Universität Hamburg
FB Informatik

Eine offene Kommunikationsstruktur für verteiltes kooperatives Arbeiten

Diplomarbeit

eingereicht bei

- Erstbetreuer -

Prof. Dr. Joachim W. Schmidt
Arbeitsbereich Softwaresysteme
Forschungsschwerpunkt Informations- und Kommunikationstechnik
Technische Universität Hamburg-Harburg

- Zweitbetreuer -

Prof. Dr.-Ing. Karl Kaiser
Arbeitsbereich Technische Informatiksysteme
Fachbereich Informatik
Universität Hamburg

von

Jörn Fornfeist
Matrikelnummer 4725449

9. Juni 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einleitung

Die Entwicklung von *Informationssystemen* hat durch das nahezu exponentielle Wachstum des *Internet* seit Mitte der neunziger Jahre einen starken Schub erfahren. Wurden in früheren Jahren in der Regel Informationssysteme für die interne Verwendung in Unternehmen oder Verwaltung entwickelt, so eröffnete das Internet nun die Möglichkeit Informationen für externe *Benutzer*, also Kunden oder Bürger, besonders aktuell und kostengünstig aufzubereiten und bereitzustellen. Eine der mögliche Definitionen für Informationssystemen wird im nachfolgenden Zitat gegeben.

> „Information ist zweckbezogenes Wissen, das durch die Anwendung von Regeln und Anweisungen auf Daten entsteht. Ein Informationssystem ist ein System zur Aufnahme, Speicherung, Verarbeitung und Wiedergabe von Informationen. Es besteht aus Gesamtheit der Daten und Verarbeitungsanweisungen. [...]“
>
> Quelle: Page et al. (1993, S. 97)

Als Einschränkung zu dieser Definition soll hier unter einem Informationssystem ein System verstanden werden, das große Mengen von Daten in Datenbanken verwaltet und diese Daten einer großen Zahl von Benutzern zu einem bestimmten Zweck zur Verfügung stellt.

Bei der Entwicklung von *Internet-Informationssysteme* haben Erfahrungen gezeigt, dass nur solche Systeme von den Benutzern gewürdigt werden, die einen echten Mehrwert gegenüber den traditionellen Informationswegen schaffen. Ein Möglichkeit der Generierung von Mehrwert stellt die gezielte und *individuelle* Informationsaufbereitung für die Benutzer dar, die gemeinhin unter dem Begriff der *Personalisierung* geführt wird. Dabei wurde die Personalisierung häufig auf die Individualisierung von *Benutzeroberflächen* oder die Erhebung von *Benutzerprofilen* für Marketing-Zwecke reduziert, wie Sehring[1] ausführt.

Eine weitere und weitaus allgemeinere Ausprägung der Personalisierung stellt die sogenannte *inhaltliche Personalisierung* dar[2]. Dieser liegen Annahmen aus der *Semiotik* und der *Erkenntnistheorie* nach Ernst Cassirer[3] zugrunde, die unter anderem davon ausgehen, dass für

[1] Sehring (2004), S. 22.
[2] Rossi et al. (2001).
[3] Schmitz-Rigal (2002).

unterschiedliche Benutzer individuelle *Zeichen* Verwendung finden müssen, um identische *Inhalte* zu übermitteln. Informationssysteme können diesem Umstand Rechnung tragen, indem sie Teile der *Sinnstrukturen* ihrer Benutzer in Form von sogenannten *Asset-Modellen* verwalten und auf dieser Basis geeignete Zeichen zur Informationsaufbereitung und -präsentation verwenden. Ein entsprechendes Modell zur *begriffs- oder konzeptorientierten Inhaltsverwaltung* wird in Schmidt et al. (2003) und Sehring (2004) entwickelt und beschrieben. Insbesondere wird in diesem Modell die Einheit von *Konzepten* und verwalteten Inhalten betont, die zu *Konzept-Inhaltspaaren*, sogenannten *Assets*, verschmolzen werden.[4]

Nun stellen die abgebildeten menschlichen Sinnstrukturen keineswegs stabile *Entitäten* dar, sondern wandeln sich je nach *Erkenntnisstand* und *Anwendungsfokus* ihres Eigentümers in mehr oder weniger schneller Periodizität. Auch Cassirer fordert in seiner Philosophie *Offenheit* und *Dynamik* für den menschlichen *Erkenntnisprozess*, was nichts anderes als die Möglichkeit der Entstehung neuer sowie die Wandlung und das Vergehen bestehender Konzepte einer Sinnstruktur zu jedem Zeitpunkt bedeutet. In *IT-Begriffen* ausgedrückt: Das verwendete *Schema* zur Modellierung der Sinnstruktur kann ebenfalls nicht stabil bleiben, sondern muss sich in jeder Phase des *Lebenszyklus* eines entsprechenden Informationssystems wandeln können.[5] Spätestens hier stoßen dann die bisherigen Vorgehensweisen und Technologien bei der Entwicklung von Informationssystemen an ihre Grenzen. Zwar stellen *Schemaevolution* sowie die *Migration* bestehender Altdaten Problemfelder dar, die bereits eingehender Betrachtung unterzogen wurden[6]. Dennoch wird für einen Evolutionsschritt der Eintritt in einen besonderen Betriebszustand vorausgesetzt, der den normalen Betriebsverlauf und damit den Zugriff durch die Benutzer unterbricht.

An dieser Stelle tritt noch ein weiterer Aspekt der Entwicklung von Informationssystemen in Erscheinung. Wie die obigen Definitionen von Informationssystemen schon angedeutet haben, besteht ein herkömmliches Informationssystem nicht nur aus dem Datenschema der verwendeten Datenbanken, sondern beinhaltet auch *Regeln* und *Anweisungen* zur Erfassung, Verarbeitung und Wiedergabe der Daten, die die *Semantik* der Daten maßgeblich bestimmen. Verändert sich nun die Sinnstruktur eines Benutzers und damit die Semantik der gespeicherten Daten, so sind Änderungen an der *Anwendungslogik* des Informationssystems unvermeidlich. Allerdings kann die manuelle Pflege und Nachführung der Logik kaum mit der Geschwindigkeit eines menschlichen Erkenntnisprozesses schritthalten. Sehring stellt deshalb einen *Modell-Compiler* vor, der auf der Basis einer speziellen *Asset-Definitionssprache*[7], zu jedem Schritt der Schemaevolution die entsprechende Logik ableitet und automatisch *generiert*[8]. Darüberhinaus beschreibt Sehring eine vollständige *Architektur*, die sich durch die Identifikation der wesentlichen *Komponenten* eines begriffsorientierten Inhaltsverwaltungsystems sowie deren zweckmäßige *Komposition* auszeichnet, um die Entwicklung und den ununterbrochenen Betrieb eines derartigen Systems zu ermöglichen. Es bleibt abzuwarten, inwieweit die dort gewonnenen Ergebnisse auch für andere Informationssysteme von Bedeutung sein können, die zwar nicht im klassischen Sinne Inhalte verwalten, welche einer *subjektiven Deutung* durch die Benutzer unterliegen, aber ganz allgemein Informationen verwalten, die kein stabiles Schema aufweisen. Insbesondere würden Entwickler von den ewig gleichen Tätigkeiten bei der Umsetzung der Anwendungslogik, wie *Objekt-relationales Mapping* und Aufbereitung und Interpretation von Daten in der Präsentationsschicht, entbunden, zumal diese häufig über

[4] Vgl. Abschnitt 2.2, Ein offenes begriffsorientiertes Modell
[5] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[6] U.a. in Wienberg et al. (2002).
[7] Sehring (2004), S. 39ff.
[8] Sehring (2004), S. 87ff.

eine Vielzahl von Komponenten verteilt und damit nur sehr schwer zu warten ist.

Im vorangegangen Teil der Einleitung wurde u.a. die inhaltliche Personalisierung als Motivation der begriffsorientierten Inhaltsverwaltung angeführt. Ein Punkt, der dabei vernachlässigt wurde, ist die Notwendigkeit, Benutzer in begriffsorienitierten Systemen zu verwalten, um die Zuordnung des individuellen *Asset-Modells* zu ermöglichen. Darüberhinaus ist auch die Abstraktion von den einzelnen Benutzern, also die *Gruppe*, die eine *gemeinsame Sicht* und damit ein gemeinsames *Asset-Modell* repräsentiert, eine sinnvolle Entität. Durch die Zugehörigkeit zu Gruppen, kann ein Benutzer seine *inhaltliche Nähe* zu der jeweiligen Gruppe und ihren Mitgliedern ausdrücken. Umgekehrt können über die Gruppe *Zugriffsrechte* und *Rollen* für gemeinsame Inhalte zugeteilt werden. Insgesamt kann auf diese Weise eine *Organisationsstruktur* aufgebaut werden, die die Grundlage für einen weiteren Aspekt der begriffsorientierten Inhaltsverwaltung schafft. Es handelt sich um das *kooperative begriffsorientierte Arbeiten*.

Kooperatives begriffsorientiertes Arbeiten bezeichnet die *koordinierte Aktivität* mehrerer *Akteure*[9] mit dem Ziel des *Erkenntnisgewinns*. Unter Akteuren sollen hier Benutzer oder Systeme verstanden werden, die eine Aktivität oder einen Prozess anstoßen. Eine Richtung dieser kooperativen Arbeit wurde bereits mit der inhaltlichen Personalisierung vorgestellt, denn die *Ableitung* einer subjektiven Sicht durch eine *Organisationseinheit* setzt das Vorhandensein der *allgemeineren Sicht* einer übergeordneten Einheit sowie die Einrichtung von *Zugriffspfaden und -rechten* voraus. Der umgekehrte Weg der kooperativen Arbeit, also die *Publikation* von Inhalten aus einer subjektiven Sicht in die allgemeinere Sicht einer übergeordneten Organisationseinheit, wird ebenfalls unterstützt. Wie der Begriff der Ableitung schon andeutet, sieht die von Sehring entwickelte Architektur für begriffsorientierte Inhaltsverwaltungssysteme eine *inkrementelle Modellevolution* auf Basis der jeweiligen *Asset-Modelle* der Organisationseinheiten vor. Zu diesem Zweck werden, von dem bereits erwähnten *Modell-Compiler*, auch Komponenten zur *Modelladaption* generiert, die die Kooperation in beider Richtungen der Organisationsstruktur unterstützen.

Ein weiterer Punkt, der Systeme zur begriffsorientierten Inhaltsverwaltung von herkömmlichen Informationssystemen unterscheidet, ist der besondere Augenmerk, der auf den sogenannten *Werkcharakter* von Inhalten gelegt wird. Damit wird die besondere Bedeutung des Entwicklungsprozesses, in dessen Kontext die beschriebenen Entitäten entstanden sind, gegenüber dem eigentlichen *Werkprodukt*, etwa Verweisen auf *multimediale Artefakte*, betont. Diese Sichtweise geht wieder auf die Theorie Cassirers zurück, die einen dynamischen Erkenntnisprozess postuliert, der stetig voranschreitet und dabei einen variablen *Erkenntnisgewinn* erzielt, ohne jedoch ein definiertes Ende zu erreichen. Erkenntnis ist in erster Linie der Prozess und weniger sein Ergebnis. Demzufolge werden alle (kooperativen) Prozesse, die auf verwalteten Inhalten operieren, ebenfalls als durch ein begriffsorientiertes System zu beschreibende Entitäten aufgefasst. Hervorzuheben ist hier, dass die Beschreibung der (kooperativen) *Asset-Prozesse* (sogenannte *Werke*) kein separates Inhaltsmodell erfordert. Vielmehr erlaubt die generische Natur des bisherigen Modells, die Beschreibung mit vorhandenen Mitteln.

1.1 Ziele der Arbeit

In Sehring (2004) wird u.a. ein Inhaltsmodell sowie eine *komponentenbasierte* Architektur für Systeme zur *begriffsorientierten Inhaltsverwaltung* beschrieben. Ein Ziel dieser Arbeit ist

[9] Vgl. *Business Conversations* in Matthes (1997).

es, basierend auf den Leistungen von SEHRING und vor dem Hintergrund der Erfahrungen mit bestehenden Prototypen dieser Klasse, die besonderen Vorzüge eines *verteilten Entwurfes* für begriffsorientierte Systeme, aufzuzeigen. Dabei sollen, neben den Anforderungen der speziellen Problemklasse, insbesondere auch die *allgemeinen Entwurfsziele* verteilter Systeme Berücksichtigung finden, wie sie in Coulouris et al. (1994) beschrieben werden.

Darauf aufbauend ist es ein weiteres Ziel dieser Arbeit, eine *prototypische* verteilte *Infrastruktur* für *kooperatives begriffsorientiertes Arbeiten* zu entwickeln und in der Programmiersprache *Java*, unter Zuhilfenahme *offener Standards*, umzusetzen. Dabei sollen Konzepte zur Steigerung von *Performanz*, *Zuverlässigkeit* und *Data Awareness*, wie *Replikation* und *Notifikation*, intensiv Verwendung finden.

Als wichtiges Ziel der Arbeit und wesentlicher Bestandteil der Infrastruktur werden ein spezielles *Werkbeschreibungsformat* sowie ein *Werkmodell* zur Beschreibung und zur Ausführung aller vorkommenden (kooperativen) Prozesse auf Inhalten entworfen und implementiert, die den *dynamischen* Aspekten und der *Werkorientierung* der begriffsorientierten Inhaltsverwaltung Rechnung tragen.

Das letzte Ziel dieser Arbeit ist, alle durch die *Entwurfsentscheidung* zugunsten einer verteilten begriffsorientierten Infrastuktur bedingten Komponenten, etwa die *Namens-, Verzeichnis- und Nachrichtenverwaltung*, sowie deren *Kooperation* nicht in einer Sonderrolle zu betrachten und umzusetzen. Vielmehr können die statischen und dynamischen Aspekte dieser Komponenten durch die vorhandenen *Asset-* und Werkmodelle beschrieben werden. Das gleiche gilt für die Komponenten zur *Benutzer-, Gruppen-, Rechte- und Werkverwaltung*, die nicht durch die verteilte Systemarchitektur bedingt sind.

1.2 Gliederung der Arbeit

Das nachfolgende 2. Kapitel stellt zunächst einige Fallstudien zum *begriffsorientierten Arbeiten* vor, bevor es ein, aus den Fallstudien abstrahiertes, generisches Modell zur *begriffsorientierten Inhaltsverwaltung* beschreibt. Dabei werden die Eigenschaften der *Offenheit* und *Dynamik* des Modells gesondert betrachtet. Anschließend werden die Abhängigkeiten der vorliegenden Arbeit von anderen Arbeiten zum begriffsorientierten Arbeiten aufgezeigt sowie ein Überblick über verwandte Themen und Gebiete gegeben.

Das 3. Kapitel zeigt die Implikationen der Eigenschaften Offenheit und Dynamik für den Entwurf einer begriffsorientierten Infrastruktur auf. Insbesondere werden die Vorteile eines *verteilten Entwurfes* herausgearbeitet sowie die Abbildung von Konzepten des begriffsorientierten Modells auf Konzepte verteilter Systeme dargelegt. Darüberhinaus wird ein *Werkbeschreibungsformat* und *Werkmodell* für Prozesse entwickelt, das der *Werkorientierung* der begriffsorientierten Inhaltsverwaltung Rechnung trägt. Abschließend führt das Kapitel in eine Architektur begriffsorientierter Systeme ein, die der vorliegenden Arbeit als Grundlage dient.

Das folgende 4. Kapitel führt mögliche Technologien für die Implementation der begriffsorientierten Infrastruktur auf und stellt eine exemplarische *Konfiguration* zusammen. Desweiteren wird eine mögliche Umsetzung der *Werkbeschreibungssprache* sowie die *prototypische* Implementation der im vorherigen Kapitel entworfenen Komponenten gemäß der genannten Systemarchitektur beschrieben.

Zum Abschluß der Arbeit fasst das 5. Kapitel die erreichten Ergebnisse zusammen, bewertet

die erfolgreiche Umsetzung der Ziele dieser Arbeit und gibt einen Ausblick auf zu untersuchende Problemfelder und mögliche Entwurfs- oder Technologiealternativen.

1.3 Danksagung

Mein Dank gilt Prof. Dr. Joachim W. Schmidt, Arbeitsbereich Softwaresysteme (STS), Forschungsschwerpunkt Informations- und Kommunikationstechnik der Technischen Universität Hamburg-Harburg für die Betreuung dieser Diplomarbeit. Weiterhin danke ich Dr. Hans-Werner Sehring, vom oben genannten Arbeitsbereich STS, für unzählige Gespräche, Ideen und Klarstellungen zum konzeptionellen Unterbau und zur formalen Korrektheit dieser Arbeit. Auch dem gesamten übrigen Team von STS gilt mein Dank für die anregende und freundschaftliche Arbeitsumgebung.

Herrn Prof. Dr.-Ing. Karl Kaiser, Arbeitsbereich Technische Informatiksysteme, Fachbereich Informatik der Universität Hamburg möchte ich meinen Dank für die Übernahme der Zweitbetreuung aussprechen.

Zum Abschluss bin ich meiner Frau, meinen Kindern und meinen Eltern zutiefst dankbar, dass sie mir die Möglichkeit zum erfolgreichen Abschluss meines ausschweifenden Studiums gegeben haben, obwohl die Kollision mit den Familieninteressen abzusehen war. Ich hoffe, dass es sich für meine Familie als eine gute Investition in die Zukunft herausstellen wird.

Kapitel 2

Begriffsorientiertes Arbeiten

Dieses Kapitel gibt im Abschnitt 2.1 zunächst einen Überblick über Projekte zum *begriffsorientierten Arbeiten*, die am Arbeitsbereich Softwaresysteme (STS) der Technischen Universität Hamburg-Harburg (TUHH) durchgeführt wurden. Anschließend führt Abschnitt 2.2 ein *begriffsorientiertes Modell* ein, das von den konkreten *Inhaltsmodellen* der beschriebenen Projekte abstrahiert und ein *generisches* Modell darstellt, welches sich durch besondere *Offenheit* auszeichnet. Der nachfolgende Abschnitt 2.3 geht auf die *dynamischen* und *kooperativen* Aspekte des offenen begriffsorientierten Modells ein und führt hierzu das Konzept des *Werkes* ein. Zum Schluss des Kapitels grenzt Abschnitt 2.5 den vorliegenden Text gegenüber anderen Arbeiten zum begriffsorientierten Modell am Arbeitsbereich STS ab und gibt einen Überblick über verwandte Themen und Gebiete.

2.1 Fallstudien: Projekte zum begriffsorientierten Arbeiten

Der Arbeitsbereich STS der TUHH hat seine Wurzeln im Bereich der Entwicklung von *Datenbanksystemen* und deren nahtloser *Integration* mit Programmiersprachen. Mit der fortschreitenden Entwicklung des *Internet*, seit Anfang der neunziger Jahre, hat sich der Fokus auf die Entwicklung von *Internet-Informationsystemen* verschoben, die in ihrer Mehrzahl auf Datenbanken als *Massenspeicherabstraktion* beruhen. Im Verlaufe dieser Refokussierung wurde eine Entwicklung über *Content-Management-Systeme*[1], *Digitale Bibliotheken*[2] und *Dokumenten-Management-Systeme*[3] vollzogen, die in jeweils spezialisierten *Problemabstraktionen* mündete. Als Gemeinsamkeit der durchgeführten Projekte kann jedoch vielfach die *Begriffsorientierung* der Modelle ausgemacht werden. Die nachfolgenden Abschnitt geben einen Überblick über die Projekte Warburg Electronic Library (WEL), Technologie- und Forschungsinformationssystem (TeFIS) und sap für Hamburg.

[1] CoreMedia (2004a).
[2] Niedereé et al. (1996).
[3] infoAsset (2004).

2.1.1 WEL - Warburg Electronic Library

Die WEL ist ein interdiziplinäres Projekt des Arbeitsbereiches STS der TUHH und des Kunstgeschichtlichen Seminars der Universität Hamburg. Ziel des mehrjährigen Projektes war es zunächst, den vom Kunsthistoriker Martin Warnke initiierten und auf Einsichten des Kunsthistorikers Aby Warburg[4] und des Philosophen Ernst Cassirer[5] beruhenden Bildindex zur politischen Ikonographie (BPI) und die sogenannte Regentenbank (Rg) durch das *Internet* einer breiteren und räumlich verteilten Gemeinschaft von Wissenschaftlern (*community*) zeitnah zugänglich zu machen.

Ein *Index* bezeichnet ein konkretes *Begriffssystem*, das in der Regel *Begriffe*, also benannte *Konzepte*, einer bestimmten *Anwendungsdomäne* zusammenfasst und miteinander in Beziehung setzt (*Taxonomie*) und eine Menge von *Entitäten* der Anwendungsdomäne, die durch das Begriffssystem *klassifiziert* werden. Im Falle des BPI handelt es sich bei der Anwendungsdomäne um das Feld der *politischen Ikonographie*. Der BPI gibt für sein Begriffsystem ein bestimmtes *Modell* von Konzepten und *semantischen Beziehungen* vor, das im Nachfolgenden kurz beschrieben werden soll.

Die ersten beiden Ebenen des BPI-Modelles werden durch *Schlagworte* und *Unterschlagworte* gebildet, die untereinander durch *Generalisierung* bzw. *Spezialisierung* in semantischer Beziehung stehen. Die nachfolgenden n ($n = 0..\infty$) Ebenen des BPI werden durch eine Hierarchie von *Stichwörtern* gebildet, die als benannte *Aggregate* von *Bildkarten* und (Unter-) Stichwörtern und nicht als benannte Konzepte der Anwendungsdomäne zu verstehen sind. Schlagworte und Unterschlagworte klassifizieren Bildkarten und Stichwörter. Umgekehrt *definieren* Bildkarten und Stichwörter Schlagworte und Unterschlagworte. Die beiden genannten semantischen Beziehungen sind jedoch, im Gegensatz zum Paar Generalisierung und Spezialisierung, nicht *invers* und stellen damit zwei echte Beziehungen und nicht die beiden *Rollen* einer binären Beziehung dar. Bildkarten sind eine Aggregation von *Metadaten* und Verweisen auf *multimediale Artefakte*. Weiterhin können auf Bildkarten Verweise auf andere benannte Konzepte enthalten sein, wie z.B. im BPI unter dem Attribut `Motiv`.

Im Zuge des Projektanalyse wurden eine Reihe von weiterführenden Konzepte gegenüber dem ursprünglichen Modell des BPI in Papierform identifiziert. Es handelt sich um die Einführung von
Medienkarten-Entitäten, die auf beliebige mediale Artefakte verweisen und deren multimedialen Inhalte direkt visualisieren sowie die *Mehrfachverschlagwortung* der Medienkarten-Entitäten. Weiterhin wurden die Konzepte der *inhaltlichen Personalisierung* von Begriffssystemen und Medienkarten-Entitäten sowie der *kooperativen Arbeit* identifiziert.

Tiefere Einblicke in die WEL werden in Niedereé et al. (1996), Bruhn (1999) und Fornfeist (2003) vermittelt. Für die weiteren Betrachtungen im Rahmen dieser Arbeit sind besonders die zuletzt genannten Konzepte von Bedeutung.

[4] Warburg-Haus (2004).
[5] Schmitz-Rigal (2002).

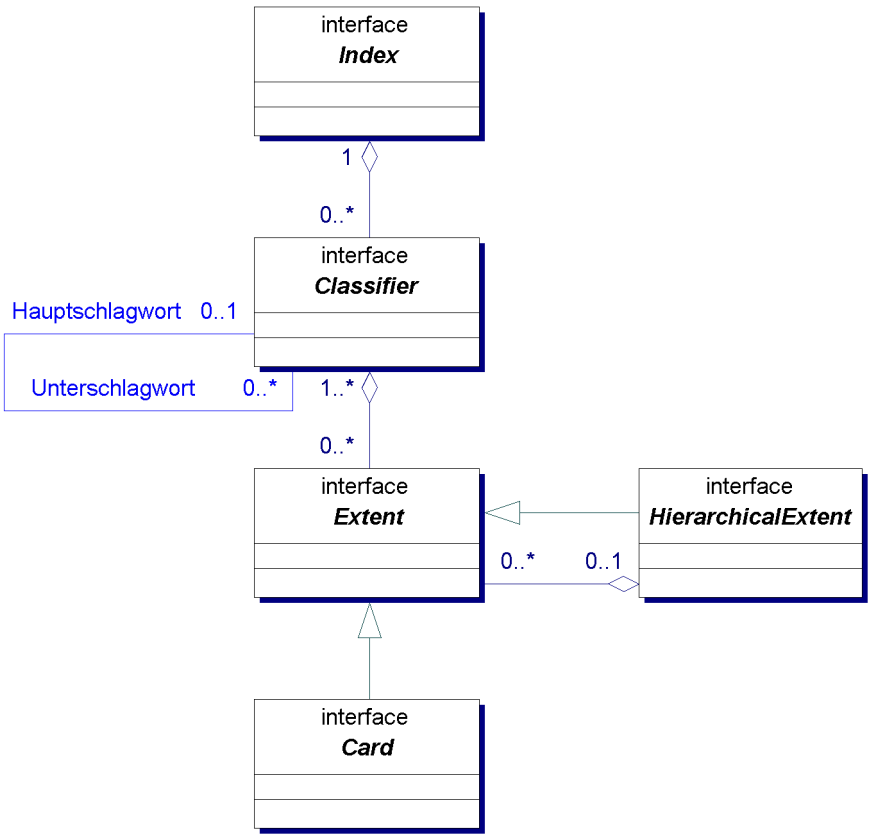

Abbildung 2.1: Ausschnitt des WEL Interface als schematisches Klassendiagramm. Quelle: Fornfeist (2003, S. 41)

2.1.2 TeFIS - Technologie- und Forschungsinformationssystem

Das Technolgie- und Forschungsinformationssystem (TeFIS)[6] wurde im Rahmen eines dreijährigen Projektes am Arbeitsbereich Softwaresysteme (STS) der Technischen Universität Hamburg-Harburg (TUHH) entwickelt, um den bis dato manuell ablaufenden *Redaktionsprozess*, zur Erstellung des regelmäßig erscheinenden Forschungsberichtes der TUHH, durch Informationtechnologie (*IT*) geeignet zu unterstützen und teilweise zu automatisieren.

Die Hochschulen sind verpflichtet einen solchen Forschungsbericht regelmäßig zu veröffentlichen und darin einen Einblick in die Tätigkeiten aller, zur Hochschule gehörenden, Arbeits- und Funktionsbereiche zu geben. Das geschieht durch Informationen über *Ressourcen* wie Mitarbeiter, Projekte, Mittel sowie Publikationen. In diesem Zusammenhang ist die Hochschule natürlich an einer größtmöglichen Qualität der Informationen gelegen, um die Hochschule angemessen zu repräsentieren. Ein *normativer* Redaktionsprozess kann zu diesem Ziel beitragen. Desweiteren lassen sich die erfassten Daten für andere Zwecke, z.B. zur Erstellung des *Internet-Auftritts* der TUHH, und Dienste, wie erweiterte Suchfunktionalitäten und Zugriff

[6] Sehring (2004), S. 177ff.

auf historische Daten verwenden (*multichannel publishing*).

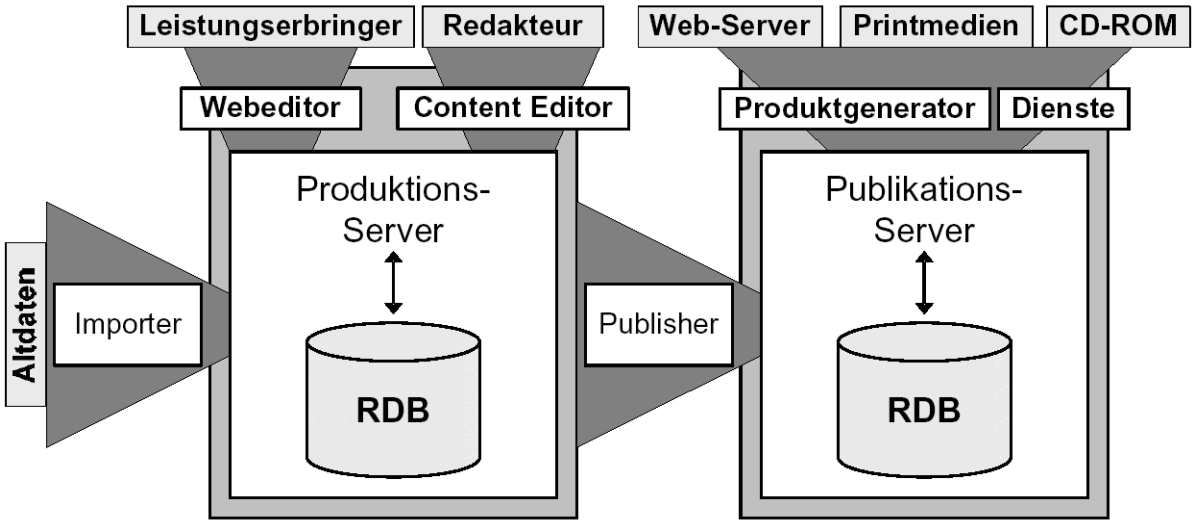

Abbildung 2.2: TeFIS Architektur.
Quelle: Sehring (2004, S. 178)

Die Architektur des Systems besteht aus zwei Servern (*Produktions- und Publikationsserver*), die jeweils eine Stufe (*stage*) des zweistufigen Redaktionsprozesses abbilden. Die zweite Stufe des Prozesses wird durch Kopieren der jeweiligen, durch einen *Redakteur* als freigegeben markierten Daten auf den Publikationsserver erreicht. Während des Kopiervorgangs werden Maßnahmen zur *Qualitätssicherung* durchgeführt. So wird beispielsweise die *Verweisstruktur* transitiv auf fehlende Ressourcen überprüft.

Im Verlauf des Projektes kristallisierte sich ein *Asset-basierter* Ansatz zur Datenspeicherung heraus, wobei *Assets* in diesem Zusammenhang, ganz generell, als werthaltige Objekte angesehen werden können. Der Forschungsbetrieb der Hochschule wird durch *Assets*, wie Arbeits- und Funktionsbereiche, Mitarbeiter, Projekte und Publikationen repräsentiert, die einer kontinuierlichen Veränderung (*Asset-Prozesse*) unterworfen sind. Dabei ist nicht allein der Ist-Zustand eines *Assets* zu gegebener Zeit von Bedeutung, sondern gleichbedeutend ist seine Entwicklunggeschichte. Im Zusammenhang mit dem *normativen* Ansatz des Redaktionsprozesses stellen *präskriptive Assets* einen weiteren Schritt zur Qualitätsicherung dar. Ein präskriptives *Asset* besitzt *offene Bindungen*, die durch den Leistungserbringer (in diesem Fall die Arbeits- und Funktionsbereiche), unter Einhaltung der, durch das *Asset* vorgegebenen, *Regeln* und Strukturen, gefüllt werden sollen. Ein präskriptives *Asset* kann somit als normatives Element sowie als Hilfestellung angesehen werden.

Einen detailierteren Überblick über das TeFIS-Projekt vermittelt Sehring[7]. Im Kontext dieser Arbeit, sind insbesondere die Konzepte *Assets*, präskriptive *Assets* und *Asset-Prozesse* von Bedeutung. Weiterhin wird in dieser Arbeit der Grundgedanke eines *nicht-monolithischen* Systems, wie es durch Produktions- und Publikationsserver seine Umsetzung findet, fortent-

[7]Sehring (2004), S. 177ff.

wickelt.

2.1.3 infoAsset Broker

Im Rahmen des Projektes SAP FÜR HAMBURG, der Einführung von SAP R/3 für das *Haushaltsmanagement* der HANSESTADT HAMBURG im Jahre 1997, wurde eine große Zahl von *Dokumenten* zur Unterstützung der *Migration* zusammengetragen. Für die Verwaltung und den Zugriff auf diese *Ressourcen*, wurde durch den ARBEITSBEREICH SOFTWARESYSTEME (STS) der TECHNISCHEN UNIVERSITÄT HAMBURG-HARBURG (TUHH), in einem separatem Projekt, der FHH INFOBROKER[8] zur Verfügung gestellt. Dieser basiert auf dem INFOASSET BROKER, einer *Standard-Software* der Firma INFOASSET, die in Kooperation mit dem Arbeitsbereich STS entwickelt wird[9].

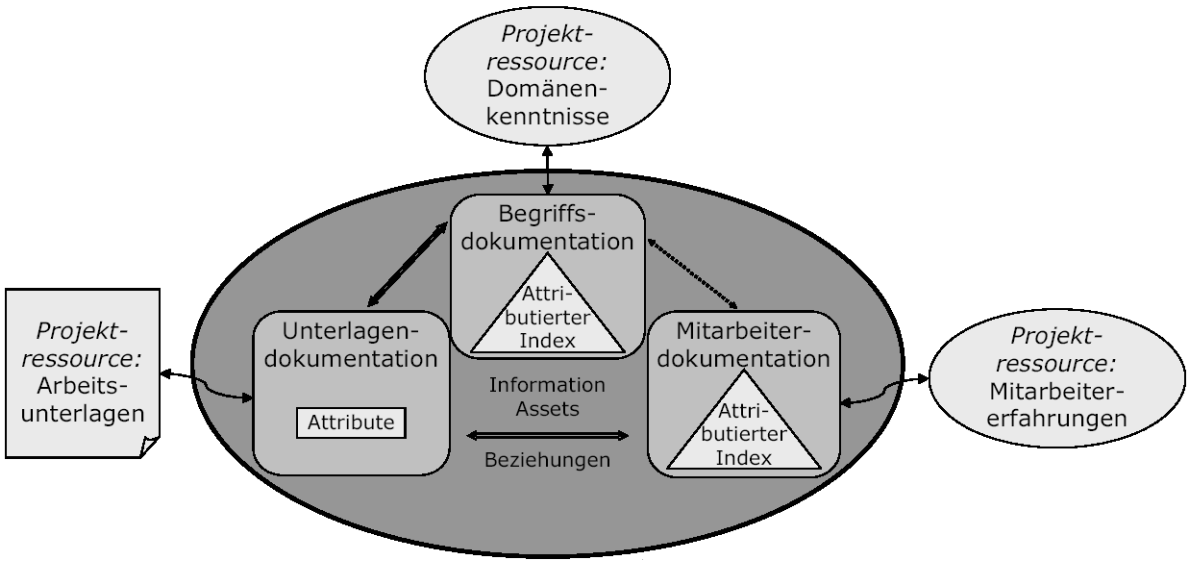

Abbildung 2.3: Ressourcen in Projekt SAP FÜR HAMBURG.
Quelle: Raulf et al. (2001)

Der INFOASSET BROKER (kurz BROKER) ist ein *Dokumenten-Management-System*, das im Gegensatz zu anderen Systemen dieser Klasse in der Lage ist, auch *strukturierte Daten* in Form von sogenannten *Assets* zu speichern und zu verwalten. Ein *Asset-Typ* definiert einen festen Satz von *Attributen*, deren Attribut-Werte ein *Asset* beschreiben. Zwischen *Asset-Typen* im BROKER können beliebige *Beziehungen* definiert werden. Eine systematische *Evolution* des, durch die *Asset-Typen* und deren Beziehungen untereinander gebildeten, *Asset-Modells* wird vom BROKER nicht unterstützt. Die ein *Asset* beschreibenden Attribut-Werte enthalten Daten über den Ist-Zustand eines *Assets*. Prozessdaten zur Entwicklung der *Assets* werden nicht systematisch gespeichert und verwaltet. Die im BROKER abgelegten Ressourcen in Form von *Dokumenten* werden ebenfalls als *Assets* betrachtet und können auf diese Weise zu anderen

[8] Raulf et al. (2001).
[9] Wegner (2002).

Asset-Typen, beispielsweise `Begriff`, `Person`, `Gruppe` und `Projekt`, in Beziehung gesetzt werden.

Die *Klassifikations-Beziehung* vom *Asset-Typ* `Dokument` zum Typ `Begriff`, sowie die *Definitions-Beziehung* in umgekehrter Richtung werden im *Asset-Modell* des BROKER zusammengefasst, vordefiniert und systematisch unterstützt. Zu Bedenken ist hier allerdings, dass beide Beziehungen nicht notwendigerweise *invers* sind[10]. Ein weitere vordefinierte Beziehung im *Asset-Modell* des BROKERS ist die zwischen *Personen* und *Begriffen*, die eine Person in die *Rolle* eines *Wissensträgers* erhebt. Weiterhin werden Personen über die Zugehörigkeit zu *Gruppen* Rollen und somit *Rechte* für den Zugriff auf *Assets* im BROKER verliehen. Dies geschieht ebenfalls über *ausgezeichnete* Elemente innerhalb des *Asset-Modells*.

Über die genannten Fähigkeiten von BROKER und *Asset-Modell* hinaus, bietet das System die Möglichkeit zur *Volltextsuche*, basierend auf einer *Volltextindizierung*, innerhalb der verwalteten Dokumente. Die Volltextindizierung ist auch die Grundlage für die *Autoklassifizierung* beim Einstellen von Dokumenten in den BROKER.

Inhaltliche Personalisierung in eingeschränkter Form erfolgt im BROKER über sogenannte *Sammelmappen*, in die Verweise auf Inhalte abgelegt, und die auf andere Arbeitsplätze übertragen werden können. Weiterhin können Ressourcen im BROKER mit *Annotationen* versehen werden, um persönliche Qualtitätsmerkmale bei der *Suche* und Verwaltung von Ressourcen systematisch einzubeziehen.

Für vertiefende Informationen zum Projekt SAP FÜR HAMBURG sei auf Raulf et al. (2001) verwiesen. Weitere Details zum BROKER können Wegner (2002) und infoAsset (2004) entnommen werden. Die in diesem Abschnitt erwähnten Konzepte der Klassifizierung von Ressourcen über Begriffe, der Inhaltsverwaltung über Assets und der inhaltlichen Personalisierung stellen bereits erste Ansätze für die in dieser Arbeit behandelte offene und dynamische Inhaltsverwaltung[11] dar. Insbesondere die explizite Formulierung von Domänenwissen, wie in Abbildung 2.3 verdeutlicht, entspricht dem hier behandelten Modell der Inhaltsverwaltung, das sich ebenfalls an Domänen orientiert[12].

2.2 Ein offenes begriffsorientiertes Modell

In Folge der in Abschnitt 2.1 beschriebenen Fallstudien zum *begriffsorientierten Arbeiten* wurden am ARBEITSBEREICH SOFTWARESYSTEME (STS) der TECHNISCHEN UNIVERSITÄT HAMBURG-HARBURG (TUHH) Überlegungen zum Entwurf eines generischen *begriffsorientierten Inhaltsveraltungssystems* angestellt[13]. Als Basis für einen derartigen Entwurf fehlte allerdings am Arbeitsbereich zunächst das Verständnis der grundsätzliche Natur des *menschlichen Erkenntnisprozesses*. Denn ohne ein adequates *Modell* dieses Vorganges und seiner Phänomene ist kein generisches System denkbar, das den Erkenntnisprozess angemessen unterstützen und dessen *Inhalte* verwalten könnte. Ein mögliches Modell, das diesen Anforderungen gerecht wird, wird in Schmidt et al. (2003) und Sehring (2004) beschrieben. Zu der dort verwendeten Begrifflichkeit des *konzeptorientierten Modells* und der in dieser Arbeit

[10] Vgl. Abschnitt 2.1.1, WEL - WARBURG ELECTRONIC LIBRARY

[11] Vgl. Abschnitt 2.2, Ein offenes begriffsorientiertes Modell; Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell

[12] Vgl. Abschnitt 2.2.1, Domänen

[13] Schmidt et al. (2001).

angewandten Terminologie *begriffsorientiertes Modell* ist anzumerken, dass es sich bei *Begriffen* um benannte *Konzepte* einer *Anwendungsdomäne* handelt[14]. Insofern bezeichnen beide Namen dasselbe Modell der Inhaltsverwaltung.

Die Eigenschaft der *Offenheit* bezeichnet im Zusammenhang mit einem Modell zur begriffsorientierten Inhaltsverwaltung die Fähigkeit, Instanzen des Modells um neue Konzepte und damit auch Begriffe erweitern oder bestehende Konzepte modifizieren zu können.[15]. Im Detail entstehen neue Konzepte durch *Differentierung* und *Spezialisierung* aus vorhandenen Konzepten oder durch *Abstraktion* aus einer Teilmenge von vorhandenen Inhalten.

Neben der hier behandelten Offenheit ist die *Dynamik*, die in Abschnitt 2.3 eingeführt wird, nach Cassirer die zweite Grundlage für den unbegrenzten Erkenntnisprozess des Menschen, der aus der fortlaufenden Differentierung, *Integration*, Spezialisierung und *Generalisierung* von Konzepten sowie der *Definition* von Konzepten durch Inhalte und der *Klassifikation* von Inhalten durch Konzepte besteht[16]. Offenheit ist somit integraler Bestandteil eines Modells zur begriffsorientierten Verwaltung von Inhalten, die im Zuge eines menschlichen Erkenntnisprozesses zu Tage gefördert wurden.

2.2.1 Domänen

Ein *Domäne* ist ein abgegrenztes (Fach-)Gebiet der menschlichen Erkenntnis in welchem eine (Fach-)Sprache und *Begriffe* dieser Sprache verwendet werden. Im Folgenden werden Domänen in *Objekt- und Subjektdomänen* unterteilt.

Objektdomänen

Objektdomänen sind Domänen, deren *Konzepte* und *Inhalte* sowie deren *Beziehungen* zueinander keiner Deutung unterliegen, sondern objektiv gegeben sind. Es handelt sich also beispielweise um mathematisch-logische, einheitlich definierte oder formal definierte Domänen.

Im Rahmen des in dieser Arbeit entwickelten Prototypen, wird das zugrundeliegende *begriffsorientierte Modell* zusätzlich zur Verwaltung von Benutzerinhalten, vor allem zur Verwaltung von Systeminhalten genutzt. Diese Systeminhalte sind naturgemaß objektiver Art und können deshalb als Teil einer oder mehrerer ausgezeichneter Objektdomänen, sogenannter Basisdomänen, angesehen werden. Mögliche Basisdomänen in einem *begriffsorientierten System* sind die *Benutzer-, Gruppen-, Rechte-, Rollen- und Werkverwaltung*[17]. In einem *verteilten System*[18], kommen weitere Basisdomänen zur *Namens-, Verzeichnis- und Nachrichtenverwaltung* hinzu.

[14] Vgl. Abschnitt 2.2.3, Begriffe und Konzepte
[15] Sehring (2004), S. 9.
[16] Cassierer (2001).
[17] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[18] Vgl. Abschnitt 3.1, Implikationen der Offenheit und Dynamik für den Systementwurf

Subjektdomänen

Subjektdomänen sind Domänen, deren Konzepte, Inhalte und Beziehungen nicht objektiv gegeben, sondern der *subjektiven Sicht* des jeweiligen *Benutzers* unterworfen sind[19]. Im Umgang mit Subjektdomänen ergibt sich folglich die Notwendigkeit der systematischen Unterstützung einer umfassenden *inhaltlicher Personalisierung*, die über das in Fallstudien am Anfang des Kapitels beschriebene Maß hinausgeht[20].

