Metadaten-Standard MARC[3] leide demnach unter einer Vielzahl an Varianten, die keine einfache Handhabung zulassen. Spezielle Metadatenstandards gibt es beispielsweise in den Bereichen der Weltraumforschung, der Teilchenphysik, der Produktinformationen und der Medizin. Als Metadaten-Standard aus dem Bereich des Internet nennt er die Dublin Core Initiative. Sie hebt sich durch Umfang und Ausrichtung von den anderen Metadaten-Konzepten ab. Auf Dublin Core greifen auch mehrere andere Initiativen zurück; darunter insbesondere auch die Open Archives Initiative[4].

1995 fand in Dublin (Ohio) der erste *Dublin Core Series Workshop* unter dem Titel "Metadata Workshop: The Essential Elements of Network Object Description" statt. Aus diesem ging die *Dublin Core Metadata Initiative* (DCMI)[5] Organisation hervor. Die fortschreitende Entwicklung der Informationstechnologie und die zunehmende Vernetzung waren der Antriebsmotor, das Konzept der Metadaten für das Internet zu definieren und einen offenen Standard für Metadaten festzulegen. Das Ziel der DCMI war und ist es erst einmal, bestehende Such- und Indexierungs-Methoden für web-basierte Metadaten zu erweitern. Zu diesem Zweck wurde das *Dublin Core Metadata Element Set* als Vorschlag für einen Metadaten-Standard veröffentlicht. Im Jahr 2003 wurde es von der ISO als Standard anerkannt[6]. Entgegen den sehr umfassenden Vorgaben anderer Metadaten-Konzepte, nimmt sich der Ansatz der DCMI recht bescheiden aus: Man beschränkt sich auf einen überschaubaren Satz beschreibender Informationen zu diversen Formen von Medien im weiteren Sinne. Unter dem Gesichtspunkt, dass die verschiedenen Formen von Medien den Zugang zu annähernd allen Informationen ermöglichen, ist das Konzept der DCMI recht umfassend. Doch die DCMI schließt bewusst auch physische Gegenstände nicht aus, ja führt sie sogar beispielhaft an[7]. Im Sinne des RDF (siehe Abschnitt 4.1.3.2) sind also durchaus Ressourcen gemeint.

Die Art der Metadaten der DCMI ermöglicht aber nur einen eingeschränkten Fokus auf die betrachteten Objekte. Metadaten-Informationen, die nicht in diesen festen Standard eingebunden werden können – beispielsweise die Funktionsweise eines Apparates –, lassen sich höchstens in die Description einbinden. Ein solches Vorgehen verhindert aber wieder die Maschinenlesbarkeit. Trotz dieses Umstands erfreut sich der Dublin Core Metadata Standard gerade bei Institutionen wie Bibliotheken, Museen und Archiven, aber auch bei Websites, deren Organisationen eine semantische Erweiterung benötigen, breiter Verwendung. Gegenüber den viel umfassenderen Metadatenstandards von Seiten der Bibliotheken wie MARC weist der Dublin Core aber einen sehr viel geringeren Umfang auf,

[3] "MAchine-Readable Cataloging", siehe `http://www.lcweb.loc.gov/marc/`, Version vom 13.01.2005

[4] Open Archives Initiative Protocol for Metadata Harvesting, siehe `http://www.openarchives.org/OAI/openarchivesprotocol.html`, Version vom 10.12.2004

[5] siehe `http://dublincore.org`

[6] registriert als ISO-Standard 15836

[7] Dublin Core Metadata Initiative: Using Dublin Core – The Elements, (26.08.2003), `http://dublincore.org/documents/2003/08/26/usageguide/elements.shtml`

was seine Handhabarkeit stark vereinfacht. Der Name "Dublin Core" ist auch genau so zu verstehen, dass nur die wesentlichen, nämlich die *Kern*-Metadaten Berücksichtigung gefunden haben. Der Dublin Core basiert – wie viele andere Metadaten-Standards auch – auf der *Extended Markup Language XML*. Dies ermöglicht grundsätzlich die flexible Einbindung in andere Formen an Metadaten, soweit diese ebenfalls in XML vorliegen. Die DCMI bietet hier eine geeignete Basis.

Das Dublin Core Metadata Element Set bildet ein Vokabular zur Darstellung der wesentlichen Informationseigenschaften. Einige Elemente wie *Description* – für eine allgemeine Beschreibung des betrachteten Gegenstands –, *Coverage* – zur Angabe von Gültigkeit oder Bezug im Sinne von Zeit, Ort oder Sachgebiet – oder *Rights* – zur Einbindung von Hinweisen auf Aspekte der Urheberschaft – erfordern keine weitere Spezifizierung ihrer Form und sind daher auch nicht ohne weiteres maschinenlesbar und somit eher nur durch Menschen verwendbar. Neben diesen bieten andere Elemente Möglichkeiten zur Ausstattung mit allgemeinen Metadaten: *Title* für den Namen des betrachteten Objekts, *Creator* für seinen Autoren beziehungsweise Erfinder, *Publisher* für die Person, die für die Zugänglichkeit oder die Verfügbarkeit verantwortlich ist, *Contributor* für eine an der Objekterstellung mit beteiligte Person, *Type* für die Art des Objekts und *Date* für das Datum der Veröffentlichung oder des zur Verfügung Stellens. Weitere Metadaten-Elemente sind in ihrem Format nicht vom Dublin Core Metadata Element Set festgelegt, da es durch andere Standards bestimmt wird; diese anderen Standards gelten aber nur für bestimmte Bereiche von Objekten – für andere Bereiche gibt es andere Standards oder Festlegungen, für wieder andere gar keine. Zu diesen Elementen zählen *Format* für die Art der Erscheinung – zum Beispiel MIME[8] oder Zeichensatz im digitalen Bereich –, der eindeutige *Identifier* – beispielsweise *ISBN*[9] oder *URI*[10] – und *Language* zur Angabe der Sprache bei Objekten mit Text. Eine letzte Sparte an Metadaten Elementen setzt die einzelnen Objekte in Verbindung zueinander: *Relation* für verwandte oder zusammengehörige Objekte, *Subject* zur Bestimmung des thematischen Gebietes oder einer Klassifizierung, *Source* zur Angabe der Objekte, die als Quelle für das beschriebene Objekt dienten. Detailiert wird das Metadata Set vom Dublin Core Metadata Initiative (2004) beschrieben.

Die erstgenannten Elementformen sind zwar flexibel, aber aufgrund ihres Mangels an Maschinenlesbarkeit wenig geeignet für den Einsatz in einer digitalen Identitäten-Infrastruktur mit automatischer Ermittlung zusammengehöriger Betrachtungsgegenstände zwischen Kommunikations- und Kooperationspartnern. Die anderen Elemente eignen sich besser für diesen Einsatz. Unkompliziert in der Verwendung ist die zweitgenannte Gruppe, weil sie natürliche und damit allgemein verständliche Grenzen setzt für Titel und Personennamen. Für die Nutzung von Elementen der dritten Gruppe muss der jeweilige Bezugsrahmen bekannt sein: Die standardisierte ISBN ist in der Verwendung beispielsweise unproblematischer als Produktseriennummern von Elektrogeräten, die unter der Vielfalt und des fehlenden Bekanntheitsgrads leiden. Die vierte Gruppe

[8] Multimedia Internet Message Extensions
[9] Internationale Standard Buchnummer
[10] Unified Resource Identifier

deckt nicht den Objekte spezifizierenden Aspekt der Metadaten ab, sondern nimmt sich der semantischen Struktur an. Für diesen Bereich sind die genannten drei Elemente aber sehr knapp. Allerdings gibt es für diese Aufgabe ja gerade die Ontologiesprachen.

4.1.2. Taxonomien und Ontologiesprachen

Im Laufe der letzten 30 Jahre hat der Bereich der Taxonomien und Ontologie-Sprachen eine fortlaufende Entwicklung erfahren. Es gab Zeiten, in denen die mit ihnen im Beginn stark verbundene *Künstlichen Intelligenz* in Miskredit kam. Prominentes Beispiel ist hier das Gedankenexperiment *Das Chinesische Zimmer* des Philosophen Searle (1986). In dieser Zeit ließ auch das Interesse an den Vorläufern der Ontologie-Sprachen nach. Aber die fehlende Überschau- und Handhabbarkeit heutiger Datenmengen hat diesem Konzept wieder neuen Auftrieb gegeben und zu Popularität verholfen. Es ist zu untersuchen, welches der aktuellen Konzepte und ihrer Umsetzungen die gesetzten Anforderungen erfüllen kann.

Das in den fünfziger und sechziger Jahren aufgekommene Paradigma der Künstlichen Intelligenz schuf eine Verbindung zwischen Taxonomien und Automaten. 1968 stellte Quillian die Frage

> What sort of representational format can permit the "meanings" of words to be stored, so that humanlike use of this meanings is possible? (Quillian, 1968, S. 227)

Aus seinen Überlegungen entwickelt er das *Semantic Memory Model*:

> A word's full concept is defined in the memory model to be all the nodes that can be reached by an exhaustive tracing process, originating at its initial, patriarchical type node, together with the total sum of relationships among these nodes (Quillian, 1968, S. 238)

Die Arten der *Beziehungen zwischen diesen Knotenpunkten* differenziert Quillian im weiteren Verlauf unter anderen in "subclass-to-superclass pointer" und "adjectively or adverbially modification pointer". Verknüpfungen, die einzelne Knotenpunkte in eine disjunktive oder konjunktive Relation bringen, hält er ebenfalls für sehr wichtig. Er wendet sich aber gegen vordefinierte hierarchische Gesamtstrukturen, da eine Betrachtung des Modells von jedem Knotenpunkt aus seinen Anfang nehmen könne. So sei jedes Wort als Knotenpunkt eine Art oberste Superklasse für alle anderen Knotenpunkte, sobald dieser Knotenpunkt in den Mittelpunkt der aktuellen Betrachtung gestellt würde.

Aufbauend auf Quillians Überlegungen entstand eine Reihe an daran anknüpfenden Entwürfen, die heute unter dem Begriff *Konzeptsprachen* und ihrer Vorläufer betrachtet werden: Die *Conceptual Dependency* von Schank (1972) stellte eine Theorie für das Verstehen natürlicher Sprachen auf Basis Semantischer Netze dar. Die Theorie der *prozedural semantischen Netze* nach Levesque (1979) beinhaltet schon alle in den Abschnitten 3.1.1 und 3.1.2 geforderten Konzepte: Durch Aggregation lassen sich Metadatenkomplexe bilden; Generalisierungen erzeugen die semantische Struktur und die Klassifizierung – der

heute übliche Begriff ist Instanziierung – bewirkt die Verknüpfung zu den Identitätsdaten (siehe Mylopoulos, 1980).

Diesen Ansatz greift die Idee der Ontologiesprachen auf und setzt ihn in die Praxis um: Sie verknüpfen einzelne Knoten durch Subklassen-Superklassen-Beziehungen – dem von Quillian (1968) benannten *ISA-Link*, welcher einfach für die englische Wendung "is a" steht – und Attribut-Beziehungen, welche zusätzliche strukturierte Elemente darstellen, die einen Knoten spezifizieren oder erweitern können. Auch Quillians Kritik an den hierarchischen Gesamtstrukturen begegnet diese Idee, und zwar durch die Domänenorientierung: Eine Ontologie beschränkt sich dabei auf Inhalte, die zu einem bestimmten Themen- oder Betrachtungsbereich – der Domäne – gehören. Dies ist sicher nicht ganz die Flexibilität, die Quillian fordert, ermöglicht aber zumindest die Einbindung eines Knotens in verschiedenen Domänen, was auch einem Perspektivwechsel entspricht.

In der Entwicklung von Ontologiesprachen haben sich zwischenzeitlich zwei Strömungen herausgebildet. Goguen (2003) macht sogar drei Gruppierungen aus und bezeichnet sie als isoliert voneinander arbeitend. Die Verbindung der dritten Gruppe – nämlich der *Semantic Web Initiative* des World Wide Web Consortiums W3C – zu den anderen beiden Gruppen hat sich allerdings auch erst ab 2003 deutlich dargestellt. Zur Tradition der Konzeptsprachen bekannte sich Brachmann (1977) mit *KL-ONE.* Er legte damit erste Konstrukte für eine Ontologiesprache vor – wenn auch damals noch nicht unter diesem Namen: Darin differenziert er drei Knotentypen in einem semantischen Netz: *primitive*, *definierte* und *individuelle* Knoten[11]. Die Gesamtheit der Knoten einer Domäne wird als *TBox* bezeichnet, wobei sich das "T" auf "Terminologie" bezieht. Mit der TBox ist also ein semantisches Schema beschrieben. Der Begriff "TBox" wurde allerdings erst später KL-ONE hinzugefügt, um die logischen und begrifflichen Zusammenhänge von den tatsächlichen Gegebenheiten abzugrenzen. Die Knoten untereinander sind verbunden durch notwendige[12] Beziehungen. Primitive und definierte Knoten unterscheiden sich darin, dass ein definierter Knoten durch die Beziehungen zu anderen Knoten auch hinreichend beschrieben ist. Mit KL-ONE wurde auch *Subsumption* als wesentliches Konzept und Erweiterung des Konzepts der Klassen-Spezialisierung eingeführt: Eine Subsumption eines Knotens X durch einen Knoten Y liegt dann vor, wenn es beweisbar ist, dass alle Instanzen von X auch Instanzen von Y sind. Der Begriff des individuellen Knotens erfuhr in verschiedenen Diskussionen der Sprache KL-ONE und darauf aufsetzenden Sprachen und deren Umsetzungen unterschiedliche Interpretationen: Zum einen wurde er als eine konkrete Ausprägung oder Instanz eines primitiven oder definierten Knotens verstanden; mithin stellte er die Basis konkreter Aussagen (*Assertionen*). Auf der anderen Seite wurde der Begriff des individuellen Knotens dahingehend interpretiert, dass er selbst nicht mehr die Instanz sei, sondern ein generisches Konzept darstelle, mit dem maximal eine Instanz verbunden ist. Dies ermöglichte es, die Assertionen von den konzeptionellen Knoten zu trennen. Letztere Interpretation setzte sich in der weiteren Diskussion durch. Für die Assertionen ist in den späteren Erweiterungen – gleichzeitig aufkommend mit

[11] Der Begriff "Knoten" für den englischen Begriff "Concept" sei hier im Zusammenhang zum Vorhergehenden und zur Vermeidung des allzu generischen Begriffs "Konzept" verwandt.

[12] "notwendig" im mathematischen Sinne

dem Begriff der TBox – die *ABox* (*assertional Box*) als ein zusätzlicher Formalismus eingebracht worden. Er ermöglicht es, konkrete Aussagen darzustellen. Die Assertionen der ABox bilden demnach tatsächliche Ausprägungen zu der durch die TBox erzeugte Terminologie. Es existiert auch eine Attributbeziehung in KL-ONE, welche als *Role-Link* bezeichnet ist. Eine solche Rollen-Beziehung ist eine Relation zweier Knoten, die durch zwei Angaben eingeschränkt wird: den Bezugsbereich (*domain*) und den Wertebereich (*range*), die für diese Beziehung gelten. Zudem ist die Anzahl der Role-Links von einem Knoten aus durch Angabe eines Kardinalitätswertes begrenzbar. Abgesehen davon, dass sie die erste formulierte Sprache dieser Art ist, hebt gerade diese letzte Möglichkeit KL-ONE gegenüber den meisten nachfolgenden und verwandten Sprachen ab. Eine detailiertere Einführung in die Sprache, auf der auch die vorangehende Zusammenfassung basiert, bieten Luck und Owsnicki-Klewe (1990).

Gegenüber der optisch ansprechenden Darstellung auf Basis von Grafen bietet eine mathematisch eindeutige Notation des Formalismus erheblich bessere Möglichkeiten zur Analyse, für Beweise oder auch nur zur korrekten Darstellung und Kommunikation über ein System. KL-ONE war zu Beginn allerdings nur informal beschrieben worden, was sich als problematisch herausstellte. Erst durch Brachmann und Schmolze (1985) wurde die abschließende formale Spezifikation von KL-ONE publiziert. Um aber Beweise oder auch nur Berechnungen wie *Inferenzen* – die Berechnung impliziter Informationen auf Basis der gegebenen expliziten Informationen – innerhalb dieses Formulismus ohne Umstand durchführen zu können, ist die Möglichkeit zur Abbildung auf ein bestehendes Kalkül notwendig. Ein solches Kalkül als Basis zu nehmen, ermöglicht auch sämtliche zu diesem Kalkül bestehenden Gegebenheiten – Beweise und Sätze – auf den Formalismus anwenden zu können. Für KL-ONE wurde dadurch auf Basis des Kalküls von Schmidt-Schauß (1989) bewiesen, dass Subsumptionen in ihr unentscheidbar seien. Eine wesentliche Weiterentwicklung einiger der in Folge von KL-ONE vielfältig aufkommenden Ontologiesprachen war daher die vollständige Abbildbarkeit auf Beschreibungslogiken (*Description Logic*s). Baader und Nutt (2003) stellen Beschreibungslogiken wie folgt dar:

> Description Logics (DLs) is the ... name for a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain (the "world") by first defining the relevant concepts of the domain (its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description) (Baader und Nutt, 2003)

Nach Borgida (1996) kann gezeigt werden, dass Beschreibungen, die mit den Konstrukten der Beschreibungslogiken erstellt werden, auf Prädikate einer Untermenge der Prädikatenlogik erster und zweiter Ordnung abgebildet werden können. So lassen sich die Konzepte der Beschreibungslogiken in Prädikatenlogik übersetzen. Weiter wird gezeigt, dass diese Untermenge entscheidbar ist. Während KL-ONE selbst noch nicht auf Beschreibungslogiken basiert und auch deren erste Nachfolger dies nur eingeschränkt tun, ist dies einigen ihrer später nachfolgenden Ontologiesprachen-Systeme gegeben – darunter als erste *KRIS* von Baader und Hollunder (1991) und *CRACK* von Bresciani et al.

(1995). Für den praktischen Einsatz optimiert wurden FaCT von Horrocks (1998b) und RACER von Haarslev und Möller (2001) entwickelt; letzteres beinhaltet auch – im Gegensatz zu allen vorher genannten – eine Implementierung des ABox-Konzepts. Gerade beim Letztgenannten steht aber nicht die Art der Modellierung im Vordergrund, sondern vielmehr die effiziente Ermittlung des Wahrheitsgehalts von Aussagen, welche durch den *Reasoner* durchgeführt wird. Dieser ermöglicht die Annahme von Aussagen in Form von Anfragen und die Ermittlung ihrer Korrektheit auf Basis einer Ontologie. Aus diesem Grund sind die Entwickler von RACER flexibel in der Sprache, in der die Modelle erstellt werden, solange nur alle Konzepte von Racer abgebildet werden können. So ist nach DAML (siehe Abschnitt 4.1.3) nun auch OWL (siehe Abschnitt 4.1.3.4) eine mögliche Sprache für RACER geworden.

Eine andere Gruppe von Sprachen, die sich nicht auf KL-ONE berufen, basiert auf dem von Genesereth (1991) vorgestellten *Knowledge Interchange Format KIF*. Diese Syntax der Sprache basiert auf Lisp. Nach Goguen (2003) umfasst sie die Ausdrucksmöglichkeiten der Prädikatenlogik erster Ordnung und wurde um einige Ausdrucksmöglichkeiten für Metadaten erweitert. Ziel von KIF ist nicht primär – wie es bei den von KL-ONE abstammenden Sprachen und Konzepten der Fall ist – die Wissensrepräsentation und die Möglichkeit zur Abfrage dieses Wissens, sondern der einheitliche Austausch von Wissensbasen. Darauf aufbauend beziehungsweise daran angelehnt wurde das *Generic Frame Protocol GFP* von Karp et al. (1995) und als dessen Nachfolger das *OKBC Knowledge Model* als Grundlage des *Open Knowledge Base Conectivity* API von Chaudhri et al. (1998) entwickelt. Eine weitere Sprache – *Ontolingua* von Gruber et al. (1990) – basiert auf KIF, setzt sich aber das Ziel, von Menschen leicht gepflegt werden zu können, und im Einsatz durch verschiedene Gruppen auch bei unterschiedlichen Modellen anwendbar zu bleiben. Sie erweitert daher KIF um die Möglichkeiten interaktiver Festlegung und Pflege von Ontologien und einer gewissen Flexibilität bei syntaktischen und semantischen Variationen verschiedener Ontologiemodelle.

4.1.3. Ontologiesprachen auf Basis von XML

Seit Luke et al. (1996) gab es mit *SHOE*[13] Bestrebungen, die Konzepte von Ontologie-Sprachen auch im Rahmen des World Wide Webs umzusetzen. Auf der einen Seite gab es dieses Bestreben und die in Abschnitt 4.1.2 schon nachgezeichnete Entwicklung im Bereich der Wissensrepräsentation und -verarbeitung; auf der anderen Seite bestehen Defizite bei den Möglichkeiten, welche das World Wide Web im Bereich der semantischen Informationen und der automatischen Verarbeitung aufweist. Dies motivierte das *World Wide Web Consortium*, sich ebenfalls dieses Themas anzunehmen. Das *Resource Description Framework RDF* war der erste Schritt, semantische Informationen in das World Wide Web einzubetten[14]. Mit der dazugehörigen Schema-Erweiterung *RDF Schema* werden Standards für Klassen und Typen festgelegt, welche Konzepte wie Vererbung und Taxonomien ermöglichen. Die *Web Ontology Language OWL* bietet darüber hinaus

[13] Abkürzung für *Simple HTML Ontology Extensions*, siehe `http://www.cs.umd.edu/projects/plus/SHOE/`

[14] HTML selbst war bisher nur mit rudimentären semantischen Elementen ausgestattet.

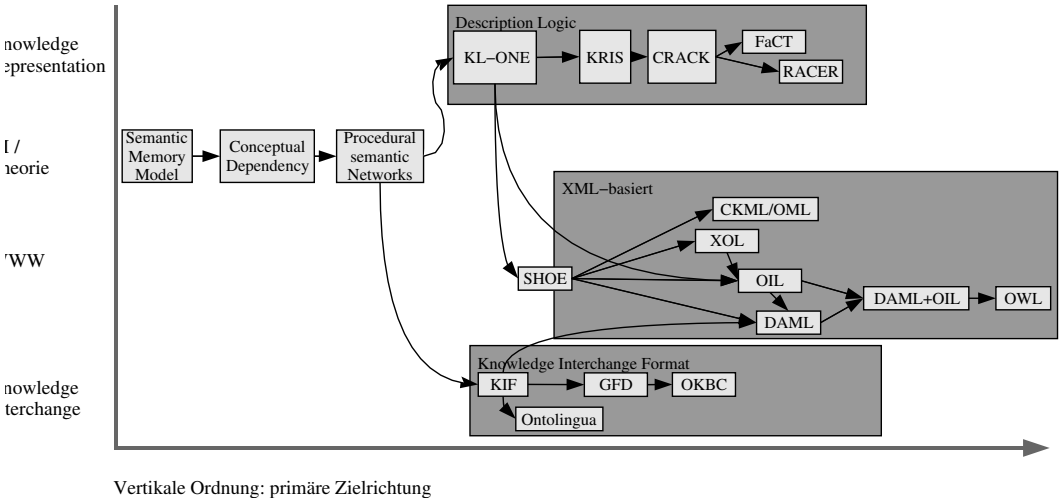

Abbildung 4.1.: Die Verbindungen zwischen den Ontologiesprachen und ihren Verläufern

in Weiterführung der Sprache *DAML+OIL* als W3C Empfehlung fast alle Aspekte, die von anderen Ontologie-Sprachen auch ermöglicht werden.

Abbildung 4.1 zeigt die Verbindungen, die zwischen den einzelnen taxonomischen Konzepten, den frühen Ontologiesprachen und den auf XML basierenden Ontologiesprachen bestehen.

4.1.3.1. Vorarbeiten

Die Idee, das World Wide Web beziehungsweise HTML mittels SHOE um die Möglichkeit zur Einbindung ontologiebasierter Semantik zu erweitern, fand Zuspruch und in Folge dessen eine Reihe von Nachfolgern. SHOEs Spezifikation war allerdings in Form einer *SGML DTD*[15] erstellt, während im gleichen Jahr der erste Entwurf für XML als eine leicht nutzbare und für eine unkomplizierte Verwendung optimierte Untermenge von SGML publiziert wurde. Spätere Sprachen wurden daher mittels XML DTD spezifiziert[16].

Auf SHOE aufbauend veröffentlichte Kent (1999) das Konzept der *Conceptual Knowledge Markup Language CKML* und ihrer Teilmenge *OML* – die *Ontology Markup Language*. Letztere ist Sprache ausschließlich zur Beschreibung von Taxonomien und Schemata. Neben SHOE als Basis wurde insbesondere auf Anlehnung an RDF(S) (siehe Abschnitt 4.1.3.2) und XOL geachtet. Die *XML-based Ontology Exchange Language XOL* wurde in etwa zur gleichen Zeit entwickelt und durch Karp et al. (1999) vorgestellt. Sie

[15] Eine *Document Type Definition* basierend auf der *Standard Generalized Markup Language*, diese Definition findet sich unter `http://www.cs.umd.edu/projects/plus/SHOE/shoe.dtd` – näheres zu SGML und DTD siehe Abschnitt 4.2.1

[16] Auch für SHOE wurde im Jahr 2000 eine XML DTD erstellt.

beruft sich als Inspirationsquelle wiederum auf OML, aber auch auf OntoLingua. Die Semantik von XOL basiert auf OKBC.

Fensel et al. (1999) stellten den *Ontology Inference Layer OIL*[17] vor. OIL basiert einerseits auf XOL und damit auch auf OKBC, andererseits war die Orientierung an den Konzepten der Beschreibungslogiken wichtig. Zudem weist OIL eine Kompatibilität zu RDFS in seiner XML-Form auf; damit liegt XML auch dieser Sprache zugrunde. OIL differenziert sich in drei Abstufungen, die in Untermengenbeziehung zueinander stehen: Die kleinste Version ist *OIL Core*, die bis auf das Fehlen des Konzepts der Reification (siehe Abschnitt 4.1.3.2) identisch ist mit RDFS. Die zweite Version *Standard OIL* erweitert RDFS' Möglichkeiten zur Modellierung. Die umfangreichste Fassung *Instance OIL* fügt der Sprache auch die Möglichkeit hinzu, Instanzen selbst zu modellieren. Ziel von OIL ist die Integration des Ontologienkonzeptes in das World Wide Web.

Eine ähnliche Zielsetzung hatten Berners-Lee et al. (2000) mit ihrer Veröffentlichung der *Darpa Agent Markup Language DAML*. Sie wollten die Basis einer Standard-Ontologiesprache für das World Wide Web aufbauen. DAML orientiert sich dabei an OIL, an SHOE und am KIF. Auch hier – wie bei OIL – ist die Basis RDFS und XML.

Wie schon in Fensel et al. (1999) angekündigt, verschmolzen die beiden Sprachen OIL und DAML zur neuen Sprache DAML+OIL. Diese Sprache erfuhr viel Zustimmung und wurde daher in einer leicht veränderten Fassung als *OWL DL* vom World Wide Web Consortium im Jahr 2004 als Empfehlung publiziert. Eine genaue Betrachtung erfolgt in Abschnitt 4.1.3.4.

4.1.3.2. Das Resource Description Framework

Das Resource Description Framework RDF wurde im Oktober 1997 vom World Wide Web Consortium als Arbeitsentwurf (Working Draft) der Öffentlichkeit vorgestellt und im Februar 1999 als W3C Empfehlung veröffentlicht. Ziel war die Schaffung einer einheitlichen Basis für Metadaten über Ressourcen im World Wide Web, um Anwendungen die automatische Nutzung und Verarbeitung von Daten in standardisierter Weise zu ermöglichen.

Der Umfang des Frameworks wurde bewusst minimal gehalten, um neben einer möglichst großen Flexibilität auch die einfache Handhabung zu ermöglichen. Das Grundkonzept des Frameworks ist die Ressourcenbeschreibung. Ressourcen – in diesem Zusammenhang – sind alle Dinge, über die eine Aussage getroffen werden kann. Dabei gibt es keine Einschränkung was beispielsweise die Repräsentation oder den Zugriff auf technischer Ebene anbelangt. Die Beschreibung von Ressourcen stellt immer eine Beziehung zwischen zwei Ressourcen dar: Eine Ressource wird durch eine andere mit Hilfe einer Relation beschrieben. Selbst diese Relation ist wiederum eine Ressource. Durch diese Verknüpfung wird ein Zusammenhang gebildet, der in vergleichbarer Form auch in der natürlichen Sprache vorhanden ist: das Tripel *Subjekt–Prädikat–Objekt*.

Ressourcen können mehreren Beschreibungen zugehören, sei dies als Subjekt, als Objekt oder auch als Prädikat – wobei nicht jede Ressource ein sinnvolles Prädikat darstellt.

[17] Eine alternative Bedeutung des Akronyms lautet *Ontology Interchange Language*

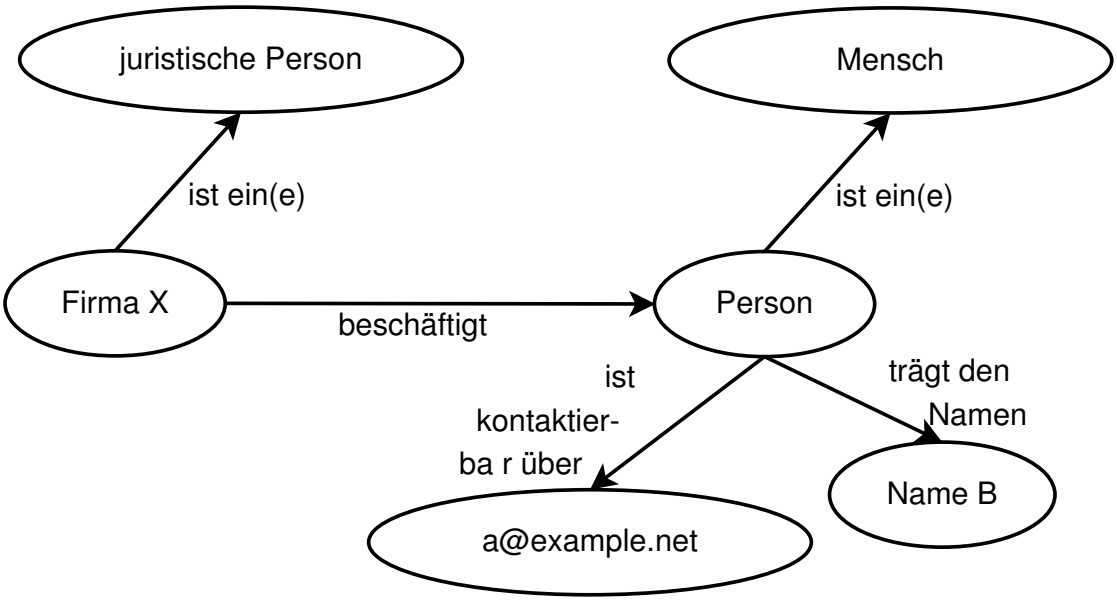

Abbildung 4.2.: Ressourcen-Netz: RDF in grafischer Darstellung

Durch diese Mehrfachverknüpfungen werden Ressourcen-Netze gebildet. Bei einem solchen Ressourcen-Netz handelt es sich um einen gerichteten Graphen wie in Abbildung 4.2.