2.2.2 Inhalte

Inhalte beschreiben *konkrete und abstrakte Entitäten* einer *Anwendungsdomäne*. Konkrete Entitäten finden in einem realen *Phänomen* ihre Entsprechung, während abstrakte Entitäten zunächst „nur“ Produkte des *menschlichen Erkenntnisprozesses* darstellen und später durch eine *extensionale*[21] oder *intensionale Definition* angereichert werden.

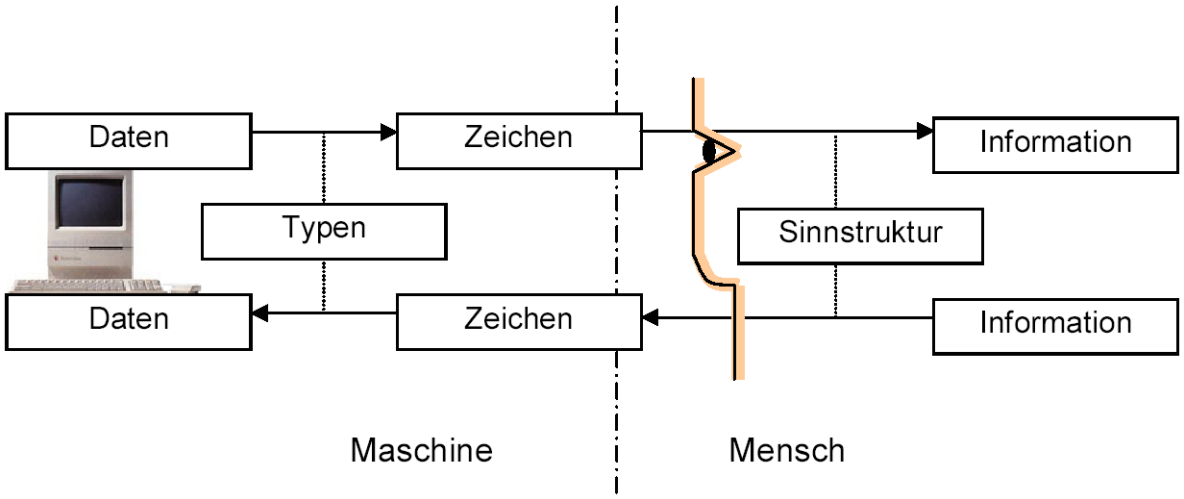

Abbildung 2.4: Mensch-Maschine-Kommunikation.
Quelle: Bachmann (2003, S. 9)

Alle *IT-Systeme* können naturgemäß nur *objektive Daten* und keine *subjektive Inhalte* verwalten. Die *Übertragung* von Daten mittels *Zeichen* und deren *Interpretation* als Inhalte erfolgt gemäß der *Sinnstruktur* des *Benutzers*. Der gleiche Inhalte kann an zwei verschiedene Benutzer durch völlig unterschiedliche Zeichen übertragen werden. Bisherige *Dokumenten- oder Inhaltsverwaltungssysteme* verwenden jedoch ein einheitliches und durch technische Gegebenheiten bestimmtes Repertoire an Zeichen zur Übermittlung von Inhalten[22].

Ein System zur *begriffsorientierten Inhaltsverwaltung* speichert im Gegensatz hierzu nicht nur Daten für die Inhaltsbeschreibung, sondern auch Teile der persönlichen Sinnstruktur eines Benutzers. Auf diese Weise wird das System in die Lage versetzt, der Sinnstruktur

[19] Vgl. Abschnitt 2.3.2, Organisationsspezifische Prozesse
[20] Vgl. Abschnitte 2.1.1, WEL - Warburg Electronic Library; 2.1.3, infoAsset Broker
[21] Vgl. Abschnitt 44, Klassifikation und Definition
[22] Vgl. Abbildung 2.4, Mensch-Maschine-Kommunikation

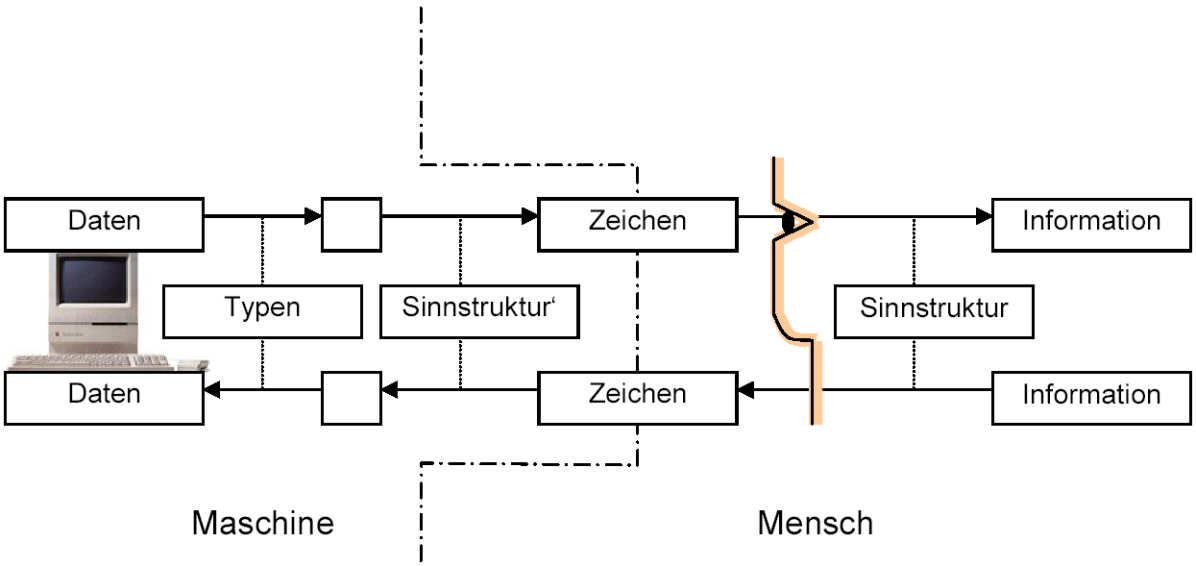

Abbildung 2.5: Mensch-Maschine-Kommunikation mit angereicherter Semantik.
Quelle: Bachmann (2003, S. 10)

des Benutzers *individuell* angepasste Zeichen zur Übertragung von Inhalten zu verwenden. Die Speicherung von Teilen der Sinnstruktur verschiebt die *Systemgrenze* in Richtung des Benutzers[23].

Zeichen können nach Peirce (1931) in zehn verschiedene Klassen von Zeichen, sogenannte *Zeichenrelationsklassen*, unterteilt werden[24]. Im Detail werden Zeichen anhand der drei Dimensionen *Zeichen-, Objekt- und Interpretantendimension* unterschieden. Auf jeder Dimension sind sogenannte *Erst-, Zweit- und Drittheiten* mit dimensionsanhängiger Benennung angeordnet. Bei Erstheiten handelt es sich um Zeichen, die *charakteristische Eigenschaften* einer *Entität* beschreiben. Zweitheiten bezeichnen *Beziehungen* zwischen Entitäten. Drittheiten stellen *regelhafte Beziehungen* zwischen Entitäten dar. Insgesamt sind nach Peirce nur zehn der 27 möglichen Kombinationen von Zeichenrelationsklassen sinnvoll, weil zwischen den Klassen eine hierarchische Abhängigkeitsbeziehung besteht. Für weitere Details sei auf Hoffmann (2004) verwiesen. Im Rahmen dieser Arbeit werden die Zeichen hauptsächlich anhand der Interpretantendimension unterschieden.

Allgemein werden Inhalte, bzw. die durch die Inhalte repräsentierten Entitäten, in dem hier vorgestellten Modell der begriffsorientierten Inhaltsverwaltung nicht nur durch ihren Ist-Zustand beschrieben, sondern im Kontext ihres *Entwicklungsprozesses* betrachtet. Diese Sichtweise ist durch den Begriff des *Werkes* charakterisiert. Ein Werk kann gewollt doppeldeutig, als Kurzform oder Verweis auf das *Werkprodukt*, also ein *multimediales Artefakt*, sowie als Prozess der Entstehung des Werkproduktes gesehen werden. Das begriffsorientierte Modell trifft keine Entscheidung über die *Signifikanz* beider *Sichten*, sondern gibt diese in die Hände des Benutzers.

Die genannte Doppeldeutigkeit bzw. die Hervorhebung der *Prozessicht* im Rahmen der be-

[23] Vgl. Abbildung 2.5, Mensch-Maschine-Kommunikation mit angreicherter Semantik
[24] Vgl. Abbildung 2.6, Zeichenrelationsklassen nach Peirce

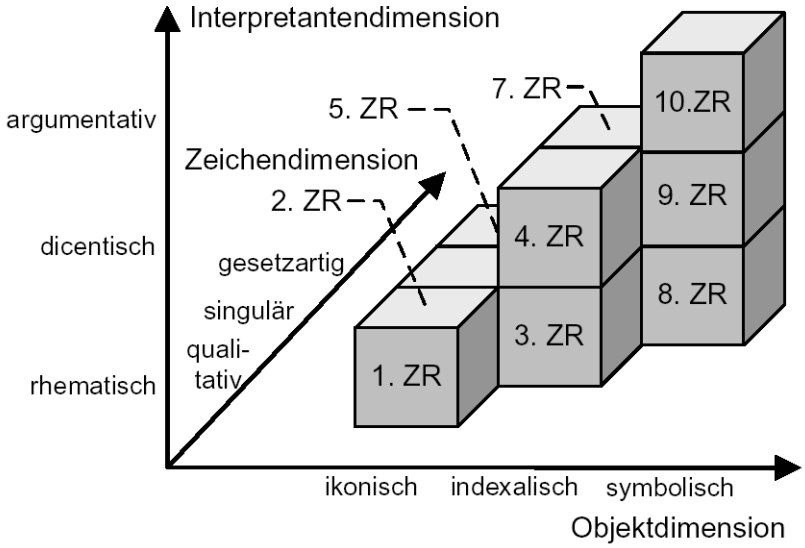

Abbildung 2.6: Zeichenrelationsklassen nach Peirce.
Quelle: Hoffmann (2004)

griffsorientierten Inhaltsverwaltung wurde bereits im WEL Projekt[25], identifiziert und steht in der Tradition der ständigen Reorganisation der Warburg-Bibliothek und des Bildindex zur politischen Ikonographie[26], welche auf der Philosophie Cassirers, des fortschreitenden menschlichen Erkenntnisprozesses[27] basiert.

2.2.3 Begriffe und Konzepte

Begriffe sind benannte *Konzepte* einer *Anwendungsdomäne*. Konzepte wiederum stellen im Zuge des *menschlichen Erkenntnisprozesses* und im *Fokus* einer bestimmten Domäne gewonnene *Abstraktionen* dar. Zwischen Begriffen bzw. Konzepten und den zuvor beschriebenen *Inhalten* bestehen die engen *Beziehungen* der *Klassifikation* und der *Definition*[28]. Letzteres hat zum Schritt der *Komposition* von Konzepten und Inhalten zu sogenannten *Assets* im Modell der *begriffsorientierten Inhaltsverwaltung* geführt, der im nachfolgenden Abschnitt ausführlicher begründet wird. In diesem Zusammenhang definiert ein Konzept oder auch *Asset-Typ* ein *Asset intensional* durch *Aggregation charakteristischer Eigenschaften* (*Erstheiten)*, Beziehungen zu anderen *Asset-Typen* (*Zweitheiten*) und *Regeln* für Erst- und Zweitheiten (*Drittheiten*).

[25]Vgl. Abschnitt 2.1.1, WEL - Warburg Electronic Library
[26]Niedereé et al. (1996); Bruhn (1999); Fornfeist (2003).
[27]Cassierer (2001).
[28]Vgl. Abschnitt 44, Semantische Beziehungen - Klassifikation und Definition

2.2.4 Assets als Einheit von Konzept und Inhalt

In Erweiterung zu Fallstudien am Beginn des Kapitels[29] wird das Konzept *Asset* im Rahmen der *begriffsorientierten Inhaltsverwaltung* redefiniert. Ein *Asset* wird nicht nur als werthaltiges Objekt oder *Inhalt* verstanden, sondern bildet eine *Einheit aus Konzept und Inhalt*. Dieser Definition eines *Assets* liegen *erkenntnistheoretische* Einsichten[30] zugrunde, die die Existenz von Konzepten ohne *definierende* Inhalte sowie die Existenz von Inhalten ohne *klassifizierende* Konzepte verneinen.

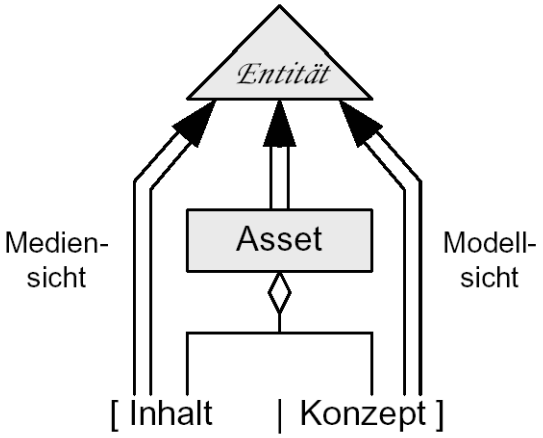

Abbildung 2.7: *Asset* als Einheit von Konzept und Inhalt.
Quelle: Sehring (2004, S. 8)

Der *menschliche Erkenntnisprozess* ist so beschaffen, dass Inhalte ohne ein Konzept nicht als Inhalte erkennbar sind. Andersherum sind Konzepte ohne sie *materialisierende* Inhalte nicht *abstrahierbar*. Diese Sichtweise führt allerdings zum bekannten *Henne-Ei-Dilemma*. Die Frage lautet, auf welche Weise konnten dann überhaupt Konzepte und Inhalte entstehen? Die mögliche Anwort einer Ausstattung des Menschen mit einem *initialen Begriffssystem* quasi per Geburt, verneint Cassirer. Stattdessen führt er sogenannte *hypothetische Konzepte* ein, die sich „alsbald“ bewähren, also Inhalte klassifizieren müssen. Zum zweiten können hier die Konzepte der *Objektdomänen*[31] genannt werden, deren Definition fest gegeben ist.

Ursprünglich verwendet Cassirer für die Einheit aus Konzepten und Inhalten den Begriff *Symbol*. Allerdings ist dieser Begriff bereits mit der *Informatik* und der *Semiotik* nach Peirce (1931) mehrfach belegt. Infolgedessen wird die Einheit aus Konzept und Inhalt im Modell der begriffsorientierten Inhaltsverwaltung nach Sehring[32] als *Asset*, in genau dem in diesem Abschnitt beschriebenen Sinne, bezeichnet. Eine formale Sprache zur Beschreibung von *Assets* wird ebenfalls von Sehring[33] definiert.

Die im Abschnitt 2.2.2 motivierte Verschiebung der *Systemgrenzen* der *Mensch-Maschine-Kommunikation* in Richtung der *Sinnstruktur* des *Benutzers*, findet mit der Einführung des

[29] Vgl. Abschnitt 2.1.2, TeFIS; 2.1.3, infoAsset Broker
[30] Cassierer (2001); Cassierer (2002b); Cassierer (2002a).
[31] Vgl. Abschnitt 2.2.1, Domänen - Objektdomänen
[32] Sehring (2004), S. 8f.
[33] Sehring (2004), S. 39ff.

Asset-Konzeptes im Modell der begriffsorientierten Inhaltsverwaltung seine konsequente Umsetzung. Ein *Asset-Modell* spiegelt Teile der individuellen Sinnstruktur eines Benutzers wieder. Die Speicherung des *Asset-Modells*, also der konzeptionellen Teile aller *Assets*, ermöglicht eine *semantisch angereicherte* Mensch-Maschine-Kommunikation über die im inhaltlichen Teil aller *Assets* beschriebenen *Entitäten* mittels *individueller* Begriffe des Benutzers.

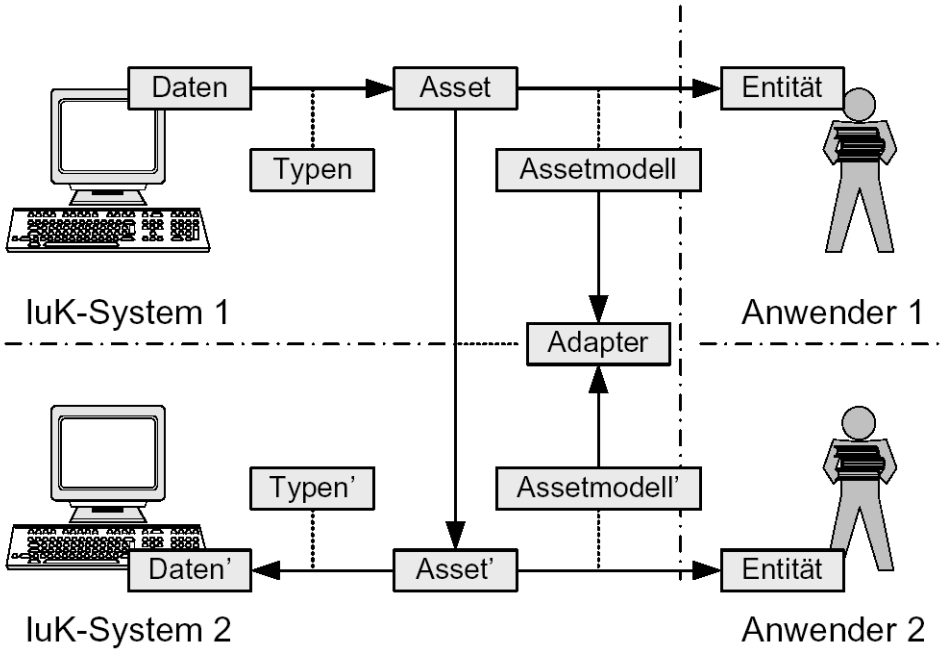

Abbildung 2.8: Mensch-Mensch-Kommunikation mittels *Asset-Modelladapter*.
Quelle: Sehring (2004, S. 18)

Die Kommunikation zwischen Benutzern eines *begriffsorientierten Systems* erfolgt über die *Adaption* der jeweiligen *Asset-Modelle* der Benutzer. Es findet also eine Umwandlung von Daten aus einem Schema in ein anderes Schema statt. Vom `Benutzer 1` an `Benutzer 2` übermittelte Zeichen, werden durch den *Modelladapter* modifiziert und können von `Benutzer 2` in seiner individuellen Sinnstruktur interpretiert werden[34]. Die gleiche Vorgehensweise kann angewandt werden, wenn sich die Sinnstruktur eines Benutzers im Zuge seines fortschreitenden Erkenntnisprozesses gewandelt hat oder sich sein *Anwendungsfokus* verschoben hat. Bestehende Inhalte können mittels Modelladaption der aktuellen Sinnstruktur angepasst werden. Insgesamt wird durch das beschriebene Modell die *unifizierende* und häufig technisch bedingte Begrifflichkeit der Maschine zur Repräsentation von Inhalten vermieden.

2.2.5 Semantische Beziehungen

Im *konzeptionellen* Teil eines *Assets* werden neben *charakteristischen Eigenschaften* (*Erstheiten*) auch *Beziehungen* zwischen *Assets* (*Zweitheiten*) und *Regeln* für Erst- und Zweitheiten (*Drittheiten*) definiert[35]. Neben diesen explizit beschriebenen Beziehungen, ergeben sich auch

[34] Vgl. Abbildung 2.8, Mensch-Mensch-Kommunikation mittels *Asset-Modelladapter*
[35] Vgl. Abschnitt 2.2.2, Inhalte

Beziehungen implizit aus dem *begriffsorientierten Modell* und der Systematik im Umgang mit *Assets*. Die möglichen Typen von Beziehungen werden nachfolgend kurz beschrieben.

Klassifikation und Definition

Unter Berücksichtigung der Definition von *Assets* als *Einheit von Konzept und Inhalt*[36], handelt es sich bei der *Klassifikation* u.a. um die Einordnung von *Inhalten* unter ein bestehendes *Konzept*, also einen *Asset-Typ*. Diese Variante der Klassifikation ist statisch insofern, als das das Konzept einer gegebenen *Asset-Instanz* nicht verändert werden kann. Die zweite Variante der Klassfikation ist die, als Klassifikationsbeziehung gedeutete und evtl. durch Drittheiten semantisch angereicherte, Zweiheit in der *intensionale Asset-Definition* (s.u.). Diese Ausprägung der Klassifikation ist dynamisch, kann also für eine gegebene *Asset-Instanz* ohne *Modellevolution*[37] verändert werden.

Die Definitionsbeziehung hat, ebenso wie die Klassifikation, die beiden Varianten der intensionalen und *extensionalen* Definition von *Assets*. Erstere definiert, wie bereits mehrfach erwähnt, ein *Asset* bzw. seien Typ durch Angabe aller charakteristischen Eigenschaften (Erstheiten), Beziehungen (Zweiheiten) und Regeln (Drittheiten). Diese Form der Definition ist statisch (s.o.). Die zweite Variante fasst eine Menge von *Assets* als extensionale Definition (*Extension*) des zu definierenden *Assets* auf. Diese Art der Definition kommt dem menschlichen Erkenntnisprozess nahe, der in der Regel Konzepte aus einer Menge von Beispielen abstrahiert, die nicht notwendigerweise minimal ist.

Ein Abgrenzung zur nachfolgenden Generalisierungs- und Spezialisierungsbeziehung stellt eine Feststellung dar, die bereits in der Fallstudie zur WEL getroffen wurde und besagt, dass Klassifikation und Definition nicht *invers* sind. Nur bei einer Teilmenge der Extension eines *Assets* handelt es sich explizit um sogenannte *Stützstellen der Domänensemantik*.

Generalisierung und Spezialisierung

Generalisierung und *Spezialisierung* stellen Beziehungen zwischen allgemeineren und spezielleren Konzepten und den sie beschreibenden *Assets* her und sind invers zueinander. Beide Beziehungen können explizit, durch ein entsprechendes *Schlüsselwort* im konzeptionellen Teil einer intensionalen *Asset-Definition*, vom Benutzer definiert werden. Es entsteht auf diese Weise eine statische *Asset-Typhierarchie*. Desweiteren können Beziehungen (Zweitheiten), in der intensionalen Definition, als Generalisierungs- und Spezialisierungsbeziehungen gedeutet und durch Drittheiten mit spezialisierter *Semantik* versehen werden. Auf diese Weise können dynamische *Asset-Typhierachien* aufgebaut werden. Als letzte mögliche Ausprägung, können extensional definierte Assets als Spezialisierung von anderen Assets gesehen werden, wenn die Extension der Ersteren eine Obermenge der Extension der Zweiteren darstellt. Auch diese Variante der Generalisierungs- und Spezialisierungsbeziehung ist dynamisch und kann ohne Modellevolution verändert werden.

[36] Vgl. Abschnitt 2.2.4, *Assets* als Einheit von Konzept und Inhalt
[37] Vgl. Anschnitt 2.3, Ein dynamisches begriffsorientiertes Modell

Aggregation

Eine *Aggregation* stellt ein *Asset* als *Kompositum* aus anderen *Assets* dar. Konkret definiert wird die Aggregation im konzeptionellen Teil eines Assets, wo die Beziehungen (Zweitheiten) des *Asset-Typs* des Kompositum zu anderen *Asset-Typen* beschrieben wird. Durch die freie Definierbarkeit einer Aggregation mit den Ausdrucksmitteln der Zweit- und Drittheiten kann eine individuelle und anwendungsorientierte Semantik der Aggregationsbeziehung geschaffen werden. Die konkrete Ausprägung wird also explizit durch einen *Benutzer* bestimmt.

Materialisierung

Eine Beziehung zwischen *abstrakten* und *konkreten Assets*[38] wird als *Materialisierung* bezeichnet. Die Materialisierung ist eine implizit hergestellte Beziehung zwischen *Assets* und beruht auf der Typgleichheit der *Asset-Instanzen* in der statisch oder dynamisch definierten *Asset-Typhierarchie*.

2.2.6 Benutzer und Gruppen

Im Abschnitt 2.2.4 wurde dargelegt, dass die Verwaltung von Teilen der *Sinnstruktur* eines *Benutzers* in Form eines *Asset-Modells* eine *semantisch angereicherte Mensch-Maschine-Kommunikation* über *Inhalte* erlaubt. Hierzu müssen natürlich auch die *Benutzer* eines *begriffsorientierten Systems* als beschreibbare *Entitäten* und zu verwaltende Inhalte erfasst werden, um ihnen unter Zuweisung ihres individuellen *Asset-Modells* Zugriff auf persönliche oder von höheren *Organisationseinheiten* abgeleitete *Assets* zu gewähren.

Die *Abstraktion* von einzelnen Benutzern führt zum Konzept der *Gruppe*, die für ihre Mitglieder *Zugriffsrechte* auf *Assets* und *Sichten* (*Asset-Modell* der Gruppe) definiert. Zusätzlich lassen sich Gruppen mit *Rollen* assoziieren, die ein Benutzer einnehmen kann. Auch Gruppen können wieder zu übergeordneten Einheiten, im Sinne der *Generalisierung*[39], zusammengefasst werden.

Gruppen und Benutzer stellen, wie die bisher beschriebenen Entitäten, *Konzept-Inhalts-Paare* dar, die durch *Assets* adäquat abgebildet werden können[40]. Deshalb liegt es nahe, das vorhandene *Modell der begriffsorientierten Inhaltsverwaltung* auch zur *Benutzerverwaltung* einzusetzen.

Die *Zugehörigkeit* zu einer anderen Gruppe kann durch die Gruppe oder den Benutzer selbst oder durch eine *Systemauthorität* festgelegt werden. Die Beschränkung auf eine *hierarchisch azyklischen Beziehungsstruktur* zwischen Gruppen und Benutzern führt nach Bachmann[41] zu einer beträchtlichen Vereinfachung der Prozesse der *Personalisierung* und *Publikation* in einem begriffsorientierten System.

[38] Vgl. Abschnitt 2.2.2, Inhalte
[39] Vgl. Abschnitt 46, Semantische Beziehungen - Generalsierung und Spezialisierung
[40] Sehring (2004), S. 22f.
[41] Bachmann (2003), S. 46.

2.3 Ein dynamisches begriffsorientiertes Modell

Die *Dynamik*, genauer die dynamische Anwendung der *Offenheit*[42], bezeichnet im Zusammenhang mit dem hier vorgestellten *Modell zur begriffsorientierten Inhaltsverwaltung* die Eigenschaft, eine Modellinstanz in allen Phasen ihres *Lebenszyklus* um neue *Konzepte* ergänzen und vorhandene Konzepte modifizieren zu können[43]. Die im System abgelegten *individuellen Asset-Modelle* können auf diese Weise mit dem *fortschreitenden Erkenntnisprozess* oder der Verschiebung des jeweiligen *Anwendungsfokus* der *Benutzer* schritthalten.

2.3.1 Asset-Prozesse

Der im Zusammenhang mit *Inhalten*[44] hervorgehobene *Werk- oder Prozesscharakter* in der Philosophie Cassirers[45] findet seinen Niederschlag natürlicherweise auch im Umgang mit *Assets*. *Asset-Prozesse* bestehen aus allen Vorgängen des Zugriffs und der Veränderung des *konzeptionellen* oder *inhaltlichen* Teils von *Assets*. So wie *Asset-Modelle* menschlichen *Sinnstrukturen* abbilden, so bilden Prozesse auf *Assets* menschliche *Erkenntnisprozesse* ab, die auf den Sinnstrukturen operieren. Diese *Asset-Prozesse* können, in rekursiver Anwendung des *Modells zur begriffsorientierten Inhaltsverwaltung*, ebenfalls wieder als zu beschreibende *Entitäten* und damit als *Assets* angesehen, die im Folgenden als *Werke* bezeichnet werden.

Als Vorgriff auf den Abschnitt 3.2.5 soll hier ein kurzer Abriss der Modellierung des *Asset-Typs* `Werk` gegeben werden. Eine graphische Darstellung kann Abbildung 3.8 entnommen werden. Im *konzeptionellen* Teil von `Werken` werden neben einem *eindeutigen* Charakteristikum `id`, im wesentlichen Verweise auf über- und untergeordneten Werke (*Teilwerke*) und zugegriffenen *Assets* verwaltet. Darüberhinaus können Werke als *Prototypen* für andere Werke fungieren. Zu guter letzt ist ein Verweis auf den *Werkausführenden* Bestandteil des konzeptionelle Teils des Werk-Typs.

Im *inhaltlichen* Teil enthalten Werke Fragmente von *Werkbeschreibungen*, die in einer *Werkbeschreibungssprache* formuliert sind, die in der vorliegenden Arbeit entwickelt wird[46]. Auf die Besonderheiten dieser Sprache und des zugehörigen Modells zur *Werkausführung*[47] sowie die Abgrenzung gegenüber dem *Prozessmodell* im *Workflow Management* wird nachfolgend eingegangen.

Abbildung 2.9 stellt das Prozessmodell im *Workflow Management*, als typisches Beispiel einer Klasse von Systemen zur Unterstützung kooperativer Prozesse, dar. Darin wird insbesondere zwischen *Prozessdefinition oder -vorschrift* und *Prozessergebnis* unterschieden. Derartige Prozesse sind also auf eine konkretes Endprodukt oder das *Aggregat* aller bis zum Ende der *Prozessausführung* vorgenommenen *Aktivitäten* ausgerichtet[48]. In dem durch Cassirer motivierten Modell der begriffsorientierten Inhalsverwaltung wird hingegen von einem ständig fortschreitenden Erkenntnisprozess ausgegangen, der kein definiertes Ende besitzt

[42] Vgl. Abschnitt 2.2, Ein offenes begriffsorientiertes Modell
[43] Sehring (2004), S. 9.
[44] Vgl. Abschnitt 2.2.2, Inhalte
[45] Cassierer (2001).
[46] Vgl. Abschnitt 3.3, Asset-Prozesse als Entitäten; Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse
[47] Vgl. Abschnitt 3.3.3, Prozessmodell
[48] Vgl. Abschnitt 2.5.2, Verwandte Themen und Gebiete - *Workflows*

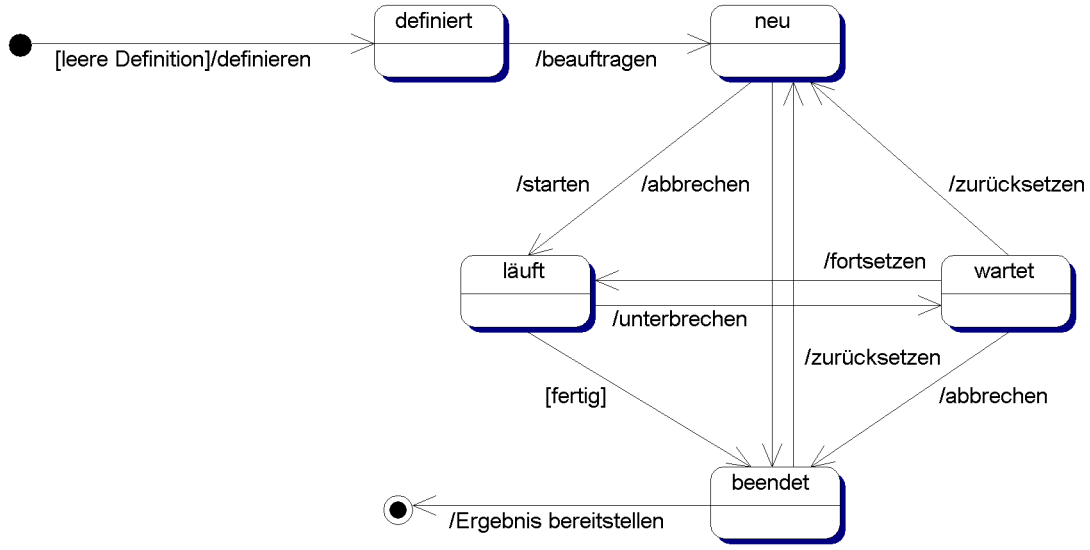

Abbildung 2.9: Prozessmodell im *Workflow Management*.

und dessen Zweck nicht in einem konkreten Endprodukt besteht. In der Folge muss allen *Zwischenprodukten* eines CASSIRER'schen Prozesses, also eines *Werkes*, ein potentieller Wert für die *Benutzer* zugemessen werden. Kein Zwischenschritt kann von der Betrachtung und damit vom Zugriff ausgeschlossen werden. Folglich taugt das herkömmliche *Prozessmodell* nicht für eine Beschreibung von Werken. Es muss ein Modell gefunden werden, das inkrementellen *Prozess- oder Werkfortschritt* angemessen beschreiben kann. Letztlich entfällt die eindeutige Abbgrenzung zwischen Prozessdefinition und -ergebnis zugunsten einer zunächst vollständig *präskriptiven* und dann zunehmend *deskriptiven* Beschreibung von Werken.

Eben dies wird in der entwickelten Werkbeschreibungssprache durch die Zahl der *offenen Bindungen* in einer Werkbeschreibung ausgedrückt. Eine derartige Werkbeschreibung kann nun *analytisch* (zu *Lern-* oder *Protokollzwecken*) oder *konstruktiv* (Planung, Vorschrift) genutzt werden. Letzteres erfolgt durch die *Ausführung* von Werkbeschreibungen, die in der Herstellung von Bindungen durch *menschliche* oder *maschinelle Akteure* besteht, bis keine offene Bindung mehr vorhanden ist oder kein Akteur mehr in der Lage ist einen Werkfortschritt zu erzielen. Dieser Zustand stellt jedoch nur eine Momentaufnahme dar, weil Werke, wie alle andere *Assets*, der *Offenheit* und *Dynamik* im *begriffsorientierten Modell* unterworfen sind. Demzufolge können Werke durch Hinzufügen neuer präskriptiver Elemente oder Veränderung deskriptiver Elemente (Bindungen lösen) „wiederbelebt“ werden. Eine Konkretisierung dieser Analysen und entsprechende Design-Entscheidungen werden in Abschnitt 3.3 geliefert.

2.3.2 Organisationsspezifische Prozesse

Organisationsspezifische Prozesse bezeichnen alle *Asset-Prozesse*, die der *Adaption* von *Inhalten*, gemäß der *Benutzer- und Gruppenstrukturen*, in einem *begriffsorientierten System* dienen. Insbesondere handelt es sich also um Prozesse, die aus eigenen *Assets* und *Assets* übergeordneter *Organisationseinheiten*, mittels verwalteter *Asset-Modelle*, *individuelle Sichten generieren*.

Weiterhin gehören Prozesse, die den umgekehrten Weg der Adaption unterstützen und die *Publikation* von *subjektiven Sichten* auf die Ebene höherer Organisationseinheiten ermöglichen, zu dieser Gruppe.

Personalisierung als subjektive Domänensicht

Unter dem Begriff *Personalisierung* werden nach Niedereé (2002) Maßnahmen zur bedarfsgerechten Anpassung eines Informationsangebotes, genauer die Auswahl, Strukturierung und Anreicherung von Inhalten sowie die Bereitstellung spezialisierter Dienste verstanden. Das *Modell der begriffsorientierten Inhaltsverwaltung* geht über diese Formen der Personalisierung hinaus. Es werden alle Vorgänge des *fortschreitenden menschlichen Erkenntnisprozesses*[49], also die *Differentierung*, *Integration*, *Generalisierung* und *Spezialisierung* von *Konzepten* sowie die *Definition* von Konzepten durch *Inhalte* und die *Klassifikation* von Inhalten durch Konzepte, unterstützt. Die *Benutzer* leiten ihre *subjektive Domänensicht*, mittels der aufgeführten geistigen Anstrengungen, *inkrementell* aus der Domänensicht einer *übergeordneten Gruppe* ab. Diese subjektive Domänensicht stellt für den Benutzer eines begriffsorientierten Systems den, seiner *Erkenntnis* entsprechenden, Zugang zu den verwalteten Inhalten dar.

Der Weg der Personalisierung führt also entlang der *hierarchisch azyklisch* organisierten Benutzer- und Gruppenstruktur[50] von der einheitlichen Gruppensicht zur subjektiven Benutzersicht. Der umgekehrte Weg, also die Publikation von subjektiven Inhalten zum Zwecke der *Kommunikation* und Kooperation mit Teilen einer übergeordneten Gruppe, wurde bisher nur kurz im Abschnitt 2.2.4 angerissen. Die *inkrementelle Subjektivierung*, also die schrittweise *Evolution* der *Asset-Modelle* in der begriffsorientierten Inhaltsverwaltung, erlaubt die *Generierung* von *Modelladaptoren* zum Zweck des Datenaustausches zwischen *verwandten Modellen*. Somit kann der zuvor beschriebene Weg der Personalisierung sowie umgekehrt eine *Publikation* von subjektiven Inhalten in die einheitliche Gruppensicht und optional eine weitere Adaption in die Benutzersicht bestimmter untergeordneter Benutzer erfolgen.

Abgrenzung zu anderen Personalisierungsformen

Die im begriffsorientierten Modell der Inhaltsverwaltung angewandte inhaltliche Personalisierung, also die Möglichkeit der fortlaufenden und individuellen Anpassung der Domänensemantik von Subjektdomänen durch die Benutzer[51], ist die generische Variante der in bisherigen *Modellen der Dokumenten- oder Inhaltsverwaltung* umgesetzten Personalisierung. In diesen Systemen beschränkt sich die Personalisierung auf die individuelle Anpassung der *Benutzeroberflächen*, die Definition von *Datenfiltern* oder die *Annotation von Dokumenten*, wie im Falle des infoAsset Brokers. So werden etwa im Fall einer *e-Commerce-Anwendung* Informationen über das *Klick- oder Einkaufsverhalten* eines Benutzers gesammelt und in einem sogenannten *Benutzerprofil* gespeichert, um dem Benutzer zu einem späteren Zeitpunkt potentiell ansprechende, also seinen Interessen entsprechende, Produkte präsentieren zu können. Die Produkte bzw. Referenzen auf die Produkte werden außerdem in einer Weise aufbereitet, die den anschließenden *Bestell- und Kaufvorgang* für den Benutzer besonders *ergonomisch* gestaltet.[52] Entsprechende Anforderungen stellen einen Spezialfall der inhaltlichen Personalisie-

[49] Cassierer (2001).
[50] Vgl. Abschnitt 2.2.6, Benutzer und Gruppen
[51] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[52] Jacobsen (2002).

rung dar, und können mit einem System der begriffsorientierten Inhaltsverwaltung umgesetzt werden.

2.3.3 Anwendungsspezifische Prozesse

Anwendungsspezifische Prozesse bezeichnen alle *Prozesse* auf *Assets*, die das Überschreiten von *Domänengrenzen*, also die *Interdomänenkooperation* unterstützen. Während der Ausführung eines derartigen Prozesses werden *Assets* verschiedener Domänen zugegriffen oder manipuliert. Die *Kooperation* wird durch das *Asset-Modell* oder die *präskriptive Prozessbeschreibung* eines anwendungsspezifischen Prozesses definiert.

Die Kategorien zur Unterscheidung von *Asset-Prozessen* in *organisations-* und anwendungsspezifische Prozesse sind orthogonal, das heißt, auch Prozesse die sowohl *Assets* verschiedener Benutzer als auch verschiedener Domänen zugreifen sind möglich. Innerhalb der anwendungsspezifischen Prozesse kann eine weitere Kategorisierung anhand des Zugriffs auf *Basisdomänen*[53] hilfreich sein. Prozesse die der Funktionalität des *begriffsorientierten Systems* dienen und auf Basisdomänen zugreifen, werden als *systemspezifisch* bezeichnet. Alle anderen anwendungsspezifischen Prozesse operieren lediglich auf verschiedenen Benutzerdomänen. Ein Beispiel hierfür ist, im Rahmen der Fallstudie der WEL, der Verweis auf eine Motivdomäne im *Asset-Typ* einer Bildkarte.

2.4 Anforderungen an begriffsorientierte Systeme

In den vorangegangen Abschnitten wurde das *Modell der begriffsorientierten Inhaltsverwaltung* nach Schmidt et al. (2003) und Sehring (2004) eingeführt. Als die zentralen Eigenschaften des Modells, die sich aus der Nachbildung des *menschlichen Erkenntnisprozesses* ergeben, wurden *Offenheit* und *Dynamik* identifiziert[54]. Das Modell ist offen, weil es die *Evolution* der, in einer *Modellinstanz*, verwalteten *individuellen Asset-Modelle* erlaubt. Es ist dynamisch, weil die Evolution jederzeit im *Lebenszyklus* einer Modellinstanz erfolgen kann. Zusätzlich werden alle *Prozesse* auf den *Entitäten*, die durch das begriffsorientierte Modell abgebildet werden, wiederum als beschreibbare Entitäten aufgefasst. Die *Prozess-Entitäten*, sogenannte *Werke*, werden durch *Assets* als *Elemente erster Ordnung* im Modell beschrieben.

In den nachfolgenden Abschnitten werden aus diesen zentralen Eigenschaften Anforderungen an *begriffsorientierte Systeme* abgeleitet.

2.4.1 Unterstützung des begriffsorientierten Modells

Zunächst muss ein System die zentrale Erkenntnis des *begriffsorientierten Modells* unterstützen, also die *Einheit von Konzepten und Inhalten* in sogenannten *Assets*[55] und damit verbunden das reichhaltige *Beziehungsgeflecht* unterschiedlichster *Beziehungstypen* zwischen *Assets*[56]. Insbesondere wird von Sehring[57] die Unterscheidung der Beziehungen *Klassifikation* und

[53] Vgl. Abschnitt 2.2.1, Domänen - Objektdomänen
[54] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[55] Vgl. Abschnitt 2.2.4, *Assets* als Einheit von Konzept und Inhalt
[56] Vgl. Abschnitt 2.2.5, Semantische Beziehungen
[57] Sehring (2004), S. 17f.

Definition gefordert.

Um ein *Asset intensional* zu *definieren*, muss die Festlegung des *konzeptionellen* Teils, also des *Asset-Typs*, durch die Auswahl von maschinenabhängigen *Basistypen* als *charakteristische Eigenschaften* (*Erstheiten*), von Beziehungstypen zu anderen *Assets* (*Zweitheiten*) und von *Regeltypen* für Erst- und Zweitheiten (*Drittheiten*) ermöglicht werden. Anschliesend muss die Erfassung von Inhalten und Zuordnung zu *Assets*, und damit zu *Asset-Typen*, erfolgen können.

Darüberhinaus ist die Beschreibung der *Organisationsstruktur*, die durch *Benutzer-* und *Gruppen-Assets* modelliert wird, von besonderer Bedeutung für die Verwaltung individueller *Asset-Modelle* zur Unterstützung einer *semantisch angereicherten Mensch-Maschine-Kommunikation*[58].

2.4.2 Unterstützung von Modellevolution

Wie bereits beschrieben, beruht das *begriffsorientierte Modell der Inhaltsverwaltung* auf einer Hypothese Cassirers[59], die von einem *fortlaufenden menschlichen Erkenntnisprozess* ausgeht. Dieser *Prozess* besteht unter anderem aus *Differenzierung* und *Integration* von bestehenden Konzepten, die im begriffsorientierten Modell als *Asset-Typen* im *konzeptionellen* Teil von *Assets* verwaltet werden. Weiterhin stellen Konzepte *Abstraktionen* dar, die im *Fokus* einer bestimmten *Anwendungsdomäne* gebildet werden. Verschiebt sich dieser Fokus, so werden auch die bestehenden Konzepte einer Wandlung unterworfen sein.

Beide Sachverhalte können und sollen mit hoher Frequenz stattfinden, weil etwa der fortlaufende Erkenntnisprozess in der Regel mit einem *Erkenntnisgewinn* einhergeht. Demzufolge ist ein System, das die Änderung der verwalteten *Asset-Typen*, zusammengefasst zu *Asset-Modellen*, nur in besonderen Phasen seines *Lebenszyklus* zulässt, nicht für die Unterstützung des beschriebenen Erkenntnisprozesses geeignet. Die *Evolution* der verwaltete *Asset-Modelle* im laufenden Betrieb, genauer in jeder Phase des Lebenszyklus, ist eine zentrale Anforderung an begriffsorientierte Systeme.

Darüberhinaus werden Mechanismen zur *Migration* bestehender *Inhalte* benötigt, die mit der anhaltenden *Modellevolution* Schritt halten können. Insbesondere müssen die *Benutzer* jederzeit Zugriff auf alle bestehenden Inhalte haben, das heißt, die Zugriffsgeschwindigkeit muss auf einem angemessenen Niveau liegen, um deren Erkenntnisprozess nicht zu behindern.

2.4.3 Unterstützung subjektiver Sichten durch Modellevolution

Der *menschliche Erkenntnisprozess*, wie er im vorangegangen Abschnitt beschrieben ist, erfolgt im *Fokus* einer *Anwendungsdomäne* und insbesondere im Fokus der bisherigen *Erkenntnis* eines Menschen, ist also ein höchst *subjektiver* Vorgang. *Ein begriffsorientiertes System* muss folglich in der Lage sein, jedem *Benutzer* sein *individuelles Asset-Modell* und Zugriff auf persönliche oder von einer übergeordneten *Organisationseinheit*[60] abgeleitete *Assets* bereitzustellen. Hierzu ist durch den Benutzer eine *Authentifizierung* und je nach Vertraulichkeit der zugegriffenen *Assets* gegebenenfalls eine *Authorisierung* gegenüber dem System notwendig.

[58] Vgl. Abschnitt 2.2.2, Inhalte
[59] Cassierer (2001).
[60] Vgl. Abschnitt 2.2.6, Benutzer und Gruppen

Damit sich ein neuer Benutzer nicht genötigt sieht, seinen Erkenntnisstand, bestehend aus *Sinnstruktur* und *Inhalten*, beschrieben durch *Asset-Modell* und materialisierende *Assets*, vollständig in ein begriffsorientiertes System einzupflegen, sollte das System entsprechende Mechanismen zur *Modellableitung* von, im Sinne der Sinnstruktur und Erkenntnis, nahestehenden Organisationseinheiten anbieten. Entsprechend ist die Ableitung von benutzereigenen *Asset-Modellen* denkbar, die im Fokus einer anderen Domäne betrieben werden und die parallel Gültigkeit besitzen. Die Ableitung des *Asset-Modells* und die anschließende *inkrementelle Modellevolution* stellen die Basis für die systematische *Kooperation subjektiver Modelle* dar, deren Anforderungen im nachfolgenden Abschnitt herausgearbeitet werden.

2.4.4 Unterstützung der Kooperation subjektiver Modelle

Die systematische *Kooperation subjektiver Modelle*[61] beruht, wie bereits gezeigt, auf der *Ableitung* von bestehenden *Asset-Modellen* und der kontrollierten, *inkrementellen Modellevolution*. Das *begriffsorientierte System* muss die *Evolutionsschritte* aufzeichnen und eventuelle Mehrdeutigkeiten in der *Interaktion* mit dem *Benutzer* durch Festlegung von *Regeln* ausräumen. Eine Möglichkeit der Verwaltung von entsprechenden Regeln ist die Beschreibung durch *Drittheiten* in *Assets*.

Nach Festlegung eines *eindeutigen Evolutionpfades*, muss das System entsprechende *Modelladaptoren* generieren, damit die *Modellkooperation*, in diesem Fall die *Personalisierung*, entlang der Beziehungen in der *Organisationsstruktur* automatisch erfolgen kann. Auch der umgekehrte Weg der Kooperation subjektiver Modelle, also die *Publikation*, muss durch Bildung der *Inversen der Modellevolution* und anschließende *Generierung* von Adaptoren ermöglicht werden. Auf diese Weise können beliebige Organisationseinheiten, die über einen Pfad in der Organisationstruktur miteinander verbunden sind, durch Zeichen, die ihrer *Sinnstruktur* angepasst sind, miteinander kommunizieren und somit auch kooperieren.

2.4.5 Unterstützung von Asset-Prozessen

Die systematische Unterstützung von *Prozessen auf Assets*, die wiederum als durch *Assets* beschreibbare *Entitäten* aufgefasst werden (*Werke*) und die alle Vorgänge des Zugriffs auf *Assets* abbilden, erfordert ein besonderes *Prozessmodell*. Wie in Abschnitt 2.3.1 dargelegt, können Werke unter anderem als Ausschnitte des *menschlichen Erkenntnisprozesses* betrachtet werden, der nach Cassierer (2001) nicht primär auf das Endprodukt, sondern auf *fortschreitende Erkenntnis* im Umgang mit den Entitäten des momentanen *Anwendungsfokus* ausgerichtet ist[62]. Neben dieser analytischen Nutzung von Werkbeschreibungen, soll auch die konstruktive Nutzung unterstützt werden, die die geplante Fortschreibung eine Werkes in die Zukunft abbildet. Für beide Arten der Nutzung besteht die Notwendigkeit *inkrementellen Werkfortschritt* abzubilden, der in der Umwandlung von *präskriptiven* in *deskriptive* Elemente der Werkbeschreibung besteht. Ein definiertes *Werkende* kann im Prozessmodel nicht vorausgesetzt werden, da Werkbeschreibungen, wie andere *Assets* auch, permanenter Veränderung unterworfen sein können.[63]

[61] Vgl. Abschnitt 2.3.2, Organisationsspezifische Prozesse
[62] Fornfeist (2003), S. 9f.
[63] Vgl. Abschnitt 3.3, Assets-Prozesse als Entitäten

Zusätzlich zur *Kooperation subjektiver Modelle*, also Werken die entlang der *Organisationstruktur verteilt* (*organisationsspezifische Werke*) sind, wie im vorangegangenen Abschnitt beschrieben, gibt es für Werke eine zweite Dimension der Verteilung. Es handelt sich hierbei um die Kooperation verschiedener *Benutzer- oder Basisdomänen*[64]. Insbesondere muss ein *begiffsorientiertes System*, das *Benutzer-, Gruppen-, Rechte-, Rollen- und Werkverwaltung* im *begriffsorientierten Modell* abbildet, anwendungsspezifische Werke zulassen[65]. Beide Dimensionen der Verteilung sind *orthogonal*, so das Werke denkbar sind, die sowohl die Kooperation subjektiver Modelle als auch mehrerer Domänen beschreiben.

2.5 Einordnung der Arbeit

Im nachfolgenden Abschnitt wird der vorliegende Text gegenüber anderen Arbeiten zum begriffsorientierten Modell am Arbeitsbereich STS abgegrenzt. Der Abschnitt 2.5.2 gibt anschließend einen Überblick über verwandte Themen und Gebiete.

2.5.1 Abgrenzung der Arbeit

Diese Arbeit basiert auf dem *Model der begriffsorientierten Inhaltsverwaltung* nach Schmidt et al. (2003) und Sehring (2004). Darüberhinaus werden durch Sehring weitere Grundlagen für die Entwicklung eines *begriffsorientierten Inhaltsverwaltungssystems*, wie die Entwicklung einer *Asset-Definitions-, Anfrage- und Manipulationssprache*, eines *Asset-Modell-Compilers* und einer Systemarchitektur[66] gelegt. Diese Arbeit beschreibt keine neuen Entwürfe oder Implementierungen für die erstgenannten Aspekte der begriffsorientierten Inhaltsverwaltung, sondern nutzt die bestehende *Infrastruktur*, um sich auf das Konzept der *Werke* und der hierdurch beschriebenen *Kooperation* über Organisations- und Domänengrenzen hinweg zu konzentrieren.

Ein weiterer Aspekt dieser Arbeit ist die besonderen Vorzüge eines *verteilten Entwurfes*, im Rahmen der vorhandenen Architektur, für das begriffsorientierte System herauszuarbeiten. Dabei spielen sowohl allgemeine Entwurfsziele *verteilter Systeme*, sowie besondere Anforderungen an begriffsorientierte Systeme eine Rolle. Es wird gezeigt, dass die von Cassirer geforderte *dynamische Anwendung der Offenheit* für den *menschlichen Erkenntnisprozess* ihre natürliche Fortsetzung in *kooperierenden, offenen und verteilten Diensten* findet.[67]

Der im Rahmen dieser Arbeit entworfene und implementierte *Prototyp* beschränkt sich auf eine exemplarische *Konfigurationen* aus manuell entwickelten *Dienstschnittstellen- und Koordinationsmodulen* sowie einem *generierten Interpretationsmodul*, das den Zugriff auf eine Standardkomponente zur Massenspeicherabstraktion realisiert[68]. Die Orientierung an der Sehring'schen Systemarchitektur[69] gestattet dennoch die Kombination mit anderen, durch den *Asset-Modell-Compiler* generierten, Modulen, die der gleichen *Asset-Schnittstelle* genügen. Damit stehen dem Prototypen zusätzlich die Fähigkeiten der begriffsorientierten Inhaltsver-

[64] Vgl. Abschnitt 2.2.1, Domänen
[65] Vgl. Abschnitt 2.3.3, Anwendungsspezifische Prozesse
[66] Vgl. auch Bachmann (2003).
[67] Vgl. Abschnitt 3.1, Implikationen der Offenheit und Dynamik für den Systementwurf
[68] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems
[69] Sehring (2004), S. 113ff.

waltung, abzüglich *Offenheit* und *Dynamik*, zur Verfügung. Darüberhinaus ist die exemplarische Konfiguration als *Machbarkeitsstudie* anzusehen, die im Erfolgsfall in der Entwicklung entsprechender *Generatoren* für das *Compiler Backend*[70] resultiert, um zukünftig das gesamte Potential der begriffsorientierten Inhaltsverwaltung zu erschließen. Insofern stellt die Orientierung an der SEHRING'schen Systemarchitektur eine Investition in die Zukunft der vorliegenden Arbeit dar[71].

2.5.2 Verwandte Themen und Gebiete

In diesem Abschnitt werden verwandte Themen und Gebiete beleuchtet, die als Gemeinsamkeit mit Werken (Prozessen auf *Assets)* in der begriffsorientierten Inhaltsverwaltung, die Fokusierung auf *langandauernde koordinierte Aktivitäten* aufweisen. Konkret sind dies *Workflows*, *Business Conversations* und *Agentensysteme.*

Workflows

Workflows sind zielgerichtete und koordinierte langandauernde Aktivitäten beliebig vieler *Akteure.* Akteure können hier Menschen und *IT-Systeme* sein. In der Regel entstammen die Akteure einer *Organisation*, das heißt *Workflows* sind nicht organisationsübergreifend. Im Gegensatz zu den, im nachfolgenden Abschnitt beschriebenen, *Business Conversations*, werden *Workflows* von einer *zentralen Instanz* gesteuert, die den Ablauf des *Workflows* auf der Basis einer *Workflow-Definition* bestimmt und den Akteuren zu gegebener Zeit ihre Aufgaben zuweist. Die Akteure sind somit keine *autonomen* Systeme.

Das Modell der *Workflows* entscheidet explizit zwischen *Workflow-Definition* und der *Workflow-Instanz.* Die *Workflow-Definition* spezifiziert einen *Kontrollfluß* mit einem definierten Ende. Desweiteren sind *Workflows* dokumentenorientiert und auf die Erstellung eines finalen *Dokumentes* zum Prozessende, als Produkt des *Workflows*, ausgerichtet. Der *Lebenszyklus* eines *Workflows* sieht nach dem Erreichen des Prozessendes, im Gegensatz zum Modell der *Business Conversations*, keine explizite *Rückmeldephase* zur Qualitätssicherung des *Workflows* bzw. seines Produktes vor. Vielmehr muss eine Rückmeldung innerhalb des *Workflows* oder außerhalb des *Workflow-Management-Systems* erfolgen. Für weitere Ausführungen zum Thema *Workflows* und *Workflow Management* sei auf van der Aalst et al. (2002) verwiesen.

Workflows unterscheiden sich in einigen Punkten zentral von dem Konzept der *Werkes*, das in der vorliegenden Arbeit beschrieben wird. Viele dieser Unterscheidungskriterien werden am Ende des nachfolgenden Abschnittes dargelegt, sofern es sich um gemeinsame Kriterien von *Workflows* und *Business Conversations* handelt. Die verbleibenden Merkmale zur Unterscheidung sollen jedoch schon hier erörtert werden.

Werke sind im Gegensatz zu *Workflows* organisationsübergreifende Prozesse, die keine zentrale Steuerung durch eine Kontrollinstanz erfahren. Akteure eines *Workflows* sind, anders als Akteure eines Werkes, somit keine *autonomen Systeme.* Weiterhin erlaubt die dezentrale Organisation von Systemen zur Werkausführung die *Migration* oder *Delegation* von *(Teil-)Werken*[72], etwa im Falle von Systemüberlastungen und -ausfällen.

[70] Sehring (2004), S. 101f.
[71] Vgl. Abschnitt 5.3, Ausblick
[72] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen

Business Conversations

Das Modell der *Business Conversations (BC)* basiert auf der *Sprechakttheorie*, insbesondere auf *Sprechakten* vom Typ *Conversations for Actions (CFA)*[73]. Ebenso wie die im vorherigen Abschnitt beschriebenen *Workflows*, handelt es sich bei *Business Conversations* um zielgerichtete und koordinierte langandauernde Aktivitäten. Im Gegensatz zu *Workflows* kennen *Business Conversations* jedoch nur die zwei Akteure *Kunde* und *Dienstleister* sowie deren binäre Beziehung. Akteure können grundsätzlich Menschen und *IT-Systeme* sein. Als Erweiterung zu *Workflows* sind *Business Conversations* in der Regel organisationsübergreifend, das heißt Kunden und Dienstleister können unabhängigen Organisationen entstammen.

Die Akteure, also *BC-Systeme* und durch Menschen gesteuerte *BC-Clients*, sind autonom und werden im Gegensatz zu *Workflow-Management-Systemen* nicht von einer zentralen Komponente gesteuert. Weiterhin sind die Akteure von *Business Conversations* grundsätzlich migrierbar, allerdings hat dieser Aspekt, durch die gesammelten Erfahrungen im Zusammenhang mit der Implementierung von *BC-Systemen* laut Hupe (2000, S. 15), an Bedeutung verloren.

Während einer *Business Conversation* tauschen die Akteure interne Inhalte (*content*) in Form von Dokumenten aus und bestimmen auf dieser Basis den Fortgang der Konversation. Dabei durchlaufen *Business Conversations* vier Phasen[74], insbesondere verfügen sie, im Gegensatz zu *Workflows*, über eine *Rückmeldephase*, die der *Qualitätsicherung* des Prozesses dient.

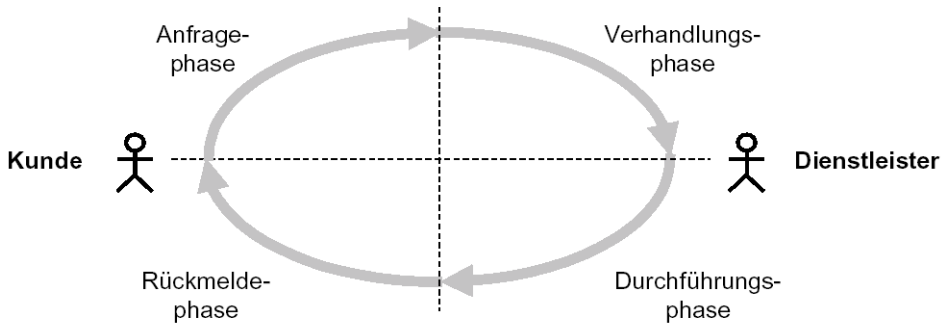

Abbildung 2.10: Phasen einer *Business Conversation*.
Quelle: Hupe (2000, S. 6)

Das Modell der *Business Conversations* unterscheidet explizit zwischen *Spezifikation*, bestehend aus *Prozess- und Inhaltsspezifikation*, und *Exemplar* (Instanz der Spezifikation). Die Prozessspezifikation definiert eine Menge von zulässigen *Dialogen*, davon jeweils mindestens ein *ausgezeichneter finaler Dialog* und ein *Abbruchdialog*. Somit verfügen *Business Conversations* über eine explizites Prozessende. Die Dialoge wiederum definieren eine Menge von möglichen *Anfragen* innerhalb eines Dialoges. Eine Anfrage legt umgekehrt die Menge der zulässigen Dialoge fest, die der Anfrage nachfolgen können. Anfragen sind folglich die kleinste Einheit des *Prozessfortschrittes* in einer *Business Conversation*.

Die Konzepte der *Nebenläufigkeit* und *Schachtelung* von *Teilkonversationen* sind nicht Teil

[73] Wegner (1998), S. 8ff.
[74] Vgl. Abbildung 2.10, Vier Phasen einer *Business Conversation*

der Spezifikation von *Business Conversations*. Deren Umsetzung hängt vom konkreten Prozessmodell einer *BC-Systemimplementierung* ab. Eine *BC-Spezifikation* definiert lediglich eine Menge von zulässigen Kontrollflüssen aus Dialogen und Anfragen die immer mit einem finalen oder einem Abbruchdialog enden.

Zum Abschluß so noch darauf hingewiesen, daß *Business Conversations akteurs- und medienunabängig* sind. Das heißt, *Business Conversations* abstrahieren von der konkreten Implementierung eines Akteurs, also der Frage ob *regelbasierter* oder *interaktiver* Akteur, und der Art (Protokolle, Formate) der verwendeten Medien zum Dokumentenaustausch. Vertiefende Ausführungen zum Thema *Business Conversations* können bei WEGNER[75] und HUPE[76] nachgelesen werden.

Zu der Abgrenzung zwischen *Business Conversations* und *Werken*, wie sie in dieser Arbeit fortgestellt werden, ist anzumerken, dass Werke kein explizites Prozessende besitzen und somit nicht auf ein finales Dokument, als Produkt des Prozesses, ausgerichtet sind. Stattdessen steht in Werken der inkrementelle Prozess- oder Werkfortschritt im Vordergrund. Konsequenterweise unterscheiden Werke deswegen auch nicht explizit zwischen Spezifikation und Exemplar, wie es im Modell der *Business Conversations* erfolgt, sondern bilden einen kontinuierlichen Weg von der *präskriptiven* zur *deskriptiven* Prozessbeschreibung ab. Desweiteren spezifizieren *Business Conversations* eine Menge von zulässigen Kontrollflüssen, wohingegen Werke nicht auf Kontroll- oder *Datenfluß* festgelegt sind. In dieser Arbeit wird die Entscheidung, zugunsten einer *Datenflussmodellierung*, im Rahmen des Entwurfes des *begriffsorientierten Systems* gefällt[77]. Abschließend sei darauf hingewiesen, dass an der Ausführung eines Werkes beliebig viele *organisationsübergreifende* Akteure beteiligt sein können, die durch Delegation von Teilwerken ihre Aufgaben zugewiesen bekommen, während *Business Conversations* nur die binäre Kunde-/Dienstleister-Beziehung kennen.

Agentensysteme

Agentensysteme basieren auf der *Sprechakttheorie* und sollen hier nur kurz vorgestellt werden. Die Entwicklung von der Sprechakttheorie zum *Modell der Agenten* sowie der Entwurf und die Implementierung eines Agentensystems werden in Wegner (1998) ausführlich dargestellt.

Agenten sind autonome und migrierbare *Software-Komponenten*, die auf Anfrage Aussagen über externes Wissen (*knowledge*) treffen oder externes Wissen manipulieren können. Dazu tauschen Agenten, ebenso wie *Akteure* bei *Business Conversations*, *Sprechakte* aus. Im Gegensatz zu den Akteuren steht bei Agenten jedoch die Fähigkeit der Migration im Zentrum des Interesses. Ein weiterer Unterschied liegt in der Auswirkung der Kooperation. Während Agenten externes Wissen verwalten und manipulieren, besteht der Zweck von *Business Conversations* in dem Austausch von internen Inhalten (*content*)[78].

Zum Abschluß sei darauf hingewiesen, dass sich die Modelle der autonomen Agenten und der *Business Conversations* vielfach überschneiden. Demzufolge können die wesentlichen Unterschiede zum Konzept der *Werke*, das in dieser Arbeit dargelegt wird, dem vorherigen Abschnitt über *Business Conversations* entnommen werden.

[75] Wegner (1998), S. 8ff.
[76] Hupe (2000), S. 5ff.
[77] Vgl. Abschnitt 3.3, Prozesse auf *Assets* als Entitäten
[78] Hupe (2000), S. 15.

Kapitel 3

Eine offene und dynamische Infrastruktur für begriffsorientiertes Arbeiten

Dieses Kapitel erörtert im Abschnitt 3.1 die Implikationen der zentralen Eigenschaften *Offenheit* und *Dynamik* für den Entwurf einer *Infrastruktur* zum *begriffsorientierten Arbeiten*. In der Folge wird die Entscheidung zugunsten einer *verteilten* Infrastruktur getroffen. Daraufhin werden die allgemeinen Entwurfsziele verteilter Systeme dargelegt.

Der anschließende Abschnitt 3.2 motiviert die Modellierung von *Domänen* und deren *Kooperation* durch Kooperation *verteilter Dienste*. Desweiteren werden sogenannte *Basis- oder Systemdienste* beschrieben, die dem Betrieb der begriffsorientierten Infrastruktur dienen und deren *Inhalte* ebenfalls im *begriffsorientierte Modell* verwaltet werden.

In Abschnitt 3.3 wird die Beschreibung von *Asset-Prozessen* durch spezielle *Assets*, sogenannte *Werke*, begründet. Darüberhinaus werden die Formen der *präskriptiven* und *deskriptiven Werkbeschreibung* zur Abbildung des *inkrementellen Werkfortschrittes* entworfen. Zu Ausführung präskriptiver Werkbeschreibungen führt der Abschnitt außerdem ein *Prozessmodell* ein.

Abschnitt 3.4 führt die Kooperation von Diensten auf die *Delegation von Teilprozessen* zurück und motiviert auf diese Weise den Entwurf von sogenannten *Teilwerken*. Anschließend werden verschiedene Arten von Teilwerken beschrieben. Zum Abschluß des Abschnittes wird die Delegation von Teilwerken, unter besonderer Berücksichtigung der allgemeinen Entwurfsziele verteilter Systeme, modelliert.

Am Ende des Kapitels stellt Abschnitt 3.5 eine Architektur für begriffsorientierte Inhaltsverwaltungssysteme nach Sehring (2004) vor, die auch für die entworfene und realisierte Infrastruktur in der vorliegenden Arbeit als Leitfaden dient.

3.1 Implikationen der Offenheit und Dynamik für den Systementwurf

Der folgende Abschnitt schließt von der dynamischen Offenheit der begriffsorientierten Inhaltsverwaltung auf den Entwurf einer verteilten und kooperativen Infrastruktur für begriffsorientiertes Arbeiten. Dabei wird auf Eigenschaften verteilter Systeme vorgegriffen, die im Abschnitt 3.1.2 zusammengefasst und in Coulouris et al. (1994) als wesentliche Ziele verteilter Systementwürfe genannt werden.

3.1.1 Verteilung und Kommunikation als Folge von dynamischer Offenheit

In den Abschnitten 2.4.2 und 2.4.3 wurden Anforderungen an begriffsorientierte Systeme formuliert, die einen offenen und dynamischen *Erkenntnisprozeß* unterstützen. Die wesentliche Aussage ist, dass die *Evolution* der menschlichen *Sinnstrukturen*, abgebildet durch *Asset-Modelle*, nachvollzogen werden muss. Da die Fortentwicklung der Sinnstrukturen in der Regel *inkrementell* erfolgt und über die Zeitabstände zwischen zwei Inkrementen keine Annahmen getroffen werden können, sie insbesondere auch sehr kurz sein können, ist ein System erforderlich, das die *Modellevolution* während seiner *Laufzeit* zulässt. Die *Nebenwirkungen* dieses Vorgangs (Unterbrechungen, Performanzeinbrüche, *Migration* von *Assets*) sollen weitgehend vor den Benutzer des Systems verborgen bleiben. Es kann in diesem Zusammenhang auch von *transparenter Modellevolution* gesprochen werden.

Nun kann ein Schritt der Modellevolution wegen der geforderten Transparenz u.U. nicht zeitnah und vollständig erfolgen, weil eine große Menge von Assets im alten Modell verwaltet wird, deren Migration viel Last auf dem System erzeugen würde. Daraus folgt das *Asset-Modelle* parallel zu betreiben sind und die Migration, je nach gewählter *Migrationsstrategie*, bedarfsweise erfolgt. Die konkrete Migration eines *Assets* wird durch einen *Adapter* vollzogen, der durch den, bei SEHRING[1] beschriebenen, *Modell-Compiler* aus den *Asset-Modellen* generiert wird. Auf diese Weise resultiert jedes Inkrement der Modellevolution in einem parallelen Modell, auf dessen *Assets* über einen entsprechenden *Modelladapter* zugegriffen werden kann.

Der beschriebene Mechanismus der Adaption wird ebenfalls eingesetzt, um die *Ableitung* einer *subjektiven Sicht* eines Benutzers aus der Sicht einer übergeordneten *Organisationseinheit* zu unterstützen. In Termen von *Asset-Modellen*, handelt es sich dabei um die Ableitung eines individuellen Benutzermodells aus dem *Asset-Modell* einer anderen Organisationseinheit. Im Gegensatz zum vorherigen Absatz wird in diesem Fall jedoch keine Migration von *Assets* vollzogen, sondern die *Sichten* (*Sinnstrukturen*), definiert durch ihre jeweiligen *Asset-Modelle*, werden überlagert.

Die zwei zuvor dargelegten Vorgänge, also die Modellevolution von Benutzermodellen und die Ableitung individueller Modelle aus übergeordneten Modellen, führen, ebenso wie die Modellierung von Basisdomänen[2] durch das begriffsorientierte Modell, je nach *Geschwindigkeit der Erkenntnisprozesse*, *Benutzerzahl* und *Verzweigungsgrad* in der Organisationsstruktur, zu einer sehr großen Zahl von parallel zu unterhaltenden Modellen und *Adaptionsmodulen*[3].

[1] Sehring (2004), S. 99.
[2] Vgl. Abschnitte 2.2.1, Objektdomänen; 2.2.6, Benutzer und Gruppen
[3] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems

Diese Erkenntnis und erste, im Projekt WARBURG ELECTRONIC LIBRARY[4] gesammelte, Erfahrungen mit der inhaltlichen Personalisierung zeigen, dass ein *monolithischer* Ansatz für den Entwurf eines begriffsorientierten Systems nicht zielführend ist. Stattdessen liegt die Abbildung auf ein *verteiltes System* nahe, in dem die auftretenden Lasten gleichmäßiger auf ein *Netzwerk* aus begriffsorientierten Teilsystemen verteilt werden kann. Je nach Umsetzung des verteilten Systems, kann so eine hohe *Toleranz* gegenüber der Erhöhung der Benutzer- und Modellzahlen im *begriffsorientierten Netzwerk* erreicht werden[5].

In einem verteilten Entwurf, der den Knoten seines Netzwerkes verschiedene begriffsorientierte Modelle zuweist, müssen die zwei in den Abschnitten 2.3.2 und 2.3.3 geschilderten Arten von kooperativen Prozessen durch Kommunikation zwischen den Knoten abgebildet werden. Je nach Aufteilung der *Asset-Modelle* können auf diese Weise erheblich Kommunikationskosten entstehen, die den Vorzügen der Verteilung entgegenwirken. Es bleibt also festzuhalten, dass die Verteilung für ein begriffsorientiertes System große Vorteile mit sich bringen kann, die jedoch gegenüber den Kommunikationskosten der notwenigen kooperativen Prozesse abzuwägen sind. Eine konkrete Zuordnung der Elemente des begriffsporientierten Modells aus 2.2 und 2.3 auf die Knoten eines verteilten Systems wird im Abschnitt 3.2.1 diskutiert.

Zum Abschluß sei noch darauf hingewiesen, dass die geforderte Offenheit für das begriffsorientierte Modell ihr Pendant in der Offenheit[6], als Entwurfsziel verteilter Systeme, findet. So wie das Modell, mittels Evolution und Adaption, offen ist für die *Kooperation* mit neuen *gedanklichen Konzepten*, so ist das System, durch definierte *Schnittstellen* und *Standards*, offen für die Kooperation mit neuen *Hard- und Softwarekonzepten*

3.1.2 Ausgewählte Entwurfsziele verteilter Systeme

Nach COULOURIS ET AL. stellen *Transparenz*, *Offenheit*, *Nebenläufigkeit*, *Skalierbarkeit* und *Fehlertoleranz* die zentralen Ziele eines verteilten Entwurfes dar. Je nach Ausprägung des konkreten Systems können sie auch in einer komplementären Beziehung zueinander stehen. Nachfolgend werden die genannten Ziele genauer Untersucht und, wie im Falle der Transparenz, weiter untergliedert.

Transparenz

Transparenz bezeichnet die Eigenschaft eines *verteilten Systems* seine Verteilung zu verbergen, und das Gesamtsystem als *kompakt* erscheinen zu lassen. Die erhöhte *Komplexität* des verteilten Systems wird somit verborgen, und die Vorteile der gesteigerten *Nebenläufigkeit*[7], *Skalierbarkeit* und *Fehlertoleranz*[8] kommen voll zum Tragen. Im Folgenden werden verschiedene Ausprägungen der Transparenz betrachtet[9].

- Transparenter Zugriff (*Access Transparency*)

[4] Vgl. Abschnitt 2.1.1, WEL - Warburg Electronic Library
[5] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit
[6] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Offenheit
[7] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Nebenläufigkeit
[8] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Fehlertoleranz
[9] ANSA (1989); ISO (1992).

Lokale und entfernte *Informationsobjekte* können über identische Operationen zugegriffen werden.

- Transparente Lokalisierung (*Location Transparency*)

  Entfernte Informationsobjekte können ohne Kenntnis ihres physikalischen Aufenthaltsortes im verteilten Gesamtsystem zugegriffen werden.

- Transparente Nebenläufigkeit (*Concurrency Transparency*)

  Verschiedene Prozesse können ohne gegenseitige Beeinflussung nebenläufig auf gemeinsame, lokale oder entfernte Informationsobjekte zugreifen.

- Transparente Replikation (*Replication Transparency*)

  Verbirgt die Existenz von *Replikaten* von lokalen oder entfernten Informationsobjekten, zur Steigerung von *Performanz* und Zuverlässigkeit, gegenüber zugreifenden Prozessen.

- Transparente Fehler (*Failure Transparency*)

  Auftretende Fehler auf *Hard- oder Softwareebene* werden verborgen und zugreifende Prozesse in die Lage versetzt, ihre Aktivitäten konsistent zu beenden.

- Transparente Migration (*Migration Transparency*)

  Informationsobjekte können ihren Aufenthaltsort im verteilten Gesamtsystem ändern, ohne andere Komponenten des Systems zu beeinflussen.

- Transparente Lastverteilung (*Performance Transparency*)

  Auftretende *Lasten* können durch dynamische Anpassungen im Gesamtsystem aufgefangen werden.

- Transparente Skalierung (*Scaling Transparency*)

  Die Größe des Gesamtsystems kann variiert werden, ohne die Struktur des Systems oder die verwendeten *Algorithmen* anzupassen.

In einigen Anwendungsfällen jedoch kann Transparenz allgemein oder eine der oben erwähnten Ausprägungen unerwünscht sein. Beispiel hierfür sind Druckaufträge in einem lokalen Netzwerk (*LAN*). Transparente Lokalisierung ist in diesem Fall nicht sinnvoll, weil der Benutzer natürlich Auskunft bekommen muss, auf welchem Drucker sein Auftrag ausgeführt wurde.[10]

Offenheit

Die Offenheit eines verteilten Systementwurfes betrifft sowohl die *Hardware-* als auch die *Software-Ebene*. Um Offenheit zu gewährleisten, müssen *Schnittstellen* des Systems spezifiziert, dokumentiert und frei zugänglich sein.

Auf *Hardware-Ebene* ist beispielsweise die Erweiterung um zusätzliche oder andersartige *Mainframes*, *Personal Computer*, *Peripheriegeräte* oder *Kommunikationskanäle* wünschenswert. Bei reinen *Software-Entwürfen*, die auf einer gegebenen *Hardware-Plattform* aufsetzen,

[10] Coulouris et al. (1994), S. 20f.

hängt die Offenheit auf *Hardware-Ebene* von der gewählten *Hardware-Plattform* und ihren *Kommunikationsschnittstellen* ab.

Auf *Software-Ebene*, beschreibt die Offenheit die Fähigkeit eines verteilten Systems, nachträglich weitere *Kommunikationsprotokolle* oder *Datenformate* zu unterstützen.

Ein offenes verteiltes System kann aus diversen *Hard- und Software-Komponenten* von unterschiedlichen Herstellern zusammengesetzt sein, sofern die Komponenten den veröffentlichten Schnittstellenspezifikationen entsprechen. Auf diese Weise kann ein *heterogenes* Gesamtsystem gebildet werden. Die Erfüllung einer Schnittstellenspezifikation durch eine *Hard- oder Software-Komponente* sollte durch die Hersteller getestet und zertifiziert sein.[11]

Nebenläufigkeit

Nebenläufigkeit hat in verteilten Systemen zwei Aspekte, die sich aus der *Unabhängigkeit* der physikalischen *Ressourcen* ergeben. Zum einen können mehrere Benutzer simultan an einem verteilten System arbeiten und in Ihrem lokalen Prozess nebenläufige *Code-Ausführung* initiieren. Zum anderen können in einem verteilten Gesamtsystem mehrere *Server-Prozesse* gleichzeitig (parallel, *Multiprozessorbetrieb*) oder quasi-gleichzeitig (*Interleave-Betrieb*) auf verschiedenen logischen oder physikalischen *Servern* ablaufen.

Nebenläufigkeit kann den *Durchsatz* eines verteilten Systems gegenüber einem nicht-verteilten Ansatz erheblich steigern, sofern die Verteilung nicht durch übermäßige *Kommunikationskosten* erkauft wird. Kommunikationskosten entstehen aufgrund von besonderen Anforderungen an die *Synchronisation* von Zugriffen auf gemeinsame Ressourcen oder der Durchührung *simultaner Aktivitäten* in einem verteilten System.[12]

Skalierbarkeit

Die Skalierbarkeit eines verteilten Systems hängt sowohl von dessen Offenheit[13], als auch von dessen Entwurf ab. Der Entwurf eines verteilten Systems kann künstliche Schranken enthalten, beispielsweise die Beschränkung der Anzahl von *Host-Adressen* im *IP-Protokoll*, die zunächst unerreichbar erscheinen, aber bei einer Skalierung des verteilten Systems zum *Engpass* werden. Limitationen dieser Art sind folglich im Entwurf sorgfältig zu planen und festzulegen. Idealerweise ist bei dem Entwurf verteilter Systeme davon auszugehen, dass weder *Software- noch Hardware-Ressourcen* irgendwelchen Beschränkungen unterliegen und in unendlicher Zahl zur Verfügung stehen.

Ein weiteres Problem der Skalierbarkeit verteilter Systeme sind die, möglicherweise überproportional mit der Zahl der Teilsysteme, wachsenden Kommunikationskosten, aufgrund der *Komplexität* verwendeter Algorithmen beim Zugriff auf gemeinsame Ressourcen oder der Koordinierung simultaner Aktivitäten[14, 15, 16].

---

[11] Coulouris et al. (1994), S. 14ff.
[12] Coulouris et al. (1994), S. 16f.
[13] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Offenheit
[14] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Nebenläufigkeit
[15] Vgl. Byzantinische Probleme in Tel (2000).
[16] Coulouris et al. (1994), S. 17ff.

Fehlertoleranz

Für den Entwurf fehlertoleranter, verteilter Systeme existieren zwei sich ergänzende Ansätze. Ein Ansatz ist die Bereitstellung *redundanter Hardware*, der andere der Einsatz von *Software*, die in der Lage ist sich von fehlerhaften Zuständen zu „erholen“ (*Software Recovery*) und ihre *persistenten* Daten in einen *konsistenten* Zustand zurückführt[17]. Beide Ansätze sind simultan zu verfolgen, wenn das Gesamtsystem als fehlertolerant gelten soll.

In verteilten Systemen ergibt sich ein Aspekt der Fehlertoleranz auf *Hardware-Ebene* natürlich, wenn *logische Server* und deren *Server-Prozesse* portabel (mobil), also nicht an feste *physikalische Server* gebunden sind oder logische *Server* redundant auf verschiedenen physikalischen *Servern* im Gesamtsystem angeordnet sind. Die Entwurfsziele Nebenläufigkeit, Skalierbarkeit[18] und Fehlertoleranz wirken in dieser Hinsicht komplementär. Der zweite Aspekt der Fehlertoleranz auf *Hardware-Ebene* sind verlässliche und fehlertolerante Kommunikationskanäle.[19]

3.2 Domänenkooperation als Kooperation verteilter Dienste

Im Abschnitt 3.1.1 wurde bereits argumentiert, dass die Aufteilung von *Asset-Modellen* und damit *Adaptions- und Migrationslasten* auf die Knoten eines *verteilten Systems* Vorteile gegenüber einem *monolithischen* begriffsorientierten Enwurf hat, dass dabei jedoch die entstehenden *Kommunikationskosten* berücksichtigt werden müssen. Die konkrete Verteilung von *Domänen* und *Asset-Modellen* der *Organisationseinheiten* soll nun im folgenden Abschnitt diskutiert werden.

3.2.1 Domänen als Dienste

In diesem Abschnitt wird zunächst begründet, warum die vorliegende Arbeit das *Client-Server-Modell* für den Entwurf und die Umsetzung einer *verteilten begriffsorientierten Infrastruktur* verwendet. Im zweiten Abschnitt, werden die Auswirkungen dieser Entscheidung für die Aufteilung von *Domänen* und *Asset-Modellen* auf die Infrastruktur diskutiert.

Client-Server- und Objekt-Modell

Für die Modellierung von verteilten Systemen existieren nach Coulouris[20] zwei konkurrierende Modelle. Es handelt sich um das *Client-Server-Modell* und das *Objekt-Modell*. Die Unterschied zwischen beiden Modellen liegt in der Verwaltung von *Ressourcen* und *Verhalten*. Im Objekt-Modell, werden ähnlich dem Ansatz in der objektorientierten Programmierung, die verteilten Ressourcen gemeinsam mit den Schnittstellen, die den Zugriff auf die Ressourcen definieren, als eine Einheiten (Objekte) verwaltet. Die Objekte können zudem *eindeutig* und *unabhängig* von ihrem *Aufenthaltsort identifiziert* werden. Folglich unterstützt das Objekt-

[17] Vgl. Transaktionen in Lockemann et al. (1987).
[18] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit
[19] Coulouris et al. (1994), S. 19f.
[20] Coulouris et al. (1994), S. 11ff.

Modell die *transparente Migration*[21] von Objekten. Die Hauptschwierigkeit des Modells liegt darin begriffen, dass mit der Migration von Objekten auch die zugehörigen *Objekt-Manager* migrieren müssen, die für die Implementation der Schnittstellen zuständig sind.

In der vorliegenden Arbeit, die auf dem *begriffsorientierten Modell* basiert, ist ein weiteres Hindernis für das Objekt-Modell zu nennen. Die verwalteten *Assets* können nur im Kontext ihres jeweiligen *Asset-Modells*, das eine individuelle Sinnstruktur widerspiegelt, sinnvoll interpretiert werden. Bei der freien Migration von *Assets* müssen die durch den *Asset-Modell-Compiler* generierten *Module*[22], als Manifestation des *Asset-Modells*, ebenfalls migrieren. Dies schliesst die, je nach *Geschwindigkeit der Erkenntnisprozesse* und *Verzweigungsgrad* in der *Organisationsstruktur*, erhebliche Zahl von *Modelladaptionsmodulen* mit ein. Zusätzlich sind Teile des *Asset-Modells dynamisch* oder *extensional* definiert[23], was die Migration einer großen Zahl von *Assets* nach sich ziehen kann. Zu guter letzt sind, bei konsequenter Umsetzung des Objekt-Modells, je nach abgebildeter *Problemdomäne*, erhebliche Datenmengen an multimedialen Inhalten zu migrieren.

Eine denkbare Alternative zu dieser Vorgehensweise stellt eine Abkehr von der bisherigen Praxis dar, *Adaptionsmodule* „nur“ zum Zweck der *Personalisierung* und *Publikation* zwischen *Organisationseinheiten* einer *Domäne* sowie zur Unterstützung des Erkenntnisprozesses einer Organisationseinheit zu unterhalten. Statt dessen werden Adaptionsmodule zu allen, im verteilten System existierenden, *Asset-Modellen* erzeugt, um *potentielle* Migrationen zu ermöglichen. Abgesehen vom *exponentiellen Wachstum* der Zahl der Adaptoren, das diese Vorgehensweise technisch undurchführbar macht, bereitet auch die Definition der *Adaptionspfade* erhebliche interpretatorische Schwierigkeiten, besonders wenn Domänengrenzen überschritten werden.

Stattdessen verwendet die vorliegende Arbeit das Client-Server-Modell für den Entwurf und die Implementation der verteilten begriffsorientierten Infrastruktur. Dieses Modell ist laut Coulouris das am besten verstandene und am häufigsten eingesetzte Modell für verteilte Systeme. Es besteht aus einer Menge von *logischen Servern*, die jeder für sich als *Ressourcen-Manager* fungieren, und einer Menge von *logischen Clients*, die über *Anfragen* (*Requests*) an logischer *Server* auf verteilte Ressourcen zugreifen. Logische *Server* können, im Rahmen der Erstellung einer *Antwort* (*Response*) auf eine Anfrage, wiederum auf logische *Server* zugreifen und nehmen dann ebenfalls die Rolle von logischen *Clients* ein.

Ein *Ressourcen-Manager* ist für die Verwaltung von Ressourcen eines bestimmten *Typs* zuständig. *Ressourcen-Manager* sind das Pendant zu *Objekt-Managern* im Objekt-Modell, mit dem wesentlichen Unterschied, dass *Ressourcen-Manager* nicht migrieren, sondern an logische *Server* gebunden sind, die wiederum auf *physikalischer Servern* ablaufen. Damit sind logische *Server* der exklusive Ort für die Verwaltung von Ressourcen in einem *Client-Server-System*.