Eine Ressource wird in RDF durch einen URI repräsentiert. Ein Objekt kann auch ein simples Element sein – beispielsweise ein einfacher String für einen Eigennamen. Über dieses Objekt kann allerdings – außer der Angabe zu seinem eigenen Wert – keine Aussage getroffen werden kann, da es selbst nicht mehr als Ressource mittels eines URI angesprochen werden kann. Einen anderen Sonderfall stellt die anonyme Ressource dar: Ist es nicht möglich, einen Bestandteil eines Tripels zu benennen – beispielweise aus Unkenntnis über diesen Teil –, will aber trotzdem eine Verbindung der anderen beiden Ressourcen zum Ausdruck bringen, lässt sich diese unbenannte Ressource stellvertretend verwenden.

Für die Verwendung in Computersystemen gibt es neben der grafischen Darstellung eine maschinenlesbare Umsetzung auf Basis von XML – entsprechend *RDF/XML* benannt. Hierbei ist jede Ressource als Auszeichnungs-Attribut oder als Element aufgeführt. Diese beiden Darstellungsvarianten sind äquivalent. Das Subjekt eines Tripels leitet mit dem Element *rdf:Description* eine Ressourcen-Beschreibung ein. Das oben dargestellte RDF in grafischer Form lässt sich demnach auch leicht in RDF/XML erstellen. Unter Einsatz der *XML Base* Technik, bei der sich wiedeholt aufkommende Namensräume mit einer zu Beginn festzulegenden Abkürzung dargestellen lassen, könnte das RDF in XML/RDF Form aussehen wie in Quellcode-Beispiel 1[18].

Es ist auch möglich, die Bestandteile eines Tripels einzeln in ihrer Funktion als Sub-

[18]Die Domains example.net und example.org sind für den Zweck solcher Beispielangaben vorgesehen und haben ansonsten keine weitere Bedeutung.

Quellcode-Beispiel 1 Ein Beispiel für RDF in XML-Form

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:bsp="http://example.org/namespace/">
  <rdf:Description rdf:about="http://example.net/Firma_X">
    <bsp:ist_eine rdf:resource
         ="http://example.net/juristische_Person" />
    <bsp:beschaeftigt rdf:about="http://example.net/Person_A" />
  </rdf:Description>
  <rdf:Description rdf:about="http://example.net/Person_A">
    <bsp:ist_eine rdf:resource="http://example.net/Mensch" />
    <bsp:traegt_den_Namen>Name_B</bsp:traegt_den_Namen>
    <bsp:ist_kontaktierbar_ueber rdf:resource
         ="mailto:a@example.net" />
  </rdf:Description>
</rdf:RDF>
```

jekt, als Prädikat oder als Objekt festzulegen. Auf diese Weise werden nicht nur Ressourcen untereinander – mittels anderer Namensräume entstammender Prädikate – in Verbindung gebracht, sondern auch innerhalb der Aussage – also des Tripels – mit RDF inhärenten Prädikaten versehen: *rdf:subject*, *rdf:predicate* und *rdf:object*. Das Subjekt für diese Prädikate ist natürlicherweise die Aussage selbst. Daher gibt es auch noch das RDF-Element *rdf:Statement*. Eine einzelne Zeile aus dem Quellcode-Beispiel 1 würde sich mit diesen Mitteln wie in Quellcode-Beispiel 2 darstellen lassen.

Quellcode-Beispiel 2 Reification eines RDF-Tripels

```
<rdf:Statement rdf:about="#triple_abc">
  <rdf:subject rdf:resource="http://example.net/Person_A" />
  <rdf:predicate rdf:resource
      ="http://example.org/namespace/ist_eine" />
  <rdf:object rdf:resource="http://example.net/Mensch" />
</rdf:Statement>
```

Über ein solches Tripel lassen sich dann noch weitere Angaben machen: Denkbar ist die Nennung des Erstellers dieser Aussage, das Erstellungsdatum oder was sonst für den Anwendungsfall sinnvoll ist. Die Aussage ist damit selbst eine beschreibbare Ressource geworden. Eine solche explizite Aussagenbeschreibung wird in RDF als *Reification* (Verdinglichung) bezeichnet.

Eine sinnvolle Ergänzung in RDF/XML stellen die Container-Elemente dar. Mit ihnen ist es möglich, eine Reihe von gleichartigen Ressourcen, welche mit einer anderen Ressource über das gleiche Prädikat verbunden sind, mit einem Rahmen-Element zu

umfassen. Damit ist nicht nur eine schnellere Bearbeitung möglich, da das statt des Ressourcen-Elements nur als Marke fungierende Listen-Element keine eigene Subjekt-Prädikat-Objekt-Verbindung darstellt, sondern auch eine bessere Lesbarkeit für Menschen gegeben. Das Container-Element bietet verschiedene Optionen zur Angabe der Art des Container wie geschlossen oder erweiterbar und ungeordnet oder geordnet.

Auf diese Weise lassen sich sämtliche Ressourcen eines beliebigen Betrachtungsgegenstandes in Verbindung bringen. Im siebenstufigen Konzept des Semantic Web nach Koivunen und Miller (2001) ist RDF die einfachste Möglichkeit, über Ressourcen etwas auszusagen. Zusammen mit RDF Schema stellt es die Ebene dar, in der aus den Ressourcen heraus das Vokabular des betrachteten Bereichs definiert wird.

4.1.3.3. RDF Schema

Die RDF Schema Language (RDFS) bietet – aufbauend auf dem Resource Desription Framework – wesentliche Konzepte zur einheitlichen Strukturierung von Ressourcen. Der Sprache entsprechend handelt es sich bei den Schema-Komponenten nicht um syntaktische Muster, sondern um einen sematischen und strukturgebenden Rahmen. Die zwei wesentlichen Aspekte, die RDFS dem bestehenden Resource Description Framework beisteuert, sind Klassen (*Classes*) und Eigenschaften (*Properties*). Klassen ermöglichen – ganz im Sinne der Taxonomien – eine Kategorisierung der Ressourcen. Eigenschaften fügen den in Klassen eingeordneten Ressourcen dann eine genaue Spezifizierung – also Metadaten – hinzu.

RDFS wurde im Jahre 1998 als Ergänzung zu RDF vom World Wide Web Consortium als Arbeitsentwurf veröffentlicht. 2004 erschien die endgültige W3C Empfehlung. RDFS ermöglicht es, ein für andere RDF-Dateien geltendes Vokabular mittels RDF selbst zu definieren. Das Kernelement – die Ressource – wird im Zusammenhang von RDFS einer Klasse zugewiesen. Die Ressource selbst ist also eine Instanz dieser Klasse. Ganz dem Konzept entsprechend, dass Ressourcen alle Dinge sind, über die etwas ausgesagt werden kann, sind auch die Klassen selbst wiederum Ressourcen. Nejdl et al. (2000) weisen darauf hin, dass hierdurch allerdings die klare Trennung zwischen Meta-Ressourcen und Ressourcen durchbrochen wird, was zu Unentscheidbarkeit führen kann: Die Klasse ist auf Grund dieser Festlegung nämlich eine Instanz ihrer selbst und die Ressource ist Superklasse und gleichzeitig Instanz der Klasse. Pan und Horrocks (2001) schlagen daher eine Untersprache von RDFS vor namens *RDFS(FA)*. FA steht dabei für "Fixed metamodel Architecture". Das Meta-Modell von RDFS(FA) orientiert sich dabei an dem von *UML*[19] und ist in vier Ebenen strikt getrennt. Auf diese Weise sind Überschneidungen zwischen den beschriebenen Objekten (Instanzen), den Klassen und Attributen, und den Metaklassen (wie die Metaklasse rdfs:class) nicht mehr möglich, und die Entscheidbarkeit bleibt erhalten. Dieser Vorschlag wird aber aller Voraussicht nach keine Auswirkung auf die Empfehlungen des World Wide Web Consortiums haben: Einerseits gibt es auch in der darüber liegenden Ebene (nach dem in Abschnitt 4.1.3.2 erwähnten siebenstufigen Konzept des Semantic Web) die Möglichkeit, mittels *OWL Full* nicht-entscheidbare Modelle zu erstellen; andererseits ist die Entscheidbarkeit bei Verwendung durch *OWL DL*

[19] Unified Modelling Language

und *OWL Lite* gegeben. Es liegen also grundsätzlich unterschiedliche Ansichten daüber vor, ob nicht-entscheidbare Modelle in irgendeiner Weise zulässig sein sollen, und in welcher Ebene solche Beschränkungen eingebunden sein müssen. Eine genauere Betrachtung zu OWL folgt in Abschnitt 4.1.3.4.

Wie in RDF/XML üblich, wird das Vokabular von RDFS über einen Namensraum definiert. Durch die enge Verwebung von RDFS mit RDF selbst, sind einige durch RDFS erst hinzugekommene Definitionen jedoch im Namensraum von RDF enthalten. Auf der anderen Seite ist der Namensraum von RDF mittlerweile durch RDFS-Definitionen ergänzt worden.

Die Basis-Klasse für alle Ressourcen in RDFS ist die Klasse *resource*. Die Klassen selbst sind zudem in der Klasse *Class* enthalten. Neben den Klassen für Literale und typisierten Daten – *Literal*, *XMLLiteral* und *Datatype* – stellt die Klasse *Property* ein weiteres Kernelement von RDFS dar. Mittels Eigenschaften lassen sich sowohl Wertebereiche und Schranken definieren – mittels *range* und *domain* –, als auch taxonomische Strukturen im Bereich der Klassen wie auch der Eigenschaften selbst. Dazu behilflich sind *subClassOf* und *subPropertyOf* für die Struktur zwischen Klassen und Eigenschaftsangaben und *type* für die Bindung der Ressourcen an die Klassen. Das Quellcode-Beispiel 1 basierte auf einem nicht näher definierten Namensraum mit dem URI "http://example.org/namespace/". Dieser lässt sich mittels RDFS definieren wie das Quellcode-Beispiel 3 zeigt.

Auf diese Weise werden die Meta-Informationen zu den Zusammenhängen von *Mensch*, *juristische Person*, *Person, traegt_den_Namen, ist_kontaktierbar_ueber* und *beschaeftigt* als Teil eines Vokabular von den inhaltlichen Informationen in Verwendung dieses Vokabulars getrennt. Dabei entsteht ein Vorteil durch Wiederverwendbarkeit und somit Standardisierung und Strukturierung des Vokabulars. Ein solches Vokabulars lässt sich als zentrale Instanz für eine Vielzahl an RDF-Dateien nutzen. Zudem lassen sich in komplexen Strukturen nicht offenbare Zusammenhänge (implizites Wissen) durch Inferenz-Algorithmen offenlegen. Noch viel weitreichender lassen sich Inferenzberechnungen ausnutzen, wenn zusätzliche Verbindungen aufgebaut werden, wie sie in OWL möglich sind.

4.1.3.4. Die Web Ontology Language

Als eine weitere Ebene in der Konzeption von Koivunen und Miller (2001) des Semantic Web ist die Ontologie-Ebene schon durch eine Spezifikation erfüllt, die den Status einer W3C-Empfehlung inne hat: Die *Web Ontology Language OWL* ist nicht nur konzeptioneller Nachfolger von DAML+OIL:

> OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL. (McGuinness und Harmelen, 2004)

Auch wenn RDFS schon die Kernelemente einer Ontologie-Sprache aufweist, bietet es keine Möglichkeiten, um verschiedene Dinge auszudrücken, welche die Ontologiesprachen und Ontologiekonzepte in Abschnitt 4.1.2 mit auszeichnen: Dazu zählen die Möglichkei-

Quellcode-Beispiel 3 Ein Beispiel für RDFS

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdfs:Class rdf:about="#Person" />
  <rdfs:Class rdf:about="#juristische_Person">
    <rdfs:SubClassOf rdf:resource="#Person" />
  </rdfs:Class>
  <rdfs:Class rdf:about="#Mensch">
    <rdfs:SubClassOf rdf:resource="#Person" />
  </rdfs:Class>
  <rdf:Property rdf:about="#beschaeftigt">
    <rdfs:domain rdf:resource="#Person" />
    <rdfs:range rdf:resource="#Mensch" />
  </rdf:Property>
  <rdf:Property rdf:about="#traegt_den_Namen">
    <rdfs:domain rdf:resource="#Person" />
    <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  </rdf:Property>
  <rdf:Property rdf:about="#ist_kontaktierbar_ueber">
    <rdfs:domain rdf:resource="#Mensch" />
  </rdf:Property>
</rdf:RDF>
```

ten zu Relationen und booleschen Operationen über Klassen und Eigenschaften, lokale Restriktionen oder eben auch die Möglichkeit zur Zusicherung der Entscheidbarkeit.

Die Unterschiede im Sprachumfang gegenüber DAML+OIL sind als marginal einzustufen und finden sich in der Umbenennung einiger Elemente wieder. Ein wesentlicher Unterschied allerdings liegt in der Aufteilung von OWL in drei Untersprachen wie sie in ähnlicher Form allerdings bei OIL existiert: OWL Lite, OWL DL und OWL full. Bei OWL DL – DL steht hier wieder für "Description Logic" – handelt es sich daher um die neue Version von DAML+OIL: Sie bietet einen mächtigen Sprachumfang, ist aber trotzdem entscheidbar, da sie nicht alle Konstrukte erlaubt: Eine Instanz kann hier nicht mehr Sub- oder Superklasse einer Klasse sein, eine Eigenschaft nicht gleichzeitig eine Klasse. Oberste Klasse, von der alle Klassen erben, ist daher nicht mehr *Resource*, welche in RDF ja selbst eine Ressource, also eine Instanz ist. Um eine schnelle Anwendbarkeit und auch eine erleichterte Einarbeitung in OWL zu ermöglichen, wurde die Untersprache OWL Lite entwickelt. Hier wurde auf bestimmte komplexe OWL-Elemente verzichtet – beispielsweise axiomatische Zusagen bezüglich der Instanzen einer Klasse oder die booleschen Klassen-Operatoren außer der Schnittmenge –, andere sind eingeschränkt –

Kardinalitätsbeschränkungen sind so nur mit 0 oder 1 festzulegen. Als dritte Sprache gibt es OWL Full, welche den gleichen Sprachumfang wie OWL DL aufweist, allerdings nicht deren Beschränkungen unterliegt. Durch diese syntaktischen Freiheiten lassen sich auch nicht-entscheidbare RDFS-Modelle in OWL abbilden; allerdings auch weiterhin mit dem Manko der Nicht-Entscheidbarkeit. Auf Grund dieses Zusammenhangs der OWL Sprachen lässt sich folgendes feststellen:

> Every legal OWL Lite ontology is a legal OWL DL ontology.
> Every legal OWL DL ontology is a legal OWL Full ontology. (McGuinness und Harmelen, 2004)

Neben der starken Erweiterung der vorgegebenen Elemente bietet OWL gegenüber RDFS eben auch die Möglichkeit zur Zusicherung der Entscheidbarkeit. Dies führt zu der realen Möglichkeit, nicht nur Ontologien abzubilden, sondern auch Berechnungen über sie vornehmen zu können und somit per Inferenz implizites Wissen zu ermitteln. Erst dadurch ergeben die diversen Sprachelemente von OWL auch einen praktischen Sinn:

Das von RDFS vorhandene Klassenkonzept wurde um zwei vordefinierte Klassen erweitert: *Nothing* als die Subklasse aller Klassen, der kein *Individual* – so werden die Instanzen in OWL benannt – angehört, und *Thing* als die Superklasse aller Klassen, der alle Individuals angehören – mit Zuordnung mindestend der Klasse Thing wird ein Individual auch deklariert. Über Klassen, Eigenschaften und Individuals kann eine Aussage zur Äquivalenz getroffen werden: *equivalentClass*, *equivalentProperty* und für Individuals *sameAs*. Von den Eigenschaften kann auch jeweils ausgesagt werden, ob sie symmetrisch, transitiv, funktional oder umgekehrt funktional sind (*SymmetricProperty*, *TransitiveProperty*, *FunctionalProperty* und *InverseFunctionalProperty).* Neben der in RDFS für Eigenschaften schon vorhandenen globalen Beschränkung auf eine Domäne kann in OWL eine entsprechende lokale Beschränkung in Zusammenhang mit einer Klasse festgelegt werden – diese kann für mindestens ein (*someValuesFrom*) oder alle Individuals (*allValuesFrom*) gelten müssen. Die Anzahl der möglichen Individuals lässt sich durch Angabe einer Kardinalität festlegen (*cardinality*). Klassen lassen sich über boolesche Operationen genauer in Bezug auf andere Klassen spezifizieren (*disjointWith*, *intersectionOf*, *unionOf* und *complementOf*). Die Elemente *equivalentClass* und *disjointWith* stellen axiomatische Zusagen zur Verfügung. Das gleiche gilt auch für die konkrete Festlegung aller aller möglichen Individuals einer Klasse (*oneOf*) und einer Eigenschaft (*hasValue*). Das oben schon verwendete Quellcode-Beispiel 1 lässt sich auch mit OWL-Elementen wie im Quellcode-Beispiel 4 erweitern.

Für die Klasse “Mensch” lässt sich so ausdrücken, dass sie verschieden ist von “juristische Person” (*disjointWith*). Für “Person” lässt sich mittels Bildung der Vereinigungsmenge sagen, dass es sich dabei immer um “Mensch” oder “juristische Person” handelt (*unionOf*). Von der Eigenschaft “beschaeftigt” lässt sich darstellen, dass sie umgekehrt funktional ist, so dass eine “Person” nur durch genau eine Instanz “beschaeftigt” sein kann (*InverseFunctionalProperty*[20]).

[20]Ist ein Objekt aus einem anderen als dem eigenen Namensraum – hier der OWL-Namensraum –, so lässt es sich nicht per XML Base ansprechen. Über die mögliche abgekürzte Form per ENTITY