Nach den bisherigen Ausführungen stellen sich *Ressourcen-Manager* und deren *logischer Server* als *zentrale Leistungserbringer* für Ressourcen eines bestimmten Typs dar. Zentrale Elemente sind, aus Gründen der *Skalierbarkeit*[24] und der *Fehlertoleranz*[25], in verteilten Systemen jedoch unerwünscht. Zu diesem Zweck werden *Dienste* als Menge von *Ressourcen-Manager* (logische *Server*) eingeführt, die gemeinsam für Ressourcen eines Typs zuständig

[21] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme - Transparenz
[22] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems
[23] Sehring (2004), S. 57ff.
[24] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit
[25] Vgl. Abschnitt 101, Ausgewählte Entwurfsziele verteilter Systeme - Fehlertoleranz

sind. Coulouris führt dazu aus:

> „[...] It is for this reason that a distinction is made between the *services* that are provided to clients an the servers that provide them. A service is considered to be an abstract entity that may be provided by serveral server processes running on separate computers and cooperating via the network.“
> Quelle: Coulouris et al. (1994, S. 13)

Aufteilung von Domänen und Asset-Modellen

Nach der Entscheidung zugunsten eines verteilten Systems in Abschnitt 3.1.1 und der Wahl des *Client-Server-Modells* für den Entwurf und die Implementation im vorgegangenen Abschnitt, bleibt die Frage zu klären, wie die Domänen und Modelle der begriffsorientierten Infrastruktur vorteilhaft auf Dienste und logische *Server* verteilt werden können. Zur Bedeutung des Begriffs „vorteilhaft“ sei auf die nachfolgende Aufstellung von Zielen verwiesen, deren lineare Ordnung vom Verfasser dieser Arbeit aufgrund subjektiver Einschätzungen festgelegt wurde. Die gewählte Ordnung ist vor dem Verlassen der *prototypischen Implementationsphase*, beispielsweise durch *Benutzerbefragungen*, zu fundieren und gegebenenfalls anzupassen, woraus dann auch Änderungen im Entwurf und der Eintritt in einen neuen *Entwicklungszyklus* resultieren können. Die meisten der aufgeführten Ziele sind allgemeine Entwurfsziele verteilter Systeme, deren genaue Bedeutung Abschnitt 3.1.2 entnommen werden kann.

1. Anwortzeiten unterhalb der Frustrationsschwelle
2. maximale Nebenläufigkeit
3. Skalierbarkeit
4. Offenheit
5. Fehlertoleranz
6. beherrschbare Komplexität
7. Eleganz durch uniforme logische *Server*
8. Transparenz

Um die *Antwortzeiten* für die Benutzer der begriffsorientierten Infrastruktur optimal zu gestalten, sind verschiedene Aspekte zu beachten. Zum einen muss die *Auslastung* eines physikalischen *Servers* und seiner logischen *Server* aufgrund parallel zu betreibender *Asset-Modelle* und deren Infrastruktur, inklusive der *Asset-Modelladaptoren*, in Betracht gezogen werden. Aus diesem Blickwinkel führt *maximale Verteilung*, also die Bereitstellung von einem physikalischen *Server* für einen logischen *Server* für ein individuelles *Asset-Modell*, zur geringsten Auslastung. Ob damit auch das Ziel der optimalen Antwortzeiten erreicht wird, ist jedoch fraglich. Denn bei dieser Aufteilung des verteilten Systems entstehen auch *maximale Kommunikationskosten*, die nur teilweise durch *Replikation* aufgefangen werden können.

Ein Beispiel für diesen Sachverhalt ist die Kommunikation zwischen den *Asset-Modellen* einer *Organisationseinheit*, die im Zuge der inkrementellen Modellevolution entstanden sind.

Für jeden Evolutionsschritt wird nun ein logischer *Server* mit der nötigen Modellinfrastruktur, insbesondere dem *Asset-Modelladapter*, betrieben. Die Kommunikation zwischen den *Servern* erfolgt, wie die Kommunikation mit logischen *Servern* anderer Organisationseinheiten und *Domänen*, über *externe Kanäle*. Je nach implementierter *Migrationsstrategie* und Geschwindigkeit des *Erkenntnisprozesses* der Organisationseinheit, bildet sich nun eine mittlere *Kommunikationskette*, deren erhöhte *Kommunikationskosten* sich gegenüber der *serverinternen* Kommunikation multiplizieren. Zudem ist jeder Zwischenschritt mit *Marshalling-* und *Unmarshalling-Kosten* belastet. Die *Replikation* der übertragenen *Assets* oder gar deren *flacher Beschreibungen* ist sinnlos, weil nach einmaligem Zugriff auf ein älteres *Asset* deren Migration in das aktuelle *Asset-Modell* erfolgt.

Aus den vorangegangenen Ausführungen ist klar geworden, das eine zu feine *Granularität*, für die Verteilung der begriffsorientierten Infrastruktur, bezüglich der Antwortzeiten des Systems Nachteile bringt. Die gesteigerte Nebenläufigkeit wird durch erhöhte Kommunikationskosten mehr als wettgemacht. Die nächste mögliche Granularität, die es zu untersuchen gilt, ist die Bereitstellung von einem physikalischen *Server* für einen logischen *Server* für einen *Kontext* einer Organisationseinheit. In diesem Szenario hat die geschilderte mittlere Kommunikationskette für die *Asset-Migration* zwar die gleiche Länge wie im ersten Szenario, jedoch fallen pro Schritt nur die *server-internen* Kommunikationskosten an, während die Kosten für *Marshalling* und *Unmarshalling* vollständig entfallen.

Bei der Kommunikation mit logischen *Servern* anderer Organisationseinheiten der gleichen Domäne[26] oder anderer Domänen[27], kann, je nach *Verzweigungsgrad* der *Organisationsstrukturen*, ebenfalls eine längere mittlere Kommunikationskette entstehen. In dieser sind, wegen der externen Kommunikation, zudem wieder alle Zwischenschritte mit den *Marshalling-Kosten* belastet. Der wesentliche Unterschied zum ersten Szenario ist jedoch die durchschnittliche Länge der Kette; der mittlere Verzweigungsgrad der Organisationsstruktur dürfte erheblich unter der mittleren Zahl der parallelen *Asset-Modelle* liegen, die durch *Evolutions- und Migrationsgeschwindigkeit* bestimmt wird. Zudem können beide Formen der Kommunikation durch *Replikation* und *Notifikation* entschärft werden. Dabei ist besonders das Zwischenspeichern möglichst kleiner Granulare flacher *Werkbeschreibungen* vorteilhaft[28]. Der Grund sind die lokal sehr begrenzten Auswirkungen der *Invalidierung* von Replikaten bei der Komposition von flachen, zusammengesetzten *Asset-* und Werkbeschreibungen. Zudem werden die *Marshalling-Kosten* eingespart. Werkbeschreibungen werden in Abschnitt 3.3 genauer erläutert.

Das zuletzt genannte Szenario scheint, aufgrund der dargestellten Argumente, die richtige Balance zwischen Auslastung (*Nebenläufigkeit*) und Kommunikationskosten zu finden, um das primäre Ziel der angemessenen Anwortzeiten zu unterstützen. Als Erweiterung zu den bisherigen Ausführungen, wird die Gesamtheit der logischen *Server* der Organisationseinheiten einer Domäne zu Diensten zusammengefasst. Diese Abbildung entspricht zwar nicht exakt dem *Client-Server-Modell*, in dem *Ressourcen-Manager* und damit logische *Server* die Ressourcen eines *Typs* verwalten, aber zwischen den Mitglieder einer Domäne besteht zumindest eine Gemeinsamkeit im *Anwendungsfokus*. Außerdem kann über *Adaptionsmodule* für Teile der Domäne ein gemeinsamer Typ hergeleitet werden. Die konkrete Aufteilung erfolgt entsprechend:

[26] Vgl. Abschnitt 2.3.2, Organisationssprezifische Prozesse
[27] Vgl. Abschnitt 2.3.3, Anwendungsspezifische Prozesse
[28] Vgl. Abschnitt 4.3.2, Replikation; Vgl. Abschnitt 5.2, Bewertung - Transparenz - Transparente Replikation

- Ein physikalischer *Server* pro logischem *Server*
- Ein logischer *Server* pro Kontext einer Organisationseinheit.
- Ein Dienst pro Domäne

Diese Aufteilung wird in nachfolgenden Arbeit als Grundlage der Verteilung genutzt. Im Abschnitt 3.5 werden *logische Server* dann mit dem Terminus *Komponenten* benannt, die sich aus einer Vielzahl von *Modulen* zusammensetzen, die wiederum der Unterstützung individueller *Asset-Modelle* dienen.

3.2.2 Basisdienste für den Systembetrieb

Die Verwaltung der *Inhalte* von *Basisdomänen* durch das *begriffsorientierte Modell* war bereits mehrfach Gegenstand der vorliegenden Arbeit[29]. Hier können nun zwei Gruppen von *Diensten* zur Erbringung der Leistungen dieser Domänen unterschieden werden. Die erste Gruppe beinhaltet *Basisdienste*, die keine Anforderungen des verteilten Entwurfes darstellen, sondern der Unterstützung *kooperativen begriffsorientierten Arbeitens* dienen. Es handelt sich dabei um den *Benutzer- und Gruppenverwaltungsdienst*, den *Rechte- und Rollenverwaltungsdienst* sowie den *Werkverwaltungsdienst*. Die zweite Gruppe umfasst Dienste, die zur Erreichung allgemeiner Entwurfsziele *verteilter Systeme* beitragen[30]. Es sind dies der *Namens- und Verzeichnisverwaltungsdienst* sowie der *Nachrichtenverwaltungdienst*.

In diesem und den nachfolgenden Abschnitten werden die Anforderungen an die genannten Basisdienste, in Form von *Anwendungsfalldiagrammen* (*Use Cases*), illustriert[31]. Die Verwaltung aller Dienste in einem einheitlichen *Inhaltsmodell* führt naturgemäß zur *redundanten* Anwendungsfällen in diesen Diagrammen. Deshalb werden die Fälle, die sich direkt auf die schematische *Asset-Schnittstelle*, die vom *Asset-Modell-Compiler* einheitlich für alle *Asset-Modelle* generiert wird[32], abbilden lassen, im Diagramm 3.1 als *elementare Grundfälle* zusammengefasst. Es sind dies Anwendungsfälle zur Erzeugung, Abfrage, Manipulation und Zerstörung von *Assets*. Die Anwendungsfälle zur Abfrage und Manipulation bedienen sich wiederum anderer Fälle für den lesenden und schreibenden Zugriff auf Charakteristika und Beziehungen von *Assets*. Die übrigen Anwendungsfälle werden in den folgenden Abschnitten unter der Überschrift ihres jeweiligen Basisdienstes behandelt. Sie dienen später u.a. zur Herleitung von *Sprachkonzepten* der *präskriptiven Werkbeschreibungsprache*[33]. Diese erhebt den Anspruch, alle *nichtelementaren* Fälle durch Kombination von elementaren Grundfällen mit Ausdrücken der Sprache in Form von Werkbeschreibungen abbilden zu können.

Ein Beispiel für eine Eigenschaft eines Modell, die sich weder in einem Klassendiagramm noch in der *Asset-Definitionssprache* nach Sehring[34], die nachfolgend zur Formulierung der *Asset-Modelle* genutzt wird, ausdrücken und sicherstellen läßt, ist die Forderung nach einem gerichteten azyklischen Graphen für die Benutzer- und Gruppenstruktur bei Bachmann[35].

[29] Vgl. Abschnitte 2.2.1, Objektdomänen; 2.2.6, Benutzer und Gruppen
[30] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme
[31] Fowler et al. (1997), S. 53.
[32] Sehring (2004), S. 101ff.
[33] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen; Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse
[34] Sehring (2004), S. 39ff.
[35] Bachmann (2003), S. 46.

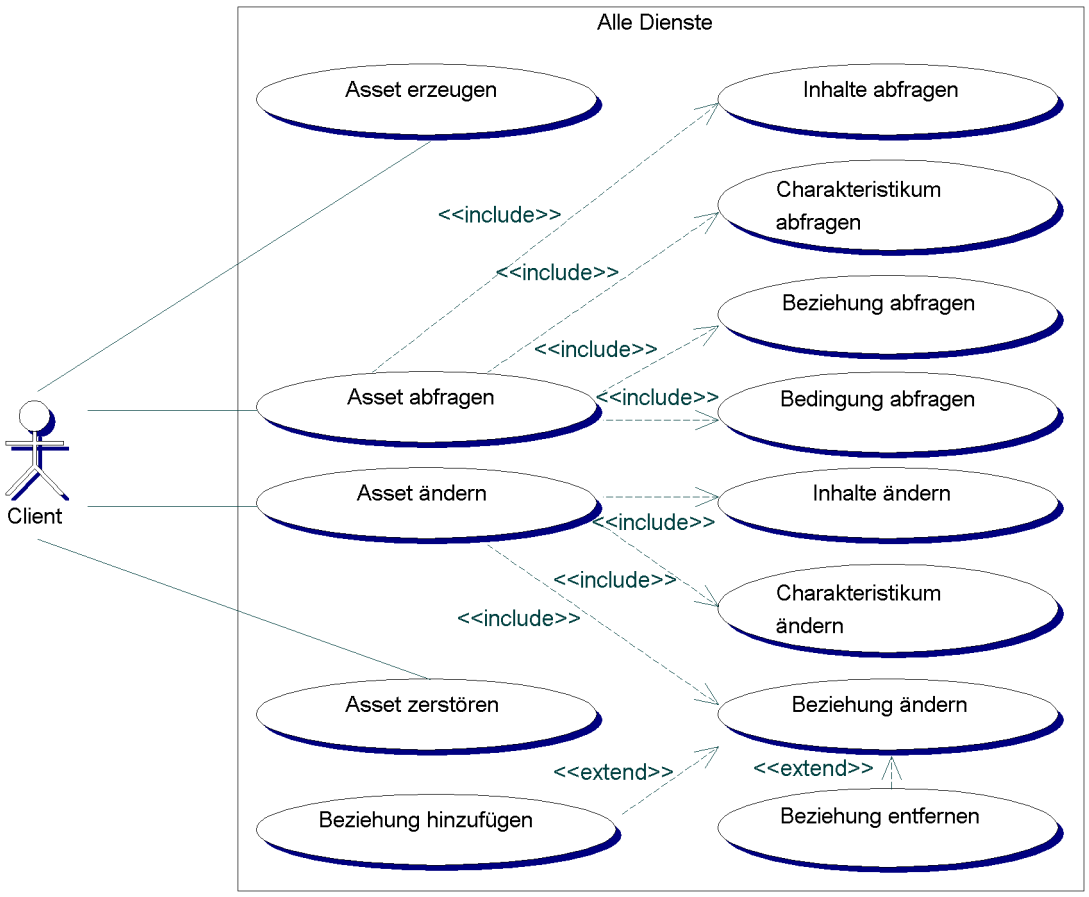

Abbildung 3.1: Elementare Anwendungsfälle der Basisdienste.

Diese Bedingung kann lediglich bei der Erzeugung neuer Beziehungen zwischen `Gruppen` und `Benutzern`, in einer Werkbeschreibung, geprüft und aufrecht erhalten werden. Entsprechend existieren für diese Vorgänge nichtelementare Anwendungsfälle in Abschnitt 3.2.3.

In Abbildung 3.2 werden die *Entitäten* aller Basisdienste sowie ihre Beziehungen untereinander in einem *schematischen Klassendiagramm*, als Gesamtbild der begriffsorientierten Infrastruktur, dargestellt. Das Diagramm wurde aus den elementaren und nichtelementaren Anwendungsfällen der jeweiligen Dienste abgeleitet und dient nachfolgend der Erstellung der *Asset-Modelle*, inklusive der *Charakteristika*, *Beziehungen* und *Bedingungen* sowie der Modellierung der Navigierbarkeit der Beziehungen.

Als Folge der Verteilung und Kooperation in der Infrastruktur, entstehen auch sogenannte *Proxy-Assets*, deren zwei Charakteristika einen eindeutigen Verweis auf ein entferntes *Asset* darstellen. Sie werden von domänenübergreifenden Werkbeschreibungen als „Brücken“ zum jeweiligen Partnerdienst genutzt und in der nachfolgenden Aufstellung der Dienste bzw. im jeweiligen Abschnitt des Anhanges anhand ihrer Dienstzugehörigkeit in separate *Asset-Modelle* zusammengefasst.

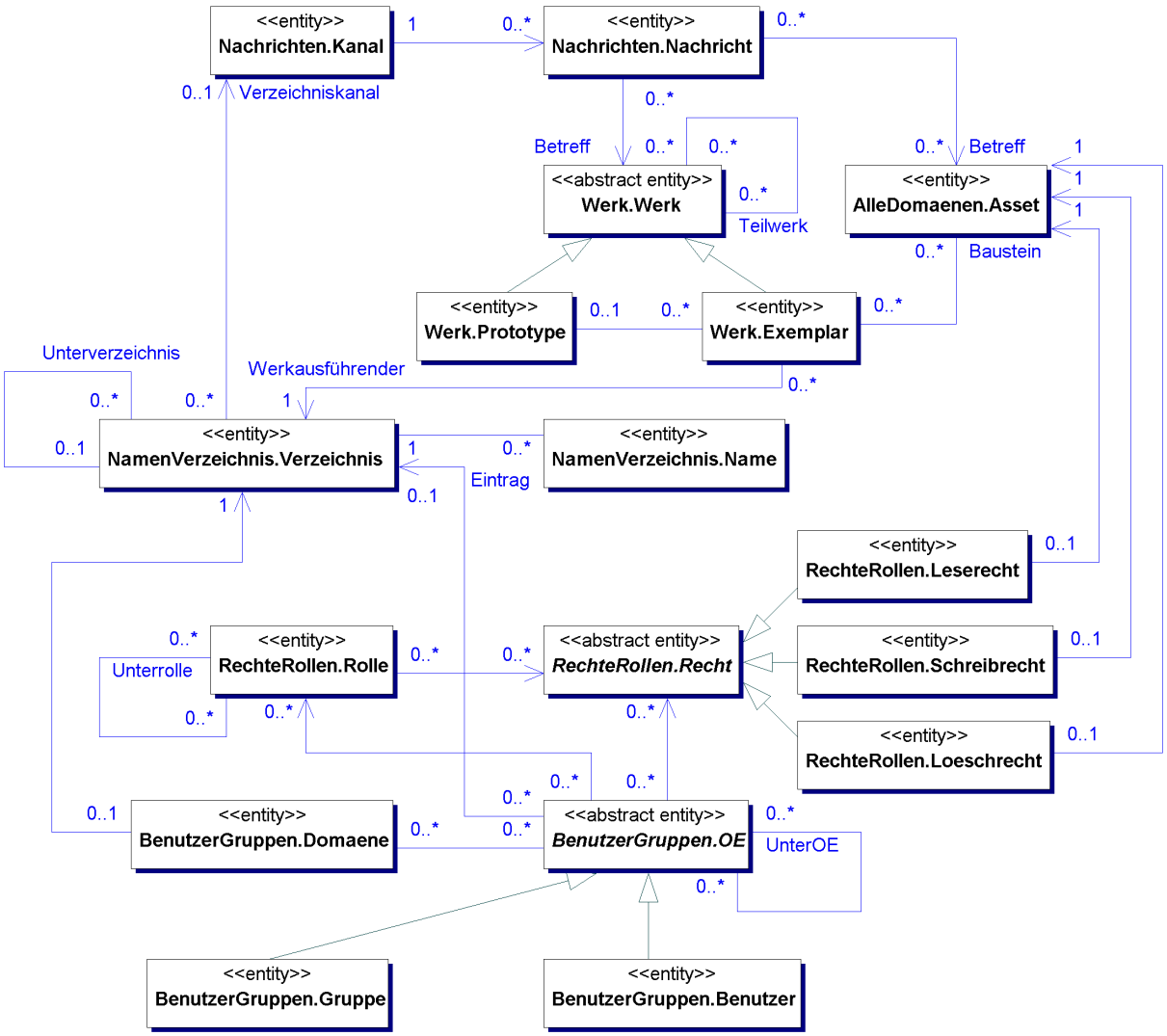

Abbildung 3.2: Entitäten der Basisdienste.

3.2.3 Benutzer- und Gruppenverwaltungsdienst

Der *Benutzer- und Gruppenverwaltungsdienst* bildet die *Organisationsstrukturen* der begriffsorientierten Infrastruktur ab, die für die Prozesse der *Personalisierung* und *Publikation*[36] von größter Bedeutung sind. Die nichtelementaren Anwendungsfälle des Dienstes können Abbildung 3.3 entnommen werden. Sie dienen zunächst der Identifikation der Entitäten mit Charakteristika, Beziehungen und Bedingungen.

Für jede verwaltete `Domäne` kann eine eigene Organisationsstruktur existieren, allerdings ist es auch sinnvoll, dass sich mehrere `Domänen` eine Struktur oder Elemente einer Struktur teilen. Als Beispiel sind hier die Benutzer- und Gruppenverwaltung und die Rechte- und Rollenverwaltung zu nennen. Die zu beschreibenden Entitäten sind demnach `Benutzer`, `Gruppen` und `Domänen`. Deren Beziehungen untereinander sowie mit den Entitäten der *Rechte- und*

[36] Vgl. Abschnitt 2.3.2, Organisationsspezifische Prozesse

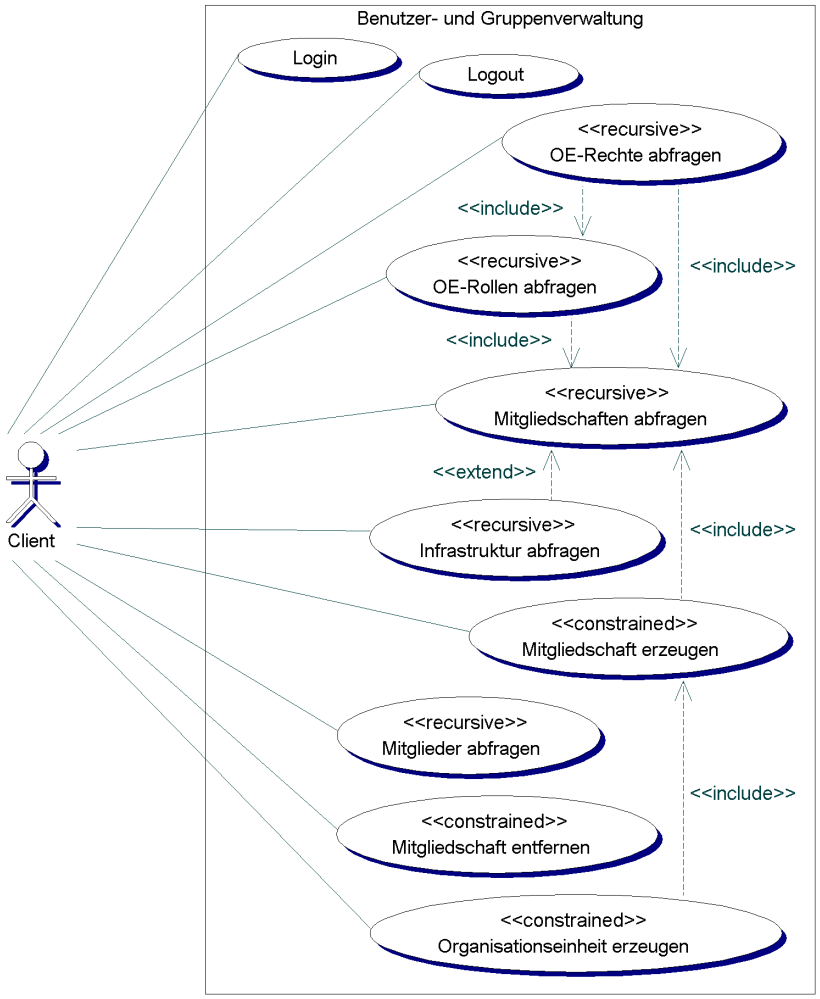

Abbildung 3.3: Anwendungsfälle der Benutzer- und Gruppenverwaltung.

Rollenverwaltung und der *Namens- Verzeichnisverwaltung* werden im Folgenden detailliert betrachtet. Desweiteren werden die Bedingungen, die für Entitäten und Beziehungen gelten, behandelt. Dabei fällt auf, dass sich die Benutzer- und Gruppenentitäten weder in ihren Beziehungen noch in den Bedingungen unterscheiden. Lediglich die Charakteristika weisen erhebliche Unterschiede auf. Demzufolge werden `Benutzer` und `Gruppen` aus einer abstrakten Entität `OE` (*Orgainsationseinheit*) abgeleitet, die die genannten Gemeinsamkeiten enthält. Als Folge können nun auch `Gruppen` `Benutzern` untergeordnet werden, was z.B. in einer Lehrer-Schüler-Beziehung sinnvoll sein kann. Insgesamt ist die Benutzer- und Gruppenverwaltung nach BACHMANN eine hierarchisch azyklische Struktur.

In Abbildung 3.4 werden die *Entitäten* des Benutzer- und Gruppenverwaltungsdienstes sowie ihre Beziehungen untereinander und zu Entitäten anderer Dienste in einem *schematischen Klassendiagramm* dargestellt. Das Diagramm ist ein Ausschnitt der Abbildung 3.2, die die gesamte begriffsorientierte Infrastruktur illustriert. Die zugehörigen *Asset-Modelle* können den Kodeabschnitten 3.1, 3.2 und 3.3 entnommen werden.

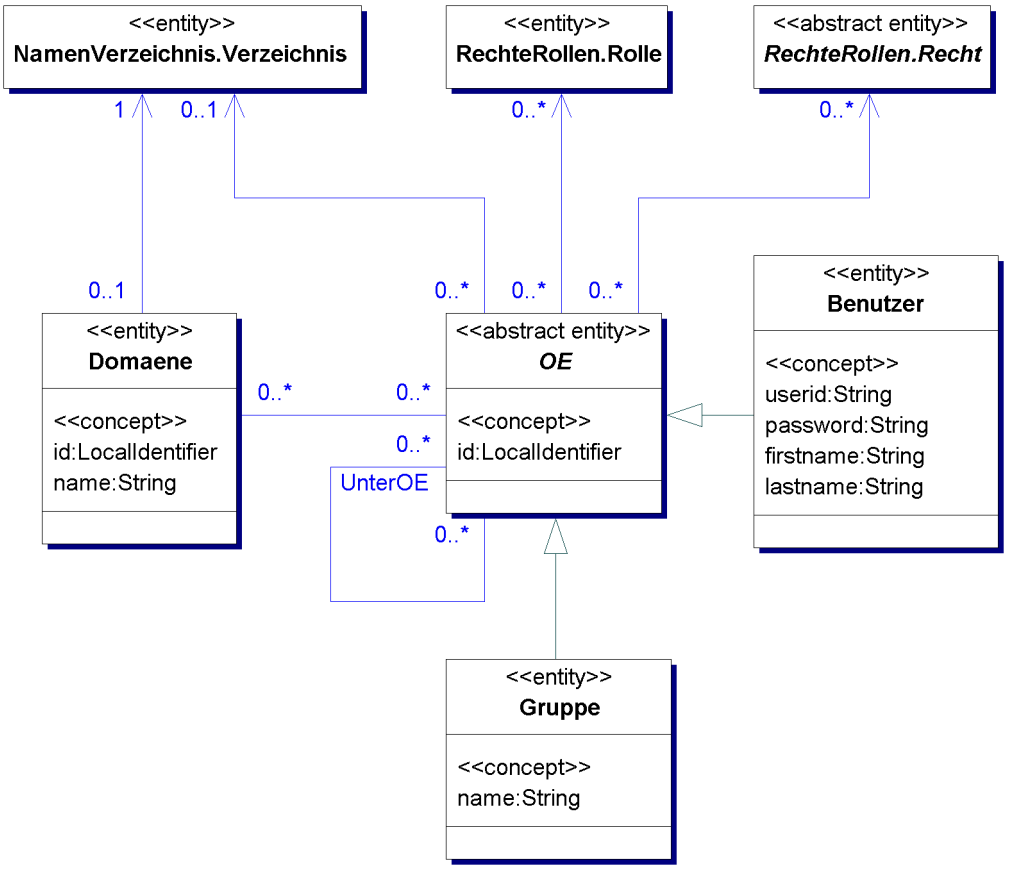

Abbildung 3.4: Entitäten der Benutzer- und Gruppenverwaltung.

- **Bedingung**: `Benutzer` werden über das Charakteristikum `userid` eindeutig identifiziert.
- **UnterOE/OberOE-Beziehung**: Diese Beziehung verbindet `Organisationseinheiten`. Die Rollen sind `UnterOE` und `OberOE`. Eine `Organisationseinheit` kann `UnterOE` von beliebig vielen `OberOEs` sein $(0..n)$. Eine `OberOE` kann beliebig viele `UnterOEs` haben $(0..n)$. Die Beziehung ist in beide Richtungen navigierbar, um den Anwendungsfällen `Mitgliedschaften abfragen` und `Mitglieder abfragen` in Abbildung 3.3 zu genügen.
- **Bedingung**: Die `UnterOE/OberOE`-Beziehung bildet einen gerichteten azyklischen Graphen. Diese Bedingung kann weder in der Abbildung 3.2 noch im nachfolgenden *Asset-Modell* des Benutzer- und Gruppenverwaltungsdienstes beschrieben werden. Sie muss demzufolge durch die nichtelementaren Anwendungsfälle `Organisationseinheit erzeugen` und `Mitgliedschaft erzeugen` sichergestellt werden.
- **Anmerkung**: Der Graph der `UnterOE/OberOE`-Beziehung wird je nach Anwendungsfall mit oder ohne *transitive Hülle* interpretiert. Ein Beispiel für den Verzicht auf die *Hülle* ist die Personalisierung, die nur über direkte Beziehungen möglich ist. Das gegenteilige Beispiel ist die Abfrage der transitiven Verwandschaftsbeziehungen zur Vermeidung der Zyklenbildung, wie sie durch die Anwendungsfälle `Mitgliedschaften abfragen` und

`Mitglieder abfragen` erfolgt. Aus diesem Grund kann keine *Subsumptions-Bedingung* als Teil des *Asset-Modells* definiert werden.

- **OE/Domäne-Beziehung**: Diese Beziehung verbindet `Organisationseinheiten` und `Domänen`. Eine `Organisationseinheit` kann zu beliebig vielen `Domänen` gehören $(0..n)$. Ein `Domäne` kann beliebig viele `Organisationseinheiten` enthalten $(0..n)$. Die Beziehung ist aufgrund der Anwendungsfälle `Infrastruktur abfragen` und `Domänenbenutzer abfragen` (elementar) in beide Richtungen navigierbar.

- **Bedingung**: Die `UnterOE/OberOE`-Beziehung darf nur zwischen `Organisationseinheiten` hergestellt werden, die eine gemeinsame `Domäne` besitzen. Diese Bedingung wird durch das *Asset-Modell* des Benutzer- und Gruppenverwaltungsdienstes sichergestellt.

- **OE/Recht-Beziehung**: Diese Beziehung verbindet `Organisationseinheiten` und *Rechte-Proxies*. Eine `Organisationseinheit` kann beliebig viele `Rechte` haben $(0..n)$. Ein `Recht` kann beliebig vielen `Organisationseinheiten` erteilt worden sein $(0..n)$. Die Beziehung ist in Richtung der `Rechte` navigierbar.

- **OE/Rolle-Beziehung**: Diese Beziehung verbindet `Organisationseinheiten` und *Rollen-Proxies*. Eine `Organisationseinheit` kann beliebig viele `Rollen` innehaben $(0..n)$. EinE `Rolle` kann beliebig vielen `Organisationseinheiten` zugewiesen sein $(0..n)$. Die Beziehung ist in Richtung der `Rollen` navigierbar.

- **Anmerkung**: Die `OE/Rechte`- und die `OE/Rollen`-Beziehung werden je nach Anwendungsfall mit oder ohne *transitive Hülle* interpretiert. Beispiel für den Verzicht auf die *Hülle* ist der Entzug von `Rechten` und `Rollen`, der immer nur für direkt zugeordnete Elemente erfolgt. Gegenteilige Beispiele sind die Anwendungsfälle `OE-Rechte abfragen` und `OE-Rollen abfragen` die zur Ermittlung der Gesamtheit aller zugeordneten Elemente eingesetzt werden. Desweiteren würde die Festlegung von *Subsumptions-Bedingungen* die Entscheidung für *ko- oder kontravariante* Kooperation[37] zwischen Diensten unnötig fixieren. Deshalb werden derartige Bedingungen nicht als Teil des *Asset-Modells* definiert.

- **OE/Verzeichnis-Beziehung**: Diese Beziehung verbindet `Organisationseinheiten` und *Verzeichniss-Proxies*. Eine `Organisationseinheit` kann ein `Verzeichnis` haben (0..1). Ein `Verzeichnis` kann zu beliebig vielen Organisationseinheiten gehören$(0..n)$. Die Beziehung ist in Richtung der `Verzeichnisse` navigierbar.

- **Bedingung**: Auf jedem Pfad der Organisationsstruktur, gebildet durch die `UnterOE/OberOE`-Beziehung, der bei einer beliebigen `Organisationseinheit` beginnt und bei einer `Organisationseinheit` ohne `OberOE` endet, muss mindestens eine `Organisationseinheit` ein `Verzeichnis` besitzen. Diese Bedingung kann weder in der Abbildung 3.2 noch im nachfolgenden *Asset-Modell* des Benutzer- und Gruppenverwaltungsdienstes beschrieben werden. Sie muss demzufolge durch die nichtelementaren Anwendungsfälle `Organisationseinheit erzeugen` und `Mitgliedschaft entfernen` sichergestellt werden.

- **Domäne/Verzeichnis-Beziehung**: Diese Beziehung verbindet `Domänen` und *Verzeichnis-Proxies*. Eine `Domäne` hat genau ein `Verzeichnis` (1). Ein `Verzeichnis`

[37] Sehring (2004), S. 177ff.

kann zu einer `Domäne` gehören (0..1). Die Beziehung ist in Richtung der `Verzeichnisse` navigierbar.

Nachdem nun die Entitäten der Benutzer- und Gruppenverwaltung, einschließlich ihrer Beziehungen und Bedingungen, detailliert betrachtet wurden, gilt es das entsprechende *Asset-Modell* in der *Asset-Definitionssprache* zu formulieren. Dabei sind auch die beiden *Proxy-Asset-Modelle* zu berücksichtigen. In diesen werden Verweise auf die entfernten `Rechte`, `Rollen` und `Verzeichnisse` in Form von *eindeutig*, in der *Infrastruktur* der jeweiligen Organisationseinheit, aufzulösenden *Asset-Charakteristika* verwaltet. Die *Auflösung* erfolgt mit Hilfe des Namens- und Verzeichnisverwaltungsdienstes[38].

Listing 3.1: *Asset-Modell* der Benutzer- und Gruppenverwaltung

```
model UserGroupManagement

from RightRoleMangement import Right, Role
from NameDirectoryManagement import Directory

class OE {

    concept characteristic id   : LocalIdentifier

            relationship superOEs   : OE*
            relationship subOEs     : OE*
            relationship domains    : Domain*
            relationship rights     : Right*
            relationship roles      : Role*
            relationship directory  : Directory

            ; *** uniqueness
            contraint lookfor OE { id = self.id } = { self }

            ; *** inversion
            constraint subOEs.superOEs >= { self }
                onviolation update subOEs { superOEs += self }

            ; *** others
            constraint (superOEs.domains - self.domains) # na

}

class User refines OE {

    concept characteristic userid       : String
            characteristic password     : String
            characteristic firstname    : String
            characteristic lastname     : String

            ; *** uniqueness
            contraint lookfor User { userid = self.userid } = { self }

}

class Group refines OE {

    concept characteristic name : String

}
```

[38] Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltungsdienst

```
class Domain {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship domainOEs  : OE*
            relationship directory  : Directory

            ; *** uniqueness
            constraint lookfor Domain { id = self.id } = { self }

            ; *** inversion
            constraint domainOEs.domains >= { self }
                onviolation update domainOEs { domains += self }

            ; *** cardinality
            constraint directory # na

            constraint lookfor Domain { directory = self.directory } = { self }

}
```

Listing 3.2: *Proxy-Asset-Modell* der Rechte- und Rollenverwaltung

```
model RightRolleManagement ; this is only a proxy model

class Right {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Right {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}

class Role {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Role {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}
```

Listing 3.3: *Proxy-Asset-Modell* der Namens- und Verzeichnisverwaltung

```
model NameDirectoryManagement ; this is only a proxy model

class Directory {
```

```
    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Directory {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}
```

Für die Umsetzung der nichtelementaren Anwendungsfälle in der *präskriptiven Werkbeschreibungssprache*[39] lassen sich bereits an dieser Stelle *Sprachkonzepte* identifizieren, die, wegen der diskutierten Struktur des Benutzer- und Gruppenverwaltung, unverzichtbarer Bestandteil der Sprache sind. Es handelt sich u.a. um die *Rekursion* und die *Mengen-Semantik*, die beide zur *Traversion* des Graphen der Organisationsstruktur benötigt werden.

3.2.4 Rechte- und Rollenverwaltungsdienst

Der *Rechte- und Rollenverwaltungsdienst* ist für die Vergabe von Rechten für den Zugriff auf *Assets* verantwortlich. Die nichtelementaren Anwendungsfälle des Dienstes können Abbildung 3.5 entnommen werden. Sie dienen zunächst der Identifikation der Entitäten mit Charakteristika, Beziehungen und Bedingungen.

Es können drei Arten von `Rechten`, nämlich `Lese-, Schreib- und Löschrechte`, für ein `Asset` festgelegt werden. Jedes dieser Rechte kann pro `Asset` höchstens ein mal vergeben werden. Weiterhin können `Rechte` zu `Rollen` zusammengefasst werden, die wiederum zu übergeordneten `Rollen` vereinigt werden können. Die zu beschreibenden Entitäten sind demnach `Leserecht`, `Schreibrecht`, `Löschrecht`, die aus der abstrakten Entität `Recht` abgeleitet sind, sowie `Rollen`. Deren Beziehungen untereinander sowie mit den Entitäten beliebiger anderer Domänen werden im Folgenden detailliert betrachtet. Desweiteren werden die Bedingungen, die für Entitäten und Beziehungen gelten, behandelt. Die Zuordnung von `Rechten` und `Rollen` zu `Organisationseinheiten` erfolgt, aufgrund der ermittelten Navigationsrichtung für diese Beziehungen, im zuvor beschriebenen Benutzer- und Gruppenverwaltungsdienst. Insgesamt ist die Rechte- und Rollenverwaltung, aus Flexibilitätsgründen eine gerichtete azyklische Struktur. Die weitere Flexibilisierung, also der Fortfall der Beschränkung auf eine azyklischen Struktur, würde jedoch unverhältnismäßige Probleme mit sich bringen. Rekursive Algorithmen zur Traversion der Struktur müssten dann mit Mechanismen zum Erkennen und Verlassen von Zyklen ausgestattet sein.

In Abbildung 3.6 werden die *Entitäten* des Rechte- und Rollenverwaltungsdienstes sowie ihre Beziehungen untereinander und zu Entitäten anderer Dienste in einem *schematischen Klassendiagramm* dargestellt. Das Diagramm ist ein Ausschnitt der Abbildung 3.2, die die gesamte begriffsorientierte Infrastruktur illustriert. Die zugehörigen *Asset-Modelle* können den Kodeabschnitten A.4 und A.5 im Anhang entnommen werden.

[39] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen; Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse

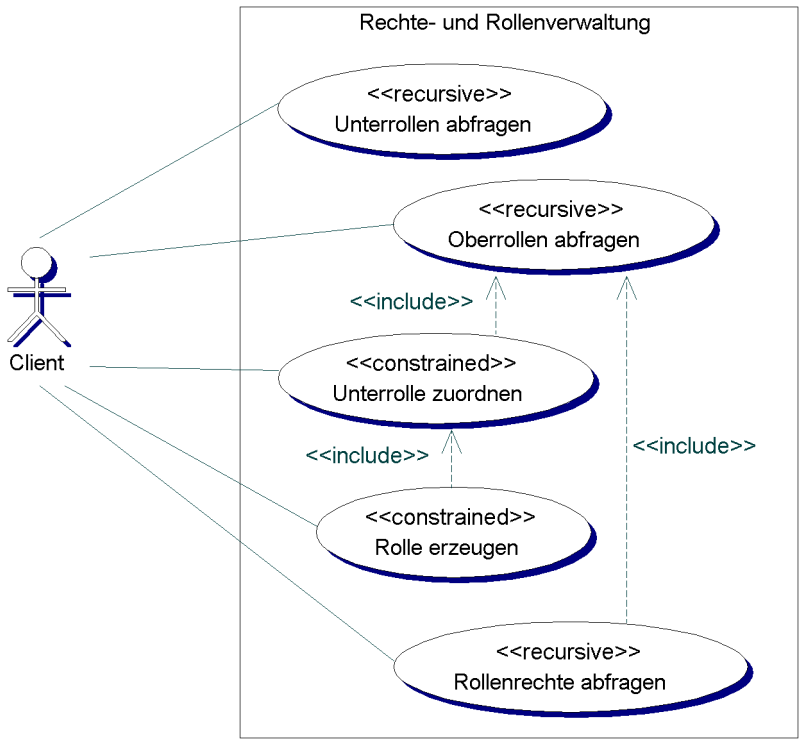

Abbildung 3.5: Anwendungsfälle der Rechte- und Rollenverwaltung.

- **(Lese-, Schreib-, Lösch-)recht/Asset-Beziehung**: Diese Beziehung verbindet `Rechte` und *Asset-Proxies*. Ein `Recht` einer Art bezieht sich auf genau ein `Asset` (1). Einem `Asset` kann höchstens ein `Recht` einer Art zugewiesen werden (0..1). Die Beziehung ist in Richtung der `Assets` navigierbar.

- **Recht/Rolle-Beziehung**: Diese Beziehung verbindet `Rechte` und `Rollen`. Ein `Recht` kann zu beliebig vielen `Rollen` gehören ($0..n$). Eine `Rolle` kann beliebig viele Rechte umfassen ($0..n$). Die Beziehung ist in Richtung der `Rechte` navigierbar.

- **Anmerkung**: Der Graph der `Recht/Rolle`-Beziehung wird je nach Anwendungsfall mit oder ohne *transitive Hülle* interpretiert. Ein Beispiel für den Verzicht auf die *Hülle* ist die Entfernung von `Rechten` aus `Rollen`, die nur für direkt zugeordnete `Rechte` erfolgt. Das gegenteilige Beispiel ist die Abfrage der Gesamtheit der `Rechte` einer `Rolle`, wie sie durch den Anwendungsfall `Rollenrechte abfragen` erfolgt. Aus diesem Grund kann keine *Subsumptions-Bedingung* als Teil des *Asset-Modells* definiert werden.

- **Unterrolle/Oberrolle-Beziehung**: Diese Beziehung verbindet `Rollen`. Die Rollen sind `Unterrolle` und `Oberrolle`. Eine `Rolle` kann `Unterrolle` von beliebig vielen `Oberrollen` sein ($0..n$). Eine `Oberrolle` kann beliebig viele `Unterrollen` haben ($0..n$). Die Beziehung ist in beide Richtungen navigierbar, um den Anwendungsfällen `Oberrollen abfragen` und `Unterrollen abfragen` in Abbildung 3.5 zu genügen.

- **Bedingung**: Die `Unterrolle/Oberrolle`-Beziehung bildet einen gerichteten azyklischen Graphen. Diese Bedingung kann weder in der Abbildung 3.2 noch im nachfol-

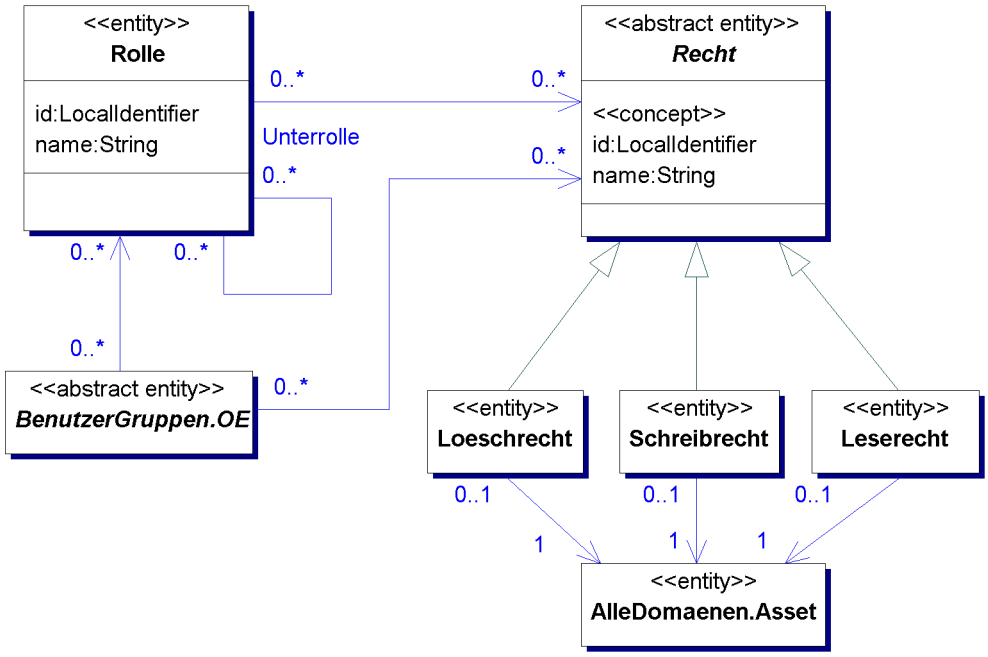

Abbildung 3.6: Entitäten der Rechte- und Rollenverwaltung.

genden *Asset-Modell* des Rechte- und Rollenverwaltungsdienstes beschrieben werden. Sie muss demzufolge durch die nichtelementaren Anwendungsfälle `Rolle erzeugen` und `Unterrolle zuordnen` sichergestellt werden.

- **Anmerkung**: Der Graph der `Unterrolle`/`Oberrolle`-Beziehung wird je nach Anwendungsfall mit oder ohne *transitive Hülle* interpretiert. Ein Beispiel für den Verzicht auf die *Hülle* ist die Entfernung von `Unterrollen` aus `Oberrollen`, die nur für direkt zugeordnete `Unterrollen` erfolgt. Das gegenteilige Beispiel ist die Abfrage der transitiven Rollenbeziehungen zur Vermeidung der Zyklenbildung, wie sie durch die Anwendungsfälle `Oberrollen abfragen` und `Unterrollen abfragen` verwirklicht wird. Aus diesem Grund kann keine *Subsumptions-Bedingung* als Teil des *Asset-Modells* definiert werden.

Nachdem nun die Entitäten der Rechte- und Rollenverwaltung, einschließlich ihrer Beziehungen und Bedingungen, detailliert betrachtet und als Ausschnitt der Abbildung 3.2 dargestellt wurden, gilt es das entsprechende *Asset-Modell* in der *Asset-Definitionssprache* zu formulieren. Dabei ist auch das *Proxy-Asset-Modell* zu berücksichtigen, in dem die Verweise auf entfernte `Assets` verwaltet werden.

Wie bei der zuvor beschriebenen Benutzer- und Gruppenverwaltung, werden auch bei der Rechte- und Rollenverwaltung *Rekursion* und *Mengen-Semantik* als unverzichtbare Bestandteile der *präskriptiven Werkbeschreibungssprache*[40] identifiziert, um die nichtelementaren Anwendungsfälle ausdrücken zu können.

[40] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen; Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse

3.2.5 Werkverwaltungsdienst

Der *Werkverwaltungsdienst* ist für die Verwaltung und Überwachung eines *Werkabhängigkeitsgraphen* zuständig, der *Awareness*[41] und *Verklemmungsfreiheit*[42] bei der Delegation von Teilwerken ermöglicht. Darüberhinaus können *Werkbeschreibungen* über den Werkverwaltungsdienst als Inhalte verwaltet (*Persistenz*) und als Prototypen genutzt werden. Die nichtelementaren Anwendungsfälle des Dienstes können Abbildung 3.7 entnommen werden. Sie dienen zunächst der Identifikation der Entitäten mit Charakteristika, Beziehungen und Bedingungen.

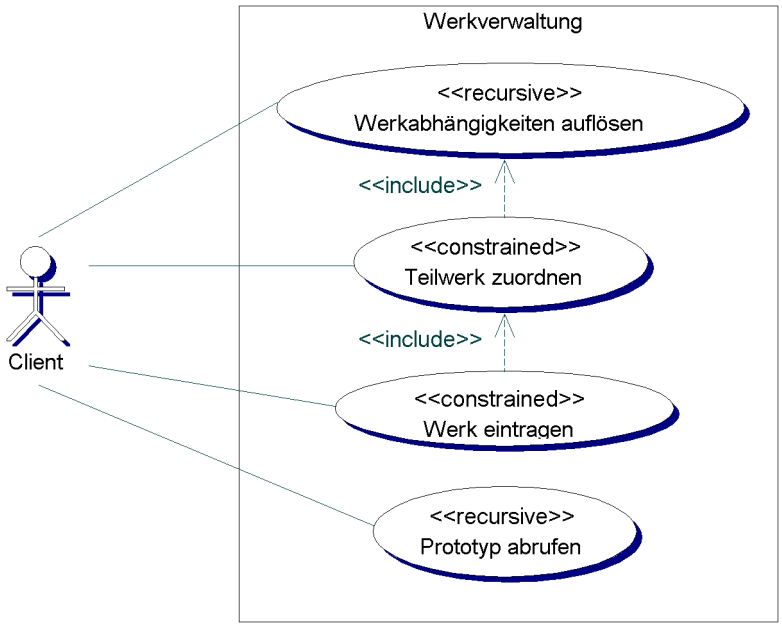

Abbildung 3.7: Anwendungsfälle der Werkverwaltung.

Der Werkabhängigkeitsgraph stellt das zentrale Mittel des Werkverwaltungsdienstes zur Erfüllung seiner Aufgaben dar. Da aus der abstrakten Entität `Werk` die Entitäten `Prototyp` und `Exemplar` abgeleitet werden, kann der Graph zweigeteilt betrachtet werden. Diese Partionierung wird auch durch entsprechende Bedingungen sichergestellt. Der eine Teil beschreibt die Anhängigkeiten zwischen `Prototypen` und dient dazu Werkbeschreibungen wahlweise flach oder tief vom Werkverwaltungsdienst abzurufen. Diese Option wird jedoch in der vorliegenden Implementation nicht genutzt und wird im Ausblick erneut aufgegriffen[43]. Der Graph der `Prototypen` wird nicht auf Verklemmungen überwacht, weil `Prototypen` nicht direkt ausgeführt werden, sondern als Blaupause für `Exemplare` dienen.

Der zweite Teil des Werkabhängigkeitsgraph bildet die Abhängigkeiten zwischen verschachtelten Werkaufrufen (`Teilwerken`)[44] ab. Ergänzt um die Beziehung zwischen *Assets* und `Exemplaren`, können auf dieser Basis Nachrichten zur *Invalidierung* von *Replikaten*[45] ver-

[41] Vgl. Abschnitt 3.4.3, Prozessdelegation mit Awareness und Data Provenence
[42] Vgl. Abschnitt 270, Verklemmungsfreiheit
[43] Vgl. Abschnitt 5.3, Ausblick
[44] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen
[45] Vgl. Abschnitt 4.3.2, Replikation

sandt und *Verklemmungen* detektiert werden. Im verklemmungsfreien Fall bildet diese Struktur einen *gerichteten azyklischen Graphen*, der auch nach dem Einfügen einer neuen Beziehung oder eines neuen `Exemplars` diese Eigenschaft beibehalten muss. Dementsprechend ist vor dem Hinzufügen einer neuen Kante, die *transitive Hülle* des (zukünftig) übergeordneten `Exemplars` zu berechnen. Wenn das (zukünftig) untergeordnete `Exemplar` Element der Hülle ist, dann ist die Bearbeitung abzulehnen.

Die beschriebene Vorgehensweise zur Vermeidung von Verklemmungen sowie der Versand von Nachrichten basiert auf der (systemweit) *eindeutigen Identifikation* von *Exemplaren*. Dieses Ziel ist in *verteilten Systemen* jedoch nicht leicht zu erreichen. Eine Lösungsmöglichkeit stellt die *Komposition* eines *Schlüssels* aus logischer Adresse des *Werkausführenden* und *Hashwert* der *lokal gebundenen*, präskriptiven Werkbeschreibung des `Exemplars`[46] dar. Diese Lösung unterstützt zudem die *Wiederverwendung* von `Exemplaren` sowie den Einsatz von *Replikationsmechanismen*[47] zur Unterstützung von *Skalierbarkeit* und *Fehlertoleranz*[48].

Die Beziehungen zwischen `Assets` und `Exemplaren` sowie zu `Verzeichnissen` des *Namens- und Verzeichnisdienstes* werden im Folgenden detailliert betrachtet. Desweiteren werden die Bedingungen, die für Entitäten und Beziehungen gelten, einschließlich spezieller Bedingungen zur Sicherstellung der Eindeutigkeit des Schlüssels, in der nachfolgenden Aufstellung der Beziehungen und Bedingungen behandelt. Die Zuordnung von `Assets` und `Exemplaren` zu `Nachrichten` und `Rechten` erfolgt, aufgrund der ermittelten Navigationsrichtungen für diese Beziehungen, im zuvor beschriebenen *Rechte- und Rollenverwaltungs-* sowie im *Nachrichtenverwaltungsdienst*.

In Abbildung 3.8 werden die *Entitäten* des Werkverwaltungsdienstes sowie ihre Beziehungen untereinander und zu Entitäten anderer Dienste in einem *schematischen Klassendiagramm* dargestellt. Das Diagramm ist ein Ausschnitt der Abbildung 3.2, die die gesamte begriffsorientierte Infrastruktur illustriert. Die zugehörigen *Asset-Modelle* können den Kodeabschnitten A.6, A.7 und A.8 im Anhang entnommen werden.

- **Bedingung**: `Exemplare` werden über das Charakteristikum `id` und die `Exemplar/Werkausführender`-Beziehung eindeutig identifiziert.

- **Bedingung**: `Prototypen` werden über die Charakteristika `id` oder `name` eindeutig identifiziert.

- **Teilwerk/Werk-Beziehung**: Diese Beziehung verbindet `Werke`. Die Rollen sind `Teilwerk` und `Werk`. Ein `Werk` kann `Teilwerk` von beliebig vielen `Werken` sein $(0..n)$. Ein `Werk` kann beliebig viele `Teilwerke` haben $(0..n)$. Die Beziehung ist in beide Richtungen navigierbar.

- **Bedingung**: Die `Teilwerk/Werk`-Beziehung bildet einen gerichteten azyklischen Graphen. Diese Bedingung kann weder in der Abbildung 3.2 noch im nachfolgenden *Asset-Modell* des Werkverwaltungsdienstes beschrieben werden. Sie muss demzufolge durch die nichtelementaren Anwendungsfälle `Werk eintragen` und `Teilwerk zuordnen` sichergestellt werden.

[46] Vgl. Abschnitt 3.3, Asset-Prozesse als Entitäten
[47] Vgl. Abschnitt 4.3.2, Replikation
[48] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme

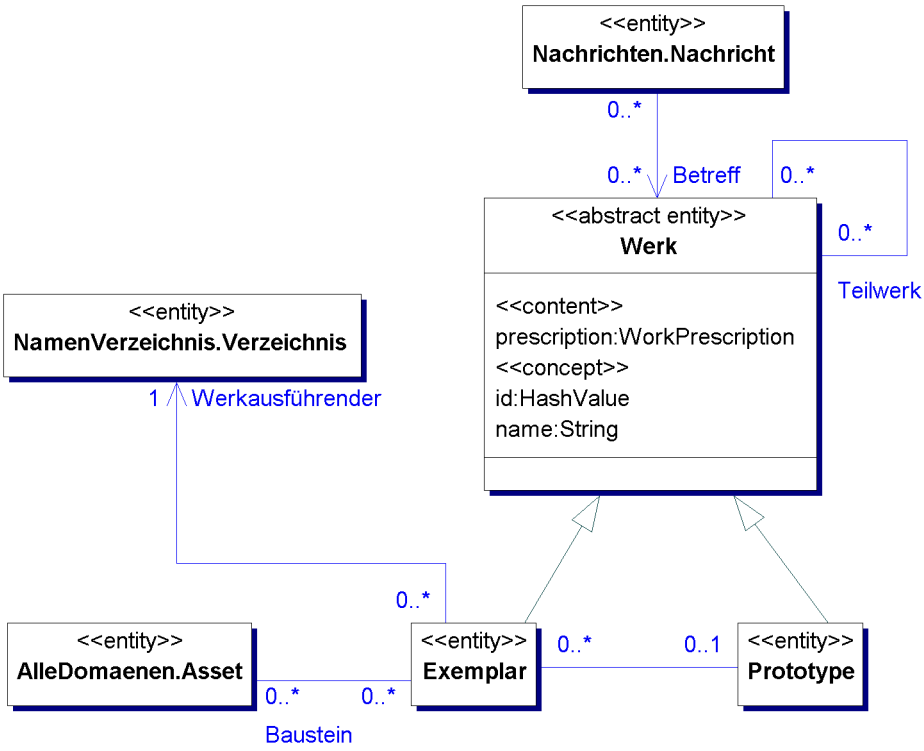

Abbildung 3.8: Entitäten der Werkverwaltung.

- **Anmerkung**: Der Graph der `Teilwerk`/`Werk`-Beziehung, der Beziehungen wird je nach Anwendungsfall mit oder ohne *transitive Hülle* interpretiert. Ein Beispiel, für den Verzicht auf die *Hülle*, ist der Versand von Nachrichten an `Werke`, der nur an direkt abhängige `Werke` erfolgt. Das gegenteilige Beispiel ist die Ermittlung der transitiven Werkabhängigkeiten zur Vermeidung von Zyklen. Aus diesem Grund kann keine *Subsumptions-Bedingung* als Teil des *Asset-Modells* definiert werden.
- **Bedingung**: Die `Teilwerk`/`Werk`-Beziehung wird für `Exemplare` verschärft. Es können nur `Exemplare` miteinander in Beziehung gesetzt werden.
- **Bedingung**: Die `Teilwerk`/`Werk`-Beziehung wird für `Prototypen` verschärft. Es können nur `Prototypen` miteinander in Beziehung gesetzt werden.
- **Prototyp/Exemplar-Beziehung**: Diese Beziehung verbindet `Prototypen` und `Exemplare`. Ein `Prototyp` kann beliebig viele `Exemplare` haben $(0..n)$. Ein `Exemplar` kann höchsten einen `Prototypen` haben (0..1). Die Beziehung ist in beide Richtungen navigierbar.
- **Baustein/Werk-Beziehung**: Diese Beziehung verbindet `Assets` und `Werke`. Die Rollen sind `Baustein` und `Werk`. Ein `Baustein` kann zu beliebig vielen `Werken` gehören $(0..n)$. Ein `Werk` kann aus beliebig vielen `Bausteinen` bestehen $(0..n)$. Die Beziehung ist beide Richtungen navigierbar.
- **Exemplar/Werkausführender-Beziehung**: Diese Beziehung verbindet `Exemplare` und `Verzeichnisse`. Die Rollen sind `Exemplar` und `Werkausführender`. Ein `Exemplar`

hat genau einen `Werkausführenden` (1). Ein `Werkausführender` führt beliebig viele `Exemplare` aus ($0..n$). Die Beziehung ist in Richtung der `Werkausführenden` navigierbar, um den Anwendungsfall `Nachrichten einspeisen` in Abbildung 3.11 zu unterstützen.

Nachdem nun die Entitäten der Werkverwaltung, einschließlich ihrer Beziehungen und Bedingungen, detailliert betrachtet und als Ausschnitt der Abbildung 3.2 dargestellt wurden, gilt es das entsprechende *Asset-Modell* in der *Asset-Definitionssprache* zu formulieren. Dabei sind auch die *Proxy-Asset-Modelle* zu berücksichtigen, in denen die Verweise auf entfernte `Assets` und `Namen` des *Namens- und Verzeichnisverwaltungsdienstes* verwaltet werden.

Auch in diesem Fall werden wieder *Rekursion* und *Mengen-Semantik* benötigt, um die nichtelementaren Anwendungsfälle als *präskriptiven Werkbeschreibungen* auszudrücken.

3.2.6 Namens- und Verzeichnisverwaltungsdienst

Der *Namens- und Verzeichnisverwaltungsdienst* ist für die Verwaltung und *Auflösung* von `Namen` zuständig, die in einer *hierarchischen* Verzeichnisstruktur angeordnet sind. Dieser Dienst wird besonders bei der *Delegation* von *Teilwerken* im Zusammenhang mit der *transparenten Lokalisierung* des *Werkempfängers* genutzt[49] und trägt damit wesentlich zur *Skalierbarkeit*[50] der *begriffsorientierten Infrastruktur* bei. Die nichtelementaren Anwendungsfälle des Dienstes können Abbildung 3.9 entnommen werden. Sie dienen zunächst der Identifikation der Entitäten mit Charakteristika, Beziehungen und Bedingungen.

Die hierarchische Verzeichnisstruktur sowie Bedingungen, die die *Eindeutigkeit* von Verzeichnisnamen und Namensschlüsseln auf der Ebene eines `Verzeichnisses` sicherstellen, ermöglichen die Verwendung von *eindeutig Pfaden* für den Verweis auf Elemente der Namens- und Verzeichnisstruktur. Die Pfade können durch *Verkettungen* von Verzeichnisnamen und an letzter Stelle stehender Namensschlüssel, die immer absolut interpretiert werden, beschrieben werden. Diese Eigenschaft machen sich auch *Proxy-Asset-Modelle* zunutze, die Verweise auf entfernte *Assets* verwalten und zu diesem Zweck einen Pfad der Namens- und Verzeichnisverwaltung als *logische Adresse* der jeweiligen Komponente interpretieren, auf der sich das *Asset* befindet.

Die zu beschreibenden Entitäten des Dienstes sind `Namen` und `Verzeichnisse`. Deren Beziehungen untereinander sowie mit den `Kanälen` des *Nachrichtenverwaltungsdienstes* werden im Folgenden detailliert betrachtet. Desweiteren werden die Bedingungen, die für Entitäten und Beziehungen gelten, behandelt. Die Zuordnung von `Verzeichnissen` zu `Domänen` und `Organisationseinheiten` sowie zu `Werken` erfolgt, aufgrund der ermittelten Navigationsrichtung für diese Beziehungen, im Benutzer- und Gruppenverwaltungsdienst[51] bzw. im Werkverwaltungsdienst[52].

In Abbildung 3.10 werden die *Entitäten* des Namens- und Verzeichnisverwaltungsdienstes sowie ihre Beziehungen untereinander und zu Entitäten anderer Dienste in einem *schematischen Klassendiagramm* dargestellt. Das Diagramm ist ein Ausschnitt der Abbildung 3.2, die die gesamte begriffsorientierte Infrastruktur illustriert. Die zugehörigen *Asset-Modelle* können den Kodeabschnitten A.9 und A.10 im Anhang entnommen werden.

[49] Vgl. Abschnitt 3.4.2, Prozessdelegation mit transparenter Lokalisierung
[50] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit
[51] Vgl. Abschnitt 3.2.3, Benutzer- und Gruppenverwaltungsdienst
[52] Vgl. Abschnitt 3.2.5, Werkverwaltungsdienst

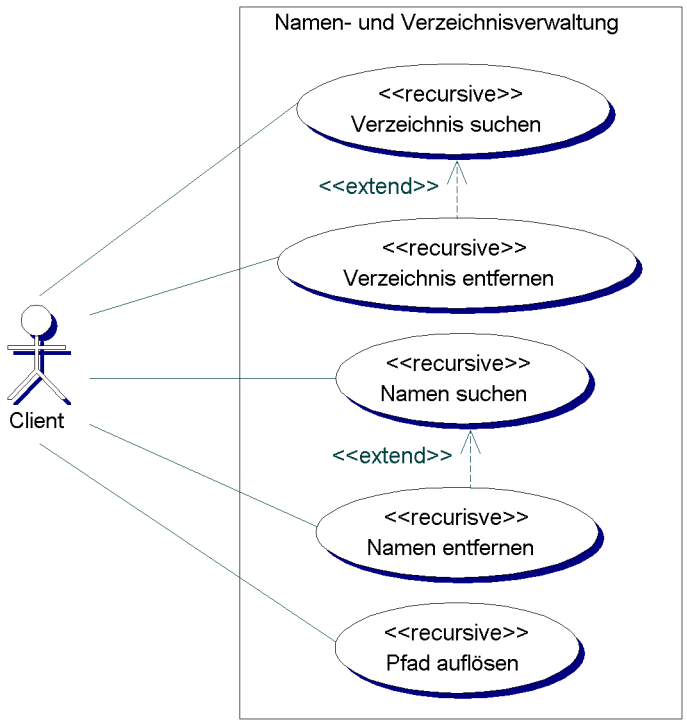

Abbildung 3.9: Anwendungsfälle der Namens- und Verzeichnisverwaltung.

- **Name/Verzeichnis-Beziehung**: Diese Beziehung verbindet `Namen` und `Verzeichnisse`. Die Rollen sind `Eintrag` und `Verzeichnis`. Ein `Eintrag` gehört zu genau einem `Verzeichnis` (1). Ein `Verzeichnis` kann beliebig viele `Einträge` haben (0..n). Die Beziehung ist beide Richtungen navigierbar, um die universelle Verwendbarkeit des Namens- und Verzeichnisdienstes nicht einzuschränken.

- **Unterverzeichnis/Oberverzeichnis-Beziehung**: Diese Beziehung verbindet `Verzeichnisse`. Die Rollen sind `Unterverzeichnis` und `Oberverzeichnis`. Ein `Unterverzeichnis` gehört zu höchstens einem `Oberverzeichnis` (0..1). Ein `Oberverzeichnis` kann beliebig viele `Unterverzeichnisse` haben (0..n). Die Beziehung ist beide Richtungen navigierbar, um die universelle Verwendbarkeit des Namens- und Verzeichnisdienstes nicht einzuschränken.

- **Bedingung**: Das Charakteristikum `Schlüssel` aller `Namen` die `Eintrag` eines `Verzeichnisses` sind, vereinigt mit dem Charakteristikum `Name` aller `Verzeichnisse` die `Unterverzeichnis` des selben `Verzeichnisses` sind, muss *eindeutig* sein, um die einfache Benennung von Pfaden zu ermöglichen. Diese Bedingungen werden durch das *Asset-Modell* des Namens- und Verzeichnisverwaltungsdienstes sichergestellt.

- **Verzeichnis/Kanal-Beziehung**: Diese Beziehung verbindet `Verzeichnisse` und `Kanäle`. Ein `Verzeichnis` kann höchstens einen `Kanal` benutzen (0..1). Ein `Kanal` kann von beliebig vielen `Verzeichnissen` benutzt werden (0..n). Die Beziehung ist Richtungen der `Kanäle` navigierbar.

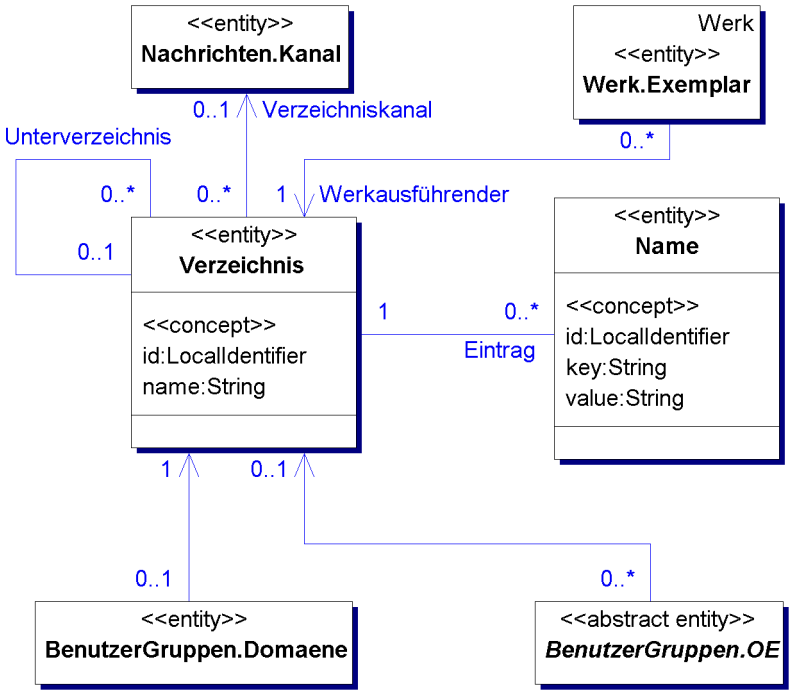

Abbildung 3.10: Entitäten der Namens- und Verzeichnisverwaltung.

- **Bedingung**: Jedem `Verzeichnis` wird genau ein `Oberverzeichnis` zugeordnet. Wird diese Beziehung nicht explizit hergestellt, dann ist ein `Verzeichnis` `Unterverzeichnis` des *Root-Verzeichnis-Assets*. Diese Bedingung wird durch das *Asset-Modell* des Namens- und Verzeichnisverwaltungsdienstes sichergestellt.

Nachdem nun die Entitäten der Namens- und Verzeichnisverwaltung, einschließlich ihrer Beziehungen und Bedingungen, detailliert betrachtet und als Ausschnitt der Abbildung 3.2 dargestellt wurden, gilt es das entsprechende *Asset-Modell* in der *Asset-Definitionssprache* zu formulieren. Dabei ist auch das *Proxy-Asset-Modell* des Nachrichtenverwaltungsdienstes zu berücksichtigen, in dem die Verweise auf entfernte `Kanäle` verwaltet werden.

3.2.7 Nachrichtenverwaltungsdienst

Der *Nachrichtenverwaltungsdienst* ist für die Zuordnung von `Nachrichten` zu `Kanälen` zuständig. Die Übertragung von Nachrichten ist im Zusammenhang mit der *transparenten Replikation*[53], der *asynchronen* Kommunikation mit Werken[54] und der *Entkoppelung* von Kommunikationspartnern zur Steigerung der *Skalierbarkeit*[55] von Bedeutung. Die nichtelementaren Anwendungsfälle des Dienstes können Abbildung 3.11 entnommen werden. Sie dienen zunächst der Identifikation der Entitäten mit Charakteristika, Beziehungen und Bedingungen.

[53] Vgl. Abschnitt 3.1.2, Transparenz - Transparente Replikation
[54] Vgl. Abschnitt 172, Asynchrone und synchrone Teilprozesse
[55] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit

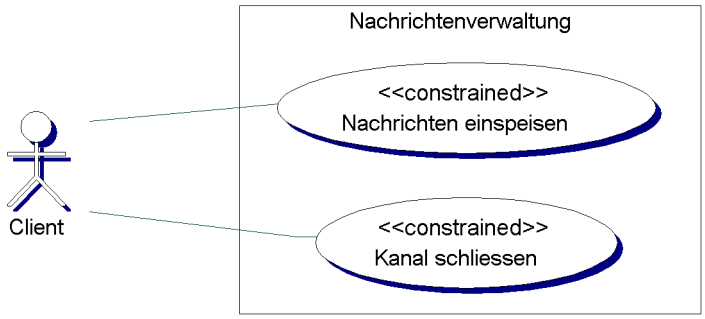

Abbildung 3.11: Anwendungsfälle der Nachrichtenverwaltung.

Diese Zuordnung von `Nachrichten` zu `Kanälen` geschieht mit Hilfe des *Werkverwaltungsdienstes*[56], der die Abhängigkeiten zwischen `Assets` und `Werken` sowie zwischen `Teilwerken` und `Werken` auflösen kann. Über den *Namens- und Verzeichnisdienst*[57] wiederum können die jeweiligen `Kanäle` der *Werkausführenden* der direkt betroffenen `Werke` ermittelt werden. In jedem dieser `Kanäle` wird genau eine Kopie der `Nachricht` abgelegt. Wichtig ist, dass nur `Nachrichten` für *direkt* betroffene `Werke` versandt werden, um die *Invalidierung* der *Replikate* sukzessive von den Blättern zur Wurzel des *Werkabhängigkeitsgraphen* vorzunehmen. An dieser Stelle könnten intelligentere Verfahren zu erheblichen Performanzsteigerungen führen[58].

Die zu beschreibenden Entitäten des Dienstes sind `Nachrichten` und `Kanäle`. Deren Beziehungen untereinander sowie mit den `Assets` und `Werken` des Werkverwaltungsdienstes werden im Folgenden detailliert betrachtet. Desweiteren werden die Bedingungen, die für Entitäten und Beziehungen gelten, behandelt. Die Zuordnung von `Kanälen` zu `Verzeichnissen` erfolgt, aufgrund der ermittelten Navigationsrichtung für diese Beziehung, im Namens- und Verzeichnisverwaltungsdienst.

In Abbildung 3.10 werden die *Entitäten* des Namens- und Verzeichnisverwaltungsdienstes sowie ihre Beziehungen untereinander und zu Entitäten anderer Dienste in einem *schematischen Klassendiagramm* dargestellt. Das Diagramm ist ein Ausschnitt der Abbildung 3.2, die die gesamte begriffsorientierte Infrastruktur illustriert. Die zugehörigen *Asset-Modelle* können den Kodeabschnitten A.11, A.12 und A.13 im Anhang entnommen werden.

- **Nachricht/Kanal-Beziehung**: Diese Beziehung verbindet `Nachrichten` und `Kanäle`. Eine `Nachricht` liegt in genau einem `Kanal` (1). In einem `Kanal` können beliebig viele `Nachrichten` liegen ($0..n$). Die Beziehung ist in Richtung der `Nachrichten` navigierbar. Die gewählte Kardinalität und Navigationsrichtung kann nicht im *Asset-Modell* ausgedrückt werden. Sie muß dementsprechend in dem nichtelementaren Anwendungsfall `Nachricht einspeisen` sichergestellt werden.
- **Nachricht/Asset-Beziehung**: Diese Beziehung verbindet `Nachrichten` und `Assets`. Eine `Nachricht` kann beliebig viele `Asset` betreffen ($0..n$). Ein `Asset` kann von beliebig vielen `Nachrichten` betroffen sein ($0..n$). Die Beziehung ist in Richtung der `Assets` navigierbar.

[56]Vgl. Abschnitt 3.2.5, Werkverwaltungsdienst
[57]Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltungsdienst
[58]Vgl. Abschnitt 5.3, Ausblick

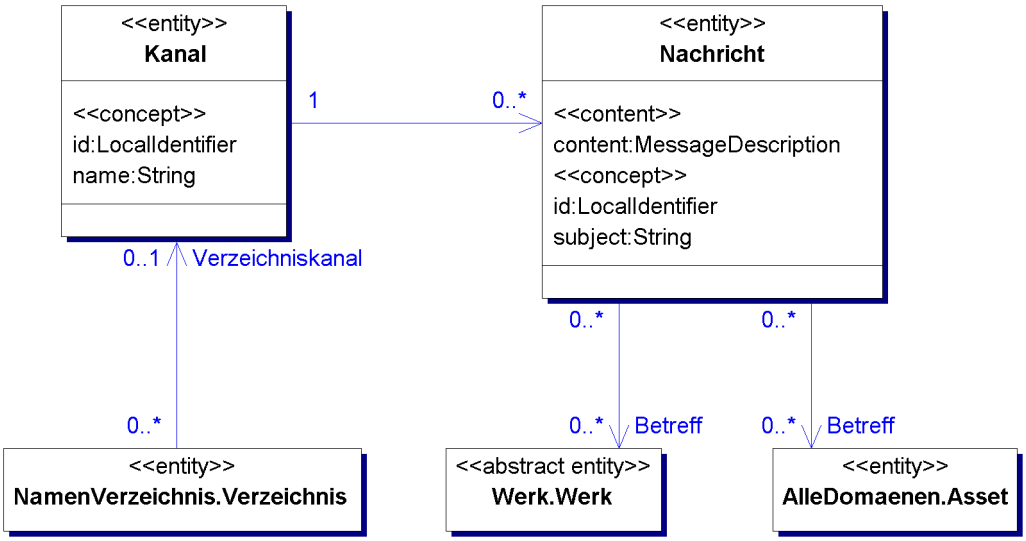

Abbildung 3.12: Entitäten der Nachrichtenverwaltung.

- **Nachricht/Werk-Beziehung**: Diese Beziehung verbindet `Nachrichten` und `Werke`. Eine `Nachricht` kann beliebig viele `Werke` betreffen $(0..n)$. Ein `Werk` kann von beliebig vielen `Nachrichten` betroffen sein $(0..n)$. Die Beziehung ist in Richtung der `Werke` navigierbar.
- **Bedingung**: Jede `Nachricht` betrifft entweder `Assets` oder `Werke`. Diese Bedingung wird durch das *Asset-Modell* des Nachrichtenverwaltungsdienstes sichergestellt.

Nachdem nun die Entitäten der Nachrichtenverwaltung, einschließlich ihrer Beziehungen und Bedingungen, detailliert betrachtet und als Ausschnitt der Abbildung 3.2 dargestellt wurden, gilt es das entsprechende *Asset-Modell* in der *Asset-Definitionssprache* zu formulieren. Dabei ist auch das *Proxy-Asset-Modell* des Werkverwaltungsdienstes zu berücksichtigen, in dem die Verweise auf entfernte `Assets` und `Werke` verwaltet werden.

3.3 Asset-Prozesse als Entitäten

Im nachfolgenden Abschnitt wird näher auf die Modellierung von *Asset-Prozessen* durch *Assets* vom Typ `Werk` eingegangen. Der Abschnitt 3.3.2 entwirft anschließend die Formen der *präskriptiven* und *deskriptiven Werkbeschreibung*, die der Abbildung des *inkrementellen Werkfortschrittes* und der *konstruktiven* und *analytischen* Nutzung von Werkbeschreibungen dienen. Zum Abschluß führt Abschnitt 3.3.3 ein *Prozessmodel* zur Ausführung von präskriptiven Werkbeschreibungen ein.

3.3.1 Beschreibung von Asset-Prozessen durch Assets

Im Abschnitt 2.3.1 wurde bereits angedeutet, dass *Asset-Prozesse*, die aus allen Vorgängen des Zugriffs und der Veränderung des *konzeptionellen* oder *inhaltlichen* Teils von *Assets* be-

stehen und somit Ausschnitte menschlicher *Erkenntnisprozesse* abbilden, als *Entitäten* aufgefasst werden, die wiederum durch *Assets* beschrieben werden. Der Typ dieser *Assets* zur Prozessbeschreibung wird als `Werk` bezeichnet und der *Domäne* der `Werkverwaltung` zugerechnet. Im Rahmen der Einführung von *Basisdiensten* für den Betrieb einer *begriffsorientierten Infrastruktur*[59] wurden, im Vorgriff auf diesen Abschnitt, bereits die *Anwendungsfälle* des *Werkverwaltungsdienstes*[60] beschrieben sowie das zugehörige *Asset-Modell*[61] in der *Asset-Definitionssprache* nach SEHRING[62] formuliert. Die Abbildung 3.8 zeigt das Modell und insbesondere das Zusammenwirken mit *Assets* anderer Modelle der begriffsorienierten Infrastruktur als Ausschnitt von Abbildung 3.2.

Genaugenommen unterscheiden sich Werke in die *Asset-Typen* `Prototyp` und `Exemplar`, die beide vom abstrakten Typ `Werk` abgeleitet sind. Gemein ist ihnen der *Verweis* auf eine *Werkbeschreibung* (`prescription`), die in einer abstrakten *Werkbeschreibungssprache* formuliert ist[63] und sich im *inhaltlichen* Teil der *Asset-Definition* befindet. Desweiteren sind die *Charakteristika* `id` und `name` und die *inversen Beziehungen* `superworks` und `subworks` zu nennen. Für die `id` und die letztgenannten Beziehungen werden im jeweiligen *Subtyp* von `Werk` noch verschärfte *Bedingungen* definiert. Zwischen `Prototypen` und `Exemplaren` besteht ebenfalls eine inverse Beziehung, deren Bedeutung nachfolgend erläutert wird.

`Prototypen` dienen dazu, wertvolle Werkbeschreibungen quasi als *Blaupausen* für die spätere Verwendung aufzubewahren. Sie werden nicht direkt ausgeführt[64] und verfügen dementsprechend über keinen Verweis auf einen *Werkausführenden* `executor`, wie dies bei `Exemplaren` der Fall ist. Stattdessen wird die Werkbeschreibung (`prescription`) in ein neues `Exemplar` kopiert, und ein Verweis vom diesem zum `Prototypen` und umgekehrt erzeugt. Die Charakteristika `id` und `name` sind für `Prototypen` jeweils eindeutig, um Anfragen sowohl für maschinelle als auch menschliche *Akteure* zu vereinfachen. Die inverse `subworks/superworks`-Beziehung definiert in diesem Falle *Abhängigkeitsbeziehungen* zwischen `Prototypen`, d.h. auf die Werkbeschreibung eines `Prototyp` wird in der eines anderen `Prototypen` verwiesen. Damit nur Beziehungen zwischen `Prototypen` vorkommen können, ist dem *Asset-Modell* eine entsprechende Bedingung zugefügt. `Prototypen` können vom Werkverwaltungsdienst wahlweise flach oder tief angefordert werden, um spätere Zugriffe auf den Dienst zu reduzieren. Weitere Überlegungen zu diesem Thema werden im Ausblick in Abschnitt 5.3 diskutiert.

`Exemplare` bilden die tatsächlich „aktiven“ Prozessbeschreibungen. Entsprechend verfügen sie über einen (Pflicht)Verweis auf die *logische Adresse* eines *Werkausführenden* (`executor`). Dieser Verweis bildet zusammen mit dem Charakteristikum `id` den *systemweit eindeutigen Schlüssel* für Werke die sich (potentiell) in Bearbeitung befinden (*Werkaufträge*). Die Eindeutigkeit des Schlüssels ist eine zentrale Anforderung für die korrekte Ausführung von Werken im Zusammenhang mit der *Delegation* von *Teilwerken*[65]. In diesem Sinne ist auch die inverse Beziehung `subworks/superworks` zu interpretieren, die die Abhängigkeiten zwischen delegierenden und delegierten Werkaufträgen modelliert. Im abgeleiteten Typ `Exemplar` des *Asset-Modells* wird dieser Beziehung eine Bedingung hinzugefügt, die sicherstellt, dass nur auf andere `Exemplare`, nicht jedoch auf `Prototypen` verwiesen werden kann.

[59] Vgl. Abschnitt 3.2.2, Basisdienste für den Systembetrieb
[60] Vgl. Abbildung 3.7, Anwendungsfälle der Werkverwaltung.
[61] Vgl. Kodeabschnitt 3.4, *Asset-Modell* der Werkverwaltung
[62] Sehring (2004), S. 39ff.
[63] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen
[64] Vgl. Abschnitt 3.3.3, Prozessmodell
[65] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen

Eine weitere wichtige Beziehung des *Asset-Typ* `Exemplar` modelliert den Zugriff auf *lokale Assets*, also Assets, die vom Werkausführenden verwaltet werden und auf die im Verlauf der Ausführung einer Werkbeschreibung zugegriffen worden ist. Dies schließt auch den lesenden Zugriff mit ein. Diese Beziehung bildet zusammen mit der zuvor genannten *Teilwerk-Beziehung* die Grundlage für *transparente Replikation*[66], weil über diesen Weg die zu invalidierenden Werkbeschreibungsreplikate identifiziert werden können.[67].

Zu guter letzt kann ein `Exemplar` einen Verweise auf seinen `Prototypen` enthalten und über diesen wiederum andere `Exemplare` (Werkaufträge) aufspüren, die die selbe Werkbeschreibung beinhalten, jedoch von einem anderen Werkausführenden bearbeitet werden. Dies kann der Erhöhung der Fehlertoleranz der begriffsorientierten Infrastruktur dienen[68].

Listing 3.4: *Asset-Modell* der Werkverwaltung

```
model WorkManagement

from AllDomains import Asset
from NameDirectoryManagement import Directory

class Work {

    content prescription : XML
    concept characteristic id   : HashValue
            characteristic name : String

            relationship superworks         : Work*
            relationship subworks           : Work*

}

class Prototype refines Work {

    concept relationship exemplars  : Exemplar*

        ; *** uniqueness
        constraint lookfor Prototype { id = self.id } = { self }

        constraint lookfor Prototype { name = self.name } = { self }

        ; *** inversion
        constraint lookfor Prototype { superworks >= self } = subworks
            onviolation update self {

                subworks += lookfor Prototype { superworks >= self }

            }

        constraint exemplars.prototype = { self }
            onviolation update exemplars { prototype := self }

}

class Exemplar refines Work {

    concept relationship prototype  : Prototype
            relationship executor   : Directory
```

[66] Vgl. Abschnitt 3.1.2, Transparenz - Transparente Replikation
[67] Vgl. Abschnitt 3.4.3, Prozessdelegation mit Awareness und Data Provenence
[68] Vgl. Abschnitt 101, Fehlertoleranz

```
        ; *** uniqueness
        contraint lookfor Exemplar {

            id          = self.id
            executor    = self.executor

        } = { self }

        ; *** inversion
        constraint lookfor Exemplar { superworks >= self } = subworks
            onviolation update self {

                subworks += lookfor Exemplar { superworks >= self }

            }

        constraint prototype.exemplars >= { self }
            onviolation update prototype { exemplars += self }

        ; *** cardinality
        constraint executor # na

}
```

3.3.2 Präskriptive und deskriptive Prozessbeschreibungen

Abschnitt 2.4.5 hat bereits einige Anforderungen an *begriffsorientierte Systeme* formuliert, die *Werke* unterstützen. Eine wesentliche dieser Anforderungen ist, dass eine Form der Werkbeschreibung gefunden wird, die sowohl *analytisch* als auch *konstruktiv* ausgewertet werden kann. Zudem richten sich Werkbeschreibungen an menschliche und maschinelle *Akteure*, die über unterschiedliche *Interpretationsfähigkeiten* verfügen. Deshalb ist sinnvoll eine *deklarative* Form der Werkbeschreibung zu wählen, in der der Interpretationsfähigkeit der *Akteure* angepasste Element nebeneinander stehen können. Die analytische Auswertung einer Werkbeschreibung kann z.B. für menschliche Akteure interessant sein, die etwas über den Erkenntnisprozeß eines anderen Akteurs lernen wollen.