Quellcode-Beispiel 4 Ein Beispiel für OWL

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
  <owl:Ontology rdf:about="">
    <owl:Class rdf:about="#Person">
      <owl:unionOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#juristische_Person"/>
        <owl:Thing rdf:about="#Mensch"/>
      </owl:unionOf>
    </owl:Class>
    <owl:Class rdf:about="#juristische_Person">
      <rdfs:SubClassOf rdf:resource="#Person" />
    </owl:Class>
    <owl:Class rdf:about="#Mensch">
      <rdfs:SubClassOf rdf:resource="#Person" />
      <owl:disjointWith rdf:resource="#juristische_Person" />
    </owl:Class>
    <owl:ObjectProperty rdf:about="#beschaeftigt">
      <rdfs:domain rdf:resource="#Person" />
      <rdfs:range rdf:resource="#Mensch" />
      <rdf:type
        rdf:resource="http://www.w3.org/2002/07/owl
        #InverseFunctionalProperty" />
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="#traegt_den_Namen">
      <rdfs:domain rdf:resource="#Person" />
      <rdfs:range
        rdf:resource="http://www.w3.org/2001/XMLSchema
        #string" />
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="#ist_kontaktierbar_ueber">
      <rdfs:domain rdf:resource="#Mensch" />
    </owl:ObjectProperty>
  </owl:Ontology>
</rdf:RDF>
```

Darüber hinaus bietet OWL Möglichkeiten zur Versionierung, zur Angabe von für Menschen lesbare Anmerkungen und zum Import anderer OWL-Ontologien. Mittels

Angabe in den Kopfzeilen sei hier der Zugänglichkeit halber verzichtet.

Angaben zur Versionierung lassen sich Angaben zu Kompatibilitäten und Inkompatibilitäten vornehmen. Die Import-Option ermöglicht es, in OWL die Konzepte der Wiederverwendbarkeit zu nutzen. Dieses schon in Abschnitt 3.1.3 erwähnte Konzept, stellt eine mächtige Technik für OWL dar, die auch gerade in einem dezentralen System Anwendung finden kann. Hier ist aber das Problem gegeben, dass eine innerhalb eines Kommunikationsablaufs importierte OWL-Ontologie in einer späteren Kommunikation nicht mehr zur Verfügung steht, da das ferne System nicht mehr erreichbar ist oder die Ontologie zwischendurch eine Änderung erfahren hat. Wird eine ferne Ontologie also nicht nur für diese eine Kommunikation benötigt, sondern in die lokale Ontologie integriert, ist auch der Import nicht nur als Verweis auf eine ferne Ontolgie durchzuführen, sondern eine Übertragung der OWL-Datei in das lokale System[21] erforderlich.

4.1.3.5. Ontologiesprachen im Vergleich

Bei der vorstehenden Betrachtung fand ein Großteil der bis heute entwickelten prägenden Ontologiesprachen und Ontologiesysteme Eingang. Dabei differenzieren diese sich einerseits über die Zielsetzung und andererseits durch die technische Umsetzung. Die Zielsetzung orientiert sich am jeweiligen Forschungsgebiet. So sind die Sprachen aus dem Bereich des Wissensmanagement auf effiziente Ontologie-Modellierung und Inferenz-Berechnung in einem geschlossenen System ausgerichtet. Die Sprachen aus dem Sektor Knowledge Interchange sind primär für den einheitlichen Austausch zwischen Wissensbasen gedacht. Die Sprachen, welche für das World Wide Web entwickelt wurden, haben die Austauschbarkeit und Ontologien-Erstellung auf Basis bestehender Internet-orientierter Standards und Empfehlungen zum Ziel. Diese Sprachen sollen die Grundlage des Semantic Web als Teil des heute schon existierenden World Wide Web aufbauen helfen.

Wie in Abschnitt 4.1.3 erläutert, sind die Web-orientierten Sprachen in ihrer Konzeption durch Vertreter der anderen beiden Bereiche inspiriert. Dies führt dazu, dass ausgereifte Konzepte zu den Themen Modellierung und Austausch in diese Sprachen integriert werden konnten. Auch innerhalb des Bereichs gab es einen intensiven Austausch und eine Orientierung untereinander, was zu einer Optimierung des aktuellen Vertreters OWL führte.

Auch die technische Basis unterscheidet sich für die drei Bereiche: Das Knowledge Interchange Format und die Sprachen und Konzepte, die auf ihm beruhen, basieren auf Lisp. Die web-orientierten Sprachen haben – abgesehen von den Anfängen von SHOE - alle XML als Grundlage; RDF, RDFS und OWL sind selbst Empfehlungen des W3C. Für den Bereich des Wissensmanagements wurde mit KL-ONE eine eigene Sprache entwickelt, welche in diversen Formen bis zu den Systemen KRIS und FaCT als eigene Konzeptbeschreibungs-Sprache weiterentwickelt wurde und bei Horrocks (1998a) detailiert beschrieben ist. Diese basiert auf XML, verwendet allerdings eine eigene proprietäre DTD. Mittlerweile gab es laut Horrocks (2003) Tests zur Abbildung eines Subsets von OWL DL auf die FaCT eigene Sprache. Die Entwickler von RACER hingegen sahen

[21] oder ein sicheres und beständiges Drittsystem, eine sogenannte *Trusted Third Party*; dies ist in dem Fall nötig, dass die Speicher-Ressourcen des lokale Systems nicht ausreichen.

sich nicht als Sprachentwickler und bedienten sich effizienter Standards wie DAML und neuerdings auch OWL.

Demnach ist OWL nicht nur innerhalb des Bereichs Semantic Web als das Ergebnis der Forschung aller Vorläufer entwickelt worden und somit entsprechend ausgereift; auch die Entwickler anderer Bereiche sehen in der Nutzung von OWL entsprechende Vorteile. OWL ist auch die Sprache, die über verschiedene Herkunftsverknüpfungen von allen drei Bereichen Konzepte integriert hat. Diese Anerkennung, Verbreitung, aber auch die Aktualität, die freie Verfügbarkeit und die Tatsache, dass OWL selbst einen Quasi-Standard darstellt, machen sie zu einem geeigneten Kandidaten zum Einsatz für verteilte Teilontologien.

4.1.4. Das Jena Framework

Als Ausgangspunkt für eine Implementierung des Ontologienkonzepts ist der Einsatz einer Bibliothek oder eines Frameworks wünschenswert, welches die OWL Spezifikation implementiert oder wesentliche Teile von ihr. Miller und Hendler (2004) von der Sematic Web Initiative des W3C stellen neben Einführung, Referenz und Beispielen auch standardkonforme Umsetzungen der OWL Empfehlung zur Verfügung. Im Abschnitt "API"[22] sind zur Zeit zwei Einträge vermerkt: Der erste ist *Cerebra* – ein API für Ontologie-Management und visuelle OWL Modellierung. Das Produkt der Firma "Network Inference" ist allerdings nicht frei zugänglich. Das zweite namens *Jena* wurde im Rahmen des Semantic Web Research Programm von den "Hewlett-Packard Labs" entwickelt. Im Gegensatz zu Cerebra ist es als Open Source freigegeben und daher frei zugänglich[23] und nutzbar. Jena ist zudem in der genannten Auflistung des W3C nicht nur unter der Rubrik API eingetragen, sondern ebenso unter "Parser/Validator" und unter "Reasoner" verzeichnet. Ein weiteres – allerdings nicht in der Auflistung des W3C genanntes – API ist das *OWL API* des *WonderWeb* Projekts[24]. Allerdings ist es noch als im Alphastadium befindlich gekennzeichnet. Von einer genaueren Untersuchung bezüglich einer Verwendung zum aktuellen Zeitpunkt sei daher Abstand genommen.

Das API Jena bietet nicht nur Möglichkeiten zur Modellierung und zum Zugriff auf OWL-Ontologien, sondern beinhaltet auch Optionen zur Validierung – laut Carroll (2003) sind diese annähernd komplett implementiert. Der Jena eigene regelbasierte Reasoner ist nach Reynolds (2003) allerdings nur als "preview version" enthalten. Jena ist ein Java-Framework; die Programmiersprache Java ist eine frei nutzbare, einheitliche und weit verbreitete Vertreterin ihrer Art. Unter all diesen Gegebenheiten bietet sich Jena als umfassendes und frei verfügbares Framework in einer geeigneten Programmiersprache zur Verwendung als Basis der ontologie-orientierten Aspekte einer Test-Anwendung an.

Die Kernaufgabe des Jena API ist die Erzeugung und Veränderung von Ontologie-Modellen. Jena bietet Klassen und Schnittstellen, durch welche Ressourcen, Eigenschaften, Klassen und alle anderen Elemente von RDF, RDFS und OWL repräsentiert werden können. Im Gegensatz zu einer früheren Fassung von Jena, die genau an der Sprache

[22] Application Programming Interface
[23] siehe `http://jena.sourceforge.net`
[24] siehe `http://owl.man.ac.uk/api.html`

DAML+OIL orientiert war, sind die Klassen und Schnitstellen der aktuellen Version 2 sprachneutral gehalten. Dies ermöglicht die Verwendung der gleichen Strukturen für verschiedene Sprachen wie RDFS, DAML+OIL und OWL. Für jede dieser Sprachen existiert ein Profil, welches eine Klasse mit dem entsprechend spezifizierenden URI verbindet – vorausgesetzt, ein entsprechendes Element ist in dieser Sprache definiert.

Ein Modell wird als Java-Objekt durch Nutzung eines Objekts, welches nach dem von Gamma (1994) beschriebenen Entwurfsmuster als *Factory* angelegt ist, unter der parametrischen Angabe eines solchen Profils erzeugt. Gleichzeitig wird dieses Modell an einen *Ontology Document Manager* gebunden. Dieser kann bei Bedarf durch weitere Parameter konfiguriert werden – beispielsweise zur Importmöglichkeit von weiteren Ontologien oder zur Einstellung der Möglichkeit zur Zwischenspeicherung (Cache) von Modellen. Er stellt auch eine Möglichkeit zur Verfügung, im Falle der Nichterreichbarkeit einer zu importierenden fernen Ontologie auf eine lokale Kopie zurückgreifen zu können. Über diese kompletten Modelle lassen sich die Operationen *Vereinigung*, *Schnitt* und *Differenz* durchführen und somit zu neuen Modellen verändern.

Eine weitere wesentliche Basis in Jena ist die Klasse *OntResource*. Ganz im Sinne von RDF erbt jede Ontologie-Element-Klasse[25] von einer solche OntRessource. Die Zirkel- und Entscheidbarkeitsprobleme aus Abschnitt 4.1.3.3 kommen in dieser Form hier - bedingt durch den Grundsatz der Einfachvererbung bei Java - natürlich nicht vor. Als relevante Ontologie-Element-Klassen gibt es zudem *OntClass*, *OntProperty* und *Individual*. Wichtig gerade auch für die Modifikation von Modellen ist die Navigation in ihnen. Hierbei basieren die wesentlichen Funktionen auf den ontologie-immanenten Verbindungen wie Unter- und Oberklasse aber auch anderen Properties. Diese Verknüpfungen von Objekten (im RDF Sinn) mit einem Subjekt liegen als ungeordnete Menge vor. Es besteht das Problem, dass die Abfrage einer von mehreren Properties einer Klasse also nicht deterministisch ist. Zu diesem Zweck gibt es in Jena Iteratoren zur Auflistung und vorübergehenden Ordnung dieser Properties. Allerdings wird diese gesamte Auflistung für größere Modelle nicht zur Anfrage empfohlen, da sie wenig performant ist. Eine Einschränkung – mittels Selektoren, die Subjekt, Prädikat und Objekt eingrenzen können – schwächt diese Performanzprobleme ein wenig ab. Eine detailierte Beschreibung des Jena API gibt Dickinson (2004).

Für aufwendigere Abfragen, bei denen mehr Performanz gefordert ist, steht in Jena eine Eigenentwicklung, die *Resource Description Query Language RDQL* zur Verfügung: Dabei handelt es sich um eine simple Abfragesprache mit Namensräumen und Filtern. Sie ist nutzbar von der Kommandozeile aus und aus Java heraus. RDQL weist eine SQL-ähnliche Syntax auf. Es ist eine Implementierung der Sprache *SquishQL* von Miller et al. (2002) welche auf der Anfragesprache *rdfDB* von Guha (2000) basiert.

[25] Der Begriff *Klasse* ist hier im Sinne einer Java-Klasse gemeint, anstatt einer Klasse innerhalb einer Ontologie.

4.2. Transformationen und Anbindung

Den zweiten Schwerpunkt im dezentralen Strukturkonzept für eine digitale Identitäten-Infrastruktur bildet die Möglichkeit zur Transformation der Strukturen der Identitätsdaten. Transformation ist schon von der Begrifflichkeit her das wesentliche Thema der *Datenverarbeitungs*-Technologie. In diesem Fall geht es speziell um Kompilierung, nämlich die regelbasierte automatische Verarbeitung der Strukturdaten zur Wandlung von einer Struktur in eine andere. Die zugehörigen Regeln wiederum sollen sich auf möglichst einfache Weise durch den Anwender erstellen lassen und zudem zu anderen in der Identitäten-Infrastruktur eingebundenen Clients übermittelbar sein.

Es existieren diverse Frameworks und Systeme, die sich der Transformation von Daten annehmen. Für regelbasierte Übersetzungen existieren speziell darauf ausgerichtete flexible Stil- und Transformationssprachen. Hier ist erneut die Eignung der verschiedenen Sprachen zu überprüfen. Die Basis sei hier wiederum die Beachtung von Offenheit, Vereinheitlichung, Verbreitung und Unabhängigkeit. Wünschenswert wäre zudem eine gute Integrationsmöglichkeit mit der bisher präferierten Technik.

Als letzter Aspekt ist der Einsatz einer geeigneten Umsetzung einer solchen Sprache sinnvoll. Im Rahmen der Untersuchung nach ausreichender Verbreitung der Sprache gilt es, eine geeignete Implementierung zu finden. Oft handelt es sich dabei um Konzepte, die beispielsweise die Verbindung mehrerer vielförmiger Datenquellen in ein System ermöglichen – Datenbank-Integrationssysteme –, die Programmquellcode optimieren oder gegebenen Umständen anpassen können – Compiler und Transcompiler – oder um solche, die textuelle in grafische Darstellungen umwandeln können. Im Bereich strukturierter Daten und Dokumente – also der Bereich, der auch die Umformung der Strukturen von Identitätsdaten betrifft – basieren einige Programme und Frameworks auf SGML beziehungsweise DSSSL, die meisten aber auf XML und XSLT.

4.2.1. Mapping: Struktur-Transformation

Die Trennung von Form und Inhalt beziehungsweise Darstellung und Funktion findet in vielen Bereichen der Informatik eine vielfältige Umsetzung: sei es in der frühen Form auf Hardware-Ebene schon 1944 bei der, dem Konzept von Neumann (1946) entgegengesetzten, *Harvard Architektur*, die – wie Lapsley et al. (1997) ausführen – die Trennung von Daten und Instruktionen mit sich brachte; sei es beim Entwurfsmuster *Model-View-Control* nach Gamma (1994) – hier kommt interaktionsbedingt mit "control" allerdings eine weitere Ebene hinzu – oder auch beim Textsatzsystem *TEX* von Knuth (2000). Bei der *Standard Generalized Markup Language SGML* findet eine solche Trennung gleich auf mehreren Ebenen statt: So gibt es mit der *Document Type Definition DTD* eine Referenz für die in einer SGML-Sprache verwendeten Elemente; und mit der *Document Style Semantics and Specification Language DSSSL*[26] existiert eine eigene Sprache für die Formatierung. Sie beinhaltet auch die Möglichkeit, die Struktur der Darstellung umzuformen oder auch ganz allgemein eine Struktur in eine andere zu transformieren. Ganz in diesem Sinne findet sich dieses Konzept auch bei XML mit XSLT.

[26] Als ISO Standard bestätigt unter der Nummer ISO/IEC 10179

4.2.2. Das Konzept der Trennung

Automatisierte Transformation und Übersetzung waren immer schon ein Schwerpunkt der Informatik. Ein bekanntes frühes Beispiel ist die Entwicklung der ersten höheren Programmiersprache: Ein Team um Backus und Herrick (1954) bei der IBM konnte zeigen, dass sich ein mit der Sprache *Fortran* erstelltes Programm und dem dazugehörigen Compiler ein ebenso optimierter Maschinencode entwickeln ließ wie durch die direkte und sehr viel aufwendigere Erstellung in Maschinensprache durch Menschen. Die Art der Transformation war hierbei noch festgelegt. Der Wunsch, Programmiersprachen auf verschiedenen Computersystemen zu verwenden, verlangte für jede einzelne Plattform zunächst noch weitgehend eine Neuentwicklung der jeweiligen Compiler. Bei regelmäßigen Erneuerungen existierender Sprachen und auch neuen Plattformen und neuen Sprachen erwies sich dieser Umstand als sehr unpraktisch. Gesucht war daher ein allgemeines Programm, dass zwar jeweils einmal für die einzelnen Plattformen existieren musste, welches aber eine allgemeingültige Beschreibung einer Sprache annehmen und daraus einen Compiler erzeugen könnte. Brooker (1960) stellte ein solches Programm vor: den ersten sogenannten *Compiler-Compiler*. Einen solchen bezeichnet Schmitz (1995) als eine Anwendung, welche auf Basis einer lexikalischen Struktur, einer kontextfreien Grammatik und Übersetzungsvorgaben durch eine Attributierung einen Compiler automatisch generieren kann. Diese Definition einer Sprache ist somit von der Programmierung des eigentlichen Compilers entkoppelt. Der Vorgang der Übersetzung führt also zwei Quellen zusammen – die formale Sprachbeschreibung und einen Quellcode in dieser Programmiersprache – und gibt eine der Sprachbeschreibung entsprechenden Übersetzung aus[27].

Dieses Konzept muss aber nicht auf Programmiersprachen begrenzt sein. In Anlehnung daran und an die unterschiedlichen Aufgaben von Autor und Layouter kam die Idee auf, eine solche Trennung auch bei Text- und Dokumenten-Satzsystemen umzusetzen. Knuth (1978) stellte das Dokumenten-Satzsystem *TEX* vor, welches einem ähnlichen Konzept folgt: Bei TEX handelt es sich um einen Makro-Prozessor; er benötigt nicht nur den Quelltext des Dokuments, der in ein druckreif gesetztes Dokument transformiert werden soll, sondern auch Angaben, auf welche Weise das Dokument gesetzt werden soll. Diese Angaben werden mit Hilfe einer TEX-eigenen Makrosprache vorgenommen und in Form von Makropaketen nutzbar gemacht. Die beiden bekannten Vertreter für solche Makropakete sind *Plain TEX* und *LATEX*.

Während TEX auf Druck spezialisiert ist und auf das entsprechende Layout, versteht sich die SGML als viel allgemeinere Dokumenten-Metasprache; sie ist der ISO-Standard[28] für eine strukturierte Dokument-Sprache. Entstanden ist SGML unter anderem aus der bei der IBM von Charles Goldfarb entwickelten *GML*, deren Wurzeln schon in das Jahr 1969 zurückreichen (siehe SGML Users' Group, 1990). SGML ermöglicht die sehr flexible Definition von Auszeichnungssprachen. Das Instrument zur Definition ist die *DTD*. Ähnlich der Attributierung beim Compiler-Compiler ermöglicht eine DTD einem SGML-Parsersystem, ein mit dieser DTD konformes Dokument zu übergeben und dieses einem

[27] Aus Performanzgründen läuft es praktisch allerdings so, dass erst der Compiler aus der Sprachbeschreibung mittels Compiler-Compiler erzeugt und dann angewendet wird.

[28] Eingetragen unter der ISO-Nummer 8879

Ausgabesystem weiterzuleiten, welches wiederum auf Basis der DTD eine geeignete Ausgabe erzeugt. Die DTD ist aber nicht für die Interpretation zuständig. Im Gegensatz zu den bisher betrachteten Konzepten sind bei SGML die Definition einer Sprache und die Interpretation beziehungsweise die Transformation oder Formatierung in die Zieldarstellung zwei voneinander getrennte Dinge.

4.2.3. Form und Funktion

Als Tim Berners-Lee 1989 die Idee für ein Hypertext-basiertes Informationsnetzwerk erdachte – aus dem sich später das World Wide Web entwickelte – und damit *HTML* als simple Auszeichnungssprache erfand, schwebte ihm Organisation und nicht Präsentation vor. Zumindest war die Präsentation so wenig in seinem Fokus, dass er nach Lie und Bos (1999) die Darstellung allein dem Anzeigeprogramm überließ – dem *Web-Client* oder *Webbrowser.* Hingegen wies die Sprache für die Struktur mit diversen Abstufungen für Überschriften, mit Absätzen, Listen und natürlich Hyperlinks schon die wesentlichen Elemente auf. Diese Elemente waren durch eine DTD auf SGML-Basis definiert. Als in Folge der zunehmenden Verbreitung des Webs auf der einen Seite und der grafischen Benutzungsoberflächen auf der anderen Seite auch der Bedarf nach Formatierung, Gestaltung und Stilvarianten aufkam und durch diverse Erweiterungen durch die populären Webbrowser auch in proprietären Dialekten von HTML Stück für Stück integriert wurden, führte das World Wide Web Consortium, angeregt durch Sperberg-McQueen und Goldstein (1994), das Konzept der Stylesheet-Sprache unter dem Namen *Cascading Style Sheet CSS* ein: die Trennung von Struktur und Format beziehungsweise von Inhalt und Umsetzung oder Darstellung.

Auch die Transformation ist ein klassisches Thema der Informatik. Eines der ersten flexibel für beliebige Zeichenketten verwendbaren Transformationskonzepte war der *Macrogenerator* von Strachey (1965). Für diesen Zweck wurde allerdings erst 1996 die Sprache DSSSL für Transformation und Formatierung für SGML als Standard[29] verabschiedet. DSSSL ist eine Untermenge der Sprache Scheme, einem Lisp-Dialekt und IEEE-Standard[30]. Laut Spezifikation[31] ist das primäre Ziel von DSSSL, ein standardisiertes Framework und Methoden bereitzustellen, um die Verknüpfung zwischen Verarbeitungsinformationen mit den Auszeichnungen von SGML Dokumenten oder Abschnitten von diesen zu ermöglichen. Da Transformation und Formatierung zwar verwandte, aber doch Gebiete mit unterschiedlicher Zielrichtung sind, wurde DSSSL in zwei unabhängige Sprachen aufgeteilt: Die DSSSL Transformation Language und die DSSSL Style Language. Zunächst erfolgte diese Trennung laut Clark (1995) noch aufgrund der Unzulänglichkeiten der Anfragesprache, erwies sich dann aber für vielfältige Zwecke als äußerst nützlich. Diese Anfragesprache – die *Standard Document Query Language SDQL* – ist eine zu-

[29] Als ISO Standard bestätigt unter der Nummer ISO/IEC 10179

[30] IEEE Std 1178-1990, `http://standards.ieee.org/reading/ieee/std_public/description/busarch/1178-1990_desc.html` – Abruf 05.11.2004

[31] Eine HTML-Fassung der DSSSL-Spezifikation (27.10.1998) liegt vor unter `ftp://ftp.ornl.gov/pub/sgml/wg8/DSSSL/readme.htm`

sätzliche Ergänzung, mit der die einzelnen Elemente innerhalb eines SGML-Dokuments zum Zweck einer Formatierung oder Transformation angesprochen werden können. Die SDQL wird von beiden Sprachen verwendet.

DSSSL unterstützt zwei verschiedene Wege der Transformation: strukturelle Umgruppierung und kontextbedingte inhaltliche Veränderung. Der eigentlichen Transformation eines SGML-Dokuments geht die Umwandlung der internen Repräsentation in eine Baumstruktur voraus – bei DSSSL *Groove* genannt. Gerade Transformationen der Umgruppierung profitieren von dieser Darstellung, da die Umgruppierung zwischen verschiedenen Ebenen durch simple Elemente erreicht werden kann: Das Dokument wird nicht mehr global betrachtet, sondern aus der Perspektive eines einzelnen Baum-Knotens, was entsprechend relative Transformationen ermöglicht, aber absolute nicht ausschließt.

Genau wie XML als Vereinfachung von SGML den breiten Erfolg dieser Metasprache ermöglichte, sollte auch statt DSSSL eine weniger aufwendige Form für eine Formatierungs- und für eine Transformationssprache zum Einsatz in Zusammenhang mit XML entwickelt werden. Sie sollte außerdem als eine XML-Sprache definiert sein. Diese Sprache bekam den Namen XSLT.

4.2.4. XSLT – Transformation auf XML Basis

XML ist nach Goldfarb und Prescod (2002) eine vereinfachte und auf Aspekte des World Wide Web angepasste Untermenge von SGML. In Verbindung mit ihr lässt sich zu Format- und Transformationszwecken zwar auch DSSSL verwenden; naheliegend ist aber die an XML angepasste Format- und Transformationssprache *Extensible Stylesheet Language XSL*. Auch bei ihr gibt es die voneinander getrennten Möglichkeiten für Formatierung und Transformation: XSL ist aufgeteilt in ihre Teilsprachen *XSLT* für *Transformations* und *XSL-FO* für *Formatting Objects*. Beide nutzen die auch in anderem Zusammenhang nutzbare Sprache *XML Path Language XPath* zur Adressierung einzelner Elemente innerhalb einer XML-Struktur. Auf XPath sei später in Abschnitt 4.2.5 noch genauer eingegangen.

Nach der Spezifikation[32] soll ein XSLT-Prozessor einem XSLT-Stylesheet entsprechend einen Quellbaum in einen Zielbaum transformieren. Eine Ausgabe des Zielbaums ist nach der Spezifikation nicht erforderlich, wird aber empfohlen. Die Basis eines XSLT-Stylesheet sind Vorlagen (*Templates*), deren Anwendung auf einen Knoten im Quellbaum mittels gegebener Muster (*Pattern*) bestimmt wird. Pro Knoten kann immer nur eine Vorlage angewendet werden. Durch den Wert der Priorität des Musters wird entschieden, welche Vorlage anzuwenden ist. Die Priorität des Musters wird ermittelt durch den Grad der Spezifizierung: Spezifische Muster erhalten gegenüber generelleren einen höheren Prioritätswert. Wird ein Knoten durch kein Muster abgedeckt, gibt es eine vorgesehene Rückfall-Vorlage (*Fallback Template*), welche den jeweiligen Knoten unverändert belässt. Eine Vorlage wird in XML-gemäßer Darstellung durch folgendes Element eingeleitet:

```
<xsl:template match="Bedingung">
```

[32] Die jeweils aktuelle XSLT-Spezifikation befindet sich unter `http://www.w3.org/TR/xslt`

Innerhalb einer Vorlage können sich einerseits buchstabengetreue Vorgaben für die Elemente des Zielbaums und andererseits *XSLT-Elemente* befinden. XSLT-Elemente bilden die programmiertechnische Basis der Vorlagen und können im simplen Fall mit Hilfe der Elemente *xsl:element* und *xsl:attribute* auch nur die buchstabengetreue Erzeugung eines Elements im Zielbaum bewirken: Mit diesen beiden lässt sich allerdings auch mittels geeigneter Angaben in der Laufzeit der Transformation ein passender Element- oder Attributname berechnen.

Auch wenn XSLT zunächst als eine Auszeichnungssprache erscheint, bietet sie mit vielerlei Mechanismen die Möglichkeiten einer Programmiersprache: So ist die Wiederverwendung vordefinierter Attribut-Mengen an beliebigen Stellen möglich. Anhand von XPath-gemäßen Ausdrücken lassen sich mit *xsl:for-each* Schleifenstrukturen umsetzen. Die Reihenfolge wird dabei – wenn nicht durch eine Sortierung anders angegeben – durch die Reihenfolge der zutreffenden Elemente im Quelldokument vorgegeben. Sortierungen wiederum sind nicht nur für Schleifen nützlich, sondern können auch für sich das Ausgabeergebnis im Zieldokument beeinflussen. Für Bedingungen sind zwei Elemente möglich: *xsl:if* wird auf einen Ausdruck angewendet, der ein boolesches Ergebnis liefert, *xsl:choose* ermöglicht die Wahl aus einer Reihe von Möglichkeiten und ist somit eine Vereinfachung verschachtelter xsl:if-Elemente.

In XSLT existieren zwei Konzepte von Variablentypen. *xsl:variable* repräsentiert das klassische lokale Variablenkonzept: Innerhalb des Elemente-Baums des XSLT-Stylesheets stellt eine Variable in einem Knoten genau den Wert dar, den die zu diesem Knoten – in der Baumstruktur in Richtung auf das Wurzelelement – nächste Wertzuweisung erzeugt hat. Erfolgte die Wertzuweisung im Wurzelknoten selbst, handelt es sich um eine globale Variable. Das andere Konzept, *xsl:param*, erlaubt die Überschreibung durch äußere Elemente und somit die parametrisierte Wertübergabe. Die Variablen können von einem der folgenden Typen sein: Zeichenkette, Zahl, boolesch, Knotenmenge oder Ergebnisbaum-Fragment. Letzterer stellt eine Besonderheit gegenüber anderen Programmiersprachen dar. Dieser Typ stellt eine Teilstruktur eines XML-Baumes dar. Festgelegt wird eine solche Variable durch Angabe des Wurzelknotens dieser Teilstruktur.

Ein spezieller Zusatz wurde XSLT beigefügt, um mit einer XSLT-Transformation wiederum ein XSLT-Dokument zu generieren. Hierbei würden ohne diesen Zusatz sämtliche buchstabentreu zu übernehmenden Elemente vom XSLT-Prozessor bei der Transformation interpretiert werden müssen. Diese zu generierenden XSLT-Elemente müssen im Ablauf der Transformation "versteckt" werden. Dieses Verstecken geschieht durch die Verwendung eines Alias-Namensraumes: Statt des Präfixes "xsl" werden solche Elemente mit einem anderen Namensraum – beispielsweise "axsl" – versehen. Dieser Namensraum muss aber auch ganz regulär als Namensraum im Header aufgeführt sein. Bei der Erzeugung wird der Alias-Namensraum in "xsl" transformiert, aber die zugehörigen Elemente buchstabentreu in den Zielbaum übertragen.

An dieser Stelle kann nicht detailierter auf die spezielle aber recht mächtige Sprache XSLT eingegangen werden. Stattdessen sei auf die Werke von Kay (2000) und Tidwell (2001) verwiesen, die als Basis für diesen Überblick dienten.

4.2.5. XPath – Navigieren im Dokumentenbaum

Obwohl wesentliche Bereiche der Spezifikation von XSLT von der XPath-Spezifikation abhängen, handelt es sich bei XPath um eine eigenständige Empfehlung des W3C, da auch andere Sprachen auf die Möglichkeiten von XPath zugreifen können sollen. Schon jetzt bedient sich *XPointer* ebenfalls der XPath-Spezifikation. Die XPath-Spezifikation beschreibt drei Bereiche: Das Hauptthema ist die Möglichkeit zur Adressierung einzelner Fragmente innerhalb eines XML-Dokuments. Dabei werden diese Elemente nicht als Teil des syntaktischen Zusammenhangs eines XML-Dokuments angesprochen, sondern als Teil der abstrakten logischen Struktur. XPath bietet zudem Möglichkeiten zum Bilden von Ausdrücken und stellt eine Funktionsbibliothek zur Verfügung. Die Form der Adressierung wurde so gewählt, dass sich leicht eine Untermenge für die Darstellung der Muster für XSLT bilden lässt.

Die abstrakte logische Struktur eines XML-Dokuments, die XPath für seine Adressierung nutzt, besteht aus einem Baum mit verschiedenen Knotentypen: Elementknoten, Attributknoten und Textknoten. Durch die Adressierung von XPath kann jeder dieser Knoten einzeln oder als Teil einer Knotenmenge adressiert werden. Diese Adressierung findet relativ auf sogenannten *Achsen* statt, ausgehend von dem Knoten, der gerade im Fokus eines Prozesses ist. Der Begriff Achse bezeichnet die Verbindung des aktuellen mit dem adressierten Element. In der Terminologie der Baumhierarchie lassen sich von diesen Achsen zu Kindern und Eltern, in der Dokumentreihenfolge zuvorkommenden und nachkommenden Geschwistern, alle Vorfahren, alle Nachfahren, zur Wurzel oder dem Knoten selbst ansprechen. Mit dieser Gruppe sind alle Elemente eines Baums adressierbar. All diese Adressierungsmethoden sind frei miteinander kombinierbar. Zudem lassen sich auch mittels Attribut- und mittels Namensraumachse die Attribute und der Namensraum eines Elements adressieren. Eine XPath-Adressierung kann neben einer ausgeschriebenen Darstellung – mit Begriffen wie *ancestor*, *child* oder *self* – auch eine abgekürzte Syntax verwenden, die sich der üblichen Syntax von Dateisystemen anlehnt: So wird aus der adressierten Knotenmenge “alle Nachkommen der Kinder des betrachteten Knotens und seiner Geschwister” aus dem komplexen Ausdruck