Eine weitere Anforderung war, dass die Werkbeschreibung den *kontinuierlichen* Verlauf des Werkfortschrittes abbildet und das dabei jeder *Zwischenschritt* eine *valide* Werkbeschreibung darstellt, die sich in ihrer Form nicht von anderen Werkbeschreibungen unterscheidet. Die Ausführung von Werkbeschreibungen kann also insbesondere zu jedem Zeitpunkt unterbrochen und wieder aufgenommen werden.[69]

In der vorliegenden Arbeit wird aus diesen Gründen entschieden, den Grad des Werkfortschrittes durch die Zahl der offenen Bindungen in einer Werkbeschreibung auszudrücken. Eine Werkbeschreibung, die offene Bindungen aufweist, wird als präskriptiv bezeichnet. Sind alle Bindungen hergestellt, so wird sie als deskriptiv bezeichnet. Dieser Zustand ist jedoch keineswegs stabil, sondern kann jederzeit durch eine Manipulation der Werkbeschreibung verlassen werden. Werden einer Werkbeschreibung neue präskriptive Elemente und damit offene Bindungen hinzugefügt, dann kann die konstruktive Werkbearbeitung fortgesetzt werden. Auf diese Weise kann eine Werkbeschreibung gleichzeitig die Vergangenheit und die (geplante) Zukunft eines Werkes beschreiben.

[69] Vgl. Abschnitt 3.3.3, Prozessmodell

Nachdem nun die grundlegenden Eigenschaften der Werkbeschreibung abgeleitet und festgelegt wurden, gilt es nun die Frage zu klären mit welchen *Ausdrucksmitteln* eine Werkbeschreibung formuliert werden soll. Dazu wird in dieser Arbeit eine *Werkbeschreibungssprache* eingeführt, deren Grundelemente in diesem Abschnitt kurz angerissen werden. Eine konkrete Implementation dieser Sprache als *XML-Dialekt* erfolgt in Abschnitt 4.2. Anschließend wird die Sprache in Abschnitt 4.3.3 für die Implementation eines exemplarischen Anwendungsfalles des Nachrichtenverwaltungsdienstes[70] verwandt.

Zunächst ist die Überlegung anzustellen, was die *Grundelemente* der Werkbeschreibungssprache sind. Da mit der Sprache *Asset-Prozesse* beschrieben werden sollen, sind dies die vier grundlegenden *Operationen* Erzeugung, Abfrage, Manipulation und Zerstörung von *Assets*, die auch mit den Anweisungen in der *Asset-Definitions-, Abfrage- und Manipulationssprache*[71] korrespondieren. Es müssen also nur deren deklarative Pendants gefunden werden. Diese weisen einen Verweis auf einen *Asset-Typ* und eine Reihe von *Parametern* auf, die zur Ausführung der jeweiligen Operation benötigt werden.

Eine *Werkbeschreibung* besteht nun aus einer *Menge von Grundelementen*, in der einige Elemente voneinander abhängen, während andere unabhängig sind. *Abhängigkeit* wird in der hier vorgestellten deklarativen Werkbeschreibungssprache durch *Schachtelung* ausgedrückt. Insgesamt entsteht ein in Richtung der Wurzel *gerichteter Baum*. Elemente zwischen denen es keinen (gerichteten) *Pfad* im Baum gibt, sind unabhängig und können *nebenläufig* ausgeführt werden.

Es gibt in einer Werkbeschreibung einen *globalen* und für jedes Grundelement jeweils einen *lokalen Sichtbarkeitsbereich*. Die lokalen Sichtbarkeitsbereiche von untergeordneten Elementen verdecken die Sichtbarkeitsbereiche von übergeordneten Elementen. In den lokalen Sichbarkeitsbereichen ist das *Asset* oder die Menge von *Assets* sichtbar, die sich als Resultat der ausgeführten Operation ergeben. Insbesondere kann auf die *Charakteristika* und *Beziehungen* der Assets zugegriffen werden. Auch die Operation zur Zerstörung von *Assets* liefert eine Ergebnismenge, nur haben diese *Assets* einen volatilen Status.

An den globalen Sichtbarkeitsbereich können *Parameterelemente* gebunden werden. Werden mehrere dieser Elemente mit dem selben Namen gebunden, so bilden sie eine Menge. Die Parameterelemente können aus allen lokalen Sichtbarkeitsbereichen zugegriffen werden. Der Zugriff erfolgt wie bei den Charakteristika und Beziehungen von *Assets* bzw. *Asset-Mengen*. Parameterelemente sind inbesondere für die nachfolgend eingeführte *Modularisierung* von Werkbeschreibungen interessant und erlauben die Übergabe von Parametern an untergeordnete Werkbeschreibungen.

Ein *präskriptives* Element entsteht, indem ein Parameter eines Grundelements eine *offene Bindung* aufweist. Ein Grundelement ist *deskriptiv*, wenn alle zugehörigen untergeordneten Elemente gebunden sind. Präskriptive und deskriptive Elemente unterscheiden sich also nur durch offene Bindungen. Offene Bindungen können einen *Verweis* auf ein Charakteristikum oder eine Beziehung des Assets im jeweiligen Sichtbarkeitsbereich enthalten. Ein Verweis wird aufgelöst, sobald die zugehörigen *Assets* verfügbar sind. Verweise werden durch besondere Symbole ausgezeichnet. Verweise auf Parameterelemente werden wie bereits erwähnt orthogonal behandelt.

Da Grundelemente auch *Asset-Mengen* zurückliefern, muss eine Mengen-Semantik in der

[70] Vgl. Abschnitt 3.2.7, Nachrichtenverwaltungsdienst
[71] Sehring (2004), S. 63.

Werkbeschreibungssprache enthalten sein. Dazu wird ein Iteratorsymbol eingeführt. Mit diesem können Elemente in einem Sichtbarkeitsbereich markiert werden. Die markierten Elemente werden entsprechend der Kardinalität der *Asset-Menge* vervielfältigt und an Stelle des alten Elementes in die Werkbeschreibung eingefügt. An jedes dieser neuen Elemente wird ein *Asset* der *Asset-Menge* gebunden. Innerhalb der neuen Elemente können nun wie üblich Verweise auf Charakteristika und Beziehungen enthalten sein. Es können auch mehrere Elemente in einem Sichtbarkeitsbereich mit dem Iteratorsymbol markiert werden. Die genannte Vorgehensweise wird auf jedes dieser Elemente angewandt und einzelne *Assets* mehrfach gebunden. Die beschriebene Mengen-Semantik gilt orthogonal für *Parametermengen*.

Grundelemente können auch sogenannte *Wächter-Elemente* enthalten. Durch diese können *logische Ausdrücke* angegeben werden, die erfüllt sein müssen, damit die zugehörige Operation ausgeführt wird. Mehrere Wächter können durch *logische Operatoren* (*and, or, not*) miteinander verknüpft werden.

Als letzte Erweiterung können die Grundelemente noch *Ausnahme-Elemente* enthalten. In diesen sind wiederum Grundelemente enthalten, die nur im *Fehlerfall* ausgeführt werden. Die Ausnahme-Elemente liegen im gleichen Sichtbarkeitsbereich, wie deren Grundelement.

Werkbeschreibungen können aus Gründen der Wiederverwendung *modularisiert* werden. Diese untergeordneten Werkbeschreibungen werden u.U. an andere Werkausführende *delegiert*[72]. Desweiteren werden Grundelemente, deren *Asset-Typ* nicht zum *lokalen Asset-Modell* des Werkausführenden passt, in eine sogenannte *ad-hoc Werkbeschreibung* verpackt und zusammen mit ihren untergeordneten Elementen an andere Werkausführende delegiert. Zudem können über diesen Mechanismus *rekursive Werkaufrufe* ausgedrückt werden.

Referenzen (Verweise auf Beziehungen) und entfernte Referenzen[73], werden durch ein spezielles Symbol markiert und bei der Bindung durch eine flache und lokale Repräsentation ersetzt. Bei entfernten Referenzen wird zudem eine ad-hoc Werkbeschreibung erzeugt und an einen zuständigen Werkausführenden delegiert. Desweiteren wird bei Zuweisungen an Beziehungen ein lokaler Stellvertreter (*Proxy*) im *Asset-Modell* erzeugt, bzw. ein bereits vorhandener Stellvertreter genutzt.

3.3.3 Prozessmodell

Im Abschnitt 3.3.1 wurde gezeigt, wie *Asset-Prozesse* durch *Assets* vom Typ `Werk` beschrieben werden können. Dabei wurde insbesondere die *Werkbeschreibung* hervorgehoben, die im *inhaltlichen* Teil eines Werkes verwaltet wird. Diese hat je nach Zahl der *offenen Bindungen*, einen mehrheitlich *präskriptiven* oder *deskriptiven* Charakter[74].

Desweiteren wurde entschieden, Werkvorschriften *deklarativ* zu formulieren, um deren analytische Nutzung (Protokoll- und Überwachungsfunktion) zu unterstützen, sie sich hauptsächlich an menschliche *Akteure* richtet. Demzufolge definiert eine Werkbeschreibung keinen *Kontrollfluss*, der die Abfolge von Bindungen eindeutig festlegt, sondern einen *Datenfluss*, der lediglich *Abhängigkeiten* zwischen offenen Bindungen modelliert und, unter Beachtung dieser Abhängigkeiten, durch alternative Kontrollflüsse verwirklicht werden kann. Dies schließt insbesondere in einem verteilten Entwurf, den die vorliegende Arbeit verfolgt, die nebenläufige

[72] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen
[73] Vgl. Proxy-Asset-Modelle in Abschnitt 3.2.2, Basisdienste für den Systembetrieb
[74] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen

Ausführung einer Werkbeschreibung oder von Teilen einer Werkbeschreibung mit ein[75].

Werkbeschreibungen werden nun je nach *Formalisierunggrad* und *Interpretationsfähigkeit* durch menschliche oder maschinelle *Akteure* ausgeführt, d.h. es wird versucht offene Bindungen herzustellen. Dabei kann jede erfolgte Bindung als *Werkfortschritt* interpretiert werden. Die Auswahl des passenden Werkausführenden, der in der Lage ist, Bindungen herzustellen und Werkfortschritt zu erzielen, erfolgt in der begriffsorientierten Infrastruktur über den Namens- und Verzeichnisverwaltungsdienst[76]. Dieser führt entsprechende Verzeichnisse über die Komponenten[77], die einem bestimmten Dienst und damit auch einer Domäne zuzurechnen sind, die wiederum eine Menge von *Asset-Typen* zusammenfasst[78].

Kann durch den Werkausführenden kein Werkfortschritt erzielt werden, so ist es möglich, Teile der Werbeschreibung in ein eigenes Werk (*Teilwerk*) auszugliedern und dieses als neuen *Werkauftrag* an einen anderen Werkausführenden zu *delegieren*. Auf diese Weise ist ein Werk nicht an einen Werkausführenden gebunden, sondern kann durch die Infrastruktur *migrieren*. Da die Entscheidung für oder gegen die *Migration* nur durch die *lokalen* Begebenheiten (Fähigkeit der Bindung) des Werkausführenden bestimmt wird, kann Werken eine gewisse *Autonomie* zugebilligt werden. Die gleicher Vorgehensweise wird im Übrigen auch genutzt, um die Granularität von Werken zu steuern, die große Auswirkungen auf die Effizienz von Replikationmechanismen hat[79].

Wie sich hier schon andeutet, stellt die Delegation von Teilwerken ein Mittel zur Kooperation mit anderen Komponenten dar. Wenn die gegenseitige Delegation nicht auf die Komponenten innerhalb eines Dienstes beschränkt wird, dann ist es auch möglich, die Kooperation von Diensten auf diese Weise abzubilden. Diese Vorgehensweise wird im folgenden Abschnitt eingehender behandelt.

3.4 Kooperation verteilter Dienste durch Delegation von Teilprozessen

Im Abschnitt 3.2 wurde die *Kooperation* von *Organisationseinheiten* verschiedener *Domänen*, also die Kooperation entlang der *Anwendungsstruktur* einer verteilten begriffsorientierten Infrastruktur, als *Kommunikation* von *Komponenten*[80] verteilter *Dienste* dargestellt. Allerdings wurde dort nicht gezeigt, wie diese Kommunikation genau verlaufen soll.

Im Abschnitt 3.3 wurden die Voraussetzungen geschaffen, *Asset-Prozesse* als Elemente erster Ordnung im *begriffsorientierten Modell* zu verwalten. Diese bilden Vorgänge des Zugriffs und der Veränderung der *konzeptionellen* oder *inhaltlichen* Komponente von *Assets* ab. Die Beschreibung erfolgt durch den *Asset-Typ* `Werk` und dessen Inhaltsteil, in dem *Werkbeschreibungen* abgelegt werden. Die Werkbeschreibungen sind in einer *Werkbeschreibungssprache* formuliert, die neben der *analytischen* Nutzung einer Werkbeschreibung auch die *konstruktive* Nutzung durch menschliche oder maschinelle *Akteure* unterstützt.

[75] Vgl. Abschnitt 95, Nebenläufigkeit
[76] Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltungsdienst
[77] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems
[78] Vgl. Abschnitt 112, Domänen als Dienste - Aufteilung von Domänen und Asset-Modellen
[79] Vgl. Abschnitt 4.3.2, Replikation
[80] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems

Eine Werkbeschreibung kann konstruktiv genutzt werden, wenn sie *offene Bindungen* aufweist, also *präskriptiv* ist, und ein Akteur in der Lage ist diese Bindungen herzustellen. Um den passenden Akteur bei der maschinelle Nutzung zu ermitteln, wird in der begriffsorientierten Infrastruktur ein *Namens- und Verzeichnisverwaltungsdienst* eingesetzt, der u.a. *logische Adressen* von Komponenten der verschiedenen *Dienste* und deren *Domänen* verwaltet[81]. Bei der Auswahl wird zudem die *Organisationsstruktur* innerhalb einer Domäne beachtet.

In Abschnitt 3.3.2 wurden dann *Sprachelemente* der *Werkbeschreibungssprache* vorgestellt, mit denen die Ausführung von vier *Grundoperationen* auf *Assets* sowie deren *Abhängigkeiten* untereinander durch *Schachtelung* ausgedrückt werden. Über die verschachtelten *Sichtbarkeitsbereiche* greifen Operationen auf die Charakteristika und Beziehungen der *Assets*, die sich als Resultat einer anderen Operation ergeben haben, zu. Zudem wurde die *Modularisierung* von Werkbeschreibungen in sogenannte *Teilwerke* vorgestellt. Ein Grund war die *Wiederverwendung* von Werkbeschreibungen. Ein zweiter wurde mit der *Delegation* von Teilwerken an andere *Komponenten* bereits kurz angerissen. Dies schloss zum einen die Delegation durch *expliziten Aufruf* eines Teilwerkes mit ein. Zum anderen werden sogenannte *ad-hoc Werke* erzeugt, wenn der angegebene *Asset-Typ* einer Operation nicht lokal aufgelöst werden kann. Die Auflösung und Verwaltung *entfernter Referenzen* erfolgt in diesem Zusammenhang über *Proxy-Asset-Modelle*[82], die als *lokale Vertreter* für entfernte *Assets* modelliert sind, und Elemente zur *eindeutigen Identifikation* der Zielkomponente und des referenzierten *Assets* kapseln.

Damit sind alle Voraussetzungen gegeben, um innerhalb einer Werkbeschreibung über die Delegation von Teilwerken auch mit Komponenten anderer Domänen zu kommunizieren und zu kooperieren. Die *Parameter-, Wächter- und Ausnahmeelemente*[83] einer Operation auf dem *Asset-Typ* einer Domäne, können mit den Werten der Charakteristika und Referenzen der Beziehungen der *Asset-Ergebnismenge* einer Operation auf dem *Asset-Typ* einer anderen Domäne gebunden werden.

Im Folgenden wird die Delegation von Teilwerken an andere Komponenten, die in der eigenen oder in einer „fremden“ Domäne liegen, genauer betrachtet. Abschnitt 3.4.1 unterscheidet dazu verschieden Arten von Teilprozessen, die in einer begriffsorientierten Infrastruktur benötigt werden. Anschließend entwirft Abschnitt 3.4.2 eine Form der Delegation von Teilwerken, die die beauftragte Komponente *transparent lokalisiert*[84]. Zum Abschluß werden die Ziele *Awareness* und *Data Provenence* im Zusammenhang mit der Delegation betrachtet und in den Entwurf eingearbeitet.

3.4.1 Arten von Teilprozessen

In diesem Abschnitt werden die verschieden Arten von *Teilwerken* behandelt, die sich im Zusammenhang mit der *Delegation* an andere Komponenten ergeben können. Es sind dies *asynchrone* und *synchrone* Teilwerke sowie *statische* und *dynamische* Teilwerke. Die genannten Arten von Teilwerken werden vom Submodul für die Werkdelegation im vorliegenden Prototyp unterstützt[85].

[81] Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltungsdienst
[82] Vgl. Proxy-Asset-Modelle in Abschnitt 3.2.2, Basisdienste für den Systembetrieb
[83] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen
[84] Vgl. Abschnitt 3.1.2, Transparenz - Transparente Lokalisierung
[85] Vgl. HTTP-Adapter in Abschnitt 4.3.1, HTTP-Dienstschnittstelle

Synchrone und asynchrone Teilprozesse

Für die Delegation von Teilwerken an andere Komponenten in der *begriffsorientierten Infrastruktur* ergeben sich grundsätzlich die Optionen der synchronen oder asynchronen Delegation. Bei der Nutzung von Inhalten aus *Basisdomänen*, z.B. zur *Lokalisierung* von Kooperationspartnern im *Namens- und Verzeichnisverwaltungsdienst* oder zur *Überwachung* der *Verklemmungsfreiheit* im *Werkverwaltungsdienst*, sollten die Werkaufrufe synchron geschehen können, weil dieses in der Regel keine *langlaufenden Prozesse* sind und im Rahmen eines Zugriffs auf die eigentlichen *Benutzerdomänen* als *Unterstützungsprozesse* dienen[86]. Zudem werden die Basisdienste, die den Betrieb der begriffsorientierten Infrastruktur ermöglichen, in der Regel über dauerhafte und leistungsfähige Netzwerkverbindungen verfügen, deren *Fehleranfälligkeit* als gering einzustufen ist. Zudem können *personalisierte* Varianten der Basisdienste als lokale Komponenten eingesetzt werden.

Dagegen sind asynchrone Teilwerke besonders bei *langlaufenden Prozessen* sinnvoll, die u.U. auch *Interaktion* mit Benutzern erfordern oder eine große Menge von kooperierenden Komponenten umfassen. Zudem sind asynchrone Teilwerke bei der Kommunikation mit Komponenten sinnvoll, die nur *temporär* über eine Netzwerkverbindung verfügen oder deren Verbindung langsam oder fehlerbehaftet ist. Insgesamt erhöht die asynchrone Kommunikation mit anderen Komponenten die *Fehlertoleranz* der begriffsorientierten Infrastruktur[87].

Als weiterer Vorteil der asynchronen Delegation von Teilwerken ist die Steigerung der *Nebenläufigkeit*[88] in der verteilten Infrastruktur zu nennen. Wenn *unabhängige* Teilwerke eines Werkes[89] an mehrere verschiedene Komponenten zu delegieren sind, dann sollte die asynchrone Variante der Kommunikation gewählt werden.

Statische und dynamische Teilprozesse

Neben der Unterteilung in synchrone und asynchrone kann auch die Unterscheidung von statischen und dynamischen Teilwerken für den praktischen Einsatz der begriffsorientierten Infrastruktur sinnvoll sein. Grundsätzlich ist es vorstellbar, dass auch *Ressourcen* in *Werkbeschreibungen* zugegriffen werden, die statischer Natur sind und nicht der Überwachung durch eine Komponente der Infrastruktur unterworfen sind (z.B. *HTML-Seiten*). Der Zugriff auf diese Ressourcen kann dann, je nach der erwarteten *Änderungsrate*, durch ein statisches oder dynamisches Teilwerk erfolgen. Statische Teilwerke werden einmalig während der *Laufzeit* einer delegierenden Komponente ausgeführt. Die zugehörigen *Replikate*[90] werden nicht *invalidiert*.

Dynamische Teilwerke können wiederum in solche unterschieden werden, die *Notifikation* über den *Nachrichtenverwaltungsdienst*[91] unterstützen, und solche die dies nicht tun. Die Erstgenannten werden in Abschnitt 3.4.3 behandelt und repräsentieren das *Standardverhalten* von Komponenten in der begriffsorientierten Infrastruktur.

Für die zweiten Gruppe von dynamischen Teilwerken, die keine Notifikation unterstützen,

[86] Vgl. Abschnitt 3.4.2, Prozessdelegation mit transparenter Lokalisierung
[87] Vgl. Abschnitt 101, Fehlertoleranz
[88] Vgl. Abschnitt 95, Nebenläufigkeit
[89] Vgl. Abschnitt 3.3.3, Prozessmodell
[90] Vgl. Abschnitt 4.3.2, Replikation
[91] Vgl. Abschnitt 3.2.7, Nachrichtenverwaltungsdienst

deren Inhalte jedoch Veränderungen unterworfen sind, muss ein *alternatives* Verfahren gefunden werden. Im vorliegenden Entwurf wird die *periodische* Invalidierung der Replikate vorgeschlagen, die dann durch das Ergebnis eines neuen *Werkaufrufes* ersetzt werden.

3.4.2 Prozessdelegation mit transparenter Lokalisierung

Die Kooperation von Komponenten in der verteilten begriffsorientierten Infrastruktur ist zweidimensional. Zum einen kooperieren Komponenten einer Domäne entlang ihrer *Organisationsstruktur*, zum anderen kooperieren Komponenten verschieden Domänen gemäß der *Anwendungsstruktur*[92]. Die Delegation von Teilwerken soll diesen beiden Dimensionen *folgen* und nicht durch die Angabe von festen Kooperationspartner in Werkbeschreibungen fixiert werden.

Stattdessen wird im vorliegenden Entwurf der Weg der *Lokalisierung* von Kooperationspartnern über einen *Namens- und Verzeichnisverwaltungsdienst*[93] verfolgt. Diese Entscheidung trägt entscheidend zur Steigerung von *Transparenz*, *Fehlertoleranz* und *Skalierbarkeit*[94] der Infrastruktur bei. Insbesondere wird diese Form der Transparenz bei COULOURIS[95] als *transparente Lokalisierung* bezeichnet.

Der Vorgang der Lokalisierung soll hier kurz skizziert werden. Die *Organisationseinheit* einer Komponente hat über den *Benutzer- und Gruppenverwaltungsdienst* und dessen Beziehung zum Namens- und Verzeichnisverwaltungsdienst (`OE/Verzeichnis-Beziehung`) Zugriff auf ein *Verzeichnis*, das die sogenannte *Infrastruktur* der Organisationseinheit verzeichnet. Darin sind die Komponenten der verschiedenen *Domänen* aufgeführt, mit denen die ursprüngliche Komponente gemäß der Anwendungsstruktur kooperieren kann. Die Auswahl des Kooperationspartners erfolgt dann über den Vergleich mit der, in der *Werkbeschreibung*, angegebenen Empfehlung für die Domäne. Bei *ad-hoc Werken* wird die Domäneangabe der gekapselten *Operation* auch für die gesamte Werkbeschreibung übernommen.[96]

Das Infrastrukturverzeichnis einer Organisationseinheit schliesst insbesondere die eigene Komponente mit ein. Auf diese Weise kann auch die Kooperation entlang der Organisationsstruktur über den Namens- und Verzeichnisverwaltungsdienst abgewickelt werden.

Sollte die Lokalisierung gemäß der genannten Vorgehensweise nicht zum Erfolg führen oder nicht gewünscht sein, so kann über den *Asset-Typ* `Domäne` im Benutzer- und Gruppenverwaltungsdienst auch ein *Domänenverzeichnis* abgerufen werden (`Domäne/Verzeichnis-Beziehung`), in dem alle Komponenten einer Domäne gelistet werden. Diese Möglichkeit ist vor allem für die Umsetzung von *Broadcasting-Techniken* im Zusammenhang mit der Lokalisierung von *migrierten Assets* interessant. Die Implementation der beschriebenen Verfahren erfolgt im Submodul für die Werkdelegation[97].

[92] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[93] Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltungsdienst
[94] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme
[95] Coulouris et al. (1994), S. 20.
[96] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen
[97] Vgl. HTTP-Adapter in Abschnitt 4.3.1, HTTP-Dienstschnittstelle

3.4.3 Prozessdelegation mit Awareness und Data Provenence

Im Abschnitt 3.4.1 wurden *dynamische* Teilwerke behandelt, deren Wesen in der *Veränderung* der zugegriffenen *Ressourcen* und damit dem *Verfall* der abhängigen *Werkbeschreibungen* besteht. In Komponenten, die Teil der begriffsorientierten Infrastruktur sind, kann wegen der dynamischen Offenheit der *Asset-Modelle*[98] ohnehin kein anderes Verhalten vorausgesetzt werden.

Aus Gründen der *Performanz*, *Skalierbarkeit* und *Fehlertoleranz*[99] wird in der begriffsorientierten Infrastruktur jedoch umfangreich von der *Replikation*[100] generierter Werkbeschreibungen Gebrauch gemacht. Um diesem Umstand Rechnung zu tragen und die *Entkopplung* der Komponenten, die mit der *transparenten Lokalisierung* in eine Kommunikationsrichtung vollzogen wurde, zu vollenden, verfügt die Infrastruktur über einen *Nachrichtenverwaltunsdienst*[101], über den u.a. Invalidierungsnachrichten für Replikate versandt werden können. Die Eigenschaft, dass Akteure über die Änderungen an zugegriffenen Ressourcen informiert werden und andere abhängige Akteure über Änderungen informieren, wird in der Literatur als *Awareness* bezeichnet[102]. In der Einschränkung auf die Invalidierung von Replikaten kann diese Eigenschaft laut Coulouris[103] auch als *transparente Replikation* bezeichnet werden.

Der Ablauf bei der Invalidierung von Replikaten über den Nachrichtenverwaltungsdienst wird hier kurz skizziert. Als Einstiegspunkt dient in diesem Fall der *Werkverwaltungdienst*[104], der die Abhängigkeiten zwischen Teilwerken und Werken (`Teilwerk/Werk`-Beziehung) sowie zwischen *Assets* und Werken verwaltet. Verändert sich ein *Asset* in irgendeiner Komponente der begriffsorientierten Infrastruktur, so können über den Werkverwaltungsdienst, unter Angabe von `id` und `type` des *Assets* sowie der *logischen Adresse* der zugehörigen Komponente, betroffene Werke ermittelt werden. Über die Beziehung `Werk/Werkausführender` zum *Namens- und Verzeichnisdienst* sowie von dort über die `Verzeichnis/Kanal`-Beziehung zum Nachrichtenverwaltungsdienst, können die zuständigen *Nachrichtenkanäle* für die *Werkausführenden* ermittelt und letztlich in jedem eine Kopie der entsprechenden Nachricht plaziert werden. Diese Kanäle werden von den zugehörigen Komponenten, genauer durch deren *Werkdelegationsmodul* periodisch abgefragt. Dieser Vorgang ist ein Beispiel für die Notwendigkeit von dynamischen Teilwerken ohne Notifikation.

Die gleiche Vorgehensweise wird auch für die *Abhängigkeitsbeziehungen* zwischen Werken angewandt. Für weitere Ausführungen zu diesem Thema und evtl. *Optimierungen* des Ablaufes sei auf Abschnitt 3.2.7 und 5.3 verwiesen. Desweiteren kann wird angemerkt, dass der Nachrichtenverwaltungsdienst auch für andere Nachrichten, als die zur Invalidierung von Replikaten, verwandt werden kann. Ein Beispiel ist wieder die *Migration* von Assets. So können die Werkausführenden von abhängigen Werken durch Nachrichten über die neue *Heimatkomponente* von *Assets* informiert werden.

Zum Abschluß dieses Abschnittes sei noch kurz auf das Thema *Data Provenence*[105] einge-

[98] Vgl. Abschnitt 2.3, Ein dynamisches begriffsorientiertes Modell
[99] Vgl. Abschnitt 3.1.2, Ausgewählte Entwurfsziele verteilter Systeme
[100] Vgl. Abschnitt 4.3.2, Replikation
[101] Vgl. Abschnitt 3.2.7, Nachrichtenverwaltungsdienst
[102] Dourish et al. (1992).
[103] Coulouris et al. (1994), S. 20.
[104] Vgl. Abschnitt 3.2.5, Werkverwaltungsdienst
[105] Bunemann et al. (2001).

gangen. Besonders für die *analytische Nutzung* von Werkbeschreibungen[106] ist es sinnvoll, den genauen Pfad einer Werkbeschreibung durch das Netzwerk der Komponenten aufzuzeichnen. Aus diesen Angaben können Informationen über die *Nutzungshäufigkeit* bestimmter Werke, bzw. deren Werkbeschreibungen, und *Assets* gesammelt und u.a. Aussagen über deren *Wert* für eine Organisationseinheit, eine Domäne oder die gesamte Infrastruktur getroffen werden[107] Deshalb wird bei der Delegation von Teilwerken das Ziel der transparenten Lokalisierung zugunsten von *Data Provenence* aufgeweicht, indem Werkbeschreibungen um Informationen über den Werkausführenden und die Art des Teilwerks angereichert werden.

3.5 Architektur eines begriffsorientierten Systems

Nachdem Abschnitt 3.2.1 die Abbildung von *Domänen* auf *Dienste* sowie von *Organisationseinheiten* einer Domäne auf *logische Server* eines Dienstes motiviert hat, ist es nun an der Zeit, die *Architektur* eines einzelnen *Servers* zu betrachten. Wie bereits vorgegriffen wurde, soll der logische *Server* einer Organisationseinheit, der deren Modelle in einem Kontext verwaltet, nach Sehring[108] als *Komponente* bezeichnet werden.

Die Architektur einer Komponente wird entscheidend von der Art und Abfolge der *Kooperation* mit anderen Komponenten bestimmt. Dieser Umstand wird im Abschnitt 3.5.1 vertiefend behandelt. Anschließend diskutiert und bestimmt Abschnitt 3.5.2 die *Grobarchitektur* von Komponenten als *multi-tier-Systeme*.

Ein Ziel der vorgestellten Architektur nach Sehring[109] ist, neben der durch Kooperation von Komponenten erreichten *Wiederverwendung* von *Assets*, auch die Wiederverwendung von *Funktionalitäten* in Komponenten zu unterstützen. Zu diesem Zweck werden in Abschnitt 3.5.3 *Module* als Einheit der Wiederverwendung von Funktionalitäten eingeführt. Im Anschluß gibt Abschnitt 3.5.4 einen Überblick über die identifizierten *Modularten*, aus denen nach Sehring, durch Festlegung einer individuelle Konfiguration, jede Komponente eines kooperativen begriffsorientierten Inhaltsverwaltungssystems kombiniert werden kann.

3.5.1 Kooperative begriffsorientierte Systeme

In *begriffsorientierten Systemen* existieren zwei Arten der *Kooperation*. Zum einen kooperieren *Organisationseinheiten* der gleichen *Domäne* gemäß ihrer *Organisationsstruktur*, um *Assets* in die *individuelle Sicht* einer Organisationseinheit zu überführen. Die Prozesse, die diese Dimension der Kooperation beschreiben, werden in der vorliegenden Arbeit als *organisationsspezifisch* bezeichnet und je nach Richtung der Kooperation in *Personalisierung* und *Publikation* unterschieden.

Zum anderen kooperieren Organisationseinheiten gemäß der Nutzungsbeziehungen zwischen *Assets* unterschiedlicher Domänen. Die beschreibenden Prozesse werden in der vorliegenden Arbeit als *anwendungsspezifisch* bezeichnet.

Darüberhinaus existieren Prozesse, die beide Dimensionen der Kooperation abbilden, al-

[106] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen
[107] Sehring (2004), S. 84ff.
[108] Sehring (2004), S. 114f.
[109] Sehring (2004), S. 113.

so sowohl entlang der Organisationsstruktur als auch entlang der Anwendungsstruktur kooperieren. Ein Beispiel für eine derartige Konstellation gibt Abbildung 3.13 (rechts oben). Die dargestellte gleichartige Ausrichtung, der Organisationsstrukturen der kooperierenden Domänen, ist jedoch keineswegs fest gegeben. Vielmehr können sie, durch die Zuordnung von *Infrastrukturen* zu Organisationseinheiten im *Namens- und Verzeichnisverwaltunsdienst*[110], in *ko- und kontravarianter* Position zueinander ausgerichtet werden. Darüberhinaus sind, je nach Anwendungsfall, auch Mischformen möglich.

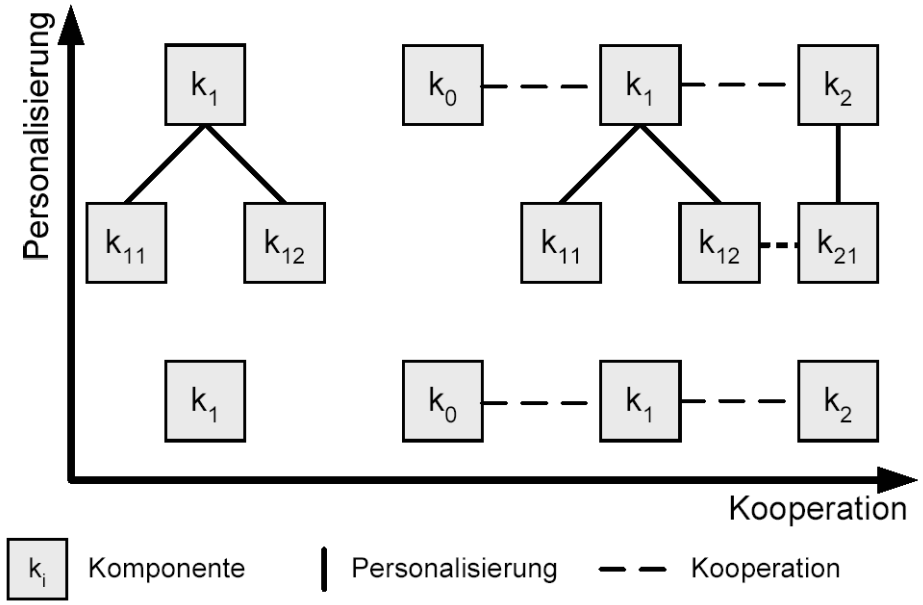

Abbildung 3.13: Kooperation von Komponenten begriffsorientierter Systeme.
Quelle: Sehring (2004, S. 115)

Nachdem dieses Kapitel schrittweise den Schluß von einem begriffsorientierten System zu einer *verteilten begriffsorientierten Infrastruktur* geführt hat, in der für jeden *Kontext* einer Organisationseinheit genau ein *logischer Server* (*Komponente*) betrieben wird[111], lassen sich die vier verschiedenen Varianten der Kooperation wie in Abbildung 3.13, als *Kommunikation* zwischen Komponenten der Infrastruktur, darstellen. Für die Umsetzung der Kommunikation wurde die *Delegation von Teilwerken* eingeführt[112], die wiederum auf der Darstellung von Prozessen durch *präskriptive Werkbeschreibungen* beruht[113].

Es bleibt nun die Frage zu klären, wie sich die beschriebenen Formen der Kooperation durch Kommunikation auf die *Architektur* einer begriffsorientierten Komponente auswirken. In der zugrundeliegenden Architektur nach Sehring[114], in der Komponenten in verschiedene, durch einen *Asset-Modell-Compiler* zu generierende und konfigurierende, *Module* unterteilt werden[115], ergibt sich u.a. die Notwendigkeit von *Adaptions-*, *Koordinations-* und *Distributi-*

[110] Vgl. Abschnitt 3.2.6, Namens- und Verzeichnisverwaltunsdienst
[111] Vgl. Abschnitt 112, Domänen als Dienste - Aufteilung von Domänen und Asset-Modellen
[112] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen
[113] Vgl. Abschnitt 3.3.2, Präskriptive und deskriptive Prozessbeschreibungen
[114] Sehring (2004), S. 113.
[115] Vgl. Abschnitt 3.5.3, Modularisierung von Komponenten in der Applikationsschicht

onsmodulen, deren Bedeutung in Abschnitt 3.5.4 geklärt wird.

3.5.2 Komponenten begriffsorientierter Systeme als multi-tier-Systeme

Vor der nachfolgenden Aufteilung von *Komponenten* der begriffsorientierten Infrastruktur in *Module*, mit dem Ziel der *Wiederverwendung* von *Funktionalitäten*, steht die Frage offen, welcher *Grobarchitektur* der Entwurf folgen soll. Es stehen prinzipiell die *2-tier* oder die *multi-tier-Architektur*, mit *Daten-, Applikations- und Präsentationsschicht*, zur Auswahl. Die *multi-tier-Architektur* bietet jedoch erhebliche Vorteile bezüglich der:

- *Skalierbarkeit*, wenn die Zahl der zugreifenden *Clients* nicht beschränkt ist, weil Zugriffskanäle auf *Standardkomponenten* als Massenspeicherabstraktionen geteilt und Zugriffe optimiert werden können.
- *Offenheit*, weil *multi-tier-Systeme* mehrere alternative *Dienstschnittstellen* für den Zugriff auf die *Anwendungslogik* bereitstellen können, ohne die Logik zu vervielfältigen und die *Performanz* zu reduzieren. *Wartbarkeit* spielt in diesem Zusammenhang keine Rolle, weil die Logik durch den *Asset-Modell-Compiler* generiert wird.
- *Wiederverwendbarkeit*, weil besonders bei der Abbildung der *Kooperation* auf *Kommunikation* zwischen Komponenten und bei der *Adaption* von verschiedenen *Asset-Modellen*, häufig wiederkehrende *Funktionalitäten* auftauchen, die bei *multi-tier-Systemen* in Bausteinen der Anwendungslogik gekapselt werden können.

Vertiefende Ausführungen zu den Vor- und Nachteilen von *2-tier-* und *multi-tier-Architektur* können Fornfeist (2003, S. 23ff) entnommen werden.

3.5.3 Modularisierung von Komponenten in der Applikationsschicht

Die *Modularisierung* von *Komponenten* der *begriffsorientierten Infrastruktur* ist aus mehreren Gründen sinnvoll. Zunächst sind die Auswirkungen der *Kooperation* durch *Kommunikation*[116] zu nennen. Sie kann je nach Organisations- und Anwendungsstruktur einer Infrastruktur zu einer starken Vernetzung zwischen Komponenten führen, die nicht notwendigerweise aus einem *homogenen Hard- und Softwareumfeld* stammen. Beispielsweise sind Komponenten denkbar, die nur zeitweise eine *Netzwerkverbindung* aufweisen und sonst im *Offline-Modus* arbeiten. Desweiteren können nicht auf jedem physikalischen System *Datenbank- oder Content-Management-Systeme* als Massenspeicherabstraktionen betrieben werden (z.B. *Laptops*). Stattdessen werden einfache dateisystembasierte Abstraktionen mit geringeren Systemanforderungen benötigt.

Diese Überlegungen, zusammen mit dem *generativen* Ansatz der Komponentenerstellung aus *Asset-Modellen*, wie er bei Sehring[117] verfolgt wird, bilden eine Begründung für die Modularisierung von Komponenten. Würde der *Asset-Modell-Compiler* monolithische Komponenten generieren, dann müsste er eine kombinatorisch wachsende Zahl von *Compiler-Backends* unterhalten, die jede erdenkliche Zusammenstellung von Systemanforderungen abdeckt. *Wart-*

[116] Vgl. Abschnitt 3.5.1, Kooperative begriffsorientierte Systeme
[117] Sehring (2004), S. 88ff.

barkeit spielt zwar bei generierten Komponenten keine Rolle, wird jedoch auf die *Metaebene* des *Asset-Modell-Compilers* verlagert.

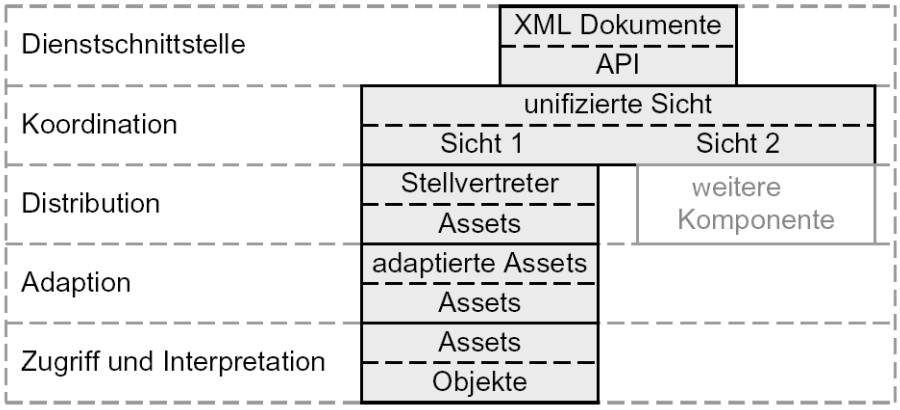

Abbildung 3.14: Modularisierung von Komponenten begriffsorientierter Systeme. Quelle: Sehring (2004, S. 121)

Der zweite Grund für die Modularisierung von Komponenten ist die Unterstützung der *Offenheit* und *Dynamik* des *begriffsorientierten Modells*. Diese resultiert in der Notwendigkeit *Modelladaptoren*, bzw. entsprechende Funktionaliät, während der Laufzeit der Komponente hinzuzufügen oder auszutauschen[118]. Ein monolithisches Kompilat müsste in diesem Fall komplett ersetzt und generiert werden, während in einer modularen Komponente lediglich Module generiert und hinzugefügt oder ersetzt werden. Die zweite Variante hat je nach *Komplexität* einer Komponente erhebliche Performanzvorteile, wobei sich die Komplexität aus einer Reihe von Parametern, wie Anzahl und Art der Kooperationen (beide Dimensionen), Geschwindigkeit der Modellevolution und Migrationsstrategie für „alte“ *Assets*, ergibt. Auf der Metaebene des *Asset-Modell-Compilers* skaliert dieser Ansatz besser als die monolithische Variante.

Als Folge dieser Überlegungen identifiziert Sehring[119] eine *vollständige* Menge von Modularten, die jede für sich eine *Basisfunktionalität* erbringt, und die zu jeder gewünschten Systemfunktionalität kombiniert werden können. Um die freie *Kombinierbarkeit* der Modularten zu gewährleisten, nutzen und bieten Modularten eine allgemeine *Schnittstelle*, die sich aus dem *Asset-Modell* ergibt. Diese Vorgehensweise wird in Abbildung 3.14 illustriert. In die gewählte Grobarchitektur eingeordnet, bilden die Module einer begriffsorientierten Komponente die *mittleren Schichten* oder auch zusammengefasst die Applikationsschicht eines *multi-tier Systems*[120].