```
parent::child::child::descendant
```

folgende komprimierte Form

```
../*/*//*
```

Die durch eine Adressierung beschriebene Knotenmenge lässt sich durch Ausdrücke weiter verarbeiten. Dabei stehen der Vereinigungsoperator, Pfadoperatoren und Filter zur Verfügung. Es lassen sich auch boolesche Werte mittels der booleschen Operationen und Relationsausdrücke berechnen sowie Zahlen mithilfe der üblichen arithmetischen Zahloperationen inklusive Modulo. Eine XPath-Implementierung muss auch einen Funktionskorpus nach der Spezifikation bieten. Dabei sind Funktionen vorgesehen für Knotenmengen, Zeichenkettenfunktionen, boolesche Funktionen und einige wenige Zahlenfunktionen. Erneut sei hier auf Kay (2000) verwiesen, der auch einen detailierten Einblick in XPath liefert.

4.2.6. XSLT Prozessoren

Ein XSLT-Prozessor ist eine Implementierung der XSLT-Spezifikation. Schon im Laufe der Entwicklung der heutigen XSLT-Empfehlung haben sich mehrere Gruppen und Einzelpersonen dieser Aufgabe angenommen. Dadurch stehen eine Reihe an XSLT-Prozessoren zur Verfügung. Es ist zu erörtern, welcher dieser Prozessoren geeignet ist. Zu dieser Erörterung seien die entsprechenden Aussagen der oben genannten Werke (Kay, 2000, S. 631ff und Tidwell, 2001, S. 3f) erneut hinzugezogen. Zur weiteren Orientierung soll der aktuelle Überblick von Brink und Tauber (2004) dienen. Es haben sich frühzeitig vier bekannte und verbreitete XSLT-Prozessoren herausgebildet: *Oracle XML Parser*, *Saxon* von Michael Kay, *Xalan* vom Apache Projekt und *xt* von James Clark (mittlerweile von Bill Lindsey betreut). Für alle vier gilt, dass sie ganz oder weitgehend die XSLT-Spezifikation erfüllen. Laut Gottlob et al. (2002) zeichnen sich alle diese aktuellen XSLT-Prozessoren auch gleich stark durch ineffiziente Bearbeitung der XPath-Anfragen aus. Jeder Transformationsvorgang ist also mit Bedacht durchzuführen. Auch in diesem Zusammenhang erweist sich eine Aufteilung in Teiltaxonomien als sinnvoll.

Den XML Parser von Oracle gibt es in zwei Versionen – einmal für C und einmal für Java[33]. Trotz der Java-Variante ist er allerdings nicht völlig plattform-unabhängig, sondern wird nur für einige Unix-Plattformen und Windows bereit gestellt. Das besondere an ihm ist die Integration von XML-Parser und XSLT-Prozessor. Er ist mit einer recht liberalen Lizenz versehen. Allerdings wird nicht der Quellcode mitgeliefert. Zudem warnt die Lizenz vor der Verbindung mit Programmen, die unter einer Open Source Lizenz stehen und damit Auswirkung auf die Lizenzgegebenheiten der daraus zu bildenden Software haben könnte. Dies ist für die Integration mit diversen anderen Software-Komponenten ein Problem und erforderte eine genaue Prüfung.

Saxon[34] hingegen ist ein quelloffener XSLT-Prozessor. Die momentan aktuelle Version 8.1.1 implementiert die Vorabversion 2.0 von XSLT (XSLT 2.0 Working Draft vom 12. November 2003). Die frühere Version 6.5.3 implementiert das derzeit aktuelle XSLT 1.0. Diese wird vom Autoren als ausgereift und sehr stabil eingestuft. Er listet einige wenige Punkte der mangelnden Konformität gegenüber der XSLT-Empfehlung auf, welche aber nur in seltenen Fällen überhaupt auftreten sollten. Saxon erhielt seinen Namen aufgrund der Festlegung auf einen an dem *SAX-Modell* orientierten XML-Parser. Das Parsing nach SAX ist ereignisorientiert (*event driven*). Im Gegensatz dazu gibt es das Parsing nach dem DOM-Modell, welches ein XML-Dokument als Ganzes im Speicher abbildet. Dies kann bei großen XML-Dokumenten problematisch sein, bietet aber Vorteile, wenn ein freier Zugriff auf die Elemente eines XML-Baums erfolgen muss.

Xalan[35] lautet der Name für das XSLT-Framework mit XSLT-Prozessor der Apache Software Foundation. Diese ist insbesondere durch die hohe Verbreitung ihres Webservers recht bekannt. Xalan entstammt einer Entwicklung der IBM unter dem Namen LotusXSL und wurde später mit einer Open Source Lizenz versehen und der Apache Software Foundation übergeben. Einerseits in kontinuierlicher Anpassung zum zugehörigen Apa-

[33] erhältlich unter `http://www.oracle.com/technology/tech/xml/xdkhome.html`
[34] zu beziehen über `http://saxon.sourceforge.net`
[35] siehe `http://xml.apache.org/xalan-j/`

che XML-Parser *Xerces*, andererseits in kontinuierlicher Fehlersuche und -behebung steht diese Software weiterhin in aktiver Entwicklung. Ein online zugängliches Bug-Tracking-System ermöglicht den genauen Einblick in jedes gelöste und ungelöste Problem, die zwar in großer Anzahl gemeldet sind, vielfach aber nicht die technische Funktion beeinträchtigen, sondern Dokumentation und Nutzbarkeit. Es fällt angesichts dessen schwer, zu beurteilen, wie weit die Konformität zur XSLT-Empfehlung eingeschränkt sein könnte. Nach eigener Aussage liegt eine vollständige Konformität mit allen verpflichtenden Teilen der Spezifikation vor. Ein Aspekt lässt Xalan gegenüber den anderen XSLT-Prozessoren insbesondere hervortreten: Für die Transformation nach XSLT ist es notwendig, das gesamte Dokument gleichzeitig im Zugriff vorliegen zu haben. Andere Prozessoren bilden stark Ressourcen verbrauchend jeden Knoten beim Parsen als Klasse der jeweiligen Programmiersprache ab und beeinträchtigen damit gerade bei größeren Quelldokumenten die Performanz stark. Xalan hingegen erzeugt ein sogenanntes *Document Table Model*, das die Knoten in Array-Form abbildet und somit nicht den Mehraufwand mit sich bringt, den die Erzeugung und Verwaltung von Klassen bedeutet. Xalan unterstützt das SAX[36] und das DOM[37].

Der Erfinder von xt[38] ist James Clark, welcher gleichzeitig die XSLT-Empfehlung ausgearbeitet hat. Xt wird von verschiedenen Stellen als sehr schnell bezeichnet. Kay (2000) äußert sich dazu in folgender Weise: “The speed of James Clark’s code is matched only by the brevity of his documentation” (Kay, 2000, S. 659). Dies weist auf das entscheidende Problem von xt hin: Die Dokumentation – in Form von javadoc – ist äußerst knapp ausgefallen. Diese Aussage gilt auch gerade im Vergleich zu den Vorgenannten Prozessoren. Die XSLT-Spezifikation ist durch xt weitgehend erfüllt; allerdings sind ein paar Beschränkungen vorhanden, welche auf den Webseiten von xt verzeichnet sind. Ein bemerkenswerter Mangel ist hierbei das Fehlen der Implementierung der Namensraum-Achse.

Neben den genannten XSLT-Prozessoren gibt es eine Umsetzung von Microsoft namens *MSXSL*, welche nicht nur den Mangel mit sich bringt, ausschließlich für das Betriebssystem *Windows* geeignet zu sein, sondern ebenfalls nicht quelloffen ist. Einen weiteren XSLT-Prozessor namens *TransforMiix* bietet die Mozilla Organization. Allerdings ist dieser fester Bestandteil des Mozilla-Browsers, und nur durch eigene Erstellung aus dem quelloffenen Code ließe sich eine eigenständige Anwendung ermöglichen.

Unter den hier genannten stechen Saxon und Xalan als quelloffene, gut dokumentierte und die Spezifikation ausreichend erfüllende XSLT-Prozessoren heraus. Xalan gewinnt durch die gleichzeitige Implementierung des DOM- und des SAX-Modells und durch die anhaltend intensive Pflege im Rahmen der Apache Software Foundation. Letztere bedeutet gegenüber Saxon – einem “Ein-Mann-Projekt” – eine größere Entwicklergruppe und damit eine kontinuierliche Qualitätsverbesserung. Aus diesen Gründen erscheint Xalan als der geeignete Kandidat für die Transformation der Metadatenstrukturkomponenten im Rahmen der Identitäten-Infrastruktur.

[36] Simple API for XML
[37] Document Object Model
[38] siehe `http://www.blnz.com/xt/`

4.2.6.1. Xalan

Unter dem Namen "Xalan" entstand ein wichtiger Pfeiler für das Apache XML Projekt. Xalan gibt es in jeweils einer Ausführung für Java und für C++. Hier soll – in Orientierung auf das schon behandelte Jena Framework – nur die Java-Version betrachtet werden.

Xalan ist ein API mit einer detaillierten Paketaufteilung. Die Methoden des Hauptpakets, Xalan Core, parsen ein XSLT-Stylesheet. Das Ergebnis wird von den Methoden des Transformations-Pakets übernommen. Diese wenden es auf ein XML an und stellen ein entsprechendes Ergebnis zur Verfügung – dabei kann es sich je nach verwendeter Methode bei der Eingabe und bei der Ausgabe um DOM, um SAX oder um einen Stream handeln. XPath ist mit einem Paket vertreten, das oben angesprochene Document Table Model und DOM und SAX ebenso jeweils mit einem. Das XSLTC-Paket bietet Möglichkeiten, den Transformationsvorgang nicht interpretierend, sondern kompilierend durchzuführen. Darüber hinaus gibt es noch Pakete für Serialisierung, Utilities und Entwicklung von Erweiterungen.

Die Basis für eine Transformation ist die Erzeugung einer *TransformerFactory*. Durch sie kann ein *Transformer*-Objekt kreiert werden, dem das XSLT-Stylesheet als Parameter übergeben wird. Der Transformer führt dann die Transformation vom Quellbaum zum Zielbaum durch. Der Quellbaum wird entweder als DOM- oder als SAX-Modell übernommen oder aber direkt aus einem Stream (beispielsweise durch das Laden einer Datei) geparst. Der Zielbaum kann dann bei Bedarf serialisiert werden, indem ein *Serializer* durch eine entsprechende Factory mit einem Parameter für das Ausgabeformat erzeugt wird.

Eine Besonderheit des Xalan-Prozessors ist die Integration des Piping-Konzepts: Mit seiner Hilfe können Transformationen leicht verkettet werden, indem der Zielbaum nicht erst in eine XML-Datei serialisiert wird, sondern umgehend als Quellbaum einer weiteren Transformation verwendet werden kann. So kann auch dann umstandslos eine Transformation durchgeführt werden, wenn zwar kein direktes XSLT-Stylesheet vorliegt, aber zwei oder mehr Stylesheets mit verkettbar zusammenpassenden Quellen und Zielen. So können zwei Teiltaxonomien auch bei nicht vorhandener Vorgabe aufeinander abgebildet werden, ohne die Erstellung eines neuen Stylesheets zu erfordern.

4.3. Übertragung

Neben der Ontologie-Organisation und der Transformation ist das dritte wichtige Element für die Kommunikation in einer Identitäten-Infrastruktur die entfernte Wiederverwendung von Teilontologien und von Transformationsregeln für den Einsatz zwischen Teilontologien. Dabei soll es nicht um die Übertragung im technischen Sinne gehen – beispielweise die Frage nach einem geeigneten Protokoll –, sondern vielmehr topologische Aspekte betrachtet werden: Wo werden die Teilontologien und Transformationsregeln vorgehalten, wie wird das Wissen darüber ausgetauscht, wie wird der Zugriff geregelt? Als zwei unterschiedliche Konzepte bieten sich eine zentrale Speicherung auf einem Server an oder die dezentrale Verteilung, die durch die Clients selbst realisiert wird.

4.3.1. Zentrale Ontologien

Eine erste Möglichkeit wäre ein zentraler Speicher für alle Teilontologien. Als einzelnes System wäre seine Funktionstüchtigkeit leicht kontrollierbar. Durch ein solches, welches eine kontinuierliche Anbindung an das Internet hätte, wäre ein ständiger Zugriff gewährleistet – bei entsprechend redundantem Vorhalt von Ersatzkomponenten ließen sich auch notwendige Wartungsarbeiten oder erneuernde Modifikationen ausfallsicher bewerkstelligen.

Ein zentrales System hätte auch positive Auswirkungen auf die Gesamtheit aller Teilontologien. Jede neue Teilontologie könnte vor Aufnahme in das System eine Prüfung durchlaufen, bei der ermittelt würde, an welchem Punkt der Gesamtontologie diese neue Teilontologie eingebunden werden soll und in welcher Beziehung sie zu anderen Teilontologien stehen könnte. Mittels der Möglichkeiten der Inferenzberechnung könnten Widersprüche zur bestehenden Gesamtontologie aufgedeckt und im Vorwege vermieden werden. Durch zentrale Kontrolle würde auch bei Beteiligung vieler Personen am Ontologien-Entwicklungsprozess und dem Zusammenkommen vieler verschiedener Teilontologien eine einheitliche Struktur ermöglicht werden. Entsprechend werden die Transformationsregeln zwischen semantisch verwandten, aber strukturell verschiedenen Teilontologien zentral gespeichert und zur Verfügung gestellt. Dem Anwender stünde so ein vielfältiger Fundus an Teilontologien zur Zusammenstellung seiner individuellen Gesamtontologie zur Verfügung; bei diesem könnte er aber immer davon ausgehen, dass er auf seiner Basis eine korrekte Gesamtontologie erzeugt.

Zentrale Systeme leiden allerdings immer unter dem Problem, ein “Single Point of Failure” zu sein: Sei es ein technisches Versagen, ein Softwarefehler oder ein Sicherheitsproblem – wenn ein zentrales System durch solche Vorfälle betroffen ist, leidet das gesamte System. Der Sicherheitsaspekt stellt für die Anwender allerdings nur bedingt ein besonderes Problem dar: Die Daten der Teilontologien sind ausschließlich Metadaten und nur sehr eingeschränkt als eine solche Art von Daten zu sehen, deren Offenlegung die Privatsphäre beeinträchtigen könnte. Immerhin könnte die individuelle Zusammenstellung an Teilontologien grobe Rückschlüsse auf die Identität ermöglichen. Ansonsten würde sich das Ausnutzen von Sicherheitslücken in Sabotage äußern und wäre in den Auswirkungen mit Softwarefehlern vergleichbar. Während eine Verzögerung des Einstellens einer neuen Teilontologie vermutlich erträglich wäre, würde eine Nutzung unmöglich, wenn Soft- oder Hardwarefehler den Zugriff auf die Gesamtontologie verhindern. Eine Abhilfe wäre eine reguläre Zwischenspeicherung der benötigten Teilontologien auf dem System des Anwenders. Da einmal eingestellte Teilontologien nicht mehr verändert werden, wäre ein erneuter Bezug vom zentralen System nur bei Datenverlust notwendig. Auf diese Weise würde das zentrale System auch erheblich entlastet werden.

Ein letzter kritischer Punkt ist die Frage nach der Zuständigkeit und der Finanzierung für ein solches zentrales System. Allerdings lehrt die Geschichte des Internets, dass Lösungen gefunden werden: Seien es von den Internet-Dienstanbietern mitgetragene oder staatlich finanzierte Dienste, seien es Forschungsprojekte, seien es durch Werbung, Gebühren oder Spenden finanzierte privat organisierte Dienste. Daher soll dieses Problem hier nicht weiter erörtert werden. Ein Aspekt darf in Bezug auf die Zuständigkeit aber

nicht ausgespart bleiben: das Problem der Bestandsgarantie. Sollte der lokale Identitätsmanager auf einem solchen zentralen System aufbauen, wäre ein sinnvoller Betrieb ohne dieses nicht möglich. Das Konzept des Identitätsmanagers bietet das Potenzial dafür, dass ein Anwender einen Großteil seiner Datenhaltung auf der Basis des Identitätsmanagers beziehungsweise dessen Ordnungsstrukturen organisiert. Die Möglichkeit zum Zugriff auf alle anderen – vom Anwender bisher nicht genutzten und daher nicht vorliegenden – Teilontologien und den zugehörigen Transformationsregeln muss also gewährleistet sein. Ein zentrales System müsste demnach eine Perspektive für einen dauerhaften Bestand aufweisen. Dies erschwert die Frage nach Zuständigkeit und Finanzierung erheblich. Die Sicherheit durch ein zentrales System scheint daher nicht gegeben.

4.3.2. Dezentrale Datenhaltung und Datenaustausch

Die Alternative zu einem zentralen System ist die dezentrale Datenvorhaltung. Die schon erwähnte lokale Zwischenspeicherung ist schon ein erster Schritt in diese Richtung. Wenn zudem ein kontrollierter Zugriff auf diese lokalen Zwischenspeicher auch von anderen entfernten Systemen ermöglicht wird und ein Verfahren zum Auffinden der so verstreuten Teilontologien und Transformationsregeln existiert, wäre ein Verzicht auf einen zentralen Speicher möglich. Eine solche verteilte Datenvorhaltung entspricht dem Konzept *Peer-to-Peer*:

> a centralized architecture, which has some drawbacks such as, the participants may experience lengthy delays if they are located far from the central server, and the organization that runs the server must deal with the security and privacy ... Adopting a P2P architecture includes some of the following benefits for software development: (1) the peer (or group) is able to have complete control of its information, (2) groups can share and duplicate information to help users with slow network connections (Bowen und Maurer, 2002)

Bei Peer-to-Peer wird das Paradigma von *Client* und *Server* in der Weise aufgelöst, dass die Clients die Aufgaben des Servers übernehmen, das zentrale System auf das minimal Notwendige – beispielsweise auf ein dynamisches Verzeichnis mit Verweis auf die jeweiligen Clients – reduziert wird oder sogar ganz wegfällt. Im letzteren Fall ersetzen geeignete Suchmechanismen den Verzeichnisdienst und suchen nach anderen Clients, um sich in einen Verbund an Peers anzukoppeln. In beiden Fällen muss der Client mit einem persistent bestehenden Identifikator versehen werden. So ist gewährleistet, dass auch nach Wechsel des Client-Systems oder im Falle dynamisch vergebener IP-Nummern der Client weiterhin erreicht werden kann.

Für den Identitätsmanager, welcher schon mit einem lokalen Zwischenspeicher für Teilontologien und Transformationsregeln versehen ist, bedeutet dies im ersten Fall, bei dem ein zentrales Verzeichnis vorhanden ist, keine besondere Änderung: Statt direkt von einem Server eine Datei zu beziehen, hält der Server einen entsprechenden Verweis auf einen Client vor, der diese Datei in seinem Zwischenspeicher vorliegen hat. Zudem sind

zwei Ergänzungen auf Seiten des Clients notwendig: Er muss eine Schnittstelle anbieten, um die lokal gespeicherten Dateien anderen Clients auf Anfrage zur Verfügung stellen zu können. Er muss sich außerdem jedesmal beim Programmstart dem Server des zentralen Verzeichnisses gegenüber zu erkennen geben, indem er seinen Identifikator dem Server übermittelt, so dass der Identifikator und die aktuelle IP auf Serverseite in Verbindung gebracht werden können. Die Aufgaben des Servers beschränken sich auf drei Bereiche: Er muss die Liste der vorhandenen Teilontologien und Transformationsregeln vorhalten, jeweils mit Verweis auf die Identifikatoren der Clients, die diese Dateien anbieten. Der Server muss zudem eine Zuweisungsliste bieten, welche einen Identifikator mit der jeweils aktuellen IP des Clients in Verbindung bringt. Bei Übernahme einer Teilontologie oder einer Transformationsregel in den lokalen Zwischenspeicher des Clients muss dies mittels seines Identifikators in die Liste des Servers aufgenommen werden, so dass dieser Client von anderen auch angefragt werden kann. Einen Ausfall eines solchen Verzeichnisservers und den gleichzeitigen Bedarf an neuen Dateien, die für eine Kommunikation benötigt werden, könnte dieses System nicht abfangen.

Die Variante ganz ohne Server verlagert auch den Verzeichnisdienst auf die Clients. Allerdings besteht hier ein Initialisierungsproblem, da ein Client zu Beginn keinen anderen Client verzeichnet hat, von dem er weitere Daten erhalten könnte. So muss wenigstens an einer zentralen Stelle ein erster Anlaufpunkt verzeichnet sein, so dass ein Client in einen Peer-Verbund finden kann. Das Peer-to-Peer-Konzept insgesamt bringt Vor- wie auch Nachteile mit sich. Die Gefahr, die vom Ausfall eines Peers ausgeht, ist in den meisten Fällen aufgrund der vorhandenen Redundanz durch andere Peers nicht gegeben. Vielmehr ist es so, dass gerade davon ausgegangen wird, dass die wenigsten Clients kontinuierlich aktiv sind. Es ist allerdings vorstellbar, dass der Versuch eines Clients unternommen wird, mit einem anderen Client auf Basis zweier verschiedener Teilontologien eine Kommunikation aufzubauen und dafür von einer dritten Partei die Transformationsregeln benötigt werden, diese aber von keinem momentan aktiven Peer zur Verfügung gestellt werden. Ein Umgehungsversuch wäre die Suche nach anderen Transformationsregeln, die sich in der Weise verketten lassen, dass die eigentlich gesuchten Regeln nachgebildet werden. Allerdings ist auch hierbei der Erfolg nicht sicher, und zudem besteht bei jeder weiteren Verkettung die Gefahr eines zunehmendes Datenverlustes, da eine Transformation einen solchen immer mit sich bringen kann. Wenn es gar keine zentrale Instanz gibt, ist auch die Gefahr der Teilung des Peer-Netzes gegeben. Dies kann beispielsweise geschehen, wenn ein Peer-Verbund sich in seiner Dynamik so entwickelt hat, dass er zwei Untergruppen herausgebildet, die nur durch einige wenige Peers verbunden sind. Koppeln diese wenigen Peers sich vom System ab, so wird aus den beiden Untergruppen je ein eigenständiger Peer-Verbund. Wenn auf diese Art beispielsweise die Hälfte der zuvor beteiligten Peers plötzlich nicht mehr erreichbar ist, dann besteht eine gute Möglichkeit, dass dieser Vorfall nicht mehr von den Redundanzen aufgefangen wird. Ein nicht zu übersehender Aspekt ist der größere programmiertechnische Aufwand, der für den Aufbau eines Peer-to-Peer-Systems notwendig ist.

4.3.3. Peer-to-Peer-Frameworks

Da die Firma Sun für Java recht schnell das Peer-to-Peer-Konzept als Trend entdeckt hat, konnte das Unternehmen im Jahr 2001 die entsprechende Erweiterung *JXTA*[39] herausgeben. Mittlerweile hat Sun JXTA mit einer Open Source Lizenz versehen und die Leitung dem unabhängigen JXTA-Projekt übergeben.

JXTA ist als Framework nach Gong (2002) in drei Ebenen aufgebaut und versucht damit das Spektrum der heute bestehenden Peer-to-Peer-Konzepte abzudecken: Die unterste Ebene *JXTA Core* baut auf einem einzelnen Client oder – eben treffender – dem Peer auf und beinhaltet die unbedingt notwendigen Komponenten für ein Peer-to-Peer-System. Ein Peer selbst trägt einen Identifikator nach dem *UUID*-Standard[40] und ist somit eindeutig identifizierbar. Die unterste Ebene definiert *Peer Groups* als virtuelle Einheiten aus Peers, die ihren Zusammenhalt aus den gemeinsam genutzten Protokollen beziehen. Als Empfänger und Sender sind *Pipes* an einen Peer gebunden, die einen asynchronen Nachrichtenaustausch ermöglichen. All diese Ressourcen können durch *Advertisements* beschrieben und öffentlich gemacht werden. Als weitere Komponente bietet das *Peer Monitoring* die Möglichkeit für eine Beschreibung über bestimmte Zustände des Peers. Die mittlere Ebene *JXTA Services* umfasst alle Formen von Basisdiensten, die in einem Peer-to-Peer-Netzwerk üblich, aber grundsätzlich optional sind; zu diesen zählen beispielweise Indizierung und Suche. Die obere Ebene *JXTA Applications* bietet beispielhafte Applikationen: Instant Messaging, File und Resource Sharing oder Auktionen. Definiert ist JXTA auf einer Reihe von Kommunikationsprotokollen. Diese dienen dem Auffinden von anderen Peers und deren Advertisements, dem Austausch von Informationen über die Peers, die Anbindung an eine Peer Group oder dem Versenden von Nachrichten.

Neben dem sehr präsenten JXTA ist es schwer, ein alternatives Peer-to-Peer-Framework für Java zu finden. Es gibt Forschungsprojekte wie *P2P-Net*[41], die für spezielle Aufgaben – hier eine Simulationsumgebung – entwickelt werden, um Peer-to-Peer-Netze untersuchen zu können. Schon aufgrund dieses Mangels an Alternativen aber auch aufgrund der offenen und standardprägenden Strategie[42] ist JXTA das geeignete Framework, um den Identitätsmanager als eine Peer-to-Peer-Komponente zu gestalten.

Allerdings ist für eine Untersuchung des Konzeptes von verteilten Ontologien und Transformationsregeln die Art der Verteilung und die Form des Zugriffs zweitrangig. Aufgrund dessen soll der Aspekt der Übertragung hier nicht näher erörtert werden.

4.3.4. Kommunikationsprotokolle

Durch die Topologie der Identitätsarchitektur ist allerdings noch nicht die Form der Nachrichten festgelegt. Gesucht ist ein Protokoll auf der Anwendungsebene, welches das

[39] Bei dem Begriff JXTA handelt es sich nicht um eine Abkürzung im eigentlichen Sinne, sondern um eine Kunstkurzform des englischen Juxtapose.
[40] Der Universally Unique Identifier ist standardisiert von der Open Software Foundation
[41] siehe `http://wwwiti.cs.uni-magdeburg.de/iti_dke/p2p/`
[42] siehe JXTA-Homepage `http://www.jxta.org`

Format von Nachrichten definiert. Wünschenswert – aufgrund der unkomplizierten Erstellung entsprechender Anfragen aus den bestehenden Ontologien – ist ein Protokoll auf Basis von XML. Dieser Forderung entspricht die Definition von Webservices:

> Ein Webservice ist ein Dienst, der mit Hilfe von XML auf der Basis von Internet-Netzwerkprotokollen erbracht wird. (Wikimedia Foundation, 2004b)

Dabei soll hier nicht auf die vielfältigen Definitionen des Begriffs Webservice[43] eingegangen werden, da nur der dafür empfohlene Standard zur Kommunikation zur Anwendung kommen soll. Die vom W3C veröffentlichte Empfehlung für Kommunikation und somit auch für die entsprechende Formatierung von Nachrichten trägt den Namen *SOAP*.

Ohne auf die vielen Möglichkeiten von SOAP genauer einzugehen, sei es hier nur als Hülle für die zu übertragenden Daten verwendet. Genau diese bietet der SOAP *Envelope*. Dieser setzt sich zusammen aus *Head* und *Body* – allerdings sind diese Festlegungen nicht weiter spezifiziert und der Head ist optional. Hier sei aber nur eine simple Variante verwendet. Es sei ein Schema zu Grunde zu legen, welches vier verschiedene Elemente umfasst: Das einleitende *ontcom* umschließt alle anderen Elemente und legt den Namensraum für die ontolgiebehaftete Kommunikation fest. *communication* ermöglicht mittels des Attributes *type* die Differenzierung zwischen Anfrage und Antwort. Entsprechend sind die vorgesehenen Werte *request* und *response*. *identifier* beherbergt das Attribut *urn*, welches entweder die Teilontologie, eines ihrer Identitätsattribute oder die Quell-Teilontologie im Falle einer Abbildung identifiziert. *content* spezifiziert durch das Attribut *type* den gewünschten oder den übermittelten Inhalt. Dabei kann es sich um eine Teilontologie ("ontology") oder um eine Abbildung ("mapping") handeln. Optional enthält es – für den Fall der Belegung des Attributs type des Elementes *communication* durch den Wert *response* – entweder eine Teilontologie oder eine Abbildung. Quellcode-Beispiel 5 stellt beispielhaft die Anfrage nach einer Teilontologie mit gegebenem URN dar.

Es existieren laut der Apache Software Foundation (2004) für SOAP unter Java diverse Implementierungen verschiedener Hersteller – darunter IBM, BEA, Borland, Oracle und Macromedia. Jene von IBM, Borland und Macromedia basieren auf der Apache eigenen Implementierung namens *Axis*. Aufgrund der Quelloffenheit und der starken Verbreitung und Anerkennung sei dies auch die Basis für die Erzeugung der Nachrichten für den Austausch von ontologiebasierten Nachrichten.

[43] siehe hiezu `http://www.jeckle.de/webServices/index.html`

Quellcode-Beispiel 5 Beispiel für eine Anfragenachricht

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env
      ="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <onefc:ontcom xmlns:onefc
         ="http://onefc.org/ontology-communication">
      <onefc:communication type="request"/>
      <onefc:identifier urn="596162616461626 14A7874615
          0325033F55704B199754D3E9076A2947775A6EE03"/>
      <onefc:content type="ontology"/>
    </onefc:ontcom>
  </env:Body>
</env:Envelope>
```

5. Ausarbeitung eines Ontologiensystem-Frameworks

Die im Kapitel 3 erstellten Konzepte finden nun mittels der in Kapitel 4 untersuchten Techniken in einen Designentwurf Eingang. Schwerpunkt seien die Organisationsstrukturen sämtlicher Identitätsdaten und die Möglichkeiten zur Transformation zwischen verschiedenen Formen von Organisationsstrukturen. Der Aspekt des Transports steht demgegenüber stärker in Verbindung mit dem Gesamtsystem einer Identitäteninfrastruktur, in das die Implementierung dieses Entwurfs integriert wird. Abhängig von diesem könnten sich jeweils verschiedene Varianten als vorteilhaft erweisen. Das Konzept der verteilten Teilontologien und ihrer Transformationen ist aber durch die Wahl der Übermittlungsform nicht direkt betroffen, sondern kann unabhängig entwickelt werden.

Im weiteren Verlauf wird mehrfach auf Dateien, Bäume oder andere Präsentationsformen von OWL und XSLT Bezug genommen. Wenn die Präsentationsform aber noch nicht sicher oder an der Stelle der Erwähnung nicht entscheidend ist, wäre die Nennung einer Präsentationsform willkürlich und für den weiteren Verlauf einschränkend. Um dieser Situation entgegenzutreten, sei eine noch unbekannte Präsentation in einer der benannten Sprachen nur mit dem Sprachnamen benannt. Dies führt beispielsweise zum – ansonsten umgangssprachlich erscheinenden – Ausdruck “Das OWL” für eine OWL-Datei respektive einen OWL-Baum.

5.1. Gesamtidee

Die Verwaltung der Teilontologien ist von der Verwaltung und Anwendung der Transformationsregeln weitestgehend unabhängig, da letztere auf die Kommunikation angewendet werden und nicht auf die Teilontologien selbst. Wenn die jeweiligen Funtionalitäten in keiner Weise voneinander abhängen, ist eine recht unabhängige Entwicklung der beiden Konzepte voneinander denkbar. Es ist aber festzustellen, dass ein Anteil der Verwaltungsaufgaben – die Verwaltung von Teilontologien wie von Transformationsregeln betreffend – ähnlich geartet ist. So sind sicherlich zwei unterschiedliche Verwaltungskomponenten zu erstellen; die Gemeinsamkeiten durch entsprechende Vererbungsverbindungen festzulegen, bleibt zu bedenken.

Diese zwei Komponenten sind der *Ontology Manager* und der *Mapping Manager*. Für beide gilt, dass nur je eine Instanz von ihnen aktiv sein sollte. Mehr als eine Instanz würde keinen erkennbaren Vorteil bieten, hingegen diverse Nachteile, gerade in Hinblick auf Persistenzerhalt und Kommunikation mit der Netzwerkschnittstelle. Die Implementierung erfolgt demnach unter Verwendung des *Singleton* Entwurfsmusters nach Gamma

(1994). Der Ontology Manager ist die zentrale Komponente zur Organisation und Verwendung der Teilontologien des Anwenders. Die Aufgaben, die zum Bereich des Ontology Managers zählen, sind in Tabelle 5.1 aufgelistet. Dies sei eine Vorüberlegung, die noch keine Festlegung bezüglich der Klassen-Organisation beinhaltet.

Aufgaben des Ontology Managers
Lokale und verteilte Suche von Teilontologien
Import gefundener neuer Teilontologien
persistente Sicherung von Teilontologien
Einlesen gesicherter Teilontologien in den Arbeitsspeicher
Annahme und Auswertung von Suchanfragen nach Teilontologien
Vergleich von Teilontologien anhand eines Identifikatoren
Übermittlung eigener Teilontologien
Aufruf des Mapping Managers bei fehlender Teilontologie

Tabelle 5.1.: Aufgaben des Ontology Managements

Der Mapping Manager ermöglicht erst die sinnvolle Verbindung zwischen zwei unterschiedlich strukturierten aber semantisch verwandten Teilontologien. Tabelle 5.2 stellt die Aufgaben für den Mapping Manager dar. Die Verbindungen oder Verknüpfungen zweier Teilontologien seien für den weiteren Verlauf einheitlich als Mapping bezeichnet. Erneut handelt es sich hierbei um eine Vorüberlegung, die noch keine Festlegung bezüglich der Klassen-Organisation beinhaltet.

Aufgaben des Mapping Managers
Lokale und verteilte Suche nach Mappings
Import gefundener neuer Mappings
persistente Sicherung von Mappings
Einlesen gesicherter Mappings in den Arbeitsspeicher
Annahme und Auswertung von Suchanfragen nach Mappings
Vergleich von Mappings anhand eines Identifikatoren
Übermittlung angefragter Mappings
Mapping von Kommunikationsfragmenten

Tabelle 5.2.: Aufgaben des Mapping Managements

Ein Aufgabenvergleich dieser beiden Komponenten erbringt eine Reihe von Gemeinsamkeiten. Von daher empfiehlt sich eine Umsetzung der beiden Manager als Klasse, die beide von einer gemeinsamen Oberklasse die verwandten Methoden erben.

Als weitere notwendige Komponenten zur Integration in einen Identitätsmanagement-Client sind Möglichkeiten zur Erstellung der Teilontologien und der Mappings zu bedenken, sowie eine geeignete Netzwerkschnittstelle. Für den Bereich der Erstellung von

Ontologien existieren diverse, darunter auch einige frei verfügbare und quelloffene Implementierungen. Passend in diesem Zusammenhang ist das Programm *Protégé*[1] mit einem OWL-Plugin, welches auf dem Jena API basiert. Als Alternative denkbar wäre auch die angekündigte Version 4.0 des dann ebenfalls auf dem Jena API aufbauenden Programms *SMORE*[2]. Mit ersterem und mutmaßlich auch mit letzterem Programm lassen sich geeignete Teilontologien erstellen. Es besteht allerdings bei diesem Vorgehen der Teilontologie-Erstellung natürlich keine Kontrollmöglichkeit, welche die Entsprechung zu den Konzepten dieser Arbeit gewährleistet. Zu diesem Zweck sei der gewählte Ontologie-Editor entsprechend abzuwandeln.

Die Erstellung eines Mappings in XSLT ist um einiges komplexer als die einer Teilontologie. Das Konzept eines Mappings ist zwar relativ simpel, aber die Erstellung einer XSLT Transition kaum für einen durchschnittlichen Anwender zumutbar; mit ihr wird schließlich über ein funktionales Konzept in den Kommunikationsfluss der Identitäten-Infrastruktur eingegriffen. Notwendig ist hier eine allgemeine Schnittstelle, welche eine simple Darstellung zur Eingabe erwartet, um diese dann in das benötigte XSLT umzuwandeln. Neben der direkten Implementierung im Quellcode bietet sich für diese Aufgabe ebenfalls XSLT an: Hierfür ist nur die einmalige Erstellung eines *Metamappings* erforderlich. Der Vorteil ist die flexible Konfigurierbarkeit ohne Eingriff in den Quellcode. Neben der Flexibilität des Eingabeformats lässt sich auf diese Weise auch festlegen, wie beispielsweise ein Mapping umgesetzt werden soll, bei welchem die Quelle zwei Attribute aufweist, das Ziel aber nur eins. Für die Erstellung dieses Metamappings bietet sich eine geeignete XSLT Entwicklungsumgebung an. Quelloffen und auf Xalan aufbauend bietet sich das Programm *Treebeard*[3] an.

Die Entscheidung für eine bestimmte Netzwerkschnittstelle hängt ab von der Entscheidung für eine Infrastruktur im Identitäten-Netzwerk. Bei Verwendung eines serverlosen Peer-to-Peer-Konzeptes bietet sich die Verwendung des schon genannten JXTA-Frameworks an; bei serverbasierten Konzepten ist eine dementsprechend anders geartete Komponente zu verwenden. Wesentlich in diesem Rahmen sei insbesondere die Festlegung einer wohldefinierten Schnittstelle, die es möglich macht, verschiedene Komponenten zu integrieren.

5.2. Teile und herrsche: Teilontologien

Die Forderung aus Abschnitt 3.1.3 nach Modularisierung der Identitätsdatengesamtstruktur in Metadatenstrukturkomponenten wird durch den in Abschnitt 4.1.3.4 dargestellten Weg OWLs erfüllt, die Klassen und Eigenschaften per URI zu verknüpfen. Dadurch kann für jede Klasse und jede Eigenschaft eine beliebige andere Klasse – respektive eine beliebige andere Eigenschaft – als Superklasse festgelegt sein.

Durch die Aufteilung der Gesamtontologie in Teilontologien muss der Frage nachgegangen werden, welchen Umfang eine Teilontologie aufweisen sollte. Die Möglichkeiten

[1] siehe `http://protege.stanford.edu`
[2] siehe `http://www.mindswap.org/~aditkal/editor2.shtml`
[3] siehe `http://treebeard.sourceforge.net`

beginnen bei einem OWL-Element und sind nach oben hin offen. Ein entscheidender Faktor spielt allerdings in diese Frage hinein und verhindert eine konkrete Antwort: Die Individualität der Ersteller. Aufgrund der bewussten Offenheit des Konzeptes zur Erstellung der Teilontologien ist hier keine sinnvolle Kontrolle oder Einschränkung im vornherein möglich oder sinnvoll – gerade zum jetzigen Zeitpunkt der Ausarbeitung lässt sich schwerlich über extreme, möglicherweise aber sinnvolle Varianten spekulieren. Es bleibt, den Anwender ausreichend zu informieren, auf dass er eigenverantwortlich die passenden Grenzen für eine zu erstellende Teilontologie wählt.

Verbunden werden alle einzelnen Teilontologien durch die Kernontologie. Aufgrund der nach OWL Spezifikation allen Teilontologien zugrunde liegenden Superklasse "owl:Thing"[4] ist dies zwar nicht zwangsläufig notwendig; es ermöglicht aber allgemeine Bestimmungen für alle lokalen Ontologieklassen. Die Kernontologie ist in der Taxonomie der Teilontologien die oberste Teilontologie, die jedem Identitätsmanagement-Client mitgegeben wird. In ihr sind ausschließlich die allgemeinste Meta-Attributklasse *identity-attribute* und die allgemeine Eigenschaft *readable-by* definiert. Die Eigenschaft *readable-by* ermöglicht für jedes Attribut jeder Klasse die Nennung einer Liste an Identitäten, die dieses Attribut auslesen dürfen. Durch diese Eigenschaft unterstützt die Gesamtontologie die Aspekte der Privatsphäre der Identität.

Eine Teilontologie muss aus verschiedenen Komponenten bestehen. Den Kern bildet mit ihren verknüpften Klassen die ontologische Struktur selbst. Eine dieser Klassen ist die Wurzelklasse dieser Teilontologie, von der alle anderen Klassen Nachkommen sind. Wichtig ist der Verweis auf alle wesentlichen Namensräume für die zu verwendenden Sprache OWL und RDF inklusive Schemaerweiterung RDFS. Da verschiedene Teilontolgien unterschiedlicher Herkunft ohne zentrale Kontrolle leicht identische Namen erhalten haben könnten, ist als drittes wesentliches Element ein immer eindeutiger Identifikator notwendig.

5.2.1. Erstellung und Bereitstellung eigener Teilontologien

Ein einheitlicher und eindeutiger Identifikator ermöglicht eine erfolgreiche Suche im lokalen und entfernten Bereich. Für die Aufgabe der Lokalisierung stellt sich der *Unified Resource Locater URL*[5] zunächst einmal als geeignet dar: Er verbindet eine weltweit eindeutige Identifizierung mit der Funktion des Lokalisierens. Allerdings sind im Falle des Einsatzes von Peer-to-Peer-Technologie – sei es servergestützt oder serverlos – für den Vorhalt von Teilontologien die einzelnen Clients der Anwender – zumindest im privaten Bereich – in den seltensten Fällen mit einem URL zu ermitteln: Meist wird ihnen nur eine dynamische IP zugewiesen, was zu gar keinem oder nur einem vorübergehend zugewiesenen URL führt.

Wichtig für einen Identifikator in einem dezentralen System ist die Gewährleistung von Eindeutigkeit trotz des Mangels an zentraler Prüfung derselben. Für diesen Fall eignet sich der *Universal Unique Identifier UUID* der Open Group (1997): Er wurde speziell für den Einsatz in verteilten Systemen – genauer für den *remote procedure call* – entwickelt.

[4] siehe Spezifikation unter `http://www.w3.org/TR/owl-ref/`

[5] spezifiziert im RFC 1738 (12.1994), siehe `http://www.faqs.org/rfcs/rfc1738.html`

Für Java bietet das JXTA Projekt eine passende Implementierung[6], da dieses als Peer-to-Peer-Framework von vornherein auf ein Identifikatoren-Konzept in einem dezentralen System angewiesen war. Die Ontologieklassen referenzieren ihre jeweiligen Superklassen allerdings per URI. Dieser Konflikt lässt sich auflösen, indem ein URI verwendet wird, der als einen Bestandteil einen UUID beinhaltet. Genau diese Lösung findet auch im JXTA Framework Anwendung: Die dortigen Identifikatoren für Peers, *Peergroups*, *Contents* und anderen Entitäten – allgemein als JXTA-IDs bezeichnet – weisen nach Gong (2002) genau eine solche Form auf:

```
urn:jxta:uuid-
59616261646162614A78746150325033F55704B199754D3E9076A2947775A6EE03
```

Als Identifikatorenkonzept findet hier der *Unified Resource Name URN* Anwendung. Dabei handelt es sich – genau wie beim URL – um eine Untermenge der Menge aller URIs[7]. Diese Untermenge eignet sich genau für die Verwendung in JXTA: Ein URN ermöglicht gerade keine Lokalisierung, aber eine eindeutige Identifizierung:

> Peer identifiers uniquely identify the peer on the JXTA network, and do not change from one session to the next. (Khan, 2004)

Ein solcher Identifikator wird jeweils für jede Teilontologie benötigt. Die Lokalisierung ist an anderer Stelle vorzunehmen und wird im Abschnitt 5.2.2 genauer untersucht. XML (und damit auch OWL) bietet keine standardisierte Möglichkeit zur Angabe eines Identifikators für das XML-Dokument an sich. Beim Bezug eines neuen OWL muss dieses aber identifizierbar sein. Es muss demnach entweder ein darüberliegendes Format erdacht werden, welches OWL und Identifikator integriert. Dies wäre dann aber nicht mehr standardisiert. Die Alternative ist die Nutzung eines geeigneten Elements innerhalb des OWL. Da mittels des Identifikators nicht nur das OWL selbst, sondern auch alle Ontologieklassen darin angesprochen werden können sollen, bietet sich der Eintrag als Namensraumangabe an. Wenn auch diese nicht für Identifikatoren gedacht sind, ist dies doch ein gangbarer Weg, ohne die Standardkonformität zu brechen oder ein zusätzliches Format einführen zu müssen. Der Namensraum sollte aber nicht wie im obigen Beispiel mit "urn" benannt werden. Eine solche Wahl suggerierte die endgültige Beschränkung, auf URL-Einträge zu verzichten. Es ist durchaus denkbar, einige schon bestehende Ontologien in das vorgestellte Konzept zu integrieren. Es sei daher weiter die allgemeinere Form des URI beibehalten. Um einem menschlichen Leser oder Ersteller nicht zu verwirren, werde zudem für den Namensraum nicht "uri", sondern der abgewandelte Bezeichner "urid" verwendet. Neben dem so selbstgesetzten "urid" stattet das Jena Framework bei der Erzeugung ein OWL automatisch mit den notwendigen und einigen optionalen Namensräumen aus: RDF, RDFS, OWL und dazu noch *RDF Site Summary RSS*, die *Jena Model Specification JMS* – sie bietet eine Möglichkeit zur Serialisierung der internen

[6] implementiert durch der Klasse net.jxta.impl.id.UUID

[7] zum Verhältnis zwischen URN, URL und URI siehe auch `http://de.wikipedia.org/wiki/Uniform_Resource_Identifier` (Version vom 13.11.2004)

Quellcode-Beispiel 6 Eine einfache Ontologieklasse für Identitäten

```
<owl:Class rdf:ID="Postkontakt">
  <rdfs:subClassOf rdf:resource="urn:onefc:uuid-
59616261646162614A787
      46150325033F55704B199754D3E9076A2947775A6EE03#Postkontakt"/>
</owl:Class>
```

Form eines Modells des Jena Frameworks –, vCard, DAML für DAML+OIL und DC für Dublin Core.

An die einleitenden Namensraum-Deklarationen schließt der Hauptteil des OWL an: die Ontologieklassen-Definitionen. Im Gegensatz zu vielen anderen XML-Sprachen wird bei OWL[8] die sprach-immanente Sruktur nur zu einem geringen Teil mittels der durch XML gegebenen Techniken beschrieben. Die Verknüpfungen der Ontologieklassen untereinander (Eltern-Kinder-Beziehung) und Querverbindungen (beispielsweise Relationen) werden stattdessen allesamt durch entsprechende Verweise auf die jeweiligen IDs vorgenommen. Ein OWL für eine Identitäten-Infrastruktur besteht daher aus einer Auflistung an Ontologieklassen – ohne weitere Verschachtelung – wie in Quellcode-Beispiel 6.

Die Erzeugung eines OWL-Modells im Jena-API wird durch die Methode *createOntologyModel* der Klasse *ModelFactory* erreicht. Diese gibt ein Objekt der Klasse *OntModelImpl* zurück. Um die entsprechende Erweiterung zur Nutzung des URI als Identifikator zu ermöglichen, muss OntModelImpl durch Methoden erweitert werden, die die entsprechende Funktionalität anbieten: *getNamespaceURI* und *getNamespaceAsURI* zur Ermittlung des Identifikators als Typ *String* und URI. Um weiterhin nahe am Modell von Jena zu bleiben, sollte die Erzeugung dieser erweiterten Klasse mit Namen *DefaultOntology* weiterhin durch die Klasse ModelFactory erfolgen. Diese muss also ebenfalls entsprechend erweitert werden. Allerdings ist ModelFactory mit dem Attribut final versehen, was das Erben von dieser Klasse verhindert. Eine Umgehung wäre das Kopieren und Modifizieren der Kopie oder die Erstellung einer Wrapper-Klasse, welche die Klasse ModelFactory kapselt und für sämtliche ihrer Methoden eine gleichnamige Methode enthält, die jeweils nur die Originalmethode entsprechend aufruft. Diese Wrapper-Klasse *OntologyBuilder* lässt sich unproblematisch durch zusätzliche Methoden ergänzen.

Die Erstellung einer Teilontologie für eine Identität innerhalb des hier beschriebenen Identitätsmanagement-Systems nutzt die Methode *createOntologyModel* der Klasse *OntologyBuilder* um ein leeres OWL-Modell als Instanz der Klasse *DefaultOntology* zu erhalten. Das OWL wird aus einer Datei mit der Methode *loadOntology* des Ontology Managers geladen. Diese Methode meldet das so erstellte Modell mit der Angabe des URI als Identifikator und des Dateinamens bei den Verwaltungsobjekten an: Der URI wird als Namensraum-Parameter dem Modell entnommen; URI und Dateiname werden in jeweils eine *Hashtable* eingetragen – *repositoryURI* und *repositoryFile* – und jeweils zusammen in einem *Properties* Objekt namens *uriFileMapping*, um diese Zugehörigkeit persistent speichern zu können.

[8] genau wie auch schon bei RDF; siehe auch die Beispiele 1, 3 und 4

Neben den Klassen lassen sich mit Properties mehrere Daten an eine Klasse anknüpfen. Dies ist wichtig für die in Abschnitt 3.1.1 eingeführten Metadatenkomplexe. Die zwei verschiedenen Arten von Properties sind *ObjectProperties*, welche jeweils eine Relation zu einer anderen Klasse darstellen, und *DatatypeProperties*. Bei letzteren handelt es sich um simple Datentypen nach XML Schema Datentypen[9] wie beispielsweise Zeichenkette oder Ganzzahl. Beide Arten von Properties könnten zur Umsetzung der Eigenschaftsdarstellung für Metadatenkomplexe genutzt werden. Der Vorteil der ObjectProperties ist die Möglichkeit, mittels ihrer Vererbungshierarchie die Eigenschaften in einen semantischen Kontext stellen zu können. Damit werden die Eigenschaften selbst zum Teil einer Taxonomie. Allerdings ist diese nur unter speziellen Bedingungen Teil der vorhandenen Identitätsontologie. Die syntaktisch-schematischen Bestandteile einer Klasse müssten an anderer Stelle der Ontologie schon erstellt worden sein. In den meisten Fällen existieren diese noch nicht und es bliebe dem Anwender überlassen, diese zu erstellen. Vom Standpunkt der Nutzbarkeit her ist die Forderung abzulehnen, für jede neue Klasse mehrere neue taxonomische Einbindungen vornehmen zu müssen und diese Taxonomien teilweise erst erstellen zu müssen. Diese neuen Taxonomien würden auch wieder die Suche nach weiteren Teilontologien mit sich bringen. Und für die Klasseneigenschaften dieser neuen Taxonomien gelte wiederum das gleiche Problem, was eine Kaskade an zu integrierenden Teilontologien bedeutete. Für jede dieser zusätzlichen Teilontologien müsste zudem eine geeignete Abbildung bestehen. Auch von der technischen Seite ist dies demnach keine sinnvolle Lösung. Zuletzt sind die Klassen dieser neuen Teilontologien auch nicht unbedingt als Subklassen – weder direkte noch indirekte – der Wurzelklasse "Identitätsattribut" definierbar.

Ein Ausweg aus diesem Anbindungs- und auch Komplexitätsproblem ist das Anlegen einer zweiten Wurzelklasse "Ding" für alle Eigenschaften. Diese Möglichkeit würde die Komplexität für den Anwender vermeiden, da alle Klassen, die für die Angaben der Eigenschaften der Identitätsontologieklassen notwendig wären, direkt von dieser zweiten Wurzelklasse "Ding" erbten. Eine solche Umsetzung erzeugt eine semantisch korrekte Gesamtontologie. Jedoch ginge der semantische Mehrwert verloren, der durch die Verwendung der ObjectProperties ja erreicht werden soll.

Als grundlegende Alternative bietet sich der Verzicht der ObjectProperties und die Verwendung von DatatypeProperties an. Die simplen Datentypen sind allgemein bekannt und stärken somit die Anwenderfreundlichkeit gegenüber individuellen Klassen. Gerade Zeichenketten sind auch flexibel und bieten damit Kompatibilität insbesondere beim Abbildungsvorgang. DatatypeProperties bieten allerdings ebenfalls keinen semantischen Mehrwert. Der relevante Unterschied ist die fehlende Option auf semantische Verknüpfung. Jede DatatypeProperty stellt somit eine ontologische Sackgasse dar. Dies ist unter Berücksichtigung der Perspektive der Identität aber vertretbar.

Einzige Ausnahme sei eine in der Kernontologie verankerte und dadurch von allen Klassen geerbte Eigenschaft *readable-by*, welche den Aspekt der Privatsphäre und des Datenschutzes berücksichtigt. In dieser Eigenschaft lassen sich die Identifikatoren aller Kommunikationspartner festhalten, denen ein Zugriff auf das jeweilige Datum erlaubt

[9] siehe `http://www.w3.org/TR/xmlschema-2/`

Quellcode-Beispiel 7 Eigenschaft zur Ontologieklasse aus Quellcode-Beispiel 6

```
<owl:Property rdf:ID="Postleitzahl">
  <rdfs:domain rdf:resource="#Postkontakt"/>
  <rdfs:range rdf:resource="&xsd;string"/>
  <owl:sub-property-of rdf:resource
      ="core-ontology#readable-by"/>
</owl:Property>
```

sein soll. Somit werden die Eigenschaften wie in Quellcode-Beispiel 7 umgesetzt[10]. Die Eingliederung erfolgt auch hier allein durch RDF-Darstellungsformen durch Anbindung mithilfe des Elements *rdf:domain*. Die XML-Struktur bleibt fast planar.

5.2.2. Bezug und Import fremder Teilontologien

Es gibt verschiedene Situationen, die einen Anwender dazu veranlassen können, seine Gesamtontologie durch eine weitere Teilontologie zu erweitern. Eine solche Übernahme einer Teilontologie kann über zwei Wege erfolgen: Einerseits durch die Suche über seinen Identifikator, andererseits durch die Suche nach einem Begriff oder Stichwort. Diese zwei Wege für die Integration einer neuen Teilontologie unterscheiden sich durch die Motivation des integrierenden Anwenders und möglicherweise in der zeitlichen Gegebenheit. Ein Anwender, der mittels Stichwortsuche seine Gesamtontologie aus eigener Motivation erweitern möchte, plant dafür vermutlich einen geeigneten Zeitraum ein und kann diesen Vorgang bei Bedarf auch abbrechen, ohne dass durch das Fehlen dieser Erweiterung direkte Konsequenzen entständen. Ein Anwender jedoch, der aufgrund einer aktuellen Kommunikationssituation seine Ontologie erweitern muss, steht eventuell unter dem Zwang, diese Erweiterung vorzunehmen, um die gewünschte Kommunikationssituation weiterführen zu können. Hier kann es auch leicht zu einer Situation mit Zeitdruck kommen, und die Motivation ist nicht aus dem eigenen Wunsch geboren, die Gesamtontologie zu erweitern.

Um die Suche nach einem Stichwort zu ermöglichen, ohne sämtliche Teilontologien jeweils analysieren zu müssen, ist die Anlage eines Verzeichnisses der Klassennamen und der Eigenschaftsnamen für jede neu erstellte oder importierte Teilontologie sinnvoll. Dieses wird in eine Gesamtliste eingetragen, die zu jedem vorhandenen Begriff den URI der entsprechenden Attributsklasse oder -eigenschaft innerhalb einer passenden Teilontologie ermitteln kann. Abhängig von der Organisationsform der Identitäten-Infrastruktur wird dieses Verzeichnis als JXTA-Service anderen Peers zur Verfügung gestellt, oder dieses Verzeichnis beziehungsweise eine Information darüber wird an den zentralen Server übertragen.

Für den Fall, dass eine aktuelle Kommunikationssituation eine neue Teilontologie erfordert, ist aber immer davon auszugehen, dass der Kommunikationspartner Daten nach einer Teilontologie verlangt, die ihm selber vorliegt. Diese Forderung wird durch die

[10] "core-ontology" sei hier zunächst nur als Platzhalter für eine geeignete Kernontologie verwendet.

Übermittlung des Identifikators der erwarteten Teilontologie deutlich. Hatte der Anwender diese Daten bisher nicht als Identitätsattribute eingepflegt, so wird er auch keine semantisch passende Teilontologie vorliegen haben. Die Beurteilung, ob diese Attribute Teil seiner Identität sind, obliegt allerdings dem Anwender selbst. Und selbst er kann vor dem Hintergrund der Möglichkeit der Vielsprachigkeit der Bezeichnungen innerhalb der Teilontologien diese nur eingeschränkt vornehmen.

Eine Anfrage nach dem Identifikator erfolgt aus genau einer Situation heraus: Innerhalb einer Kommunikation übermittelt der Kommunikationspartner die Anfrage nach einem Attribut oder die Antwort mit einem Attribut inklusive seines URI. Wenn weder die Teilontologie dieses Identitätsattributes beim Ontology Manager eingetragen ist, noch sich ein entsprechendes Mapping zu einer lokal vorhandenen Teilontologie finden lässt, dann muss die entsprechende Teilontologie mittels des URI beschafft werden. Zunächst ist zwar davon auszugehen, dass der anfragende beziehungsweise antwortende Kommunikationspartner selbst diese Teilontologie zur Verfügung stellen könnte; das Konzept schreibt aber nicht zwangsläufig eine synchrone Kommunikation vor, so dass nicht gewährleistet ist, dass die Teilontologie auf diesem Wege umgehend erhältlich wäre. Ein weitere Aspekt ist die Frage nach Privatsphäre: Wieviel Information gibt ein Anwender ungewollt preis, wenn er eine Teilontologie von seinem Kommunikationspartner einfordert? Daher ist die Weiterleitung an die Netzwerkschnittstelle als von der bestehenden Kommunikation unabhängige Anfrage nach der betreffenden Teilontologie vorzuziehen. Auch im Sinne einer klaren Schnittstelle gegenüber dem Identitätsmanagement-Client stellt sich dieses Vorgehen als nützlich dar: Der Client richtet die Anfrage oder Antwort nach einem passenden Mapping an den Mapping Manager mit dem Parameter “URI des Attributes”. Der Mapping Manager übernimmt die Aufgabe, zu klären, ob kein Mapping nötig ist – weil die Teilontologie vorliegt –, ein passendes Mapping gefunden werden kann oder eine neue Teilontologie importiert werden muss.

Hier sei nun dieser Import näher betrachtet: Die entscheidende Orientierung gibt der Identifikator der zu importierenden Teilontologie. Dieser Identifikator befindet sich immer in der Namensraum-Angabe zum URI der gesuchten Teilontologie. Bei jenen dem System bekannten Teilontologien ist er zudem in den drei in Abschnitt 5.2.1 genannten Verwaltungsobjekten eingetragen: *repositoryURI* für die Identifizierung und *repositoryFile* für die Lokalisierung aller geladenen Teilontologien und dem persistenten *uriFileMapping* für die Lokalisierung aller als Datei abgespeicherten Teilontologien. Bei Abfrage über den URI gibt *repositoryURI* das entsprechende *DefaultOntology* Objekt oder *null* zurück. Im letzteren Fall wird die Anfrage an das *uriFileMapping* weiter gereicht, welches den Dateinamen der gesuchten Teilontologie zurück gibt – und damit die Datei lädt – oder ebenfalls *null*. Gibt es in beiden Fällen *null* zurück, ist die Teilontologie dem System nicht bekannt.

Ziel einer entfernten Anfrage nach einer Teilontologie ist der Erhalt und die Entgegennahme derselben durch eine geeignete Methode. Abhängig von der Art der Anfrage ließe sich die Entgegennahme automatisch oder mit Nutzerintegration gestalten. Automatik bietet sich bei der Anfrage nach dem URI an, da genau eine Antwort richtig ist. Weitere möglicherweise eintreffende Antworten sind nur Doubletten. Sie ließen sich höchstens durch Vergleich zur Fehlererkennung nutzen.

Das Ergebnis der Importanfrage nach Stichwort muss hingegen der Anwender nicht nur selbst initiieren, sondern auch selbst auswerten. Schon die semantischen Korrektheit könnte sich mittels der ontologischen Struktur nur bedingt erkennen lassen: Durch die Möglichkeit zur mehrfachen semantischen Einordnung widerspricht eine andere als der vom Anwender vorgesehenen Einordnung nicht der Eignung als passende Teilontologie. Zudem soll der Anwender hier gerade die Möglichkeit zur Wahl einer Teilontologie haben, die seinen Vorstellungen entspricht. Hier erhält er im Falle des Erhalts mehrerer Teilontologien also die freie Entscheidung.

Für den Import ist insbesondere zu beachten, dass die Import-Funktion nicht kaskadierend ist; das heißt, dass eine – über einen Import in die bestehende Gesamtontologie eines Anwenders integrierte – neue Teilontologie nicht die Notwendigkeit mit sich bringt, eine oder mehrere nachfolgende Teilontologien importieren zu müssen. Dies wird dadurch ermöglicht, dass zwar Subklassen auf ihre Superklasse verweisen, aber nicht anders herum. Es wird somit durch eine Teilontologie keine Sub-Teilontologie eingefordert. Anders verhält es sich mit einer eventuellen nicht vorhandenen Super-Teilontologie. Da das jeweilige Wurzelelement einer Teilontologie auf eine Superklasse verweisen könnte, welche nicht Teil der Gesamtontologie ist und somit lokal nicht vorhanden wäre, müssen zwei Dinge gewährleistet sein: Eine vorhandene Superklasse wird entfernt, und die gewünschte Einbindung in die lokale Gesamtontologie wird durch Hinzufügen eines Verweises auf die entsprechende Superklasse zur Wurzelklasse der zu importierenden Teilontologie durchgeführt.

Der Vorgang des Importierens läuft entsprechend der Anfragen entgegengesetzt ab: Eine über die Netzwerkschnittstelle bezogene Teilontologie wird als Datei mit dem UUID-Teil ihres URI als Namen[11] in einem festgelegten Verzeichnis abgelegt. Der Dateiname wird dem Ontology Manager übergeben, damit dieser mit seiner Methode *loadOntology* die neue Teilontologie einbeziehen kann. Dieser lädt sie, liest den URI aus und trägt sie in den Verwaltungsobjekten *repositoryURI*, *repositoryFile* und *uriFileMapping* ein, um sie für andere Methoden zugänglich zu machen. Somit wird das selbe Vorgehen angewendet wie bei neu erstellten Teil-Ontologien nach Abschnitt 5.2.1.

Eine vom Kommunikationspartner eingeforderte Teilontologie wird aber nicht nur in die Gesamtontologie integriert, sondern bietet gleichzeitig auch eine Basis für ein Schema zur Eingabe der erwarteten Daten. So findet die Dateneingabe selbst kontinuierlich im Kommunikationsprozess statt, ohne dass der Anwender einen zusätzlichen Arbeitsaufwand damit hätte.

Die Integration einer neuen Teilontologie hingegen erfordert eine passende Einordnung, die dem Anwender überlassen bleibt. Für dieses Unterfangen muss er einen Einblick in die Gesamtontologie seiner Identität erhalten und eine geeignete Klasse auswählen, deren Subklasse die Wurzelklasse der neu integrierten Teilontologie werden soll. Dieser Vorgang sollte für einen Anwender eine Ausnahme darstellen, da er weder trivial ist – nach Marshall und Shipman (2003) ist die konsistente ontologische Einordnung nicht zwangsläufig gegeben[12] –, noch ist er anwenderfreundlich: Eine über einen langen Zeitraum gewachsene

[11] Der Doppelpunkt des URN kann nicht als Teil des Dateinamens verwendet werden.
[12] siehe auch Abschnitt 3.2.3

Ontologie ist schwerlich für einen Anwender zu überblicken. Zum Zweck der Integration einer neuen Teilontologie stellt der Ontology Manager die Methode *connectOntology* zur Verfügung. Sie trägt als neues Element *SubClassOf* der Wurzelklasse dieser Teilontologie den URI der ausgewählten Klasse ein und entfernt die vorhandenen entsprechenden Einträge.

Als anwenderfreundliche Alternative ist eine Integration nach vorhandenem *SubClassOf* Element denkbar: Der Anwender übernimmt nicht nur die gerade erhaltene Teilontologie, sondern fordert sämtliche Super-Teilontologien an. Die eingetragenen Superklassenverweise sind dann auch lokal beim Anwender gültig.

5.3. Einbindung fremder Teilontologien durch Mapping-Methoden

Zwei Möglichkeiten zur Integration einer Teilontologie ergeben sich aus dem Konzept. Einerseits gibt es die Option des Wunsches auf Seiten des Anwenders nach einer neuen Teilontologie auf Grund des bisherigen Mangels einer Entsprechung im lokalen System. Hier wird die bestehende Gesamtontologie um diese neue Teilontologie erweitert. Der andere Fall liegt vor, wenn der Datenaustausch mit einem Kommunikationspartner ansteht, aber eine gemeinsame Ontologie fehlt.

Für letzteren Fall bestehen nochmals zwei Optionen aus Anwenderperspektive: Entweder hat der Anwender die Identitätsattribute, die für einen Austausch vorgesehen sind, ganz oder teilweise auf Basis einer anderen Teilontologie in seiner Identität vorliegen, oder er hat bisher noch keine der betroffenen Identitätsattribute in seine Identität aufgenommen. Im letzteren Fall kann die Teilontologie des Kommunikationspartners übernommen und ebenfalls in die Gesamtontologie – wie in Abschnitt 5.2.2 dargestellt – integriert werden.

Der erste Fall, dass beide Kommunikationspartner unterschiedliche Teilontologien zum gleichen Identitätsattributsraum besitzen, erfordert ein anderes Vorgehen. Hier müssen bei gleichzeitigem Erhalt der ontologischen Strukturen beider Kommunikationspartner geeignete Abbildungen gefunden werden. Dies kann grundsätzlich sowohl von der einen als auch von der anderen Seite aus geschehen. Eine geeignete Abbildung kann schon lokal vorliegen, von einer dritten Partei bezogen werden oder durch Eigenerstellung erzeugt werden.

5.3.1. Integration neuer Teilontologien

Mit einem angefragten oder übertragenen Attribut übermittelt der Kommunikationspartner auch immer den URI der – im Rahmen seiner Gesamtontologie – zugehörigen Teilontologie. Anhand diesem URI und unter Berücksichtigung aller URIs der beim Anwender vorliegenden Teilontologien lässt sich ein eventuell vorhandenes Mapping finden, ohne den Anwender selbst mit solchen Fragen zu belästigen. Eine simple Option wäre die Suche nach allen Mappings, die als Quell-Teilontologie die angefragte Teilontologie akzeptieren. Dies würde – abhängig von der Zahl unterschiedlicher Mappings zu dieser

Quell-Teilontologie – eine nicht unerhebliche Netzlast auslösen.

Durch eine zweite Option wäre diese Netzlast begrenzbar: Bei einer schrittweisen Ausweitung des Suchraums werden nicht nur der URI der angefragten, sondern auch die der lokal vorliegenden Teilontologien berücksichtigt. Zunächst muss die Wurzelklasse der mit dem angefragten oder übertragenen Attribut übermittelten Teilontologie des Kommunikationspartners ermittelt werden. Dies setzt die Suche dieser Teilontologie selbst voraus, die nach dem Verfahren aus Abschnitt 5.2.2 durchgeführt wird. Allerdings wird hierbei diese neue Teilontologie nicht integriert, sondern nur zur vorübergehenden Analyse zwischengespeichert. Die Angabe *SubClassOf* der Wurzelklasse dieser temporär vorhandenen Teilontologie weist als Attribut den URI der Teilontologie auf, welche die Superklasse dieser Wurzelklasse beinhaltet. Ist diese Superklasse Teil der Gesamtontologie des Anwenders und besitzt diese Klasse Subklassen in anderen Teilontologien, so besteht aufgrund der semantischen Nähe eine erhöhte Wahrscheinlichkeit, dass eine dieser Subklassen verwandt ist mit der vom Kommunikationspartner verwendeten Teilontologie. Entsprechend wahrscheinlicher ist dann auch die Existenz einer Abbildung zwischen einer vorhandenen und dieser verwendeten Teilontologie. Bei Nichterfolg kann dieser Vorgang eine Teilontologie-Ebene höher erneut durchgeführt werden. Dies lässt sich bis zur Wurzel der Gesamtontologie des Anwender fortsetzen. Dies bedeutete zwar, dass dann sämtliche lokal vorhandenen Teilontologien des Anwenders als potenziell semantisch verwandt angesehen werden; der Suchraum – die Ziel-Teilontologie betreffend – bliebe aber immerhin auf die lokal existenten Teilontologien beschränkt. Nachteil dieser zweiten Option ist allerdings der erhebliche Mehraufwand und die Wahrscheinlichkeit zur Wiederholung der Anfrage an die Netzwerkschnittstelle. Dies könnte eine massive Verzögerung der gesamten Anfrage mit sich bringen. Der Fokus bleibe daher auf die erstgenannte simple Option gerichtet. Bei Fund eines geeigneten Mappings kann dieses nach dem Vorgehen im folgenden Abschnitt 5.3.2 eingesetzt werden.

Die mögliche Gegebenheit, in der kein Mapping gefunden werden kann, ist die Situation, dass die angefragten Attribute zwar vorliegen, es für die Kombination der beiden Teilontologien noch kein Mapping existiert oder momentan nicht erreichbar ist. Da dies weder durch Mapping Manager oder Ontology Manager noch durch den Identitätsmanagement-Client ermittelt werden kann, aber dem Anwender selbst bekannt ist, sei in dem in Abschnitt 5.2.2 genannten Dateneingabeschema auch die Option zur Übernahme bestehender Attribute enthalten. Durch die mittels dieser Übernahme verknüpften Attributsklassen zweier Teilontologien kann ein Mapping erstellt werden.

5.3.2. Mapping der Teilontologien

Neben den Teilontologien selbst sind die Mappings von einer Teilontologie auf eine andere Teilontologie das zweite Kernelement der Ontologieunterstützung für eine Identitäten-Infrastruktur. Die Suche nach einem geeigneten Mapping aus Abschnitt 5.3.1 orientiert sich wiederum an den URIs der Teilontologien. Eine Repräsentation als Klasse oder als XML kann durch zwei Identifikatoren gekennzeichnet sein. Die Benennung eines jeden Mapping für das Speichern als Datei wird durch eine geeignete Kombination der beiden URIs der betroffenen Teilontologien erzeugt. Als einfache Variante werden die

beiden URIs verkettet. Durch die Reihenfolge der Verkettung wird bestimmt, welche die ursprüngliche Quell-Teilontologie ist und auf welche Ziel-Teilontologie das Mapping erfolgen soll. Bei der Verwendung des Mappingnamens als Dateinamen ist bei bestimmten Dateisystemen zu berücksichtigen, dass die Verwendung dieser Verkettung aufgrund ihrer Länge von mindestens 128 Zeichen Probleme mit sich bringen kann. Für einem solchen Fall muss ein Verzeichnis von lokal zugewiesenen Namen und den URI-Verkettungs-Namen angelegt werden. Ein solches Problem steht zwar bei aktuellen Betriebssystemen nicht zu befürchten. Dass die Länge des Pfades 255 Zeichen[13] überschreitet, lässt sich aber teilweise nicht vermeiden. Aus diesem Grund und aus Gründen der Performanz wird eine Datei *MappingRepository* angelegt: Ihre Verwendung verhindert bei Start des Identitätsmanagement-Clients und Prüfung auf Vorhandensein eines bestimmten Mapping den direkten Zugriff auf das Dateisystem für jedes einzelne Mapping.

Drei Aspekte sind im Zusammenhang mit den Mappings zu beachten: Zunächst ist zu untersuchen, auf welche Weise – im Falle einer Kombination von Teilontologien, für die es noch kein Mapping gibt – die Mappings erstellt werden. Desweiteren erfordert die Wiederverwendung bestehender Mappings geeignete Möglichkeiten zur Verbreitung und zum Import. Abschließend sei zu untersuchen, wie die Mappings eingesetzt werden.

5.3.2.1. Erstellung neuer Mappings

Ein Mapping beinhaltet eine Reihe Transformationsanweisungen bezüglich einer Quell- und einer Ziel-Teilontologie: Es soll jeweils eine Anfrage oder Antwort, die ursprünglich entsprechend der Quell-Teilontologie formuliert war, so umgeformt werden, dass diese eine entsprechende Anfrage oder Antwort an die Ziel-Teilontologie darstellt. Eine Anfrage wie auch eine Antwort hat als umzuformenden Anteil ausschließlich den URI zum Inhalt, welcher sich aus dem URI der Teilontologie und – getrennt durch ein "#" – dem Namen der Ontologieklasse oder der Eigenschaft zusammensetzt. Der URI der Quell-Teilontologie wird hierbei durch denjenigen der Ziel-Teilontologie ersetzt. Ebenso wird der Name der Klasse, wie er in der Quell-Teilontologie verwendet wird, in den der Ziel-Teilontologie entsprechenden gewandelt. Da die Anfrage selbst keine strukturellen Informationen beinhaltet, führt dies auch bei einer die Semantik ändernden Abbildung nur zu einer syntaktischen Änderung der Anfrage. Auf diese Weise sind allerdings nur 1:1-Mappings möglich. Ein n:1-Mapping muss die Attribute mehrerer Kommunikationsantworten verknüpfen. Entsprechende Anfragen müssen also ebenfalls gekoppelt erfolgen. Dies führt dazu, dass beide Mappings einer Kommunikation eine entsprechende Kombinationsregel enthalten müssen: Das Mapping für die Anfrage nach einem kombinierten Attribut in die eine Kommunikationsrichtung erzeugt zwei geteilte Anfragen; das Mapping für die entsprechende Antwort in die andere Kommunikationsrichtung verknüpft zwei Antworten entsprechend zu einer Rückmeldung. Da nicht gewährleistet ist, dass die beiden zusammenzufügenden Attribute auch zusammen übertragen werden, führt dies nicht zwangsläufig zum Erfolg. Es bieten sich mehrere Lösungen für dieses Problem: Die simpelste ist die Beschränkung auf 1:1-Mappings; in einer anderen Lösung interveniert das Mapping-Objekt selbst, wenn die beiden zusammengehörigen Attribute nicht

[13] eine Beschränkung, die zumindest bei älteren Versionen von Microsoft Windows noch besteht

Quellcode-Beispiel 8 Transformationsanweisung

```
<transform>
  <source>
    uuid-
596162616461626 14A78746150325033F55704B199754D3E9076A29477
        75A6EE03#Quellklasse_1
  </source>
  <target>
    uuid-
596162616461626 14A78746150325033F55704B199754D3E9076A29477
        75A63F78#Zielklasse_1
  </target>
</transform>
```

gleichzeitig übertragen werden; eine dritte Lösung lässt die Übertragung der jeweils unvollständigen Teil-Attribute weg und überlässt der anfragenden Stelle, diese Anfrage zu wiederholen oder darauf zu verzichten. Letztere Lösung lässt sich per XSLT ermöglichen, die ersten beiden müssen in der Erstellung beziehungsweise im Zugriff auf das Mapping im Kommunikationsfluss beachtet werden.