Die konkrete Zusammenstellung einer Komponente, aus Ausprägungen der verschiedenen Modularten, wird durch den *Asset-Modell-Compiler* auf der Basis einer *Konfiguration* vorgenommen. Für jede Modulart stehen dem *Compiler* dabei alternative *Compiler-Backends* zur Auswahl, beispielsweise das *JDBC-* und *CAP-Backend*, zur Generierung von Modulen für den Zugriff auf relationale Datenbanken und das *Content-Management-System* CoreMedia CAP, als Ausprägungen des I-Moduls. Die verschiedenen Modularten und ihre Basisfunktio-

[118] Vgl. Abschnitt 2.4.4, Unterstützung der Kooperation subjektiver Modelle
[119] Sehring (2004), S. 119ff.
[120] Vgl. Abschnitt 3.5.2, Komponenten begriffsorientierter Systeme als multi-tier-Systeme

nalitäten werden im nachfolgenden Abschnitt behandelt.

Zum Abschluß sein noch darauf hingewiesen, dass der Austausch von Modulen zur *Laufzeit* einer Komponente zustandslose Module erfordert. Diese Anforderung harmoniert gut mit der Forderung nach zustandslosen Servern in verteilten Systemen, um deren *Skalierbarkeit* zu verbessern[121].

3.5.4 Modularten

Dieser Abschnitt stellt eine *vollständige Menge* von Modularten zur *individuellen Kombination* von Komponenten vor. Die identifizierten Modularten *Interpretationsmodul*, *Adaptionsmodul*, *Koordinationsmodul*, *Distributionsmodul* und *Dienstschnittstellenmodul* unterscheiden sich anhand ihres Leistungsumfangs, der nachfolgend kurz beschrieben werden soll. Weitere Details können[122] entnommen werden.

Interpretationsmodule

Interpretationsmodule (*I-Module*) realisieren den Zugriff auf *Standardkomponenten* in der *Datenschicht* der *multi-tier Architektur*. Dabei nutzen sie *Schnittstellen* der Standardkomponenten, beispielsweise die *JDBC-Schnittstelle* bei Datenbanksysteme oder das *Scripting API* bei der CoreMedia Content Application Platform.

Zusätzlich fungieren I-Module als *Wrapper*, die die jeweiligen Standardschnittstellen kapseln und deren Leistungen gemäß der Schnittstelle des *Asset-Modells* bereitstellen, die von allen Modularten geteilt wird.

Als drittes bilden I-Module die *Asset-Modelle* von Komponenten auf die jeweiligen *Datenabstraktionen* der Standardkomponenten ab. In Fällen, in denen die Leistungen einer Standardkomponente nicht für die *Interpretation* als *Asset-Modell* genügen, kann entweder das zuständige I-Modul reichhaltiger gestaltet werden oder die Leistungen mehrerer I-Module (bzw. gekappselter Standardkomponenten) können über ein Koordinationsmodul gebündelt werden.

Adaptionsmodule

Adaptionsmodule (*A-Module*) vollziehen die Anpassung der *Assets* eines Modells an ein anderes *Asset-Modell* aufgrund definierter *Adaptionsregeln*. Diese Leistung wird bei der *organisationspezifischen Kooperation*[123], also *Personalisierung* und *Publikation*, sowie bei der Unterstützung der *inkrementellen Modellevolution* und der Migration von *Assets*[124] innerhalb einer Komponente benötigt.

Wie bereits in Abschnitt 3.5.1 ausgeführt, werden Adaptionsmodule im Rahmen der Personalisierung und Publikation nur von den untergeordneten *Organisationseinheiten* eingesetzt, um eine bessere Verteilung der Lasten in der begriffsorientierten Infrastruktur und damit eine

[121] Coulouris et al. (1994).
[122] Sehring (2004).
[123] Vgl. Abschnitt 2.3.2, Organisationsspezifische Prozesse; 3.5.1, Kooperative begriffsorientierte Systeme
[124] Vgl. Abschnitt 2.4.2, Unterstützung von Modellevolution

bessere *Skalierbarkeit*[125] zu erreichen. Die genaue Vorgehensweise bei der Adaption von *Erst-, Zweit- und Drittheiten* innerhalb ihrer *Zeichenrelationsklasse*[126] oder beim Wechsel der Klasse werden in Sehring (2004) und Bachmann (2003) behandelt.

Koordinationsmodule

Koordinationsmodule (*C-Module*) sind Module zur *Delegation* und *Rekombination* von Anfragen an zwei untergeordnete Module. Sie werden insbesondere bei der Umsetzung der kooperativen Aspekte begriffsorientierter Inhaltsverwaltungssysteme benötigt. Zu Koordination von mehreren (> 2) untergeordneten Modulen, können C-Module geschachtelt werden.

Im Rahmen dieser Arbeit wird ein ähnliches Modul als Submodul der Dienstschnittstelle[127] eingesetzt, um *Teilwerke* zu delegieren und zu rekombinieren[128]. Im Gegensatz zu C-Modulen erlaubt dieses jedoch von vornherein die Koordination beliebig vieler Teilwerke. Auf weitere Unterschiede geht der nächste Absatz ein.

Die Tätigkeiten eines C-Moduls kann als Umsetzung des *Strategiemusters* in Gamma et al. (1995, S. 373ff) interpretiert werden, das für die *Kapselung* von Algorithmen verwendet wird. Der *Koordinationsalgorithmus* kann bei SEHRING[129] *deklarativ* beschrieben werden und ist Teil der *Konfiguration* einer *Komponente*, die gemäß eines *Asset-Modells* zusammengestellt wurde. Während der *Lebensdauer* einer Konfiguration ist die Strategie somit *statisch*. Im Gegensatz dazu, erhält das Submodul zur Werkdelegation und -rekombination in der vorliegenden Arbeit seine *Koordinationsstrategie* durch *präskriptive Werkbeschreibungen*. Die Strategie ist deshalb auch innerhalb einer Konfiguration *dynamisch*.

Distributionsmodule

Distributionsmodule (*D-Module*) dienen dem *räumlich entfernten* Zugriff auf *Assets* kooperierender Komponenten. Die Kooperation kann dabei im Rahmen *organisations- und anwendungsspezifischer Prozesse* stattfinden[130]. D-Module nehmen keine Änderungen an den übertragenen *Assets* vor, sondern bilden diese durch *Marshalling-* und *Unmarschalling-Operationen* exakt ab. Eine Anpassung der übertragenen *Assets*, z.B. im Rahmen der *Personalisierung*, erfolgt durch Kombination mit einem entsprechenden A-Modul auf Seiten der Komponente der *untergeordneten Organisationseinheit*.

D-Module bestehen aus zwei Teilen, von denen jedes einem Kooperationspartner zugeordnet ist. Genaugenommen gehört auch das *gekapselte* Netzwerk, mit seinen *Hard- und Software-Komponenten* zum D-Modul. Das gekapselte Netzwerk bleibt jedoch für die *Nutzer* eines D-Moduls verborgen (*transparent*) , was zusammen mit der Gestaltung des nachfolgend beschriebenen Dienstschnittstellenmoduls, die *Offenheit*[131] eines begriffsorientierten Systems ausmacht. Je nach *Einsatzumfeld*, können angepasste Netzwerk-Protokolle und zugehörige D-Module ausgewählt werden, ohne die restlichen Module einer Komponente zu beeinflussen.

[125] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit
[126] Vgl. Abschnitt 2.2.2, Inhalte
[127] Vgl. XMLWorker in Abschnitt 4.3.1, HTTP-Dienstschnittstelle
[128] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen
[129] Sehring (2004), S. 125f.
[130] Vgl. Abschnitt 3.5.1, Kooperative begriffsorientierte Systeme
[131] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Offenheit

Dienstschnittstellenmodule

Dienstschnittstellenmodule (*S-Module*) erfordern *Offenheit*, weil sie, im Gegensatz zu D-Modulen, primär für die Kommunikation mit Fremdsystemen, einschließlich *menschlicher Benutzer*, gedacht sind, über deren Interpretationsfähigkeit keine Aussage getroffen werden kann[132]. Aus diesem Grund bietet sich an dieser Stelle der Austausch von *XML-Dokumenten*, mit ihrer Fähigkeit zur *verbosen* Datenbeschreibung und zur Trennung von Daten und *Schema*, über offene und einfache Standardprotokolle an.

Über S-Module können beliebige Schnittstellen auf die interne, durch das *Asset-Modell* definierte Modulschnittstelle abgebildet werden. Insofern bilden S-Module das Gegenstück zu I-Modulen, die den Zugriff auf Standardkomponenten in der Datenschicht realisieren. Zudem können S-Module, je nach Art des zugreifenden Fremdsystems, den Übergang zur Präsentationsschicht der *multi-tier Architektur* darstellen.

Die vorliegende Arbeit nutzt für den Entwurf und die Implementierung des S-Moduls die Übertragung von XML-Dokumenten durch die *Post-Methode* des *HTTP-Protokoll*[133]. Damit ist die realisierte Schnittstelle besonders einfach und kann von Benutzern, die nur über einen *Webbrowser* verfügen, ebenso wie von anderen Komponenten der begriffsorientierten Infrastruktur, unmittelbar verwendet werden.

[132] Sehring (2004), S. 137.
[133] Vgl. Abschnitt 4.3.1, HTTP-Dienstschnittstelle

Kapitel 4

Implementation eines offenen und dynamischen begriffsorientierten Systems

In diesem Kapitel wird zunächst ein Überblick über verwendete Technologien für *Daten-, Applikations- und Präsentationsschicht* gegeben. Anschließend wird die Werkbeschreibungssprache mit der Extensible Markup Language (XML) umgesetzt. Zum Abschluß wird die *Diensteschnittstelle* als S-Modul für das HTTP-Protokoll in Java implementiert und die Umsetzung der Basisdienste mit den vorhandenen Mitteln der präskriptiven Werkbeschreibung vollzogen.

4.1 Technologie für die Systemimplementation

In Abschnitt 3.5 wurde eine Architektur für *begriffsorientierte Systeme* nach Sehring[1] vorgestellt, an der sich der vorliegende Prototyp orientiert. Diese kann wiederum als Ausschnitt einer *multi-tier Architektur* angesehen werden und ist dort in der oder den mittleren Schichten angeordnet (*Applikationsschicht(en)*). Die nachfolgende Aufstellung der verwendeten Technologien und Produkte orientiert sich deshalb an der klassischen Aufteilung der multi-tier Architektur in *Daten-*, *Applikations-* und *Präsentationsschicht*. Die Vorteile der verwendeten Architektur gegenüber der 2-tier Architektur können Abschnitt 3.5.2 oder Fornfeist (2003, S. 23ff) entnommen werden.

4.1.1 Standardtechnologie für die Datenschicht

In der Datenschicht des vorliegenden Prototypen eines begriffsorientierten Systems für *kooperatives Arbeiten* wird das Produkt CoreMedia Content Application Platform[2] eingesetzt, das sich durch hohe *Skalierbarkeit*[3] und einfache Bedienung auszeichnet. Dieses

[1] Sehring (2004), S. 113ff.
[2] CoreMedia (2004a).
[3] Vgl. Abschnitt 95, Ausgewählte Entwurfsziele verteilter Systeme - Skalierbarkeit

Produkt ist insbesondere für die Verwaltung von großen Beständen an multimedialen Inhalten geeignet und hat sich auf diesem Feld bereits vielfach bewährt[4]. Unter anderem wurde das Produkt auch in einer früheren Version in dem, als Fallstudie für begriffsorientiertes Arbeiten in Abschnitt 2.1 aufgeführten, Projekt WARBURG ELECTRONIC LIBRARY eingesetzt. Ein weiterer wichtiger Grund für die Verwendung von *CoreMedia CAP* kann dem nachfolgenden Abschnitt entnommen werden.

4.1.2 Technologie für die Applikationsschicht

Wie im Kapitel 3 mehrfach angesprochen, baut der vorliegende Prototyp auf der Arbeit über begriffsorientierte Systeme von SEHRING auf[5]. Diese umfasst ein begriffsorientiertes Modell, eine *Asset-Sprache* und einen *Asset-Modell-Compiler* der verschiedene Module generiert, die im Abschnitt 3.5.4 beschrieben werden.

Nach dieser Erklärung kann nun auch der zweite Grund für die Verwendung von *CoreMedia CAP* in der Datenschicht nachgeliefert werden. Aus der Arbeit von SEHRING war bereits ein entsprechendes *Compiler-Backend* vorhanden, das der Generierung von Interpretationsmodulen für den Zugriff auf *CoreMedia CAP* dient[6]. Dieses Modul greift über das sogenannte *Scripting-API* auf die CAP zu und bildet damit die Brücke zwischen Daten- und Applikationsschicht.

Nun handelt es sich bei den generierten Modulen nicht um direkt ausführbare Software, sondern um *Java-Quellkode*, der durch einen *Java-Compiler* in *Bytecode* übersetzt und in einer *Java Virtual Machine* ausgeführt werden kann[7]. Entsprechend verwendet der vorliegende Prototyp für die Implementation ebenfalls *Java*, um die Integration mit den generierten Modulen zu vereinfachen. Zudem kann der vorliegende Prototyp als *Machbarkeitsstudie* für ein entsprechendes *Compiler-Backend* angesehen werden, das die automatische *Generierung* von Modulen erlaubt, die kooperatives Arbeiten in der hier beschriebenen Weise unterstützen.

Nach den Ausführungen in Abschnitt 3.5.4 wird der Prototyp als Dienstschnittstellenmodul (S-Modul) eingestuft, das der Kommunikation mit *Akteuren* dient, über deren technologischen und *interpretatorischen* Fähigkeiten keine Aussagen getroffen werden können. Dementsprechend werden in SEHRING *offene Standardprotokolle* für die Anbindung von *Clients* vorgeschlagen[8]. Zudem sollte das verwendete Datenformat die deklarative Beschreibung der Daten erlauben. Der Prototyp verwendet entsprechend dieser Empfehlungen die Übertragung von *XML-Dokumenten*[9] über das *HTTP-Protokoll*[10].

Für die performante Verarbeitung von *HTTP-Requests* wird der APACHE WEBSERVER[11] in Kombination mit dem *Servlet-Container* JAKARTA TOMCAT[12] eingesetzt, der ebenfalls auf *Java-Technologie* beruht.

[4] CoreMedia (2004b).
[5] Sehring (2004).
[6] Sehring (2004), S. 87ff.
[7] Sun Microsystems, Inc..
[8] Sehring (2004), S. 127f.
[9] W3C (2004a).
[10] W3C (2004b).
[11] ASF (2004e).
[12] ASF (2004a).

4.1.3 Standardtechnologie für die Präsentationsschicht

Das im Abschnitt zuvor beschriebene *S-Modul* bildet im implementierten Prototypen die Brücke von der *Applikationsschicht* zur *Präsentationsschicht*. Da die vorliegende Arbeit ihren Fokus auf der Untersuchung der *kooperativen Arbeit* von *Komponenten* hat, ist die weitere *visuelle* Aufbereitung von übertragenen Werkbeschreibungen nicht Teil der Umsetzung. Entsprechend erfolgt der Zugriff durch menschliche *Akteure* über einfache aber dafür sehr verbreitete *HTTP-Clients* wie *Webbrowser*.

4.2 Prozessbeschreibungssprache für Asset-Prozesse

In diesem Abschnitt soll nun eine deklarative *Werkbeschreibungssprache* eingeführt werden, mit der *präskriptive* Anteile von Werkbeschreibungen formuliert werden können, die von der nachfolgend vorgestellten *HTTP-Dienstschnittstelle*[13] verarbeitet werden.

Im Folgenden werden die Elemente der Werkbeschreibungssprache aufeinander aufbauend und anhand von Beispielen eingeführt. Die Beispiele sind in einer *XML-artigen* Syntax formuliert, die jedoch von unnötigem Balast befreit ist, der in diesem Zusammenhang nicht zielführend ist. Die Beispiele stellen also keine validen *XML-Dokumente* dar, auch wenn dies im nachfolgenden praktischen Einsatz vonnöten ist. Der Grund für die Verwendung von *XML* als Werkbeschreibungsformat liegt in der Fähigkeit zur *verbosen* Datenbeschreibung und zur Trennung von Daten und *Schema*. Auf diese Weise können Werkbeschreibungen auch mit Elementen versehen werden, die nicht für die *Interpretation* durch den Werkausführenden vorgesehen sind, sondern der Interpretation durch einen menschlichen *Akteur* dienen und somit den *analytischen* Wert einer Werkbeschreibung erhöhen.

Werkbeschreibung

Zunächst besteht eine Werkbeschreibung aus einem *Tag* `work-prescription`, der die umschlossenen Elemente als Teile einer Werkbeschreibung ausweist. Darüberhinaus besitzt der `Tag` einige Attribute wie `id` und `executor`, die zusammen einen eindeutigen Schlüssel für den *Werkauftrag* bilden. `executor` ist dabei eine `logische Adresse` des Werkausführenden, die u.a. zu Überwachungszwecken gebunden wird. Desweiteren ist eine Domäne spezifiziert (`domain`), die aber lediglich eine Empfehlung für die Zieldomäne des Werkauftrages darstellt. Zuletzt ist ein Name (`name`) für den vorliegenden Auftrag angegeben, der im Falle der Ableitung von einem `Werk-Prototypen` den Namen des Prototypen enthält.

```
<work-prescription  id="H6dJS752"
                    executor="/PI/STS/JF_SERVER"
                    domain="PI"
                    name="getSubTopics">
...
</work-prescription>
```

Nachdem nun die „Hülle“ für eine Werkbeschreibung deklariert wurde, kann das Innere mit Elementen gefüllt werden, die sich entweder an die Werkausführenden richten oder als Beschreibung für menschliche *Akteure* gedacht sind. Im folgenden werden die vier Elemente

[13] Vgl. Abschnitt 4.3.1, HTTP-Dienstschnittstelle

beschrieben, die die Ausdrücke der *Asset-Anfrage- und Manipulationssprache* repräsentieren und zur lokalen Interpretation durch den Werkausführenden geeignet sind. Mit Hilfe dieser Grundelemente, ergänzt um nachfolgend diskutierte Sprachkonzepte, werden später die Werkbeschreibungen aller *nichtelementaren Anwendungsfälle* aus Abschnitt 3.2.2 ausgedrückt.

Asset-Erzeugung (`create`)

Der `create` Tag dient der Erzeugung von *Assets*. Er enthält die Attribute `domain` und `type`, das den *Asset-Typ* der zu erzeugenden Assets verbindlich für alle untergeordneten Elemente des Tags festlegt. Es können an dieser Stelle nur *Asset-Typen* der lokalen Domäne interpretiert werden. Sollte dies nicht zutreffen, so wird der Werkauftrag vom Werkausführenden als sogenanntes *ad-hoc Werk* weitergereicht[14].

Es gibt zwei Arten der Erzeugung von *Assets*. Zum einen können *Assets* durch Angabe einer Menge von Prototypen erzeugt werden, die als Kopiervorlage für die neuen *Assets* verwendet werden. Dies geschieht durch Unterordnung von `reference`-Tags, die durch den Werkausführenden gemäß des `type`-Attributs interpretiert werden. Die zweite Möglichkeit der Erzeugung von *Assets*, ist die explizite Angabe von Bindungen für die Charakteristika und Beziehungen des neuen *Assets* durch eine Menge von `characteristic`- oder `relationship`-Tags. Das `characteristic`-Tag verfügt über die Attribute `name` und `value`. Der Wert des `value`-Attributs wird dabei gemäß dem Zieltyp des Charakteristikums, mit dem entsprechenden Namen im *Asset-Modell*, interpretiert.

```
<create domain="PI" type="Topic">
    ...
    <reference>/PI/STS/JF_SERVER:Topic:3572</reference>
    <reference>/PI/STS/JF_SERVER:Topic:3571</reference>
    ...
    <!-- *** XOR *** -->
    ...
    <characteristic name="name" value="My_Topic"/>
    <relationship name="subtopics">
        <reference>/PI/STS/JF_SERVER:Subtopic:1378</reference>
        <reference>/PI/STS/JF_SERVER:Subtopic:1562</reference>
        ...
    </relationship>
    ...
</create>
```

Die Bindung einer Beziehung erfolgt über das `relationship`-Tag, dass ein `name`-Attribut enthält und dem wiederum eine Menge von `reference`-Tags untergeordnet sind. Die Referenzen werden gemäß des Zieltyps der Beziehung, mit dem entsprechenden Namen (`name`-Attribut) im *Asset-Modell*, interpretiert. Die Auflösung erfolgt, innerhalb einer *Komponente*[15] der Infrastruktur, über den Typ des *Assets* und eine implizit oder explizit modellierte, eindeutige `id`. In der vorliegenden Arbeit werden die `ids` stets explizit und unter Angabe einer Bedingung, die die Eindeutigkeit des Schlüssels sicherstellt, definiert. Entfernte Referenzen werden in einem nachfolgenden Unterabschnitt behandelt. Sollte die *Kardinalität* der Beziehung, durch die Anzahl der `reference`-Tags, überschritten werden, so werden die überschüssigen Referenzen ignoriert.

[14] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen
[15] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems

Asset-Abfrage (`lookfor`)

Der `lookfor`-Tag dient dem Auffinden von *Assets*. Er enthält die Attribute `domain` und `type`, wobei letzteres den *Asset-Typ* der aufzufindenden Assets verbindlich für alle untergeordneten Elemente des Tags festlegt. Es können an dieser Stelle nur *Asset-Typen* der lokalen Domäne interpretiert werden. Sollte dies nicht zutreffen, so wird der Werkauftrag vom Werkausführenden als sogenanntes *ad-hoc Werk* weitergereicht[16].

Es gibt zwei Arten der Suche nach *Assets*. Zum einen können *Assets* durch Angabe einer Menge von Beispielen gesucht werden. Dies geschieht durch Unterordnung von `reference`-Tags, die durch den Werkausführenden gemäß des `type`-Attributs interpretiert werden. Deren gebundene Charakteristika und Beziehungen werden als Bedingungen für die Suche verwendet. Die zweite Möglichkeit der Suche nach *Assets*, ist die explizite Angabe von Bedingungen für die Charakteristika und Beziehungen der gesuchten *Assets* durch eine Menge von `constrain-char`- oder `constrain-rel`-Tags, die als *Teilterme* über Termkonnektoren (`connector`-Tag) miteinander verknüpft werden. Das `constrain-char`-Tag verfügt über die Attribute `name`, `operator` und `value`. Die Auswahl des Wertes für das `operator`-Attribut hängt dabei vom Typ des Charakteristikums, mit dem Namen `name` im *Asset-Modell*, ab. Das gleiche gilt für die Interpretation des Wertes des `value`-Attributs. Mögliche Operatoren sind z.B. `less` oder `equal` bei einem Typ `Integer` des Charakteristikum.

```
<lookfor domain="PI" type="Topic">
    ...
    <reference>/PI/STS/JF_SERVER:Topic:3572</reference>
    <reference>/PI/STS/JF_SERVER:Topic:3571</reference>
    ...
    <!-- *** XOR *** -->
    ...
    <connector log-op="or">
        <connector log-op="and">
            <constrain-char name="name"
                            operator="equals"
                            value="Herrscherbild"/>
            <constrain-rel name="subtopics" operator="references">
                <reference>/PI/STS/JF_SERVER:Subtopic:1378</reference>
                <reference>/PI/STS/JF_SERVER:Subtopic:1562</reference>
                ...
            <constrain-rel/>
        </connector>
        <connector log-op="not">
            <constrain-char name="name" operator="equals" value="Landschaft"/>
        </connector>
    </connector>
    ...
</lookfor>
```

Die Einschränkung einer Beziehung erfolgt über das `constrain-rel`-Tag, das ein `name`- und ein `operator`-Attribut enthält und dem wiederum eine Menge von `reference`-Tags untergeordnet sind. Die möglichen Werte für das `operator`-Attribut sind diesem Fall auf `references` und `referenced` beschränkt. Die angegebenen Referenzen werden gemäß des Zieltyps der Beziehung, mit dem Namen `name` im *Asset-Modell*, interpretiert. Wie dies genau geschieht, wurde bereits bei der Beschreibung des `create`-Tag erörtert.

Die Verknüpfung der einzelen `constrain-char`- und `constrain-rel`-Tags erfolgt, wie be-

[16] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen

reits erwähnt, über den `connector`-Tag, der mit dem Attribut `log-op` versehen ist. Dieses kann die Werte `and, or` und `not` annehmen. Da die vorliegende *deklarative Syntax* keine *Sequenzen* kennt, müssen die logisch zu verknüpfenden Teilterme den entsprechenden Termkonnektoren untergeordnet werden. Bei den *binären logischen Operationen* `and` und `or` handelt es sich natürlich um zwei untergeordnete Teilterme, während `not` nur ein Teilterm ungeordnet ist. Die Teilterme können über die Konnektoren beliebig tief geschachtelt werden. Die Vorgehensweise, *Schachtelung* als Ausdrucksmittel für *Abhängigkeiten* zwischen den Elementen der Werkbeschreibungssprache einzusetzen, wird in einem späteren Unterabschnitt wieder aufgegriffen.

Asset-Manipulation (`update`)

Der `update`-Tag dient dem Manipulieren von *Assets*. Er enthält die Attribute `domain` und `type`, wobei letzteres den *Asset-Typ* der zu manipulierenden Assets verbindlich für alle untergeordneten Elemente des Tags festlegt. Es können an dieser Stelle wie üblich nur *Asset-Typen* der lokalen Domäne interpretiert werden. Sollte dies nicht zutreffen, so wird der Werkauftrag vom Werkausführenden als sogenanntes *ad-hoc Werk* weitergereicht[17].

```
<update domain="PI" type="Topic">
    ...
    <reference>/PI/STS/JF_SERVER:Topic:3572</reference>
    <reference>/PI/STS/JF_SERVER:Topic:3571</reference>
    ...
    <characteristic name="name" value="My_Topic"/>
    <relationship name="subtopics">
        <reference>/PI/STS/JF_SERVER:Subtopic:1378</reference>
        <reference>/PI/STS/JF_SERVER:Subtopic:1562</reference>
        ...
    </relationship>
    ...
</update>
```

Der `update`-Tag enthält zwei Bereiche. Zum einen eine Menge von untergeordneten `reference`-Tags, die auf die übliche Art und Weise über das `type`-Attribut interpretiert werden. Sie Grenzen die Menge der zu manipulierenden *Assets* ein. Der zweite Bereich beschreibt das *Wie* der Manipulation, durch Angabe einer Menge von `characteristic`- und `relationsship`-Tags. Deren Deutung erfolgt analog zum `create`-Tag.

Asset-Zerstörung (`delete`)

Der `delete`-Tag dient der Zerstörung von *Assets*. Er enthält die Attribute `domain` und `type`, wobei letzteres den *Asset-Typ* der zu manipulierenden Assets verbindlich für alle untergeordneten Elemente des Tags festlegt. Die Einschränkung auf *Asset-Typen* der lokalen Domäne gilt wie üblich. Die einzigen Element des Tags sind untergeordnete `reference`-Tags, die die Menge der zu löschenden *Assets* eingrenzen. Die Interperation erfolgt wie bei den anderen Grundelementen der Werkbeschreibungssprache.

```
<delete domain="PI" type="Topic">
    ...
    <reference>/PI/STS/JF_SERVER:Topic:3572</reference>
    <reference>/PI/STS/JF_SERVER:Topic:3571</reference>
```

[17] Vgl. Abschnitt 3.4, Kooperation verteilter Dienste durch Delegation von Teilprozessen

```
    ...
</delete>
```

Datenfluß, Nebenläufigkeit und Schachtelung

Die Schachtelung der vorgenannten Tags dient der Definition einer Abhängigkeitsbeziehung zwischen diesen. Ein untergeordneter Tag wird erst ausgeführt, wenn der übergeordnete Tag ausgeführt und die dabei die offenen Bindungen geschlossen hat. Im nachfolgenden Beispiel wird das `lookfor`-Tag also vor dem `delete`-Tag ausgeführt. Insgesamt wird durch die Abhängigkeitsbeziehung von Tags ein in Richtung der Wurzel *gerichteter Baum* aufgespannt. Tags zwischen denen es keinen (gerichteten) *Pfad* im Baum gibt, sind unabhängig und können *nebenläufig* ausgeführt werden.

```
<lookfor domain="PI" type="Subtopic">

    <constrain-char name="name" operator="equals" value="Herrscher zu Pferd"/>

    <lookfor domain="PI" type="Topic">
        <connector log-op="and">
            <constrain-char name="name"
                            operator="equals"
                            value="Herrscherbild"/>
            <constrain-rel name="subtopics" operator="references">
                <reference>@SELF</reference>
            </constrain-rel>
        </connector>

        <delete domain="PI" type="Topic">
            <reference>@SELF</reference>
        </delete>

        <!-- some analytical output -->
        <deleted>id: @id</deleted>

    </lookfor>

</lookfor>
```

Sichtbarkeitsbereiche, offene Bindungen und Verweise

Es gibt einen *globalen Sichtbarkeitsbereich* in einer Werkbeschreibung der durch das äußere `work-prescription`-Tag gebildet wird. Innerhalb dieses Sichbarkeitsbereiches stellen die vier *Grund-Tags* jeweils einen *lokalen Sichtbarkeitkeitsbereich* dar. Im Beispiel im Unterabschnitt Datenfluß, Nebenläufigkeit und Schachtelung befindet sich das `delete`-Tag im Sichtbarkeitsbereich des `lookfor`-Tags und hat demzufolge Zugriff auf die Charakteristika und Beziehungen des *Assets*, das sich als Ergebnis des `lookfor`-Tags ergeben hat und an den Bereich gebunden ist. Ein untergeordnetes Tag innerhalb des `delete`-Tags hat dagegen keinen Zugriff auf das *Ergebnis-Asset* von `lookfor`. Die lokalen Sichtbarkeitsbereiche verdecken sich also jeweils. Auf *Assets*, die an Sichtbarkeitsbereiche gebunden sind, kann nur lesend zugegriffen werden. Schreibende Zugriffe müssen über die Tags `create`, `update` und `delete` ausgeführt werden.

Offene Bindungen werden durch das Symbol `@` verbunden mit einem Verweis auf Charakteristikum oder Beziehung des lokal sichtbaren Assets versehen. Im Beispiel aus dem Unterab-

schnitt über Datenfluß, Nebenläufigkeit und Schachtelung ist dies u.a. der Ausdruck `@id`. Der Ausdruck `@SELF` stellt eine Besonderheit dar, die im Unterabschnitt über lokale und globale Referenzen behandelt wird.

An den globalen Sichbarkeitsbereich des `work-prescription`-Tag werden keine *Assets* gebunden. Stattdessen kann mit Ausdrücken der beschriebenen Form auf Parameter zugegriffen werden, die innerhalb der Werkbeschreibung über einen `parameter`-Tag deklariert wurden. Dieser verfügt über die Attribute `key` und `value`. Nachfolgend wird ein Beispiel für diese Vorgehensweise gegeben. Die dargestellte Methode ist insbesondere für die *Parameterweitergabe* an untergeordnete Werkbeschreibungen geeignet, die im Unterabschnitt Teilwerke beschrieben wird. Wenn mehrere `parameter`-Tags mit dem gleichen `key`-Attribut übergeben werden, so bilden diese eine Menge. Beim Zugriff auf diese Parametermenge über `@`-Ausdrücke wird genauso verfahren wie bei *Asset-Mengen*. Die Mengensemantik wird im nachfolgenden Abschnitt verdeutlicht.

```
<work-prescription  ... >
    <parameter key="topic_name" value="Herrscherbild"/>

    <connector log-op="and">
        <constrain-char name="name"
                        operator="equals"
                        value="@topic_name"/>

        <delete domain="PI" type="Topic">
            <reference>@SELF</reference>
        </delete>

    </lookfor>

</work-prescription>
```

Mengensemantik

Im Unterabschnitt Datenfluß, Nebenläufigkeit und Schachtelung wurde ein Detail aus Vereinfachungsgründen ausgelassen, das nun beachtet wird. Das `lookfor`-Tag kann, wie alle anderen *Grund-Tags* auch, Mengen von *Assets* zurückliefern. Dann lautet allerdings die Frage wie mit diesen Mengen umgegangen bzw. wie mit `@`-Ausdrücken auf Charakteristika und Beziehungen verwiesen werden kann. Wird eine Werkbeschreibung so wie in dem genannten Beispiel formuliert, dann wird durch `@`-Ausdrücke auf das erste Element der Menge verwiesen. Dies ist jedoch nicht immer erwünscht. Sollen alle *Assets* der Menge genutzt werden, so muss der `reference`-Tag im Beispiel mit einem besonderen Iteratorsymbol `@ITER` markiert werden. Der Tag wird dann entsprechend der Kardinalität der *Asset-Menge* vervielfältigt und an Stelle des alten Tags in die Werkbeschreibung eingefügt. An jeden dieser neuen Tags wird ein *Asset* der *Asset-Menge* gebunden. Innerhalb der neuen Tags können nun wie üblich Verweise auf Charakteristika und Beziehungen stehen. Es können auch mehrere Tags in einem Sichtbarkeitsbereich mit dem Iteratorsymbol markiert werden. Die genannte Vorgehensweise wird auf jeden dieser Tags angewandt und einzelne *Assets* mehrfach gebunden. Aus Gründen der Validität der Werkbeschreibung in `XML` ist es nötig das Iteratorsymbol in ein beliebiges Attribut zu verpacken.

```
<lookfor domain="PI" type="Topic">

    <constrain-char name="name" operator="equals" value="Herrscherbild"/>
```

```
    <delete domain="PI" type="Topic">
        <reference dummy="@ITER">@SELF</reference>
    </delete>

</lookfor>
```

Geschützte Anweisungen

Der `guard`-Tag kann jedem Grund-Tag und geschachtelten `work-prescription`-Tags hinzugefügt werden, um eine bedingte Ausführung auszudrücken. Ebenso wie beim `lookfor`-Tag können hier auch längere, über logische Operatoren verknüpfte (`constrain` u. `connector`-Tag) logische Terme ausgedrückt werden. Die angegebenen Werte für die linke (`left`-Attribut) und rechte Seite (`right`-Attribut) eines `constrain`-Tags werden entsprechend der Typen des Operators (`operator`-Attribut) interpretiert. Optional kann auf den Wert des Attribut `left` noch ein regulärer Ausdruck (`left-reg-exp`-Attribut) angewandt werden, dessen erstes Klammerpaar an den deklarierten Operator weitergereicht wird.

```
<lookfor domain="PI" type="Topic">
    <guard>
        <connector log-op="and">
            <constrain  left="..."
                        operator="equals"
                        rigth="..."/>
            <constrain  left="..."
                        left-reg-exp="(*)"
                        operator="equals"
                        rigth="..."/>
        </connector>
    </guard>

    <constrain-char name="name"
                    operator="equals"
                    value="Herrscherbild"/>

</lookfor>
```

Ausnahmebehandlung

Der `exception`-Tag kann jedem Grund-Tag zugeordnet werden, um eine Ausführung der untergeordneten Tags im Ausnahmefall zu erreichen. Der Sichtbarkeitsbereich des Tags, der den `exception`-Tag enthält, ist für die im `exception`-Tag enthaltenen Tags zugreifbar.

```
<lookfor domain="PI" type="Topic">
    <exception>

            <lookfor ...>

                ...

            </lookfor>

    </exception>

    <constrain-char name="name"
                    operator="equals"
                    value="Herrscherbild"/>
```

```
</lookfor>
```

Teilwerke

Unterhalb von `work-prescription`-Tags oder Grund-Tags können wiederum `work-prescription`-Tags angeordnet sein. Diese können bereits expandiert sein (sogenannte *ad-hoc Werke*) oder nur einen Verweis auf einen Prototypen in Form des `name`-Attributs enthalten. Über das `parameter`-Tag können diese Teilwerken Parameter übergeben werden. Wird ein `work-prescription`-Tag einem Grund-Tag untergeordnet, so kann es ebenso wie andere Grund-Tags, per `@`-Ausdruck auf Charakteristika und Beziehungen von gebundenen *Assets* verweisen.

```
<work-prescription domain="PI" ... >
    <parameter key="topic_name" value="Herrscherbild"/>
    <parameter key="subtopic_name" value="Herrscher zu Pferd"/>

    <lookfor domain="PI" type="Subtopic">

        <constrain-char name="name"
                        operator="equals"
                        value="@subtopic_name">

        <lookfor domain="PI" type="Topic">
            <connector log-op="and">
                <constrain-char name="name"
                                operator="equals"
                                value="@topic_name"/>
                <constrain-rel  name="subtopics"
                                operator="references">
                    <reference dummy="@ITER">@SELF</reference>
                </constrain-rel>
            </connector>

            <!-- *** expand via prototype *** -->
            <work-prescription  domain="PI"
                                name="deleteTopic">
                <parameter key="topic_ref" value="@SELF" dummy="@ITER">
            </work-prescription

            <!-- *** OR *** -->

            <!-- *** ad-hoc *** -->
            <work-prescription  domain="PI"
                                name="deleteTopic">
                <parameter key="topic_ref" value="@SELF" dummy="@ITER"/>

                <delete domain="PI" type="Topic">
                    <reference dummy="@ITER">@topic_ref</reference>
                </delete>

            </work-prescription

        </lookfor>

    </lookfor>

</work-prescription>
```

Lokale und entfernte Referenzen

In vielen vorangegangenen Beispielen wurde bereits der Ausdruck `@SELF` verwendet. Dieser wird bei der Bindung durch eine Repräsentation einer Referenz auf ein *Asset* ersetzt. Die Repräsentation besteht in der vorliegenden Implementation aus der Verkettung von logischer Adresse der *Heimatkomponente*, dem Typ und der *Id* des *Asset*. Bei entfernten Referenzen wird zudem eine *ad-hoc* Werkbeschreibung erzeugt und an einen zuständigen *Werkausführenden* delegiert. Desweiteren wird bei Zuweisungen an Beziehungen ein lokaler Stellvertreter (*Proxy*) im *Asset-Modell* erzeugt, bzw. ein bereits vorhandener Stellvertreter genutzt.

4.3 Dienste in der Applikationsschicht

In diesem Abschnitt wird die Implementation der *HTTP-Dienstschnittstelle* als gemeinsame Schnittstelle aller Basisdienste[18] in der Applikationsschicht behandelt. Anschließend wir in 4.3.2 auf die Besonderheiten eingegangen, die sich im Zusammenhang mit der *client-* und *server-seitigen* Replikation von Werkbeschreibungen in der Dienstschnittstelle ergeben. Zum Abschluß zeigt 4.3.3 die Umsetzung der Basisdienste auf Basis der implementierten *HTTP-Dienstschnittstelle* anhand eines exemplarischen Anwendungsfalles des *Nachrichtenverwaltungsdienstes*[19].

4.3.1 HTTP-Dienstschnittstelle

Wie bereits erwähnt, kann der vorliegende Prototyp als *Machbarkeitsstudie* für ein entsprechendes *Compiler-Backend* des *Asset-Modell-Compilers*[20] angesehen werden, das die automatische *Generierung* von *S-Modulen*[21] erlaubt, die kooperatives Arbeiten in der hier beschriebenen Weise unterstützen. Demzufolge sind die nachfolgend beschriebenen Bausteine als Submodule eines S-Moduls anzusehen. Eine grafische Darstellung der Verteilung und Zusammenarbeit der Submodule und Module kann Abbildung 4.1 entnommen werden.

Das *server-seitige* S-Modul nimmt *HTTP-Post-Requests* von dem *HTTP-Adapter* des *client-seitigen* S-Moduls entgegen. Der *Request* enthält eine Werkbeschreibung[22] in *XML-Format*[23]. Der *HTTP-Adapter* hat zuvor eine Komponente für die Kooperation über den *Namens- und Verzeichnisverwaltungsdienst*[24] ermittelt und anschließend eine eindeutige `id` für den Werkauftrag aus der *lokal gebundenen* Werkbeschreibung generiert (*Hashwert*). Danach wird der Werkauftrag beim *Werkverwaltungsdienst*[25] angemeldet und auf Verklemmungsfreiheit überprüft. Wird keine zukünftige Verklemmung detektiert, dann wird der Auftrag an die kooperierende Komponente in Form eines *HTTP-Requests* verschickt. Auf beiden Seiten der Kooperation können bei diesem Vorgang auch Replikationsmechanismen greifen, die im nachfolgenden Abschnitt erläutert werden.

[18] Vgl. Abschnitt 3.2.2, Basisdienste für den Systembetrieb
[19] Vgl. Abschnitt 3.2.7, Nachrichtenverwaltungsdienst
[20] Sehring (2004), S. 87ff.
[21] Vgl. Abschnitt 3.5.4, Modularten
[22] Vgl. Abschnitt 233, Werkbeschreibung
[23] W3C (2004a).
[24] Vgl. Abschnitt 3.2.6, Nachrichten- und Verzeichnisverwaltungsdienst
[25] Vgl. Abschnitt 3.2.5, Werkverwaltungsdienst

Der *Request* wird von *Webserver* an den nachgelagerten *Servlet-Container* weitergeleitet, der die zuständige Methode am *HTTP-Servlet* aufruft. Dieses leitet die enthaltene Werkbeschreibung an den *XMLWorker* weiter. Nun erfolgt die Zerlegung der Werkbeschreibung mit Hilfe eines *Tag-Handlers*, der über eine Aufstellung der verfügbaren *Werkbeschreibungselemente* und der zulässigen *Asset-Typen* verfügt. Werden Grundelemente ermittelt, die wegen eines unzulässigen Typs nicht lokal ausgeführt werden können, dann werden diese in *ad-hoc Werke* gekapselt. Die nichtexpandierten Teilwerke werden mit Hilfe des *HTTP-Adapters*, der *Prototypen* vom Werkverwaltungsdienst abruft, expandiert.

Anschließend werden alle möglichen lokalen Bindungen mit Hilfe des bereits erwähnten *Tag-Handlers* vorgenommen, der zu diesem Zweck auf das *Interpretationsmodul*[26] zugreift. Dieses ist durch einen *Asset-Modell-Compiler* aus dem entsprechenden *Asset-Modell* generiert und stellt eine *Asset-Schnittstelle* für den Zugriff auf das Modell zur Verfügung.