Die Erstellung einer Abbildung geschieht durch Gegenüberstellung der Quell-Teilontologie mit der Ziel-Teilontologie. Eine Erstellung direkt in XSLT ist aufgrund der zu vermutenden mangelnden Kenntnis des Anwenders nicht wünschenswert. Stattdessen ist eine Variante mit simplen XML-Tags, eine Tabelle oder eine geeignete grafische Eingabemöglichkeit zu wählen. Hier können alle zueinander passenden Eigenschaften und Klassen verknüpft werden. Als geeignetes Zwischenformat wird eine XML-Datei angelegt mit Elementen wie in Quellcode-Beispiel 8 dargestellt. Ein solches wird auch bei der Mappingerstellung durch Dateneingaben seitens des Anwenders entsprechend Ende Abschnitt 5.3.1 angelegt.

Die Übertragung in die benötigte XSLT-Datei kann durch XSLT selbst durchgeführt werden. Hierfür ist lediglich die einmalige Erstellung einer XSLT-Datei zu diesem Zwischenformat nötig. Dies ermöglicht auch eine Flexibilität für das Zwischenformat, was nachträgliche Änderungen ohne Quellcode-Änderungen ermöglicht. Nach der Transformation liegt eine XSLT-Datei vor, welche für jede zugehörige Anfrage verwendet werden muss. Dabei wird für jede Anfrage durch den XSLT-Prozessor die entsprechende Transformation durch den *Match*-Vorgang ausgewählt. Quellcode-Beispiel 9 zeigt beispielhaft die Mappingregel für die Umwandlung der Nennung der Klasse “Nick” einer Teilontologie in die Nennung der Klasse “Spitzname” einer anderen Teilontologie.

Da der thematische Bereich, den eine Teilontologie abdeckt, durch keine Maßgaben festgelegt ist, sind diesbezügliche Überschneidungen gut möglich: Die Entsprechnungen der Klassen einer Quell-Teilontologie könnten auf der Zielseite über zwei oder noch mehr Teilontologien verteilt sein. Wenn dies der Fall ist, dann ist für jede einzelne betroffene Kombination aus Quell-Teilontologie und beteiligten Ziel-Teilontologien jeweils ein XSLT

Quellcode-Beispiel 9 Auszug der XSLT-Datei zur Transformation der Anfragen

```
<xsl:template match="j.1:ptype">
  <xsl:choose>
    <xsl:when test="@rdf:resource= '596162616461626 14A787461503
                25033F55704B199754D3E9076A2947775A6EE03#Nick'">
      <identifier urn= "596162616461626 14A78746150325033F557
                  04B199754D3E9076A29489A6B7FF14#Spitzname"/>
    </xsl:when>
    ...
  </xsl:choose>
</xsl:template>
```

zu erzeugen. Diese Trennung kann durch einfache Sortierung nach dem URI der Ziel-Teilontologie schon direkt nach der Eingabe am Zwischenformat vorgenommen werden.

Abschließend wird das neu erstellte Mapping dem Mapping Manager übergeben. Der Vorgang ist identisch mit dem Vorgang, welcher dem Import folgt und in Abschnitt 5.3.2.2 beschrieben wird.

5.3.2.2. Verbreitung und Wiederverwendung von Mappings

Die Form der Suche nach einem geeigneten Mapping kommt der Suche nach einer Teilontologie gleich: Es wird zunächst durch den Mapping Manager geprüft, ob sich das Mapping im Speicher befindet. Ist dies nicht der Fall, wird eine entsprechende Datei lokal gesucht. Ist auch eine solche nicht vorhanden, wird die Anfrage an die entsprechende Netzwerkschnittstelle übergeben. Dabei wird der Anfrage – wie schon in Abschnitt 5.3.1 aufgeführt – nur der URI der Quellontologie mitgegeben. Die Netzwerkschnittstelle leitet ungefiltert alle Mappings mit dieser Quellontologie an den Mapping Manager zurück. Ist auch mittels dieser Anfrage kein Erfolg zu erzielen, existiert kein geeignetes Mapping und der Anwender ist selbst gefordert wie in Abschnitt 5.3.2.1 erörtert.

Der Import eines Mappings ähnelt in seinem Ablauf auch wieder dem Import einer Teilontologie: Über die Netzwerkschnittstelle werden die Mappings angenommen und unter der Kombination der beiden URIs – dem URI der Quell-Teilonotologie und dem der Ziel-Teilontologie – zu einem URI mittels der Methode *import* des Mapping Managers im Properties Objekt *uriFileMapping* mit einem geeigneten Dateinamen verknüpft und dann als Datei unter diesem Namen im Standardverzeichnis gespeichert. Die die Mappingsuche auslösende Methode erhält *TRUE* als Wert zurück und erhält somit die Bestätigung, dass eine Abbildung im *uriFileMapping* enthalten ist.

Falls keine gefunden wurde, übergibt der Mapping Manager die Suche nach einer entsprechenden Teilontologie dem Ontology Manager. Denkbar wäre auch zuvor ein weiterer Suchvorgang durch den Mapping Manager nach einem indirekten Mapping. Dafür würden zwei Anfragen mit jeweils einem unbestimmten Identifikator erfolgen müssen. Hierfür müsste es allerdings einen Weg geben, nach dem Mapping per Ziel-URI suchen

zu können, was so durch das Kommunikationsprotokoll nach Abschnitt 4.3.4 bisher nicht vorgesehen ist. Als Antwort kämen alle Mappings zurück, die den ersten Teilontologie-Identifikator ebenfalls als ersten aufweist, und all jene, welche den zweiten ebenfalls als zweiten aufweist. Der Mapping Manager würde die eingehenden Mapping-Identifikatoren entsprechend zwei *Array* Objekten übergeben und dann prüfen, ob sich bei den variablen Teil-Identifikatoren zwischen den zwei Gruppen der gleiche URI finden lässt. Aufgrund der dadurch verursachten gegenüber der konkreten Anfrage stärkeren Netzlast, der Notwendigkeit zur Aufsplittung der Suche nach Quell-URI und Ziel-URI und der Gefahr eines durch Informationsverlustes wenig nutzbaren Ergebnisses sei aber von diesem Mechanismus abzusehen. Stattdessen sei der Anwender umgehend aufgefordert, eine passende Abbildung zu erstellen. Auch wenn dies im jeweiligen Vorfall eine Einbuße an Nutzbarkeit mit sich bringt, wird die gesamte Infrastruktur durch die neue direkte Abbildung gestärkt.

Um anderen ebenfalls die Wiederverwendung dieses Mappings zu ermöglichen, gibt es als Teil des Mapping Managers auch öffentliche Methoden zum Zugriff auf das *uriFileMapping*. Dies ermöglicht die Prüfung einer an die Netzwerkschnittstelle gerichteten Anfrage nach einem Mapping. Zudem erhält die Netzwerkschnittstelle hierdurch direkten Zugriff auf die entsprechende Datei und leitet das XSLT an die anfragende Stelle weiter.

5.3.2.3. Anwendung der Mappings

Mappings kommen dann zum Einsatz, wenn zwei Kommunikationspartner sich über den Inhalt ihrer Kommunikation abstimmen wollen, aber verschiedene Teilontologien für diesen vorliegen. Initiiert wird die Kommunikation im Rahmen einer Kommunikationssitzung, wie sie Sack (2005) beschreibt, durch Kontaktaufnahme von Kommunikationspartner A zu Kommunikationspartner B: A übermittelt dabei eine Anfrage bezüglich eines Identitätsattributes unter Nennung der Ontologieklasse dieses Attributs in der Übermittlung. Dazu nutzt A den URI dieser Ontologieklasse. Im Falle des Vorhandenseins der entsprechenden Teilontologie auch bei B hat die ontologiegestützte Kommunikation bereits begonnen.

Im anderen, im Fall des Fehlens auf Seiten von B wird ein passendes Mapping entsprechend Abschnitt 5.3.2.2 ermittelt. Nun wird der URI der zugehörigen bei B lokalen Teilontologie im Rahmen der Kommunikation – beispielsweise eine Antwort auf eine Anfrage – an A übermittelt, um dort gleichfalls die Verwendung einer Abbildung auf diese Teilontologie zu erwirken. Da ein Mapping nicht symmetrisch angelegt sein muss – ein Attribut der einen Teilontologie kann eine, keine oder mehrere Entsprechungen in der anderen Teilontologie aufweisen –, wird diese entgegengesetzte Kommunikation unabhängig behandelt. Die Transformation findet also entsprechend auf der Seite von A statt.

Falls nicht nur menschliche Anwender die Identitätsmanagement-Clients nutzen, sondern auch automatische Services, besteht die Möglichkeit einer Kommunikation zwischen Mensch und Service. In diesem Fall sollten die Mappings beider Kommunikationsrichtungen auf der Seite des Clients des menschlichen Anwenders durchgeführt werden. Im Falle

nicht vorhandener Mappings wäre nur der Mensch befähigt, beide Mappings zu erstellen. Zudem besteht bei einem Service die Gefahr, durch eine Vielzahl an Abfragen genutzt zu werden; sollte jede dieser Anfragen durch eine XSLT-Transformation begleitet sein, dürfte sich dies als ein massives Performanzproblem darstellen. Auf diesen Fall sei im weiteren Verlauf aber nicht weiter eingegangen und vielmehr eine Kommunikation zweier Menschen angenommen. Auch hierbei erweist sich die symmetrische Verteilung der Transformationsvorgänge allerdings als potenziell problematisch: Durch gezielte Anfrage könnte ein Kommunikationspartner ermitteln, welche Teilontologien ein anderer lokal vorliegen hat. Es wäre auch vorstellbar, dass Teilontologien und Mappings als Cookie-Ersatz missbraucht werden, indem für jede Sitzung eine neue Teilontologie erzeugt wird, um mit deren Hilfe den Nutzer später identifizieren zu können[14]. Für die prototypische Umsetzung seien trotzdem solche Befürchtungen außer Acht zu lassen. Vom konzeptionellen Standpunkt her ist es irrelevant, wer welche Transformation vornimmt: einer der beiden Kommunikationspartner oder möglicherweise auch eine dritte vertrauenswürdige Partei. Daher sei hier nur auf die symmetrisch verteilten Transformationsvorgänge eingegangen.

Ein Mapping muss zwei Funktionen erfüllen: Einerseits müssen Anfragen an Identitätsattribute auf Basis einer Ontologieklasse auf die lokal verwendete Ontologieklasse abgebildet werden; andererseits müssen die zugehörigen Antworten entsprechend abgebildet werden. Zwar sind diese beiden Fälle als Kommunikationsform zu unterscheiden; sie unterscheiden sich aber bezüglich des Zugriffs auf den URI der betroffenen Ontologieklasse nicht. Im Anfragefall soll das der Ontologieklasse zugewiesene Identitätsattribut im weiteren Verlauf ausgelesen werden, im Antwortfall wird ein entsprechendes Attribut entgegengenommen. Auf die Verwendung im Identitätsmanagement-Client selbst oder eine ihn nutzende Anwendung sei hier nicht weiter eingegangen. Somit reduzieren sich die zwei genannten Funktionen für den Mapping Manager auf eine Funktion.

Das Mapping wird also nicht auf die Teilontologie im Vornherein selbst angewendet, sondern auf die jeweilige ontologiebehaftete Kommunikation. Dabei gibt es genau zwei verschiedene Formen von Feldern, die angefragt werden können: Ontologieklassen und deren Eigenschaften. Obwohl die Eigenschaften fest an eine Klasse gebunden sind, ermöglichen sie ebenfalls den direkten Zugriff über ihren URI. Unabhängig von Art der Anfrage und des betroffenen Feldes muss nur der URI entsprechend transformiert werden. Bei der Transformation handelt es sich immer um eine *n:1*-Beziehung zwischen Quell-URI und Ziel-URI mit kleinem n (meist 1). Von dieser Anforderung her würde also eine Hashtable oder dergleichen ausreichen. Aus Gründen der Standard-Konformität, des XML-basierten Formates und der besseren Möglichkeiten zur Erweiterbarkeit sei aber XSLT weiterhin das zu verwendende Format. Nur hiermit lassen sich auch Regeln definieren, auf welche Weise mehrere Quell-Attribute zu einem Ziel-Attribut verbunden werden. Es bieten sich zudem flexiblere Möglichkeiten zur Berechnung bei XSLT, die durch andere Konzepte nicht gegeben sind.

[14] siehe auch die Aspekte zum Thema Privatsphäre in Abschnitt 5.2.2.

5.4. Die Netzwerkschnittstelle

Die Netzwerkschnittstelle bietet die Möglichkeit zum Austausch von Teilontologien und Mappings. Aus Sicht des Ontologie und Mapping Managements stellt sie – abgesehen von der Eigenerstellung – die Möglichkeit zum Erhalt lokal nicht vorhandener Teilontologien und Mappings dar. Sie ist unabhängig von der Netzwerkanbindung des Identitätsmanagement-Clients, was nicht bedeutet, dass diese beiden keine Ähnlichkeit aufweisen dürfen. Das beschriebene Konzept erlaubt jedoch diverse Möglichkeiten der Implementierung, wie sie schon in Abschnitt 5.1 umrissen sind. Dies erfordert grundsätzlich eine unabhängige Implementierung.

Der Zugriff auf die Netzwerkschnittstelle erfolgt über wenige Methoden. Den Kern bilden die beiden Möglichkeiten zur Anfrage von Teilontologien und zur Anfrage von Mappings. Beide erwarten als Parameter einen URI. Zunächst erscheint es sinnvoll, die Klassen, welche diese Methoden ansprechen, für eine Anfrage als Listener nach dem entsprechenden Entwurfsmuster nach Gamma (1994) zu registrieren. Da jedoch die Kommunikation ohne eine Rückmeldung nicht ontologiebasiert weiterlaufen kann, sei darauf zu verzichten und die Antwort als gewöhnlicher Rückgabewert umzusetzen. Neben den beiden Anfragen mittels URI als Parameter kommt noch die dritte Anfragemöglichkeit hinzu: Eine Anfrage nach einer Teilontologie zu einem Suchwort kann diverse Ergebnisse unterschiedlicher Qualität erbringen. Da weder die Qualität noch die Eignung für den jeweiligen Zweck automatisch, sondern nur durch den Nutzer festgestellt, und nur durch ihn auch die beste Auswahl getroffen werden kann, ist für dieses Vorgehen eine entsprechende Nutzerschnittstelle anzulegen. Das Ergebnis der Wahl des Anwenders ist erneut genau eine Teilontologie, welche durch die aufgerufene Methode zurückgegeben wird.

Es ist zu überlegen, ob zur Vermeidung mehrerer Instanzen der Netzwerkschnittstelle diese als Singleton zu implementieren sei. Mögliche Implementierungen könnten dies erfordern; allerdings schränkt das Konzept der Schnittstelle nicht die Zahl ihrer Instanzen ein. Von daher sei dieses Entwurfsmuster nicht verpflichtend vorgegeben.

5.5. Integration der Komponenten

Anhand zweier Auszüge des Klassendiagramms und einem beispielhaften Sequenzdiagramm sollen die Integration der beschriebenen Komponenten und deren Anwendungsmöglichkeiten verdeutlicht werden.

5.5.1. Die Klassen im Überblick

Anhand des Klassendiagramms in Abbildung 5.1 lässt sich der Vererbungszusammenhang zwischen *OntologyManager* und *MappingManager* erkennen. Auf *Ontology*, *Mapping* und deren Schnittstelle *Item* sei weiter unten genau eingegangen. Die abstrakte Superklasse *ItemManager* enthält schon folgende für die beiden Subklassen *OntologyManager* und *MappingManager* notwendigen Variablen und Methoden:

- *instance* gibt den Hinweis darauf, dass die Implementierungen der geerbten Klassen

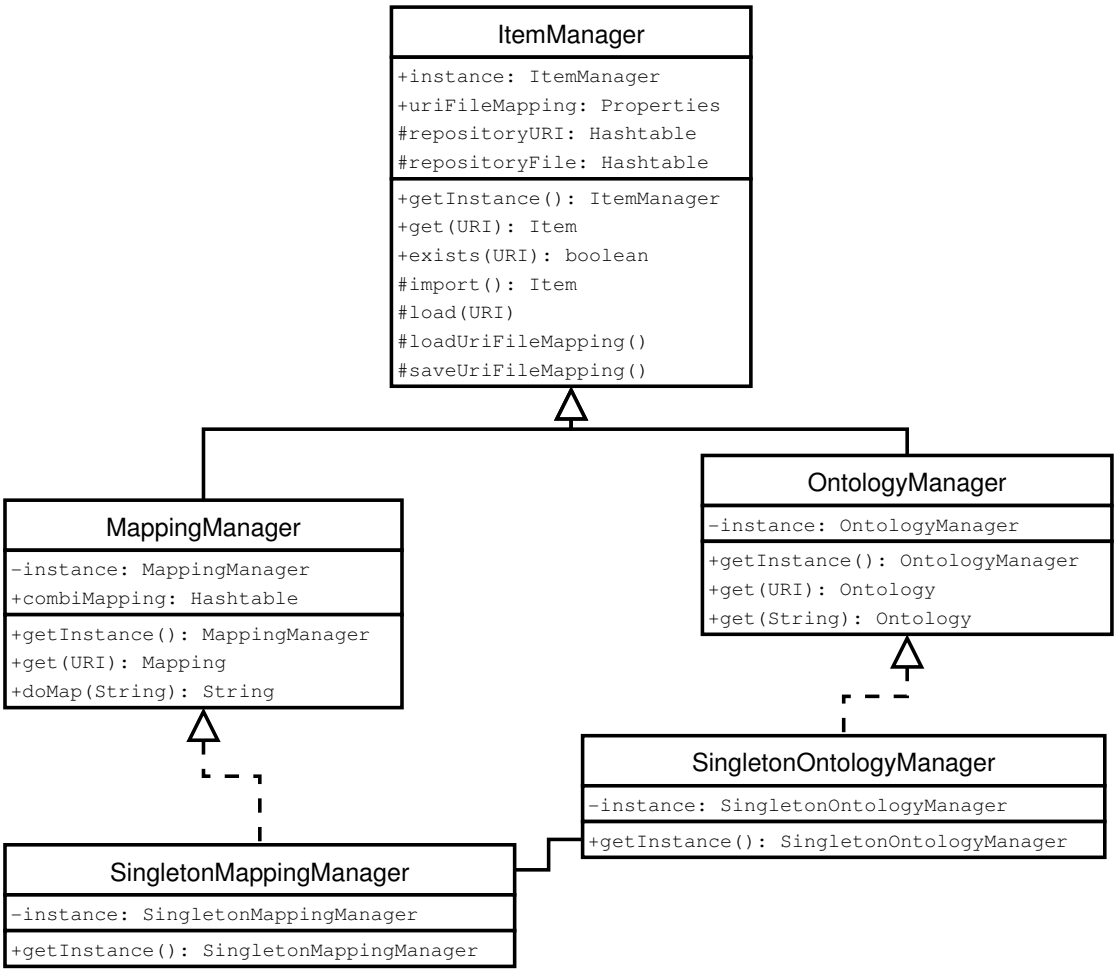

Abbildung 5.1.: Klassenverbindungen zwischen den Manager-Komponenten

dem Entwurfsmuster Singleton folgen sollen.

- *uriFileMapping* ist ein Properties Objekt, dessen Schlüssel jeweils ein Dateiname sein soll und dessen Wert dann der entsprechende URI ist.
- Zwei geschützte Hashtables – *repositoryFile* und *repositoryURI* – bieten die Möglichkeit, geladene *Items* über den Dateinamen und über den URI zu erreichen.
- Die für die Singleton-Implementierung wichtige Methode *getInstance* übernimmt die Aufgabe des Konstruktors, soweit die Klasse noch nicht instanziiert wurde. Ist dies allerdings geschehen, gibt sie die Referenz auf genau diese eine Instanz zurück.
- Mit der Methode *get* erhält der Aufrufende das *Item* zurück, welches der als Parameter mitgegebene URI identifiziert. Mit der entsprechenden Implementierung wird die in Abschnitt 5.2.2 beschriebene Kaskade an lokalen und verteilten Anfragen ausgelöst und der Import mittels der *import* Methode vorgenommen.
- *exists* prüft anhand des URI, ob das angesprochene *Item* überhaupt existiert. Dabei beschränkt sich die Suche allerdings auf den lokalen Bereich. Dies ermöglicht vor einer Anfrage per *get* eine Entscheidung, ob eine verteilte Suche überhaupt gewünscht ist.
- *import* erledigt die für die zukünftige Verwaltung notwendigen Schritte für ein neu angenommenes *Item*: Eintragung in die jeweiligen Repositories und lokales Speichern auf einem persistenten Medium. *import* wird als geschützte Methode nur von *get* verwendet.
- Mittels *load* – ebenfalls eine geschützte Methode – lässt sich ein *Item* nach Bedarf durch Angabe seines URI vom persistenten Medium laden.
- *loadUriFileMapping* und *saveUriFileMapping* sorgen für die Sicherung auf einem persistenten Medium und dessen erneute Nutzung.

Abgesehen von den Varianten in hier nicht weiter zu erwähnenden Hilfsmethoden werden der *MappingManager* und der *OntologyManager* neben den jeweiligen Ausprägungen – das *Item* als *Mapping* oder *Ontology* und die *Instance* als *SingletonOntologyManager* oder *SingletonMappingManager* – durch wenige weiter notwendige Variablen und Methoden ergänzt:

Der *OntologyManager* wird nur durch eine zusätzliche *get* Methode ergänzt. Mit ihr lassen sich Teilontologien per String suchen.

Der *MappingManager* ist ergänzt durch die Hashtable *combiMapping*, die ArrayList Objekte als Werte enthält. Der Schlüssel ist hier immer der URI einer Quell-Ontologie. Jedes im Laufe einer Kommunikation benötigte *Mapping* wird in dieses ArrayList Objekt eingetragen und steht damit als *Mapping* zur Verfügung. Wichtig ist hierbei insbesondere der Fall von *Multi-Mappings,* der die Verwendung des Objekttyps ArrayList erklärt: Eine Quell-Teilontologie wird auf zwei oder mehr Ziel-Teilontologien gemappt, für welche jeweils ein *Mapping* mit seinem URI in die ArrayList eingetragen wird. Beim späteren

Mappen müssen alle in Frage kommenden *Mappings* berücksichtigt werden, damit ein Kommunikationsfragment erfolgreich gemappt werden kann.

Die ArrayList *openAttributes* ist ebenfalls für den Fall notwendig, dass eine Teilontologie auf mehrere andere Teilontologien gemappt wird. Im Verlauf der Auswahl der geeigneten *Mappings* hält *openAttributes* die Attribute der Quell-Teilontologie fest, welche noch nicht von dem im Aufbau befindlichen Multi-Mapping abgedeckt werden. Erst wenn diese – nur temporär verwendete – ArrayList leer ist, deckt das Multi-Mapping den gesamten Attributsraum der Teilontologie ab.

Die Methode *doMap* extrahiert aus dem übergebenen String-Parameter *communicationFragment* den URI für das passende *Mapping*. Der String wird zudem in ein *StreamSource* Objekt umgewandelt und im Aufruf der Methode *mapIt* dieses *Mappings* selbst als Parameter übergeben. Das zurückzuerhaltende *StreamResult* Objekt wird als Rückgabewert von *doMap* zu einem String umgewandelt.

Zu erwähnen bleibt noch die beim *MappingManager* etwas irreführende Bezeichnung der geerbten Methode *exists*. Aus Perspektive des Identitätsmanagement-Clients wäre ein Methodenname "isMappable" treffender, da die Methode einen URI aus einer Teilontologie erhält, aber nichts über deren Existenz zurückgibt. Aus Sicht des *MappingManagers* ist der URI allerdings als URI einer Quell-Teilontologie für das entsprechende *Mapping* zu werten. Der *MappingManager* trifft mit seinem Rückgabewert demnach eine Aussage über die Existenz des entsprechenden *Mappings*.

Das Klassendiagramm in Abbildung 5.2 richtet den Fokus auf *Mapping* und *Ontology*. Die abstrakte Klasse *Item* dient der passenden Assoziation für den *ItemManager*. Auch wenn *Ontology* und *Mapping* als gemeinsame Basis ein XML besitzen, sind sie doch recht verschieden. *Item* wird hier nur als Schnittstelle beschrieben, während die Implementierungen von *Ontology* und *Mapping* in unterschiedlichen Klassenverbindungen stehen. Einzig die Möglichkeit zum Abspeichern per *save*, der Bezug zum jeweiligen Manager (*manager*) und die eigene URI als Identifikator *itemID* sind vorgegeben.

Ontology erbt von der Klasse OntModelImpl des Jena Frameworks und bietet daher diverse Möglichkeiten, welche für eine Teilontologie nach bestehender Konzeption nicht nötig sind. Um eine Kompatibilität zu den Jena eigenen Klassen zu gewährleisten, bleiben alle Methoden in ihrer ursprünglichen Form erhalten. Die vorgegebenen Methoden bieten vielfältige Navigationsmöglichkeiten zwischen den einzelnen Knoten, die sich bei der Verwaltung der Attribute des Identitätsmanagement-Clients einsetzen lassen. Aus Gründen der Kompatibilität wird auch die Erzeugung mittels des Entwurfsmusters Factory beibehalten. Zudem erfordert die Initialisierung von Ontologien auf Basis des allgemeinen Jena-Modells eine komplexe Erzeugung, was den Beibehalt dieses Entwurfsmusters sehr sinnvoll macht. Zu diesem Zweck muss allerdings die als final attributisierte ModelFactory durch eine entsprechende Modifizierung ersetzt beziehungsweise gewrappt werden.

Die Klasse *Mapping* ist wesentlich schlanker gestaltet. Ihren Kern bildet ein XSLT-Stream in Form eines StreamSource Objekts, welches bei Instanziierung aus der entsprechenden Datei geladen wird. Darüber hinaus ergänzt *Mapping* die Vorgaben von *Item* noch um die Methode *mapIt*, welche die Anwendung des XSLT-Streams auf den Eingangsparameter umsetzt. Hierbei wird die Transformationsfunktion Xalans in Anspruch

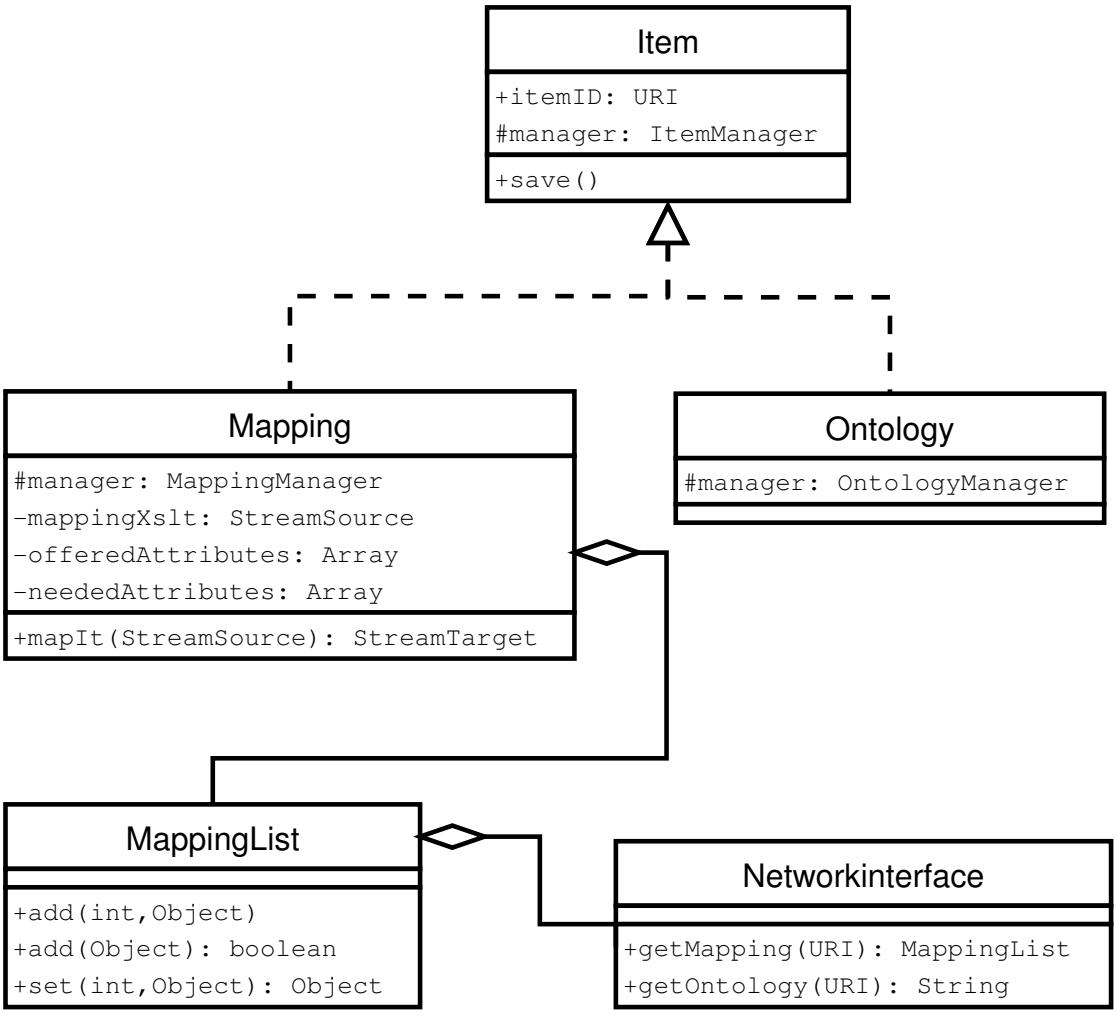

Abbildung 5.2.: Klassenverbindungen zwischen *Ontology* und *Mapping*

genommen. Zudem sind weitere Variablen in den Fällen, in denen ein einzelnes *Mapping* nicht zur Transformation der Attribute einer Teilontologie ausreicht, für die Verwaltung von Multi-Mappings zuständig. Mit *offeredAttributes* bietet das *Mapping* Objekt ein Array genau der Attribute, deren Mapping durchgeführt werden kann. *neededAttributes* listet als zweites Array solche Attribute auf, die in der Quell-Teilontologie zwar enthalten sind, aber in diesem *Mapping* nicht auftauchen.

Die Klasse *MappingList* ist ein spezialisierter Wrapper um die Standardklasse ArrayList. Bevor sie ihrer Superklasse Objekte durchreicht, wird versucht, diese nach Xalans StreamSource zu casten. Misslingt dies, wird eine entsprechende Ausnahmebehandlung eingeleitet. Die beiden Methoden zum Hinzufügen von Collections werden mit *false* abgewiesen.

Das *Networkinterface* stellt die Anbindung an das gemeinsame Ontologien- und Mapping-Austauschnetz dar. Für angefragte *Mappings* gibt es alle gefundenen *Mappings* in Form einer *MappingList* zurück. Eine gefundene Teilontologie wird einfach als String zurückgegeben. Die Erzeugung des entsprechenden Objekts *Ontology* übernimmt der *OntologyManager* selbst.

5.5.2. Kommunikation zwischen den Objekten

In der Interaktion der beiden Manager ist der *MappingManager* die kontrollierende Komponente. Sie stellt dem Identitätsmanagement-Client die Methode *exists* zur Verfügung, die nicht nur eine Aussage über das lokale Vorhandensein eines *Mappings* zurückgibt, sondern gleichzeitig auch versucht – für den Fall, dass dieses *Mapping* nicht lokal vorliegt –, dieses zu beschaffen. Erst bei Misserfolg erhält der *OntologyManager* eine entsprechende Anfrage durch den *Mapping Manager*. Das Sequenzdiagramm in Abbildung 5.3 verfolgt den Nachrichtenverlauf in genau so einem Misserfolgsfall detailliert.

Der Identitätsmanagement-Client (*Identity Management Client*) erhält die Anfrage, an einer Kommunikation basierend auf einer Teilontologie mit dem URI *UUID1* teilzunehmen. Mit diesem URI als Parameter ruft er die Methode *exists* des *MappingManagers* auf. Der Rückgabewert wird vom Typ boolean sein, welcher einfach nur angibt, ob die Kommunikation auf irgend eine Weise ontologiebasiert hergestellt werden kann. Dabei ist es für den Identitätsmanagement-Client vollkommen egal, ob die verwendeten Teilontologien gleich sind, ein *Mapping* vorliegt beziehungsweise beschafft wird oder eine neue Teilontologie importiert wird.

Der *MappingManager* ist die Komponente, die diese verschiedenen Möglichkeiten testet: Zunächst leitet die *exists* Methode die Anfrage mit gleichem URI als Parameter an die Methode *exists* des *OntologyManagers* weiter. Dieser prüft ausschließlich das lokale Vorhandensein der Teilontologie mit diesem URI. Dazu überprüft er *repositoryUri* und *uriFileMapping*: Wenn es im es im *repositoryUri* enthalten ist, dann liegt das entsprechende *Ontology* Objekt schon im Speicher vor; falls es im *uriFileMapping* vorhanden ist, dann befindet sich eine entsprechende Datei auf dem persistenten Medium. Im letzteren Fall wird die Teilontologie nun auch in den Speicher geladen. Ist weder das eine noch das andere der Fall, erhält der *MappingManager* den Wert *false* zurück, ansonsten *true*. Ist der Rückgabewert *true*, dann kann vom *MappingManager* ebenfalls *true* an den

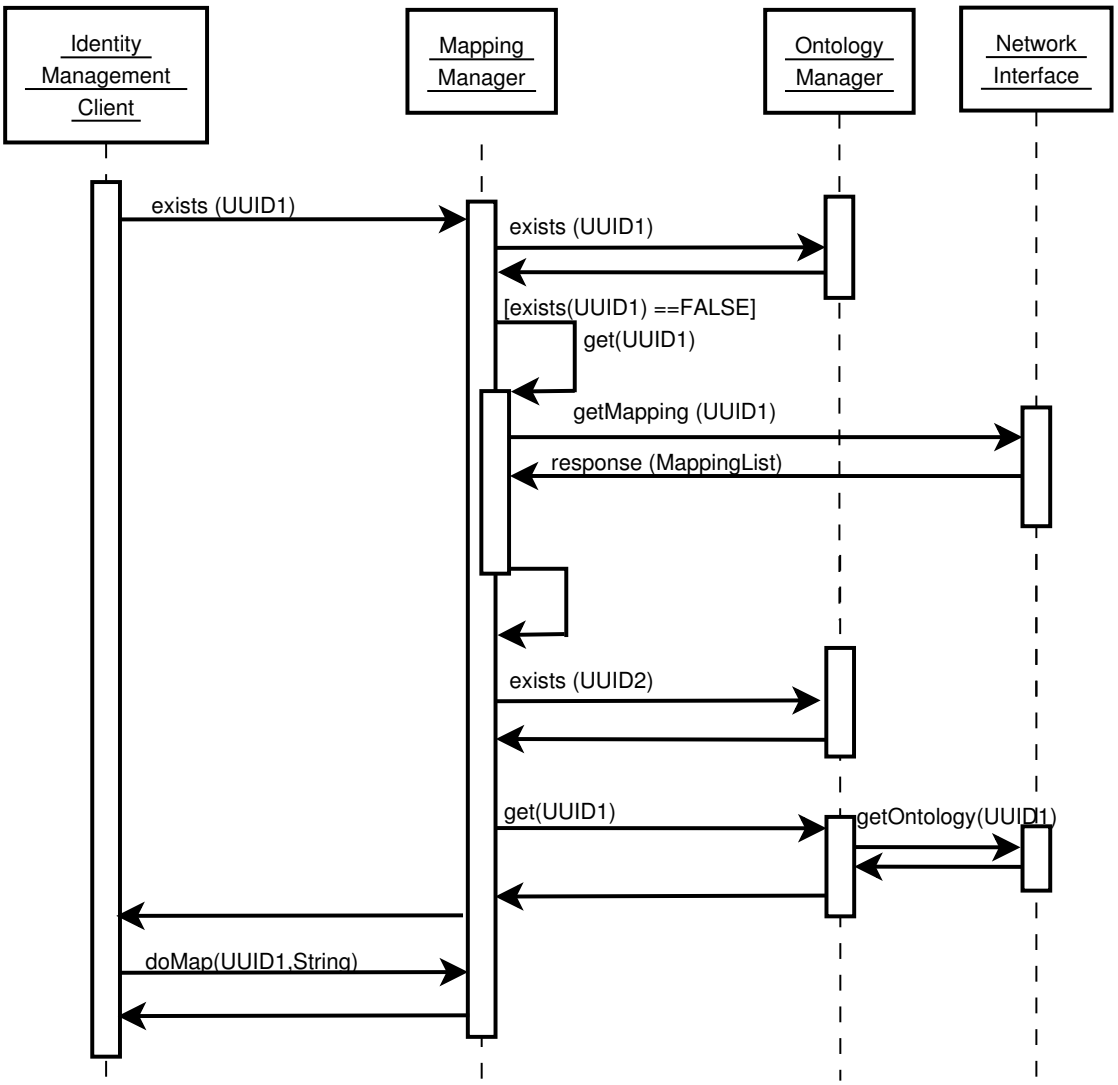

Abbildung 5.3.: Sequenzdiagramm zur Vorbereitung einer gemappten Kommunikation

Identitätsmanagement-Client zurückgegeben werden: Die Teilontologie liegt vor und die Kommunikation auf Basis dieser Teilontologie kann begonnen werden. In der Hashtable *combiMapping* wird zuvor durch den *MappingManager* ein neuer Eintrag vorgenommen: *UUID1* ist der Schlüssel dieses Eintrags; für den Wert wird eine ArrayList erzeugt, deren einziger Eintrag wiederum *UUID1* ist. Mit diesem Eintrag kann der *MappingManager* beim späteren Aufruf seiner Methode *doMap* durch den Identitätsmanagement-Client erkennen, dass diese Teilontologie überprüft wurde und das Mappen nicht nötig ist.

Ist keine entsprechende Teilontologie vorhanden – der Rückgabewert des *OntologyManagers* ist *false* –, dann überprüft der *MappingManager* das lokale Vorhandensein eines geeigneten *Mappings*. Dabei verwendet er ebenfalls den URI *UUID1* als Parameter. Zunächst werden die im Speicher befindlichen *Mappings* untersucht: Es werden alle *Mappings* durchlaufen, deren Quell-URI gleich *UUID1* ist. Für jedes einzelne wird das Array *neededAttributes* betrachtet: Bei leerem *neededAttributes* ist das *Mapping* bezüglich der Attribute der Quell-Teilontologie komplett. Eine weitere Suche kann unter der Bedingung, dass die durch den zweiten URI des *Mappings* identifizierte Ziel-Teilontologie lokal vorhanden ist, abgebrochen werden: Der *OntologyManager* überprüft dies erneut durch Aufruf seiner Methode *exists* durch den *MappingManager*. Für diese Kombination wird in der Hashtable *combiMapping* ein neuer Eintrag vorgenommen: *UUID1* ist der Schlüssel dieses Eintrags; für den Wert wird eine ArrayList erzeugt, deren einziger Eintrag der URI dieser Ziel-Teilontologie ist. Mit dieser Eintragung ist das *Mapping* aktiviert.

Gibt es kein *Mapping* mit leerem *neededAttributes* oder fehlen bei allen vorhandenen *Mappings* mit leerem *neededAttributes* die entsprechenden Ziel-Teilontologien, so werden alle weiteren *Mappings* – unter Abfrage von *uriFileMapping* – vom persistenten Medium geladen, die auch *UUID1* als URI haben, um sämtliche lokal vorhandenen *Mappings* entsprechend prüfen zu können. Wenn auch keines dieser *Mappings* ein leeres *neededAttributes* aufweist, so gilt die abschließende lokale Suche eine geeignete Kombination unter allen in Frage kommenden *Mappings*: Dazu wird aus den vorhandenen *Mappings* mit *UUID1* als URI eines mit der kleinsten Anzahl an Einträgen in *neededAttributes* gewählt. Auch hier gilt für alle gefundenen *Mappings*, dass zum verzeichneten Ziel-URI die entsprechende Teilontologie lokal vorhanden sein muss, was immer wieder mit der Methode *exitsts* des *OntologyManagers* ermittelt werden kann. Die im *neededAttributes* enthaltenen Einträge des entsprechenden *Mappings* werden in eine temporäre ArrayList *openAttributes* kopiert. Diese Einträge werden nun in den *offeredAttributes* der anderen *Mappings* mit lokal vorhandener Ziel-Teilontologie gesucht. Dort jeweils gefundene Einträge werden aus *openAttributes* wieder entfernt. Dies wird durchgeführt, bis *openAttributes* kein Element mehr enthält. Der URI einer jeden Ziel-Teilontologie der *Mappings*, die zu einer Reduzierung der Einträge von *openAttributes* beigetragen haben, inklusive dem zuvor ermittelten *Mapping* mit dem kleinsten *neededAttributes* wird in eine ArrayList aufgenommen, welche als Wert der *combiMapping* Hashtable mit *UUID1* als Schlüssel eingetragen wird.

Ist es jedoch nicht möglich, mittels der lokal vorhandenen *Mappings* sämtliche Elemente aus *openAttributes* zu entfernen, wird die Methode *getMapping* der Netzwerkschnittstelle mit *UUID1* als Parameter aufgerufen. Die Netzwerkschnittstelle führt eine

Suche im angebundenen Netz durch. Das Ergebnis der Suche sind kein, ein oder mehrere *Mappings*. Dabei wird direkt aus dem empfangenen Stream jeweils ein StreamSource Objekt erzeugt, welches in eine spezialisierte ArrayList, der *MappingList* eingetragen wird. Die *MappingList* besteht aus StreamSource Objekten, und stellt den Rückgabewert von *getMapping* dar. Durch den *MappingManager* wird aus jedem einzelnen StreamSource Objekt ein *Mapping* Objekt erzeugt. Der Konstruktor übernimmt die Aufgabe, das *Mapping* in die Hashtables *repositoryUri* und *repositoryFile* und die Properties *fileUriMapping* einzutragen und das *Mapping* per *save* auf das persistente Medium zu speichern. Bei Erfolg – ein oder mehrere Funde und damit auch Einträge in der zurückgegebenen ArrayList – wird die Prozedur der lokalen Suche erneut durchgeführt. Ist diese aufgrund der vorgenommenen Ergänzungen ebenfalls erfolgreich, erhält der Identitätsmanagement-Client *true* zurück. Damit ist die ontologiebasierte Kommunikation sicher gestellt.

Ist die lokale Suche erneut nicht erfolgreich oder gab es von der Netzwerkschnittstelle überhaupt kein *Mapping*, so wird die Methode *get* des *OntologyManagers* aufgerufen. Mit ihr erhält der *OntologyManager* ebenfalls *UUID1* als Parameter, welcher jetzt die gesuchte Teilontologie identifiziert. Der *OntologyManager* ruft die Methode *getOntology* der Netzwerkschnittstelle auf und erhält von dieser eine oder – im ungünstigen Fall – keine Teilontologie. Im Falle dieses endgültigen Misserfolgs gibt der *OntologyManager* an den *MappingManager* den Wert *false* zurück, welches dieser an den Identitätsmanagement-Client weiterreicht. Die ontologiebasierte Kommunikation ist in diesem Fall nicht möglich. Dieser Fall sollte allerdings nur eintreten, wenn die Netzwerkschnittstelle Anbindungsprobleme aufweist oder dem Anwender durch die Netzwerkschnittstelle eine Möglichkeit zur Ablehnung einer neuen Teilontologie zur Verfügung gestellt wird. Im anderen Fall erhält der *OntologyManager* eine Teilontologie von der Netzwerkschnittstelle in Form eines Strings. Er vergibt an diese neu erhaltene Teilontologie einen Dateinamen, speichert sie auf dem persistenten Medium und trägt sie in die Hashtables *repositoryUri* und *repositoryFile* und in die Properties *fileUriMapping* ein. Abschließend erhält der *MappingManager* den Wert *true* zurück. Er nimmt in der Hashtable *combiMapping* einen neuen Eintrag vor: Der Schlüssel dieses Eintrags ist der URI der gefundenen und importierten Teilontologie *UUID1*. Für den Wert wird eine ArrayList erzeugt, deren einziger Eintrag wiederum *UUID1* ist. Der *MappingManager* gibt *true* an den Identitätsmanagement-Client als Rückgabewert zurück.

Hier wird deutlich, dass zwar eine neue Teilontologie vorliegen kann, die entsprechenden Attribute allerdings nicht verknüpft sein müssen. Da die Ontologie die Attribute nicht kennt, aber umgekehrt die Attribute an die Ontologie gebunden sind, ist diese Verknüpfung nicht vom *OntologyManager* vornehmbar. Bevor der Identitätsmanagement-Client also die Kommunikation weiterführt, ist jeweils für eine angefragte Ontologieklasse die Prüfung nötig, ob ein entsprechendes Attribut zugewiesen ist. Für den Fall, dass zu den in der neuen Teilontologie enthaltenen Attributsklassen schon entsprechende Attribute der Identität vorhanden sind, kann aus deren bisherigen Verknüpfungen in Verbindung mit den Zuweisungen zur neuen Teilontologie ein neues Mapping erstellt werden.

Als Teil jeder einzelnen Antwort des Identitätsmanagement-Clients im Rahmen seiner Kommunikation ruft dieser die Methode *doMap* des *MappingManagers* auf. Übergeben

wird der URI *UUID1* der Ziel-Teilontologie und der String, welcher übermittelt werden soll. Das *Mapping* wird umgesetzt und das entsprechend gemappte Kommunikationsfragment zurückgegeben. Auch wenn die Ziel-Teilontologie vorliegt, kann *doMap* aufgerufen werden, da der *MappingManager* anhand des entsprechenden Eintrags in *combiMapping* ermitteln kann, dass das *Mapping* zwar geprüft wurde, aber nicht nötig ist. Er gibt den String mit dem somit ungemappten Kommunikationsfragment zurück. Für den Identitätsmanagement-Client bleibt die Durchführung des *Mappings* somit transparent.

Auch an dieser Stelle wäre es denkbar – im Falle eines nicht vorhandenen Attributs –, statt des gemappten Strings einen Leer-String oder ein ähnlichen Hinweis für das Fehlen des Attributs an den Identitätsmanagement-Client zurückzugeben. Aus mehreren Gründen, kann der *MappingManager* allerdings hier keine Kontrollfunktion durchführen: Zu allererst kann auch das Fehlen eines Attributes einer angefragten Ontologieklasse einen Eingriff in die Privatsphäre bedeuten, denn dem Anfragenden wird dadurch bekannt, dass sich der Anwender bisher noch nicht mit diesem Attribut beschäftigt hat. Es ist also davon auszugehen, dass auch die Übermittlung eines nicht vorhandenen Attributes durch die Privacy Komponente (siehe Rickert, 2004) überprüft wird. Zweitens ist es für den *MappingManager* nicht zu erkennen, mit welcher Intention die Übertragung eines leeren Attributs erfolgen sollte. Zuletzt ist es im Falle der Übertragung dem Kommunikationspartner überlassen, das Fehlen eines Attributes zu beanstanden.

Die hier beschriebene Sequenz der Beschaffung eines *Mappings* oder eine neuen Teilontologie deckt alle wesentlichen Fälle der ontologiebasierten Kommunikation in einer Identitäten-Infrastruktur ab.

5.6. Integration als Framework

Die ausgearbeiteten Komponenten stellen sich insgesamt als Framework zur Unterstützung von Kommunikation durch ein flexibles Ontologiensystem dar. Dabei wird es zwischen einer kommunizierenden Komponente – dem Identitätsmanagement-Client – und einer Netzwerkschnittstelle eingebettet. Das *onefC* Projekt[15] stellt eine prototypische Refererenzimplementierung zur Verfügung, welche die Einbindung des Ontologiensystem-Frameworks ermöglicht und vorsieht. Durch die starke Wandlung der Referenzimplementierung im Verlauf dieser Arbeit ist deutlich geworden, wie wichtig eine möglichst überschaubare und flexible Architektur ist. Die zusammenfassende Darstellung der Schnittstellen und Anforderungen an die integrierenden Komponenten soll dies nochmals betonen.

5.6.1. Schnittstellen und Anforderungen

Auf Seiten der Netzwerkschnittstelle sind zwei öffentliche Methoden erforderlich: *getMapping* fordert zu einer bestimmten URI die entsprechenden *Mappings* in Form einer *MappingList* an; *getOntology* sucht nach einer Teilontologie ebenfalls anhand einer URI.

[15] Literatur zum Projekt: siehe `http://vsis-www.informatik.uni-hamburg.de/projects/onefc/`

Als Rückgabewert gibt es im letzteren Fall einen String, der zur Erzeugung des entsprechenden *Ontology* Objektes genutzt wird.

Die Netzwerkschnittstelle ist somit recht stark an die Klassen und Einschränkungen des Ontologiensystem-Frameworks gekoppelt. Es stellt trotz der flexiblen Umsetzbarkeit bezüglich der Identitäten-Infrastruktur auch einen integralen Bestandteil des Frameworks dar. *Mapping* und *MappingList* müssen bei der Implementierung der Klasse *Networkinterface* verwendet werden. Die Übergabe einer Teilontologie als String stellt sich als einfacher dar: Hier wird – wenn nicht eine Kontrolle durch redundante Ergebnisse mit integriert wird – nur das erste Suchergebnis direkt als Rückgabewert ohne weitere Bearbeitung übergeben.

Auf der Anbindungsseite zum Identitätsmanagement-Client ist die Einbindung grundsätzlich simpler: Der Client hat die Möglichkeit, mittels *exists* die Kommunikation auf Basis einer Teilontologie einzufordern und diese – soweit die Einforderung erfolgreich verlaufen ist – mit der Methode *doMap* auch durchzuführen. Da die Art der Kommunikation auch die Form der Mappings bestimmt, sind diese völlig unabhängig in XSLT zu definieren, was eine weitgehende Entkoppelung ermöglicht. Die beiden Methoden *exists* und *doMap* lassen sich statusfrei verwenden. Allerdings führt das zu einer Versicherung des Mappings bei jeder einzelnen Anfrage. Die Alternative ist eine entsprechende Speicherung der URI, die als abbildbar ermittelt wurde, auf Seiten des Clients.

Bei den Teilontologien ist die Möglichkeit zur Entkoppelung nicht gegeben, da die Attribute der Identitäten, die durch den Identitätsmanagement-Client verwaltet werden, jeweils auf einen URI der Gesamtontologie verweist. Von Vorteil sind aber die durch das Jena-Framework gegebenen Navigationsmethoden, die sich so für die lokale Gesamtontologie der Identitäten nutzen lassen.

5.6.2. Zusammenfassung

Abschließend lässt sich feststellen, dass es nicht die Architektur des Ontologiensystem-Frameworks ist, welche es kennzeichnet, sondern die Integration mehrerer aktueller Standards, Techniken und Frameworks. Die Flexibilität dieser Mittel und eine Herangehensweise mit einem anderen Blickwinkel ermöglichen die vorliegende Konzeption. Das Ontologiensystem-Framework stellt sicher keinen zwangsläufig notwendigen Teil in einer Identitäten-Infrastruktur dar, ermöglicht aber die unabhängige und individuelle Nutzung einer solchen Infrastruktur und unterstützt somit über den heutigen Stand der Nutzung hinausgehende Ideen und Konzepte des Identitätsmanagements. Die überschaubare Konzeptionierung ermöglicht auch eine anwenderfreundliche Nutzbarkeit, welche teilweise durch Transparenz, teilweise durch erfassbare Strukturierung gegeben ist. Die Entscheidung, wie komplex die Teilontologien werden und damit potenziell auch die Gesamtontologie des einzelnen Anwenders, obliegt allerdings den jeweiligen Erstellern. Das Ontologiensystem-Framework ermöglicht eine Integration verschiedener semantischer und schematischer Strukturen, die Anwender unterschiedlicher Sprache, Kultur und Vorstellung von Ordnungssystemen leichter verbinden hilft.

6. Fazit und Ausblick

Ein Identitäten-Netzwerk im virtuellen Raum basiert zwar auf einer Form von Identitäten, deren Kern die mathematische Identität eines Identifikators ist; es erweist sich aber nur dann als nutzbringend, wenn diese digitalen Identitäten durch Attribute angereichert sind.

Bei digitalen Identitäten handelt es sich vielfach um Stellvertreter für reale Personen. Diesen ist mit ihrer Individualität oftmals der Wunsch zu eigen, Dingen und Daten des persönlichen Umfelds eine individuelle Ordnungsstruktur zu verleihen. Bisher war dies im digitalen Umfeld teilweise nicht möglich, teilweise mit Einschränkungen verbunden; sei es, um einen gemeinsamen Standard etablieren zu können, sei es aufgrund von Defiziten in den technischen Möglichkeiten.

Die vorliegende Arbeit weist einen Weg auf Basis aktueller Standards und Technologien zur Vermittlung der Bestrebungen zur Vereinheitlichung und zur Individuierung. In diesem abschließenden Fazit werden die zu Beginn dieser Arbeit aufgeworfenen Forderungen und Ziele der Umsetzung gegenübergestellt und die Umsetzung bewertet. Dabei sei zwischen der technischen Realisierbarkeit und der praktischen Umsetzung differenziert.

Über die Bewertung des Erreichten hinaus seien die möglichen Konsequenzen näher beleuchtet. Hier sind konkrete Nutzungsformen genauso zu berücksichtigen wie solche Ausprägungen von Visionen, die diese Arbeit als Basis oder zur Orientierung nutzen können.

6.1. Ergebnisse dieser Arbeit

Als Ergebnisse dieser Arbeit werde zunächst einmal die prototypische Umsetzung analysiert. Insbesondere sind jene Aspekte zu beleuchten, in denen die Umsetzung der Konzeption widerspricht, sie nur teilweise erfüllt oder einen alternativen Weg wählt. Da es sich hier nur um eine prototypische Umsetzung handelt, sei als zweites ein Blick auf die Einschränkungen gegenüber einem einsatzfähigen System untersucht. Mit Blick auf ein solch potenzielles System werden abschließend verschiedene Formen von Nutzung und Integration in heute bestehende Anwendungsstrukturen und die daraus resultierenden Auswirkungen erörtert.

6.1.1. Ansatz und Verwirklichung

Der Ansatz des Konzeptes dieser Arbeit liegt in der massiven Ausweitung desjenigen Bereichs, der als Identitätsdaten betrachtet werden kann. Identitätsdaten im weiteren Sinne sind Daten, welche einer Identität des Anwenders semantisch sinnvoll zugeordnet werden können. Diese Daten können im Rahmen einer Identitäten-Infrastruktur auch

dem kommunikativen Austausch dienen. Dies kann in der direkten Übertragung solcher Daten erfolgen. Es kann auch die Übermittlung anderer Daten beeinflussen. Um aus einer solchen Kommunikation einen Nutzen ziehen zu können, ist es notwendig, dass beide Seiten eine gemeinsame Ebene der Verständigung finden. Die Identitätsdaten müssen einer semantischen Ordnung unterliegen, welche die an einer Kommunikation beteiligten Systeme kennen.

Auf der anderen Seite sind Identitätsdaten und gleichermaßen auch deren Ordnung eine individuelle Angelegenheit des einzelnen Anwenders. Je größer der Datenraum für Identitätsdaten ist, desto deutlicher kommen solche individuellen Unterschiede zum Vorschein. Hier eine Standardisierung einzuführen, bedeutet Grenzen zu setzen, die einen Anwender bevormunden, so er denn überhaupt ein solches System nutzte.

Der Widerspruch, der aus dem Wunsch nach gemeinsamer Nutzung von unterschiedlich organisierten Datenbeständen herührt, kann durch das Ontologiensystem-Framework – wie schon zu Beginn dieser Arbeit verlangt – überwunden werden. Die Gesamtstruktur der Identitätsdaten eines Anwenders auf Ontologienbasis in Teilstrukturen zu zerlegen, eröffnet die Möglichkeit, diese Teilontologien zu individualisieren und gleichzeitig mittels Transformationsmechanismen Austauschbarkeit der betroffenen Daten zu gewährleisten. Dies erfüllt das in Abschnitt 2.2 geforderte umfassende Strukturkonzept. Durch eine entsprechend feine Aufteilung wächst der Transformationsaufwand nicht mit der Größe des Identitätsdatenbestand, da immer nur der für die Kommunikation relevante Teil der Indentitätsdatenstruktur verarbeitet werden muss.

Dabei ermöglicht das Konzept des Ontologiensystem-Frameworks, den zusätzlich entstehenden Aufwand entsprechend Abschnitt 2.3.1 für den Anwender gering zu halten. Eine neue Teilontologie kann eingepflegt werden, indem die Dateneingabe in einer Weise erfolgt, wie sie auch heutzutage bei Web-Formularen oder in grafischen Anwendungsschnittstellen üblich ist. Ein Mapping als Basis für die Transformation kann vollkommen transparent importiert werden.

Wünscht ein Anwender jedoch zu einem Themenbereich eine andere Teilontologie, als jene, welche ihm durch ein solches Formular vorgeschlagen wird, so steht es ihm frei, diese über ein geeignetes Stichwort selbst zu suchen. Er kann auch seine persönliche Teilontologie auf Basis von OWL eigenhändig erstellen. Dies ist allerdings kein leichtes Unterfangen, soweit nicht ein geeignetes Unterstützungsprogramm – beispielsweise ein angepasster Teilontologien-Editor – zur Verfügung steht.

Dem Anwender steht es also frei, zwischen den Möglichkeiten zu wählen, entweder die Organisation und Anwendung der Teilontologien transparent im Hintergrund ablaufen zu lassen oder selbst einzugreifen und Kontrolle über die ontologischen Strukturen seiner Daten auszuüben. Es wird nicht versucht, diesen Konflikt im Rahmen der Umsetzung des Ontologiensystem-Frameworks zu lösen. Stattdessen wird dem einzelnen Anwender diese Entscheidung überlassen. Dies gilt für jede Teilontologie einzeln.

Liegt dem Anwender eine Teilontologie zu einem Thema schon vor und möchte sein Kommunikationspartner auf diese Daten mittel eines anderen Datenschemas zugreifen, so ist dies mit der Transformation durch ein schon vorhandenes oder beschaffbares Mapping möglich. Nur der eine Anwender, der eine Kommunikation mit dieser Kombination aus zwei Teilontologien das erste Mal durchführt, ist gezwungen, die Zuordnungen für

ein Mapping anzugeben und dadurch erst das Mapping zu erstellen. Bis auf diesen einmaligen Vorgang handelt es sich daher um ein Konzept, welches Funktionen wie das bei Molva et al. (1992) dargestellte *SingleSignOn* oder eine aktive Formulardatenverwaltung ermöglicht.

Die in Abschnitt 3.1.3 angesprochene Mehrdimensionalität wird durch die Option erreicht, mehrere Ontologieklassen mit einem Attribut zu verknüpfen. Es bedeutet keinen Widerspruch, wenn ein Attribut mehreren Ontologieklassen zugewiesen ist. Allerdings bleibt diese Zusammengehörigkeit der Ontologie gegenüber verborgen: Nicht für sie sind die Attribute sichtbar, sondern für die Attribute die Ontologieklassen. Dies erfordert auf der Seite der Attributeingabe einen Mechanismus zur Verhinderung von Mehrfacheingaben, um spätere Inkonsistenzen vermeiden zu können. Die Mehrdimensionalität, die gleichzeitig eine solche Mehrbelastung für den Anwender bedeutet, tritt aber nur in den Fällen ein, in denen er sich seine eigene Teilontologie anlegt oder eine Teilontologie von sich aus sucht. Innerhalb einer Kommunikation mit schon bestehenden und verbreiteten Teilontologien wird ein passendes Mapping gefunden, ohne dass der Anwender belangt wird. Es wird also nur der Anwender belastet, der sich schon zu zusätzlichem Aufwand bereit gezeigt hat.

In Abschnitt 3.2.1 wird die lokale dezentrale Vorhaltung als Alternative zu einer zentralen Speicherung genannt und für die folgende Konzeption als Basis verwendet. Die dort geforderten lokalen Möglichkeiten zur Verwaltung und Berarbeitung der Identitätsdatenstrukturen sind Teil der Umsetzung. Auf den gleichfalls aufgeführten Datenschutz und die Datensicherheit wird zwar an den entscheidenden Stellen eingegangen, jedoch mit der Eigenschaft als Prototyp nicht vollständig umgesetzt. Insbesondere könnte ein Kommunikationspartner die lokale Existenz bestimmter Teilontologien bei einem Anwenders ausspionieren. Dies gelingt ihm allerdings nur, wenn er den Anwender dazu bringen kann, dass er als scheinbar vertauenswürdige Instanz bestimmte Daten aus den jeweils zu ermittelnden Teilontologien erhält. Es ließen sich dann sogar speziell präperierte Teilontologien in die Gesamtontologie des Anwenders einbringen, welche den einzelnen Anwender zu einem späteren Zeitpunkt – beispielsweise schon anhand des URI – wiedererkennen lassen können.

Auch die in Abschnitt 3.2.1 genannten Zugriffszeiten sollten eigentlich kein Problem sein. Wenn aber ein vielfach aufgesuchter Dienst in einer Identitäten-Infrastruktur für jede Anfrage eine Transformation mit der Interpretationssprache XSLT durchführen muss, so wird dessen Performanz erheblich reduziert. Aus diesem und dem vorgenannten Grund des Datenschutzes müssen in einer produktiven Umgebung sämtliche Transformationsaufwendungen einer Kommunikation von deren Initiator übernommen werden. Dies verhindert die Herausgabe der lokalen Teilontologien des Anwenders und vermeidet gleichzeitig die Überlastung viel genutzter Dienste.

6.1.2. Möglichkeiten und Defizite zum praktischen Einsatz

Die in dieser Arbeit dargelegte Konzeption wie auch ihre Umsetzung nutzen die Technik der Ontologien. Diese findet aber bisher nur Verwendung für die Strukturierung und Identifizierung von Attributen. Dieses Defizit in der Anwendung findet seine Gründe

einerseits in der ursprünglichen Zielsetzung, andererseits im Bestreben, dem Anwender – auch dem, der eigene Teilontologien erstellen möchte – eine möglichst niedrige Hürde im Einsatz des Ontologiensystem-Framework aufzubürden. Ein weiterer Grund sind zudem die Grenzen dieser Arbeit und die prototypische Umsetzung. Für einen weiteren Ausbau des Konzeptes lassen sich aber einige Konzepte der Ontologien vorteilhaft nutzen, ohne dass sie gleichzeitig eine Belastung für den Anwender mit sich bringen.

Ontologien in OWL Light und OWL DL bieten – wie in Abschnitt 4.1.3.4 dargelegt – den Vorteil der Entscheidbarkeit. Mithilfe entsprechender Inferenzprüfungen ließen sich Widersprüche innerhalb der Gesamtontologie eines Nutzers aufspüren. Dies wäre insbesondere der Fall, wenn zwei Attribute zu zwei äquivalenten Ontologieklassen verschieden sind. Aber es wären auch komplexere Berechnungen durchführbar.

Für den Fall, dass ein Nutzer per Stichwortsuche importierte Teilontologien besitzt, ist eine volle oder teilweise Redundanz zu anderen Teilontologien möglich. Durch einen lokalen Einsatz von Mappings auf solcherart importierte Teilontologien könnte diese Redundanz gefunden werden. Dabei wird zu einer neu importierten Teilontologie einfach die Netzwerkschnittstelle mit der Suche nach einem Mapping zu dieser Quell-Teilontologie aufgerufen. Wenn es ein Mapping von dieser zu einer schon vorliegenden Teilontologie gibt, dann liegt eine Redundanz vor. Die betroffenen Attribute der neu importierten und der dazu abbildbaren Teilontologie können dann zu einem einzelnen gewandelt werden.

Die Bandbreite der Möglichkeiten zur Inferenz-Nutzung geht sogar noch darüber hinaus. Es könnten wirklich implizite Daten ermittelt und explizit gemacht werden. Ein simples Beispiel ist die Angabe des Geburtstags in Kombination mit dem aktuellen Datum. Unter normalen Umständen müsste der Anwender bei Bedarf sein Geburtsdatum als auch sein Alter angeben. Da sich letzteres aus ersterem berechnen lässt, ist die Angabe des Alters unter geeignetem Einsatz von Inferenzen nicht mehr notwendig. Allerdings ist hier Vorsicht geboten oder ein Mechanismus mit Bedingung zu einer Nachfrage an den Anwender einzusetzen. Der Besitz eines Autos legt den Umstand nahe, dass der Besitzer auch Besitzer eines Führerscheins ist. Dies muss aber nicht immer der Fall sein.

Es lässt sich einwenden, dass bei sorgsamer Erstellung einer Teilontologie solche Trugschlüsse nicht möglich gemacht werden sollten. Da aber jeder Anwender eine Teilontologie beisteuern kann, ist ein Weg notwendig, die daraus entstehenden Inferenzen zu verhindern. Im genannten Beispiel wäre also eine entsprechende Anfrage an den Anwender von Vorteil, bevor ein solcher Schluss Eingang in die Gesamtontologie findet. All diese zusätzlichen Angaben müssten nicht nur in den Teilontologien verankert sein, sondern auch eine Entsprechung in der Umsetzung erlangen.

Wesentlich für den praktischen Einsatz ist das Vorhandensein einer kritischen Menge an Teilontologien und die entsprechende Verbreitung des ontologiebasierten Identitätsmanagement-Clients. Eine Abschätzung, wann dieser kritische Punkt erreicht sein könnte, geht über den Rahmen dieser Arbeit hinaus. Es stellt sich aber die Frage, wie sich das Erreichen dieses Punkts beschleunigen ließe. Neben dem eigentlichen Framework müssten schon eine Anzahl an Teilontologien zu im Internet stark verbreiteten identitätsrelevanten Themen vorhanden sein. Günstig wäre auch ein Programm oder – noch leichter zugänglich – ein Webservice, welcher durch unkomplizierte Angabe von Ontologieklassen eine Teilontologie erzeugt. Eine Verwendung und damit auch

Erstellung durch kommerzielle Anbieter dürfte erst in einer späteren Entwicklungsphase wahrscheinlich sein.

Schon zum jetzigen Zeitpunkt gibt es Schemata, welche die Funktion einer Teilontologie teilweise erfüllen und zudem verbreitet und bekannt sind. Zu diesen zählen insbesondere die Ontologie des Dublin Core Metadata Element Set (siehe Abschnitt 4.1.1), der vCard Standard (siehe Abschnitt 2.1.2) und auch die Ontologie des FOAF-Projektes (siehe Abschnitt 1.1). Allerdings erfüllen sie nicht die Form von Teilontologien, wie sie in dieser Arbeit dargelegt sind: Sie sind inhaltlich anders aufgebaut, und sie befinden sich an einer zentralen Stelle. Dank XSLT wäre aber auch hier ein Mapping möglich. Und mittels einer angepassten Netzwerkschnittstelle könnten auch solche besonderen URIs – nämlich URLs – berücksichtigt werden. Ein solches Vorgehen hätte durch einen leichteren Anschluss an bestehende Konzepte eine bessere Aussicht auf Akzeptanz.

6.1.3. Auswirkungen

Anwendungen der heutigen Zeit bedienen sich vielfach eines Bereichs der individuellen Anpassung. Darunter befinden sich vorübergehende und beständige Speicherung der Aktivitäten des Anwenders: Dies ermöglicht die Funktion “Rückgängig” vieler Programme und die “History” in navigierenden Anwendungen wie Webbrowsern oder diverse Dokumenten-Betrachtern. Interessanter als indentitiätsrelevante Daten stellen sich die bewusst zur Speicherung vermerkten Daten dar – beispielsweise Lesezeichen oder Anmerkungen.

Viele Programme wie auch Webdienste und -anwendungen ermöglichen oder fordern die Eingabe persönlicher Daten. Aber auch schon das Dateisystem bietet das Anlegen einer individuellen Ordnungsstruktur an. All diese Daten lassen sich im weitesten Sinne als Identitätsdaten oder doch zumindest als individuell strukturierte Daten auffassen. Eine koordinierte oder zentrale Organisation dieser Daten mittels Identitätsmanagement vermeidet mehrfachen Aufwand und auch Inkonsistenzen.

Als solche können sie auch für eine Kommunikation sinnvoll verwendbar sein. Dies kann sich in Form einer Unterstützung dieser Kommunikation darstellen oder als Teil der Kommunikation selbst genutzt werden. Die Möglichkeiten der gemeinsamen Nutzung von identitätsbasierten Daten sind variantenreich vorstellbar, aber bisher noch wenig im Einsatz. Neben zugehörigen Begriffsbestimmungen führt Wollenweber (2004) diverse Beispiele zur kollaborativen Nutzung des World Wide Webs an, darunter *Empfehlungssysteme* und *Social Navigation*. Über die kooperative Zusammenarbeit von beispielsweise Projektgruppen ist innerhalb einer Identitäten-Infrastruktur auf Basis der Identitätsdaten auch schon die Bildung einer solchen Gruppe denkbar. Dies setzt eine entsprechende Teilontologie und die Fähigkeit zur Auswertung auf seiten des Identitätsmanagement-Clients voraus. Alternativ kann auch eine andere die Gesamtontologie nutzende Anwendung zum Einsatz kommen. Das Problem, welches sich aus der Gesamtheit dieser Systeme ergibt, ließe sich mittels der ontologiebasierten Identitäten-Infrastruktur bewältigen:

> Mit der wachsenden Zahl persönlich bedeutsamer Informationen wird es da-

> bei immer schwieriger, diese dem eigenen Arbeitsstil entsprechend abzulegen, so daß ein effizienter, an Aufgaben orientierter Zugriff gewährleistet ist. (Wollenweber, 2004, S. 196)

Nicht nur die verteilte Kooperation und Kommunikation in Netzwerken können von einem Ontologiensystem-Framework profitieren; auch lokale Dienste und Anwendungen kämen dafür in Frage. Es stellt sich hier die Frage nach der Verallgemeinerung oder Öffnung der Betrachtungsweise des Identitätsbegriffs: Wenn lokale Anwendungen oder sogar Bereiche des lokalen Systems als Identität eine eigene Gesamtontologie besäßen, ließen sich mit diesen eine ebensolche ontologiebasierte Kommunikation etablieren wie mit Identitäten in einem Netzwerk. Dies würde die Individualisierung der Darstellungsstrukturen sämtlicher Parameter dieser Anwendungen oder des Systems ermöglichen – ein erleichterter Zugang für alle Anwender ist auf dieser Basis denkbar.

6.2. Visionen

In einem populärwissenschaftlichen Artikel versuchen Berners-Lee et al. (2001) die Vorteile und Möglichkeiten des Semantic Webs zu erklären. Darin beschreiben sie die verschiedenen Komponenten und Techniken, die das Semantic Web ausmachen sollen. Ihr Schwerpunkt liegt dabei auf den Sprachen wie RDF, OWL oder auch den noch zu entwickelnden Sprachen und Konzepten für Regeln, Beweise und Vertrauen. Lokalisiert werden die in diesen Sprachen formulierten Dokumente primär im Web. Eher am Rande werden die Herausforderungen und auch die erweiterten Möglichkeiten auf Seiten der Anwender betrachtet. Die *Software Agenten* eines Anwenders kümmern sich in dem dargestellten Szenario um dessen Belange und Wünsche.

> The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. The effectiveness of such software agents will increase exponentially as more machine-readable Web content and automated services (including other agents) become available. (Berners-Lee et al., 2001)

Das ontologiebasierte Identitäten-Management stellt sich in diesem Zusammenhang als eine wichtige Ergänzung zu den bisher durchgeführten Forschungen zum Semantic Web dar. Es bietet das Potenzial zur Einbindung jeder per URI erreichbaren Ontologie, indem ein entsprechendes Mapping diese in eine lokal vorhandene Teilontologie übersetzt oder sie sogar als Teilontologie selbst integriert. Die bisher im Rahmen der Semantic-Web-Initiative entwickelte Basis stellt das gewissermaßen *leblose* Fundament zur Verfügung. Durch Agenten in Form von ontologiebasierten Identitätsmanagement-Clients erlangen Identitäten Zugriff auf die Strukturen, die das Semantic Web ausmachen sollen. Das Semantic Web hätte damit die Chance, nicht nur einen semantisch vernetzten Wissensraum zu bilden, sondern eine durch seine Benutzer bevölkerte gemeinschaftliche Plattform. Dabei wäre das Semantic Web nur eine Ebene, auf der sich die Identitäten begegnen können.

Die direkte Begegnung wird eine wesentliche Rolle spielen, ohne dass die Anwender auf die Vorteile des Semantic Web verzichten müssen.

Dem Internet wäre so eher die Möglichkeit gegeben, die Vision des vielzitierten von McLuhan (1964) geprägten Begriffes des *Globalen Dorfes* mit zu erfüllen.

A. Anhang

A.1. Inhalt der CD

Die beigefügte CD enthält neben dem vorliegenden Text in digitaler Form (in den Formaten PDF und PostScript) den Quelltext der programmiertechnischen Ausarbeitung des Ontologiensystem-Frameworks. Zudem wurden sämtliche Quellen, welche in digitaler Form verwendet worden sind, im druckfreundlichen PostScript-Format abgespeichert. Die Bibliographie liegt ebenfalls im BibTEX-Format digital vor.

Der Inhalt der CD gliedert sich wie folgt:

- CD-Hauptverzeichnis
  - Diplomarbeit
  - Quellen
    - bibtex
    - digital

A.2. Urheberrecht

Diese Diplomarbeit steht unter der Creative Commons Lizenz "Urhebernennung – Nichtkommerziell – Gegenseitigkeit 1.0". Der kurze offizielle allgemein verständliche Referenztext[1] sei zuerst aufgeführt. Ein abschließender Verweis auf die formal korrekte Fassung folgt.

A.2.1. Referenztext

Sie dürfen:

- den Inhalt vervielfältigen, verbreiten und öffentlich aufführen
- Bearbeitungen anfertigen

Zu den folgenden Bedingungen:

- Namensnennung. Sie müssen den Namen des Autors/Rechtsinhabers nennen.
- Keine kommerzielle Nutzung. Dieser Inhalt darf nicht für kommerzielle Zwecke verwendet werden.
- Weitergabe unter gleichen Bedingungen. Wenn Sie diesen Inhalt bearbeiten oder in anderer Weise umgestalten, verändern oder als Grundlage für einen anderen Inhalt verwenden, dann dürfen Sie den neu entstandenen Inhalt nur unter Verwendung identischer Lizenzbedingungen weitergeben.

Im Falle einer Verbreitung müssen Sie anderen die Lizenzbedingungen, unter die dieser Inhalt fällt, mitteilen. Jede dieser Bedingungen kann nach schriftlicher Einwilligung des Rechtsinhabers aufgehoben werden. Die gesetzlichen Schranken des Urheberrechts bleiben hiervon unberührt.

A.2.2. Erläuterung und Haftungsausschluss

Dies ist eine Zusammenfassung des Lizenzvertrags in allgemeinverständlicher Sprache. Die komplette und einzige juristisch formale Fassung findet sich unter der WWW-Adresse `http://creativecommons.org/worldwide/de/translated-license`.

Der oben aufgeführte Referenztext ist kein Lizenzvertrag. Es ist lediglich ein Referenztext, der den zugrundeliegenden Lizenzvertrag übersichtlich und in allgemeinverständlicher Sprache wiedergibt. Der Referenztext selbst entfaltet keine juristische Wirkung und erscheint im eigentlichen Lizenzvertrag nicht.

[1] entnommen der Creative Commons Website
`http://creativecommons.org/licenses/by-nc-sa/2.0/de/`

A.3. Danksagung

Ich bedanke mich bei Prof. Dr. Lamersdorf und Prof. Dr. von der Heide für die Übernahme der Begutachtung dieser Diplomarbeit. Mein Dank gilt allen Mitarbeitern des Arbeitsbereichs VSIS am Fachbereich Informatik, die mir Motivation und viele Anregungen gegeben und mir die Nutzung der technischen und räumlichen Ausstattung ermöglicht haben – insbesondere Anne Awizen und Tobias Baier.

Mein spezieller Dank gilt Juliane Techen, die mir auf jede erdenkliche Weise Unterstützung gab und mir gerade auch in kritischen Phasen Motivation, Rat und Entlastung zukommen ließ.

Ein abschließender Dank geht an die Open-Source-Community und die Menschen, welche die Idee quelloffener Software fördern. Diese Diplomarbeit wurde mit LyX auf Basis von LaTeX auf einem Debian Gnu/Linux erstellt. Das Layout basiert auf KomaScript.

Literaturverzeichnis

Alden, R., Ames, G., Arai, M., Bartlett, S. W., Carroll, D., Chang, L.-J., Dawson, F., Dobson, K., Feldstein, S., Ganguly, A., Goo, B., Goyal, A. K., Hand, G., Howes, T., Joseph, M., Kelly, K., Letuan, P., Megowan, P., Mori, T., Pandya, R., Ralston, G., Rummel, S., Santullo, M., Seraphin, V., Seely, D., Shmunis, V., Stevens, D., Watkins, M., und Widlewski, H. (1996). vCard - The Electronic Business Card. Specification 2.1, versit Consortium.

Apache Software Foundation (2004). Axis/Compare - Apache Web Services. http://wiki.apache.org/ws/Axis/Compare.

Aristoteles (1998). *Organon*, Band Kategorien, Hermeneutik oder vom sprachlichen Ausdruck. Meiner.

Baader, F. und Hollunder, B. (1991). KRIS: Knowledge Representation and Inference System. *SIGART Bulletin*, 2(3):8–14.

Baader, F. und Nutt, W. (2003). *The Description Logic Handbook*, Kapitel Basic Description Logics, Seiten 47–100. Cambridge University Press.

Backus, J. W. und Herrick, H. (1954). IBM 701 Speedcoding and other Automatic-Programming Systems. In *Proceedngs of the ONR Symposium Automatic Programming for Digital Computers*, Seiten 106–113. Office of Naval Research.

Baier, T., Zirpins, C., und Lamersdorf, W. (2003). Digital identity: How to be someone on the net. In palma dos Reis, A. und Isaias, P., Herausgeber, *e-Society 2003*, Band II, Seiten 815–820. IADIS International association for development of the information society, IADIS Press.

Barg, P. (1996). Automated Inference of DATR Theories. In Bock, H.-H. und Polasek, W., Herausgeber, *Data Analysis and Information Systems: Statistical and conceptional appraches*. Gesellschaft für Klassifikation e.V., Springer-Verlag.

Berners-Lee, T. (2000). Metadata Architecture. http://w3.org/DesignIssues/Metadata.

Berners-Lee, T., Harmelen, F. v., Horrocks, I., Brickley, D., Dean, M., Decker, S., Fikes, R., Hayes, P., Heflin, J., McDermott, D., und Swick, R. R. (2000). DAML-ONT Initial Release. http://www.daml.org/2000/10/daml-ont.html.

Berners-Lee, T., Hendler, J., und Lassila, O. (2001). The Semantic Web. *Scientific American*. http://www.sciam.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

Bontchev, V. V. (1998). Methodology of Computer Anti-Virus Research. Diplomarbeit, Universität Hamburg.

Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, (82):353–367.

Bowen, S. und Maurer, F. (2002). Using peer-to-peer technology to support global software development – some initial thoughts. In *Proceedings of the ICSE Int. Workshop on Global Software Development*.

Brachmann, R. J. (1977). What's in a concept: structural foundations for semantic networks. *International Journal of Man-Machine Studies*, 9(2):127–152.

Brachmann, R. J. und Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216.

Bresciani, P., Franconi, E., und Tessaris, S. (1995). Implementing and testing expressive description logics: a preliminary report. In *Proceedings of the 1995 Description Logig Workshop*, Seiten 131–139.

Brink, L. v. d. und Tauber, J. (2004). XML software - xslt engines. http://www.xmlsoftware.com/xslt.html.

Brooker, T. (1960). High level Programming Languages on the Manchester Computers. Technical report, University of Manchester.

Carroll, J. (2003). RE: WOWG: Need to know about your implementations – ASAP. http://lists.w3.org/Archives/Public/www-webont-wg/2003May/0057.html.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., und Rice, J. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of AAAI*, Seiten 600–607.

Clark, J. (1995). Report on the W3C style sheet workshop: DSSSL and DSSSL Lite on the Web. http://w3.org/Style/951106_Workshop/report1.html#clark.

Dawson, F. und Howes, T. (1998). vCard MIME Directory Profile. Request for Comments 2426, The Internet Society.

Dickinson, I. (2004). *The Jena 2 Ontology API*. Hewlett-Packard Development Company. http://jena.sourceforge.net/ontology/index.html.

Dillon, A. und Gushrowsik, B. (2000). Genres and the Web: Is the personal home page the first uniquely digital genre? *Journal of the American Society for Information Science*, Seiten 202–205.

Döring, N. (1999). *Sozialpsychologie des Internet*. Hogrefe.

Dublin Core Metadata Initiative (2004). DCMI Frequently Asked Questions. http://dublincore.org/resources/faq/.

Dumbill, E. (2002). XML Watch: Finding friends with XML and RDF. *xmlhack.com*. http://www-106.ibm.com/developerworks/xml/library/x-foaf.html.

Fensel, D., Harmelen, F. v., und Horrocks, I. (1999). OIL: A Standard Proposal for the Semantic Web. Technical Report IST-1999-10132, Vrije Universiteit Amsterdam.

Foucault, M. (1971). *Die Ordnung der Dinge. Eine Archäologie der Humanwissenschaften (Les mots et les choses)*. Suhrkamp Verlag.

Frege, G. (1892). Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, Seiten 25–50. vorliegende Ausgabe: Vandenhoeck und Ruprecht, 1994.

Gamma, E. (1994). *Design patterns : elements of reusable object-oriented software*. Addison-Wesley.

Genesereth, M. R. (1991). Knowledge Interchange Format, Version 3.0, Reference Manual. In Allen, J., Fikes, R., und Sandewall, E., Herausgeber, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Seiten 599–600. Morgan Kaufmann.

Goguen, J. (2003). SEEK-notes - Notes for the SEEK Project. http://www.cse.ucsd.edu/groups/tatami/seek/.

Goldfarb, C. F. und Prescod, P. (2002). *XML Handbook*. Prentice Hall.

Gong, L. (2002). Project JXTA: A Technology Overview. Technical report, Sun Microsystems. http://www.jxta.org/project/www/docs/jxtaview_01nov02.pdf.

Gottlob, G., Koch, C., und Pichler, R. (2002). Efficient algorithms for processing XPath queries. In *Proceedings of the 28th international conference on very large data bases (VLDB 2002)*.

Gruber, T., Pang, D., und Rice, J. (1990). Ontolingua: A language to Support Shared Ontologies. Technical Report KSL-90-84, Knowledge Systems Laboratory, Stanford University.

Gruber, T. R. (1993). A Translation Approach to portable Ontology Specifications. Technical Report KSL 92-71, Stanford University.

Guha, R. V. (2000). *rdfDB Query Language*. http://guha.com/rdfdb/query.html.

Haarslev, V. und Möller, R. (2001). Description of the RACER system and its applications. In Goble, C., McGuinness, D. L., Möller, R., und Patel-Schneider, P. F., Herausgeber, *Working Notes of the 2001 International Description Logics Workshop*, Seiten 132–141.

Halevy, A. Y. (2003). Data Integration: A Status Report. In Weikum, G., Schöning, H., und Rahm, E., Herausgeber, *BTW 2003: Datenbanksysteme für Business, Technologie und Web*, Seiten 24–29. Bonner Köllen Verlag.

Horrocks, I. (1998a). *FaCT Reference Manual - Version 1.6*. Department of Computer Science, University of Manchester.

Horrocks, I. (1998b). The FaCT system. In Swart, H. d., Herausgeber, *Proceedings of the 2nd International Conference on Analytic Tableaux and Related Methods*, Band 1397 of *Lecture Notes in Artificial Intelligence*, Seiten 307–312. Springer.

Horrocks, I. (2003). Re: More results. http://lists.w3.org/Archives/Public/www-webont-wg/2003Sep/0168.html.

Jeffery, K. G. (1998). Metadata: An Overview and some Issues. In *11th ERCIM Database Research Group Workshop on Metadata for Web Databases*, Band W002 of *ERCIM Workshop Proceedings*. European Research Consortium for Informatics and Mathematics.

Karp, P. D., Chaudhri, V. K., und Thomere, J. (1999). XOL: An XML-Based Ontology Exchange Language. http://www.ai.sri.com/pkarp/xol/xol.html.

Karp, P. D., Myers, K., und Gruber, T. (1995). The generic frame protocol. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Seiten 768–774.

Kay, M. (2000). *XSLT Programmer's Reference*. Programmer to Programmer. Wrox Press.

Kent, R. E. (1999). Conceptual Knowledge Markup Language: The Central Core. In Gaines, B. R., Kremer, R. C., und Musen, M. A., Herausgeber, *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management*.

Khan, F. (2004). Wireless messaging with JXTA, Part 1: Using JXTA technology. http://www-106.ibm.com/developerworks/wireless/library/wi-jxta/.

Knuth, D. E. (1978). *Stanford Computer Science Report*, Kapitel Tau Epsilon Chi, a system for technical text, Seite 198 ff. Stanford University.

Knuth, D. E. (2000). *The TeXbook*. Addison Wesley.

Koivunen, M.-R. und Miller, E. (2001). W3C Semantic Web Activity. In Hyvönen, E., Herausgeber, *Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*, Seiten 27–43. HIIT Publications.

Kunze, C. P. (2003). Digitale Identität und Identitäts-Management. Diplomarbeit, Universität Hamburg. Abschnitt: Identität in Psychologie und Sozialwissenschaften - 3.1.1-3.1.2.

Lapsley, P., Bier, J., Shoham, A., und Lee, E. A. (1997). *DSP Processor Fundamentals: Architectures and Features*. IEEE Press Series on Signal Processing. IEEE Press.

Levesque, H. J. (1979). *Associative Networks*, Kapitel A procedural Semantics for Semantic Networks. Academic Press.

Lie, H. W. und Bos, B. (1999). *Cascading Style Sheets - designing for the Web*. Addison Wesley.

Linnaeus, C. (1735). *Systema naturae*, Band Lugduni Batavorum. Bokförl. Rediviva. Jahr der vorliegenden Ausgabe: 1977.

Luck, K. v. und Owsnicki-Klewe, B. (1990). KL-ONE: Eine Einführung. Technical Report IWBS Report 106, Wissenschaftliches Zentrum der IBM Deutschland, Institut für Wissensbasierte Systeme.

Luke, S., Spector, L., und Rager, D. (1996). Ontology-Based Knowledge Discovery on the World-Wide Web. In Franz, A. und Kitano, H., Herausgeber, *Working Notes of the Workshop on Internet-Based Information Systems at the 13th National Conference on Artificial Intelligence*, Seiten 96–102. AAAI Press.

Madhavan, J., Bernstein, P. A., und Rahm, E. (2001). Generic schema matching with cupid. In *The VLDB Journal*, Seiten 49–58.

Marshall, C. C. und Shipman, F. M. (2003). Which Semantic Web? In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, Seiten 57–66.

McGuinness, D. L. und Harmelen, F. v. (2004). OWL Web Ontology Language - Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

McIllroy, M. D. (1969). Mass Produced Software Components. In Naur, P. und Randell, B., Herausgeber, *Software Engineering*, Seiten 138–156. Nato Science Committee, NATO Scientific Affairs Division.

McLuhan, M. (1964). *Understanding Media - The Extensions of Man*. MIT Press.

Miller, E. und Hendler, J. (2004). Web Ontology Language (OWL). http://w3.org/2004/OWL/.

Miller, L., Seaborne, A., und Reggiori, A. (2002). Three Implementations of SquishQL, a Simple RDF Query Language. Technical Report HPL-2002-110, Hewlett-Packard Labs.

Molva, R., Tsudik, G., Herreweghen, E. V., und Zatti, S. (1992). Kryptoknight authentication and key distribution system. In *ESORICS*, Seiten 155–174.

Mylopoulos, J. (1980). A Perspective for Research on Conceptual Modelling. In *Proceedings of the workshop on Data abstraction, databases and conceptual modelling*. ACM Press.

Nejdl, W., Wolpers, M., und Capelle, C. (2000). The RDF Schema Specification Revisited. In *Proceedings of Modellierung*.

Neumann, J. v. (1946). Principles on Large Scale Computing Machines. In A. H. Taub, Herausgeber, *John von Neumann - Collected Works*, Band V, Seiten 1–33. Pergamon Press.

Open Group (1997). *DCE 1.1: Remote Procedure Call*. Number C706 in CAE Specification. The Open Group, technical standard edition.

Pan, J. Z. und Horrocks, I. (2001). Metamodeling Architecture of Web Ontology Languages. In Cruz, I. F., Decker, S., Euzenat, J., und McGuiness, D., Herausgeber, *Proceedings of the First Semantic Web Working Symposium*, Seiten 131–149.

Quillian, M. R. (1968). *Semantic Information Processing*, Kapitel 4. Semantic Memory, Seiten 227–270. MIT Press.

Quine, W. v. O. (1979). *Von einem logischen Standpunkt: neun logische-philosophische Essays*. Ullstein, peter bosch edition.

Reynolds, D. (2003). [Fwd: Re: [Jena-devel] RE: WOWG: Need to know about your implementations – ASAP]. http://lists.w3.org/Archives/Public/www-webont-wg/2003May/0068.html.

Rickert, T. (2004). Integration von Datenschutzmechanismen in Identitäteninfrastrukturen. Diplomarbeit, Universität Hamburg.

Sack, A. (2005). Ein Session-Konzept für eine Identitäts-Infrastruktur. Diplomarbeit, Universität Hamburg.

Schank, R. C. (1972). Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 3(4).

Schmidt-Schauß, M. (1989). Subsumption in KL-ONE is undecidable. In Brachman, R. J., Levesque, H. J., und Reiter, E., Herausgeber, *Proceedings of the 1st International Conference on the Principles of Knowledge Representation and Reasoning*, Seiten 421–431. Morgan Kaufmann.

Schmitz, L. (1995). *Syntaxbasierte Programmierwerkzeuge*. B.G. Teubner.

Searle, J. R. (1986). *Geist, Hirn und Wissenschaft*. Suhrkamp Verlag.

SGML Users' Group (1990). A Brief History of the Development of SGML. http://www.sgmlsource.com/history/sgmlhist.htm.

Spafford, E. H. (1989). The Internet Worm Incident. In *ESEC '89 Proceedings of the 2nd European Software Engineering Conference, University of Warwick*.

Sperberg-McQueen, C. M. und Goldstein, R. F. (1994). HTML to the Max A Manifesto for Adding SGML Intelligence to the World-Wide Web. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*. National Center for Supercomputing Applications.

Strachey, C. (1965). A General Purpose Macrogenerator. *The Computer Journal*, 8(3):225–241.

Tidwell, D. (2001). *XSLT*. O'Reilly.

Wikimedia Foundation (2004a). Wikipedia - die freie Enzyklopädie: Taxonomie. http://de.wikipedia.org/wiki/Taxonomie.

Wikimedia Foundation (2004b). Wikipedia - die freie Enzyklopädie: Web Service. http://de.wikipedia.org/wiki/Web_Service.

Wikimedia Foundation (2005). Wikipedia - die freie Enzyklopädie: E-Mail. http://de.wikipedia.org/wiki/E-Mail.

Wittgenstein, L. (1953). *Philosophische Untersuchungen*. Suhrkamp Verlag. Jahr der vorliegenden Ausgabe: 1999.

Wollenweber, F. (2004). Kollaborative Nutzung des World Wide Webs. Diplomarbeit, Universität Hamburg.

Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Zudem erkläre ich, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliothek des Fachbereichs Informatik der Universität Hamburg einverstanden bin.

Hamburg, den 2. Februar 2005

Gordian Kaulbarsch