Nachdem die lokalen Bindungen erfolgt sind, können die vorhandenen Teilwerke an den *HTTP-Adapter* weitergereicht werden, der diese, wie am Anfang dieses Abschnittes beschrieben, an andere Komponenten delegiert. Eine Besonderheit ist noch zu erwähnen: Während der Herstellung der lokalen Bindungen werden die zugegriffenen *Assets* über ihre eindeutige `id` protokolliert. Diese Informationen werden, neben den Informationen über die Werkabhängigkeiten, ebenfalls im Werkverwaltungsdienst vorgehalten. Darüberhinaus wird jeder Werkfortschritt einer Werkausführung im Werkverwaltungsdienst persistiert. Dies dient dem Zweck einen fehlertoleranten Server zu erreichen, der nach Systemabstürzen seine Arbeit wieder aufnehmen kann, indem er seine Werkaufträge vom Werkverwaltungsdienst abruft[27]. Entsprechend werden Nachrichten, die von der *CoreMedia CAP*[28] an das I-Modul gegeben werden, nicht lokal verarbeitet, sondern über den *HTTP-Adapter* an den Nachrichtenverwaltungsdienst weitergereicht. Dieser ermittelt mit Hilfe des Werkverwaltungsdienstes das betroffene Werk und plaziert eine Nachricht im Nachrichtenkanal des Werkausführenden.

4.3.2 Replikation

Die Replikation von Werkbeschreibungen wird im vorliegenden Prototypen intensiv genutzt, um die großen Kommunikationskosten, die mit der starken Verteilung der Dienste und Komponenten einhergeht, beherrschen zu können. Es werden *client-* und *server-seitige* Replikationsmechanismen eingesetzt, die beide Replikate von Werkbeschreibungen im *XML-Format*[29] zwischenspeichern, um Produktions- und Kommunikationskosten zu sparen.

In Abbildung 4.1 ist *client-seitig* ein *Cache* zu erkennen, der vom *HTTP-Adapter* genutzt wird. Der Adapter füllt den *Cache* mit Replikaten von Werkbeschreibungen und stößt auch deren Invalidierung an, indem er periodisch Nachrichten beim Nachrichtenverwaltungsdienst abruft und diese entsprechend auf seinen *Cache* anwendet. Anschließend werden die Invalidierungsnachrichten an den *XMLWorker* weitergeleitet, weil zu invalidierende Werkbeschreibungen sowohl im *client-seitigen* als auch im *server-seitigen Cache* liegen können, wenn Komponenten sich selbst beauftragt haben. Die Verwendung des *client-seitigen Cache* ist für die Submodule des *S-Moduls* transparent.

[26] Vgl. Abschnitt 3.5.4, Modularten
[27] Vgl. Abschnitt 101, Ausgewählte Entwurfsziele verteilter Systeme - Fehlertoleranz
[28] CoreMedia (2004a).
[29] W3C (2004a).

Server-seitig wird diese Aufgabe mit Hilfe des *Rewrite-Moduls*[30] des *Apache Webservers* erledigt. Das *Rewrite-Modul* ist so konfiguriert, das zunächst im Dateisystem nach einer Datei mit der eindeutigen `id` eines Werkauftrags als Name gesucht wird. Wird diese gefunden, so wird der Inhalt der Datei als Antwort an den *Client* zurückgeschickt ohne die Anfrage an den *Servlet-Container* weiterzureichen. Ansonsten wird die Anfrage wie üblich an den *Servlet-Container* weitergegeben. Der Schlüssel für den Einsatz dieser Technik ist die folgende Direktive im *Rewrite-Modul*:

```
RewriteCond /<some_path>/<cache-ctx>/<work-id> !-f
```

Diese überprüft die Existenz einer Datei mit dem gegebenen Pfad und Namen im Dateisystem. Im vorliegenden negierten Beispiel wird mit der Regelauswertung fortgefahren, wenn die Datei nicht existiert. In diesem Falle wird dann der *Request* an den *Servlet-Container* weitergereicht.

Die Invalidierung des *Cache* bzw. das Überschreiben der Replikate mit neuen Versionen der Werkbeschreibungen wird durch das Submodul *XMLWorker* über den *Dateisystemcache* angestoßen.

4.3.3 Implementation der Basisdienste

Die Implementation der *Basisdienste* ist bereits zum großten Teil durch den Entwurf der entsprechenden *Asset-Modelle* in der *Asset-Definitionssprache*[31] in Abschnitt 3.2.2 erfolgt. Die zugehörigen *Interpretationsmodule* werden durch den *Asset-Modell-Compiler* in Form von *Java-Quellkode* erzeugt. Anschließend werden dieser durch einen herkömmlichen *Java-Compiler* in *Bytecode* übersetzt und in einer *Java Virtual Machine* ausgeführt[32].

Die eigentliche Implementationsarbeit für die Basisdienste besteht anschließend nur noch darin, die beschriebenen *nichtelementaren Anwendungsfälle*[33] in der *Werkbeschreibungssprache* aus Abschnitt 4.2 zu formulieren. Die soll nachfolgend am Beispiel des Anwendungsfalls `Nachrichten einspeisen` des Nachrichtenverwaltungsdienstes gezeigt werden.

Listing 4.1: Anwendungsfall `Nachrichten einspeisen` als Werkbeschreibung

```
<work-prescription domain="WorkManagement" name="sendAssetMessage">
    <parameter key="asset_id" value="@id"/>
    <parameter key="asset_type" value="@type"/>
    <parameter key="asset_nds_path" value="@nds_path"/>
    <parameter key="message_subject" value="@subject"/>

    <lookfor domain="WorkManagement" type="Asset">
        <connector log-op="and">
            <constrain-char name="id" operator="equals" value="@asset_id"/>
            <connector log-op="and">
                <constrain-char name="id"
                                operator="equals"
                                value="@asset_id"/>
                <constrain-char name="nds_path"
                                operator="equals"
                                value="@asset_nds_path"/>
            </connector>
        </connector>
```

[30] ASF (2004b).
[31] Sehring (2004), S. 39ff.
[32] Sun Microsystems, Inc..
[33] Vgl. Abschnitt 3.2.2, Basisdienste für den Systembetrieb

```
        <parameter key="asset_ref" value="@SELF" dummy="@ITER"/>

        <lookfor domain="WorkManagement" type="Exemplar">
            <constrain-rel name="assets" operator="referenced">
                <reference>@SELF</reference>
            </constrain-rel>

            <lookfor domain="DirectoryManagement" type="Directory">
                <constrain-rel name="exemplars" operator="referenced">
                    <reference dummy="@ITER">@SELF</reference>
                </constrain-rel>

                <lookfor domain="MessageMangement" type="Channel">
                    <constrain-rel name="directories" operator="referenced">
                        <reference dummy="@ITER">@SELF</reference>
                    </constrain-rel>

                    <create domain="MessageManagement" type="Message">

                        <characteristic name="subject" value="@message_subject">
                        <relationship name="channel">
                            <reference dummy="@ITER">@SELF</reference>
                        </relationship>
                        <relationship name="assets">
                            <reference dummy="@ITER">@asset_ref</reference>
                        </relationship>

                    </create>

                </lookfor>

            </lookfor>

        </lookfor>

    </lookfor>

</work-prescription>
```

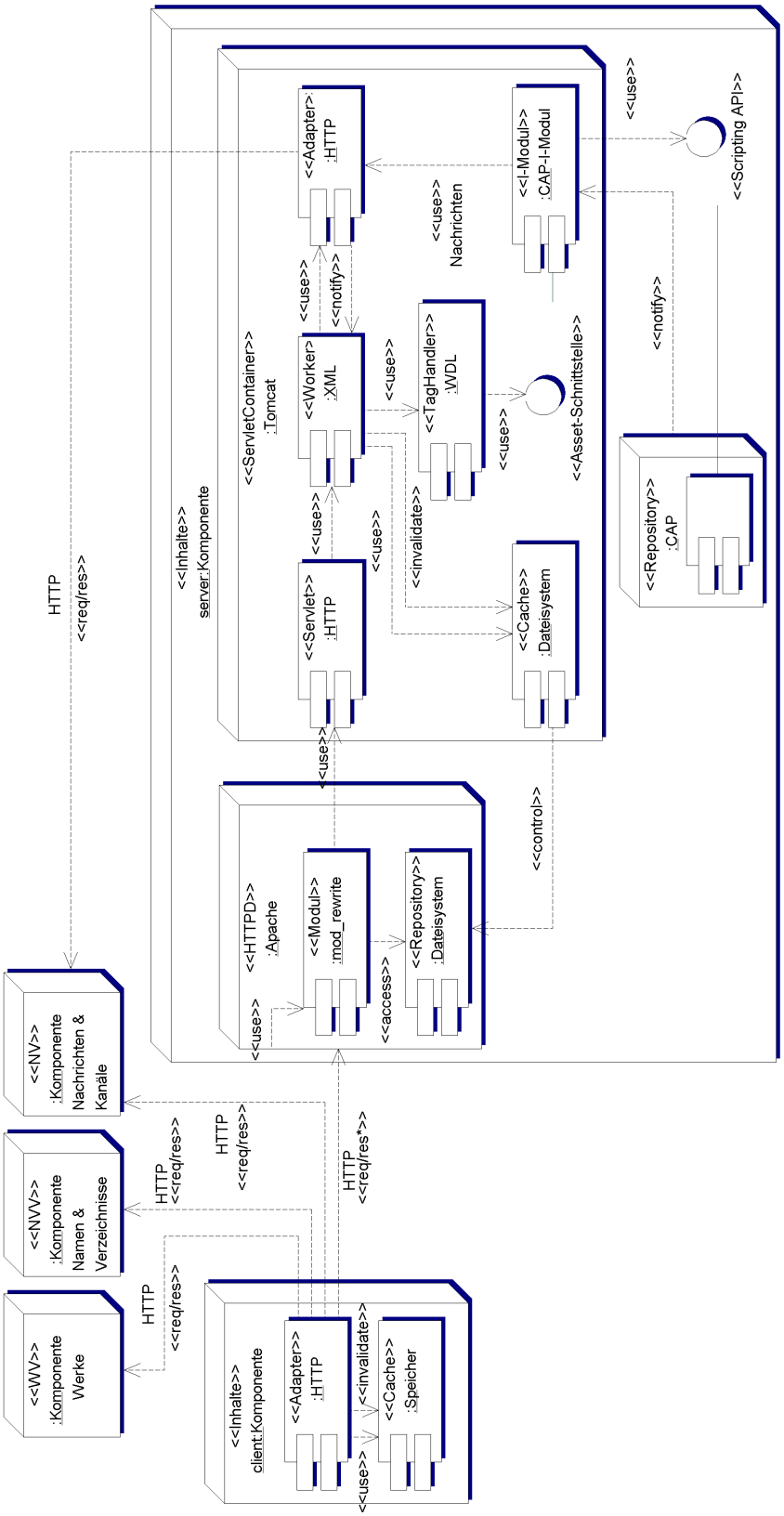

Abbildung 4.1: Verteilung und Kooperation von Komponenten und Modulen.

Kapitel 5

Zusammenfassung, Bewertung und Ausblick

Dieses Kapitel fasst die gewonnenen Erkenntnisse im Abschnitt 5.1 zusammen. Anschließend wird die vorliegende Arbeit in Abschnitt 5.2 im Hinblick auf die im Abschnitt 1.1 formulierten Ziele bewertet. Zum Abschluß gibt Abschnitt 5.3 einen Ausblick auf zu untersuchende Problemfelder und mögliche Entwurfs- oder Technologiealternativen, die sich im Verlaufe dieser Arbeit ergeben haben.

5.1 Zusammenfassung

Die vorliegende Arbeit gibt im 2. Kapitel einen Überblick über drei Projekte zum *begriffsorientierten Arbeiten* am Arbeitsbereich STS der TUHH. Daraufhin wird das, aus Veröffentlichungen von Schmidt et al. (2003) und Sehring (2004) hervorgegangene, *offene begriffsorientierte Modell* dargelegt, dessen Elemente eine reiche Modellierungsgrundlage für die Modellierung begriffsorientierter Anwendungsdomänen bilden. Insbesondere ist hier die Vereinigung von *Konzepten* und *Inhalten* zu *Assets* hervorzuheben. Die *Offenheit* des Modells besteht in der Anforderung, neue Konzept einführen sowie bestehende Konzept redefinieren zu können. Technisch resultiert dieses in der Fähigkeit eines implementierenden Systems, *Schema- oder Modellevolution* zuzulassen.

Desweiteren wird die *dynamische Anwendung* der Offenheit als eine wesentliche Anforderung an das Model eingeführt, um den *menschlichen Erkenntnisprozess* im Sinne Cassirers[1] abbilden zu können. Als Folge ergibt sich die Modellevolution während der Laufzeit eines *begriffsorientierten Systems*, die die *Benutzer* in ihrer Nutzung des Systems nicht einschränkt. Die Anwendung durch den Benutzer besteht in der *Ableitung* eines *individuellen Modells* aus dem Modell einer übergeordneten *Organisationseinheit*, um die *Sinnstruktur* des Benutzers möglichst nahe wiederzuspiegeln und eine semantisch reichere *Mensch-Maschine-Kommunkation* zu ermöglichen.

Ein weiterer Aspekt des offenen und dynamischen Modells ist nach Cassirer seine *Werkorientierung*, d.h. die Hervorhebung der Entwicklungsprozesse von *Assets* neben den eigentlichen

[1] Schmitz-Rigal (2002).

Werkprodukten, also den durch *Assets* beschriebenen Inhalten. Mögliche Dimensionen der Kategorisierung dieser *Werke* sind die *Kooperation subjektiver Modelle* einer Domäne, also die *Personalisierung* und *Publikation*, sowie orthogonal die *Kooperation über Domänengrenzen*. Als Resultat der Werkorientierung wird ein *Prozessmodell* zur Beschreibung *inkrementellen Werkfortschrittes* entwickelt, das den Übergang von *präskriptiven* zu *deskriptiven Werkbeschreibungen* (nach)vollzieht. Diese Werkbeschreibungen werden ebenfalls als durch *Assets* beschreibbare *Entitäten* eingeführt. Am Ende des Kapitels erfolgt eine Abgrenzung des vorliegenden Textes gegenüber anderen Arbeiten und ein Überblick auf verwandte Themen und Gebiete.

Im 3. Kapitel dieser Arbeit werden die Implikationen der Offenheit und Dynamik für den *Systementwurf* beleuchtet. Insbesondere wird ein verteilter Systementwurf als eine mögliche und besonders günstige Alternative für eine begriffsorientierte Infrastruktur dargestellt. Ein verteilter Entwurf hat, neben den speziellen Zielen für *kooperatives begriffsorientiertes Arbeiten*, allgemeine Ziele die sich aus der Verteilung ergeben. Dabei handelt es sich nach Coulouris et al. (1994) um *Transparenz*, *Offenheit*, *Nebenläufigkeit*, *Skalierbarkeit* und *Fehlertoleranz*, deren Erreichung am Ende dieses Abschnittes diskutiert wird.

Anschließend wird die Abbildung der Domänenkooperation auf die *Kooperation verteilter Dienste* gezeigt und eine allgemeine Diensteschnittstelle spezifiziert. Danach folgt die Enführung von *Basisdiensten* für den Betrieb der verteilten begriffsorientierten Infrastruktur. Es handelt sich um die *Benutzer- und Gruppenverwaltung*, die *Rechte- und Rollenverwaltung*, die *Werkverwaltung*, die *Namens- und Verzeichnisverwaltung* sowie abschließend die *Nachrichtenverwaltung*. Darüberhinaus wird die präskriptive und deskriptive Werkbeschreibung durch *Assets* sowie deren Prozessmodell entworfen. Die Kooperation von Domänen erfolgt in diesen Werkbeschreibungen durch *Delegation* von *Teilwerken*. Bei der Delegation spielen Aspekte wie *transparente Lokalisierung*, *Awareness* und *Data Provenence* eine Rolle.

Am Ende des Kapitels wird eine Architektur für begriffsorientierte Systeme beschrieben, die sich horizontal in *Komponenten* für Knoten der Infrastruktur und vertikal in *Module* für unterschiedliche Verantwortlichkeiten in einer Komponente gliedert. Eine konkrete Komponente wird auf der Basis einer individuellen *Konfiguration* durch eine *Drei- oder Mehrschichtarchitektur* aus Modulen aufgebaut. Module zur Umsetzung des speziellen Prozessmodells sind in der *Applikationsschicht* angeordnet.

Das 4. Kapitel gibt zunächst einen Überblick über verwendete Technologien für *Daten-, Applika- tions- und Präsentationsschicht*. Anschließend wird die Werkbeschreibungssprache mit XML umgesetzt. Zum Abschluß wird die *Diensteschnittstelle* als S-Modul für das HTTP-Protokoll in Java implementiert und die Umsetzung der Basisdienste mit den vorhandenen Mitteln der präskriptiven Werkbeschreibung vollzogen.

5.2 Bewertung

In den folgenden Abschnitten wird die Erreichung der allgemeinen Entwurfsziele *verteilter Systeme* nach Coulouris et al. (1994), also *Transparenz*, *Offenheit*, *Nebenläufigkeit*, *Skalierbarkeit* und *Fehlertoleranz*, sowie des besonderen Zieles *Verklemmungsfreiheit* für die Ausführung von präskriptiven Werkbeschreibungen in der vorliegenden Infrastruktur untersucht. Die adäquate Umsetzung der Anforderungen an begriffsorientierte Systeme wird durch die Nutzung der *Asset-Definitions-, Anfrage- und Manipulationssprache*, des *Asset-Modell-Compilers* sowie der

Architektur für begriffsorientierte Systeme nach SEHRING sichergestellt.

Transparenz

Die Transparenz, also das *Verbergen* von Auswirkungen der Verteilung der *begriffsorientierten Infrastruktur*, hat eine Reihe von unterschiedlichen Facetten, die im Folgenden Punkt für Punkt diskutiert werden. Teilweise ergänzen oder überschneiden sich die Ausführungen mit den nachfolgenden Abschnitten über Nebenläufigkeit, Skalierbarkeit und Fehlertoleranz. Dieser Sachverhalt ist, nach Meinung des Autors dieser Arbeit, eine natürliche Folge der konsequenten Umsetzung der letztgenannten Entwurfsziele, um deren Nutzen nicht durch erhöhte *Komplexität* im Umgang mit dem System zu unterminieren.

- Transparenter Zugriff (*Access Transparency*)

  Über die Delegation von *präskriptiven Teilwerken*, in Verbindung mit der *transparenten Lokalisierung*, wie im nachfolgenden Abschnitt geschildert, können lokale und entfernte *Assets* orthogonal zugegriffen werden.

- Transparente Lokalisierung (*Location Transparency*)

  Die Delegation von präskriptiven Teilwerken erfolgt unter Verwendung einer *logischen Adressierung* für den *Werkausführenden*. Die Abbildung der logischen auf die *physikalische Adresse* des Werkausführenden wird, unter Einbeziehung des Benutzer- und Gruppenverwaltungsdienstes durch einen *Verzeichnis- und Namensverwaltungsdienst* in der begriffsorientierten Infrastruktur geleistet. Kenntnis über die physikalische Adresse des Werkausführenden erlangt die delegierende Komponente in der *deskriptiven Werkbeschreibung* nur zu *Überwachungszwecken*.

- Transparente Nebenläufigkeit (*Concurrency Transparency*)

  Transparente Nebenläufigkeit wird in der vorliegenden begriffsorientierten Infrastruktur noch nicht ausreichend verwirklicht. Zwar unterstützen *Datenbanken* oder *Content Management Systeme*, die in der Regel die Persistenzmechanismen der*Datenschicht* in einer *Komponente* der Infrastruktur bilden, lokale *Transaktionen* nach dem *ACID-Prinzip* und damit auch deren wechselseitige *Isolation*, allerdings fehlt die Unterstützung für *verteilte Transaktionen* mit *Two-Phase-Commit-Protokoll*.

  Unter den gegebenen Umständen ist eine isolierte Transaktion und damit transparente Nebenläufigkeit auf die Gesamtheit der lokalen Anweisungen im Kontext eines Werkes beschränkt. Fehlt die Unterstützung von *ACID-Transaktionen* in der Datenschicht, etwa bei der Verwendung eines dateisystembasierten *Repositories*, dann entfällt auch die transaktionale Isolation, für die Gesamtheit der lokalen Anweisungen eines Werkes, gegenüber Anweisungen nebenläufiger Werke auf der Komponente. Das abstrahierende *I-Modul* kann diese Fähigkeit nicht ausgleichen.[2]

- Transparente Replikation (*Replication Transparency*)

  Replikation wird in der vorliegenden begriffsorientierten Infrastruktur intensiv, in Form von *Caching*, zur Steigerung der *Performanz* und *Zuverlässigkeit* eingesetzt. Als *Replikationsgranular* wird die Werkbeschreibung genutzt. Über die Definition von *Ad-hoc-Werken* können zudem beliebig kleine Teile eines Werkes zwischengespeichert werden.

[2] Vgl. Transaktionen in Lockemann et al. (1987).

In Verbindung mit einem *Notifikationsdienst*, ist für Werke mit lesendem Zugriff auf *Assets* die Replikation transparent. Werkbeschreibungen mit schreibendem Zugriff auf *Assets* werden, über die Zuordnung einer Lebensdauer mit dem Wert Null, von der Replikation ausgeschlossen.[3]

- Transparente Fehler (*Failure Transparency*)

  Auftretende *Netzwerk- oder Protokollfehler* werden in der verteilten Infrastruktur durch das *Dienstschnittstellenmodul* verborgen. Dazu trägt besonders die *client-seitige* Replikation von Werken bei, wie im vorangegangenen Abschnitt geschildert. Über den Namens- und Verzeichnisverwaltungsdienst, in Verbindung mit dem Benutzer- und Gruppenverwaltungsdienst, ist es im Fehlerfall sogar möglich, alternative Komponenten zu ermitteln und Teilwerke an diese zu delegieren. Erst im endgültigen Fehlerfall, der individell pro Komponente und Dienstschnittstellenmodul definiert werden kann (*Timeout*, Anzahl der Wiederholungen, zulässige Alternativen usw.), wird ein Fehler an die delegierende Komponente durchgereicht.

  Über die Verwendung von *ACID-fähigen Repositories* zur persistenten Verwaltung aller *Assets*, inklusive der Werkbeschreibungen, sowie entsprechender *Initialisierungsroutinen* wird die Toleranz gegenüber auftretenden *Hard- und Software-Fehlern* gesteigert. Allerdings ist auch hier wieder anzumerken, dass die Konsistenz der über die Komponenten der Infrastuktur verteilten *Assets* letztlich nur durch das Konzept der verteilten Transaktionen gesichert werden kann[4].

- Transparente Migration (*Migration Transparency*)

  Assets aus Anwendungsdomänen können ihren Aufenthaltsort, innerhalb ihrer Domäne in der verteilten begriffsorienierten Infrastuktur, ändern, ohne die Ausführung einer Werkvorschrift zu blockieren. Hierzu trägt sowohl die transparente Lokalisierung[5] über Namens- und Verzeichnisverwaltungsdienst, als auch die Verwendung von *Broadcasting-Techniken* zur Lokalisierung eines konkreten *Dienstanbieters* bei. Allerdings kann die Ausführung von Werkvorschriften u.U. erheblich verzögert werden, weil umfangreiche Kommunikation mit anderen Diensten der Infrastruktur nötig wird. Auf die Migration von *Assets* aus Basisdomänen wird im Ausblick eingegangen.

- Transparente Lastverteilung (*Performance Transparency*)

  Die flexible Reaktion auf auftretende *Lasten* in der verteilten Infrastruktur, erfolgt durch Maßnahmen während der Lokalisierung von Dienstanbietern. Damit bleibt die Lastverteilung, ebenso wie die Lokalisierung für den *Dienstnutzer* transparent. Im Detail sorgen Anweisungen, die *Strategien zur Lastverteilung* implementieren und auf Einträgen zu durchschnittlicher *Auslastung* und *Verbindungskosten* basieren, im Namens- und Verzeichnisverwaltungsdienst für eine entsprechende Steuerung der *Werksströme*. Die Kennwerte zu Auslastung und Verbindungskosten sowie die Verweise auf alternative Dienstanbieter können, durch Dienstnutzer oder den Dienstanbieter selbst, während der Laufzeit dynamisch angepasst werden, um veränderten Bedingungen im Gesamtsystem Rechnung zu tragen.

- Transparente Skalierung (*Scaling Transparency*)

[3] Vgl. Abschnitt 4.3.2, Replikation
[4] Vgl. Abschnitt 5.2, Zusammenfassung und Bewertung - Transparenz - Transparente Nebenläufigkeit
[5] Vgl. Abschnitt 5.2, Zusammenfassung und Bewertung - Transparenz - Transparente Lokalisierung

Alle Dienste der verteilten begriffsorientierten Infrastruktur sind auf die uneingeschränkte *Erweiterbarkeit* des Gesamtsystems ausgrichtet. Die Partitionierung der Infrastruktur anhand der *Benutzer- und Gruppenstruktur* sowie die *lose Kopplung* der Dienste über Namens-, Verzeichnis- und Nachrichtenverwaltungsdienst, erlauben die einfache und schnelle Erweiterung einer *Benutzer- oder Gruppeninfrastruktur* um alternative oder neue Dienstanbieter bei *Engpässen*. Entsprechende Einträge im Namens- und Verzeichnisverwaltungsdienst sowie deren Nutzung bleiben, außer zu *Überwachungszwecken* in Werkbeschreibungen, für die Dienstnutzer verborgen.

Offenheit

Die vorliegende prototypische Infrastruktur ist in vielerlei Hinsicht *offen*. Eine Vertiefung der unterschiedlichen Aspekte der *Offenheit* erfolgt gemäß der Kategorien *Hardware* und *Betriebssytem*, *Software* und *Schnittstellen* sowie *Netzwerk* und *Protokolle*.

Was die *Hardware* und das Betriebssystem anbelangt, so wurde der Prototyp in der Programmiersprache *Java* implementiert[6], was den Einsatz auf jeder *Hardware*- und Betriebssystemkombination zulässt, die über eine *virtuelle Maschine* für *Java* verfügt. Bei der Vielzahl der verfügbaren Kombinationen stellt dieser Punkt aber keine bedeutende Enschränkung der Offenheit dar.

In der Kategorie der Software und deren Schnittstellen orientiert sich der vorliegende Prototyp an der *Drei- oder Mehrschichtarchitektur*[7], die auf der untersten Ebene eine *Datenschicht* aufweist. Diese Datenschicht wird über sogenannte *Interpretationsmodule (I-Module)* und deren Nutzung von offenen Schnittstellen zugegriffen. Im vorliegenden Prototypen handelt es sich um das *Content-Management-System* CoreMedia CAP[8] und dessen *Scripting API*, das wiederum auf einem *CORBA API* beruht. Für ein *relationales Datenbanksystem* würde das I-Modul entsprechend eine *JDBC-Schnittstelle* nutzen. Generell ist jede Art von Massenspeicherabstraktion in der Datenschicht denkbar, für die in *Java* ein entsprechendes I-Modul entwickelt werden kann. In diesem Falle kann allerdings nicht mehr von wirklicher Offenheit des Systems gesprochen werden, weil u.U. erhebliche Entwicklungsarbeit erforderlich wird.

Auf der obersten Ebene der Architektur, in der *Präsentationsschicht*, befinden sich die sogenannten *Dienstschnittstellenmodule (S-Module)*, die eine offene Schnittstelle für den Austausch von *XML-Dokumenten*, über *Standardprotokolle* wie *HTTP*, *SOAP*[9] und *WebDav*[10] aufweisen. An dieser Stelle ist Offenheit gefragt, weil der Kreis der zugreifenden Systeme und deren Interpretationsfähigkeit nicht bestimmt werden kann. Insofern ist auch die Wahl zugunsten von *XML-Dokumenten*, mit ihrer Fähigkeit zur *verbosen* Datenbeschreibung und zur Trennung von Daten und *Schema* zu argumentieren. Über S-Module ist die Interaktion zwischen menschlichen *Benutzern* und einem begriffsorientierten System, direkt oder über einen entsprechenden Client, zu realisieren[11]. Der vorliegende Prototyp nutzt die Variante des Austausches von *XML-Dokumenten* zwischen *Webbrowser* und System über das *HTTP-Protokoll*.

Was die Offenheit in Bezug auf das Netzwerk bzw. die verwendbaren Protokolle betrifft,

[6] Vgl. Abschnitt 4.1, Technologie für die Systemimplementation
[7] Vgl. Abschnitt 3.5, Architektur eines begriffsorientierten Systems
[8] CoreMedia (2004a).
[9] ASF (2004c).
[10] webdav.org (2004).
[11] Sehring (2004), S. 137.

so gilt das gleiche wie beim Zugriff auf die Datenschicht. Generell sind alle Kombinationen denkbar, für die in *Java* entsprechende *APIs* vorhanden sind. Wegen seiner Universalität in einem *heterogenem Hardwareumfeld*, die auf dem *TCP/IP-Protokollen* beruht, sowie der hohen *Verfügbarkeit* entsprechender *Clients* (*Webbrowser*) stellen das HTTP-Protokoll und darauf aufbauende Protokolle prinzipiell eine gute Wahl dar. Der Einsatz von *SOAP* und *WebDav* sollte wegen der besonderen Unterstützung der Versionierung in Erwägung gezogen werden[12].

Nebenläufigkeit

Die *Nebenläufigkeit* wird im Wesentlichen durch die *Delegation* von *asynchronen Teilwerken*[13] an verschiedene Komponenten der begriffsorientierten Infrastruktur, durch die werkausführende Komponente, erreicht. Je nach Definition einer präskriptiven Werkbeschreibung, bzw. ihres Datenflusses, kann ein hoher Grad an Nebenläufigkeit erzielt werden. Die Auflösung des definierten Datenflusses erfolgt durch das Submodul zur Zerlegung einer Werkbeschreibung[14]. Eine weitere Form der Nebenläufigkeit ergibt sich bei dem Fehlschlag der *Bindung* einer Menge von *Assets* an eine *Werkanweisung*. In diesem Falle wird ein elementares *ad-hoc* Werk per *Broadcast* an andere Komponenten der Domäne geschickt, um, im Falle einer vorliegenden Zugriffsberechtigung, die Bindung vornehmen zu lassen. Das Problem der *Verklemmungsfreiheit* im Zusammenhang mit der Delegation von *Teilwerken* wird im Abschnitt 270 eingehender betrachtet.

Skalierbarkeit

Skalierbarkeit wird in der entwickelten Infrastrukur durch die *lose Kopplung* von Komponenten über den Namens- und Verzeichnisverwaltungsdienst sowie über den Nachrichtenverwaltungsdienst erreicht. Kenntnis über die *physikalische Adresse* einer beauftragten Komponente erlangt der Delegierende in der deskriptiven Werkbeschreibung nur zu *Überwachungszwecken*. Eine notwendige Ausnahme bildet lediglich die Hinterlegung der Adresse des jeweiligen Namens- und Verzeichnisverwaltungsdienstes sowie des entsprechenden Verzeichnisnamens als *Bootstrap* in jeder Komponente. Auf diese Weise können die Komponenten, durch das Submodul zur Delegation von Teilwerken[15], in Form eines *synchronen Teilwerkes* den jeweils gültigen *Werkempfänger* lokalisieren. Zusätzlich kooperiert die Namens- und Verzeichnisverwaltung mit der Nachrichtenverwaltung, um den zugehörigen logischen Nachrichtenkanal ermitteln und im Rahmen des vorgenannten Werkes mitliefern zu können.

Aus der geschilderten Vorgehensweise ergiben sich zwei weitere Bestandteile des *Bootstraps*, die die Skalierbarkeit der vorliegenden Infrastruktur einschränken. Es handelt sich um die Hinterlegung der physikalischen Adresse des für die Kommunikation mit dem Namens- und Verzeichnisverwaltungsdienst zuständigen Nachrichtenverwaltungsdienstes sowie des zugehörigen *logische Nachrichtenkanals*. Ohne diese Maßnahme könnte der zuvor beschriebenen Prozess nicht als eine herkömmliche präskriptive Werkbeschreibung behandelt werden, die der *Replikation* auf Seiten des Werkdelegierenden unterliegt, weil eine Invalidierung veralteter Informationen nicht erfolgen würde. *Replikation* stellt jedoch besonders in diesem Falle eine

[12] Vgl. Abschnitt 5.3, Ausblick
[13] Vgl. Abschnitt 172, Arten von Teilprozessen - Asynchrone und synchrone Teilprozesse
[14] Vgl. XMLWorker u. TagHandler in Abschnitt 4.3.1, HTTP-Dienstschnittstelle
[15] Vgl. HTTP-Adapter in Abschnitt 4.3.1, HTTP-Dienstschnittstelle als S-Modul

Notwendigkeit dar, weil die Lokalisierung des Werkempfängers sowie des logischen Nachrichtenkanals jeder Werkdelegation vorangeht und die Kommunikationskosten ohne *Replikation* vervielfacht würden.

Weitere Vorteile der transparenten Lokalisierung von Komponenten und des zugehörigen logischen Nachrichtenkanals ergeben sich mit der erhöhten Fehlertoleranz, die im nachfolgenden Abschnitt behandelt wird, und mit der Möglichkeit der Lastverteilung, die auf Strategien zur Auswertung von dynamischen Verfügbarkeits- und Kostendaten beruht.

Fehlertoleranz

Die erste Ausprägung der *Fehlertoleranz* ergibt sich, wie die zuvor beschriebene Skalierbarkeit, aus der transparenten Lokalisierung von Komponenten. *Dauerhafte Fehler* im zugrundeliegenden Netzwerk können in der vorliegenden Infrastruktur durch die *Umleitung* auf andere Komponenten aufgefangen werden. Umleitungen können neben ihrer statischen Einrichtung, durch Änderung der physikalischen Adresse, auch durch die bereits beschriebenen Strategien zur Lastverteilung eingerichtet werden, indem einer Komponente sehr hohe *Kosten* (im Vergleich zu den Alternativen) oder eine sehr niedrige *Verfügbarkeit* zugeordnet werden. Insofern stellen *Fehlerstrategien* nur einen Spezialfall der Lastverteilungsstrategien dar.

Die zweite Variante der Fehlertoleranz betrifft das*Dienstschnittstellenmodul* und dessen *Robustheit* gegenüber *sporadisch* auftauchenden Netzwerkfehlern, die durch Maßnahmen wie *Replikation*, Wiederholung(en) und *Timeouts* behandelt werden können. Das Dientsschnittstellenmodul der vorliegenden Infrastruktur erlaubt eine individuelle *Konfiguration* mit den genannten Parametern.

Ein dritter Aspekt ist die Toleranz gegenüber auftretenden Fehlern auf *Hardware-* und *Software-Ebene*, die einen *Ausfall* der Komponente bewirken und deren *Neustart* notwendig machen. Eine wesentliche Anforderung zur Erreichung dieses Zieles ist die *Zustandslosigkeit* von Servern[16]. Diese Anforderung wird jedoch in der vorliegenden Arbeit nicht vollständig verwirklicht. Die Server behalten aus Performanzgründen während des Betriebes ein lokales Replikat des Werkabhängigkeitsgraphen[17], der generierten Werkbeschreibungen und des zugehörigen *DOM Trees*[18], der sich bei der Zerlegung der Werkbeschreibungen ergeben hat.

Um das Ziel der Fehlertoleranz dennoch zu erreichen, werden die Werkbeschreibungen bei jedem erzielten Werkfortschritt über den Werkverwaltungsdienst persistiert. Im Fehlerfall können diese als Bestandteil des *Bootstrap* einer Komponente vom Werkverwaltungsdienst abgerufen und die zugehörigen Hilfsstrukturen im *Server* aufgebaut werden.

Verklemmungsfreiheit

Ein wesentliches Argument für die Einführung eines *Werkverwaltungsdienstes* stellt neben der Fehlertoleranz, wie im vorangegangenen Abschnitt beschrieben, die Überwachung von *Verklemmungen* dar. Innerhalb einer präskriptiven Werkbeschreibung kann der *Datenfluß* wegen der *Schachtelung* von Anweisungen und Teilwerken nur *zyklenfrei* formuliert werden[19]. Aller-

[16] Coulouris et al. (1994).
[17] Vgl. Abschnitt 3.2.5, Werkverwaltungsdienst
[18] W3C (2004a).
[19] Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse

dings kann die *nebenläufige* Delegation von Teilwerken (auch ad-hoc Werken) zu Zyklen im *Werkabhängigkeitsgraphen* und damit zu Verklemmungen in der vorliegenden Infrastruktur führen. Eine Leistung des Werkverwaltungsdienstes ist es, diese Zyklen auf Basis der *lokal gebundenen* Werkbeschreibungen zu erkennen und deren weitere *Ausführung* zu unterbinden. Der urspüngliche Werkempfänger, der die Beschreibung beim Werkverwaltungsdienst zur Persistierung eingereicht hat, verweigert dann ebenfalls die weitere Ausführung der präskriptiven Werkbeschreibung.

5.3 Ausblick

Dieser Abschnitt gibt einen Ausblick auf zu untersuchende Problemfelder und mögliche Entwurfs- und Implementationsalternativen, die sich im Verlaufe dieser Arbeit ergeben haben.

- Der Komplex des *schreibenden Zugriffs* auf *Assets* sowie die Zusammenfassung einer Menge von Zugriffen zu *Transaktionen* mit *ACID-Eigenschaft*[20] ist weder für einen nichtverteilten noch für einen verteilten begriffsorientierten Entwurf hinreichend untersucht. Die in dieser Arbeit beschriebenen Dienste der verteilten Infrastruktur beruhen auf Modell, Sprache, *Modell-Compiler* und Architektur der konzeptorientierten Inhaltsverwaltung nach Sehring (2004), die auf keiner Ebene Transaktionen oder ein vergleichbar mächtiges Konzept vorsieht. Es ist unklar wie eine Integration in die bestehende Infrastruktur vorgenommen werden soll, ohne Modellbrüche zu erzeugen. Gleichwohl würden (verteilte) Transaktionen große Vorteile für die *Fehlertoleranz* und *Transparenz* (*transparente Nebenläufigkeit*) mit sich bringen.

- Die *intensionale Definition* des Werkkonzeptes im *Asset-Modell* der Werkverwaltung, modelliert u.a. die `Teilwerk/Werk`-Beziehung zwischen `Werken` und die `Baustein/Exemplar`-Beziehung zwischen `Exemplaren` und `Assets`. Eine Modellierung der einzelnen *Anweisungen* zur Erzeugung, Suche, Änderung und Zerstörung von *Assets* erfolgt nicht. Die Auswirkungen dieser alternativen Modellierung sollten untersucht werden.

- Für das *Dienstschnittstellenmodul* ist *Offenheit* gefragt, weil der Kreis der zugreifenden Systeme und deren Interpretationsfähigkeit nicht bestimmt werden kann. Hier ist also ein *deklaratives* Format zum Austausch von Daten über ein Standardprotokoll von Vorteil. In der vorliegenden Infrastruktur handelt es sich um *XML-Dokumente*, die über das *HTTP-Protokoll* ausgetauscht werden. Um den Aufwand, zur Erzeugung dieser *XML-Dokumente* aus den verwalteten *Assets*, einzusparen und eine besonders performante Zwischenspeicherung zu ermöglichen, werden die Dokumente als *Replikationsgranulare* genutzt. Die Zwischenspeicherung kann somit ganz früh, in der vorliegenden Infrastruktur im *Webserver* der Dienstschnittstelle, erfolgen. Andererseits wird auf diese Weise nicht das kleinste mögliche Granular im Modell, also die einzelne Anweisung zum *Asset-Zugriff*, genutzt. Möglicherweise sind hier also noch Optimierungsspielräume vorhanden.

- Der vorliegende Entwurf der Diensteschnittstelle und dessen Implementierung nutzt, mit der Übertragung von *XML-Dokumenten* durch die *Post-Methode*, nur einen geringen Bruchteil des *HTTP-Protokolls*. Es wäre alternativ auch die Nutzung weiteren Anweisungen des Protokolls denkbar, die u.U. direkt auf Anweisungen der *Asset-Sprache*

[20] Lockemann et al. (1987).

abgebildet werden können. Desweiteren ist der Einsatz höherer, auf *HTTP* aufbauender, Protokolle wie SOAP und WebDav möglich, die mit einem größeren Funktionsumfang ausgestattet sind. Insbesondere ist hier die Unterstützung der *Versionierung* zu erwähnen, die im Zusammenhang mit der Beschreibung *inkrementellen Werkfortschrittes* und der Speicherung von erkenntnisreichen (Zwischen)Werkbeschreibungen besondere Bedeutung erlangen könnte. Zum Abschluß soll das Feld der *WebServices*[21] nicht unerwähnt bleiben, das mit der entwickelten Infrastruktur, neben der offenen, standardisierten Schnittstelle, auch Gemeinsamkeiten in der Bereitstellung von Diensten aufweist. Auch dieses könnte eine interessante Erweiterung der vorliegenden Arbeit darstellen.

- Die Migration von *Assets* aus *Basisdomänen* wurde bisher noch nicht eingehend untersucht. In diesem Zusammenhang sind eine ganze Reihe von fatalen Szenarien denkbar, die das ordnungsgemäße Funktionieren einzelner Komponenten oder der gesamten begriffsorientierten Infrastruktur unterbinden. Insbesondere ist hier die Migration von *Assets* der Namens-, Verzeichnis- und Nachrichtenverwaltungsdienste zu nennen, die eine zentrale Rolle in der verteilten Infrastruktur einnehmen. Eine Klassifizierung und Untersuchung der möglichen Szenarien wäre wünschenswert.

- Die Formulierung der *Personalize-* und *Publish-Anweisungen* durch präskriptive Werkbeschreibungen, die sich nur der vier elementaren Anweisungen der *Asset-Sprache* zur Erzeugung (`create`), Abfrage (`lookfor`), Änderung (`update`) und Zerstörung (`delete`) von *Assets* bedienen, würde die Nutzung der entwickelten Infrastruktur für die *Kooperation* subjektiver Modelle ermöglichen. Hierzu wäre allerdings eine eingehende Untersuchung der unterschiedlichen Szenarien, bei *Personalisierung* und *Publikation*[22] von *Assets*, und deren Abbildung auf Werkbeschreibungen anzuraten.

- Bei dem Vergleich der *Asset-Definitions-, Anfrage- und Manipulationsprache* nach Sehring (2004) und der Werkbeschreibungssprache in der vorliegenden Arbeit fällt deren Ähnlichkeit auf, denn die zweite beschreibt lediglich Ausdrücke der ersten Sprache in einer eigenen Syntax. Trotzdem folgt ein recht aufwendiges *Parsing* und eine anschließende Abbildung des resultierenden *DOM-Tree's* auf eine generierte *Asset-Schnittstelle*. Diese Leistung könnte jedoch auch, ohne die beschriebene Indirektion, durch einen erweiterten *Asset-Modell-Compiler* erbracht werden, was zu einer beschleunigten Ausführung der Werkbeschreibungen führen würde. Die Erweiterung des *Compilers* müsste Konzepte wie *Blöcke*, *hierarchische Namensräume*, *geschützte Anweisungen* und *Nebenläufigkeit* umfassen, die bereits mit der Werkbeschreibungssprache dargelegt wurden[23]. Neben der gesteigerten Performanz, zeichnet sich diese Lösung durch die verbesserte *Natürlichsprachlichkeit* und den Schutz des *Asset-Codes* vor unbefugtem Zugriff aus.

- In der vorliegenden begriffsorientierten Infrastruktur stellt der *Werkverwaltungsdienst* eine *zentrale Ressource* für den Betrieb der gesamten Infrastruktur dar. Zentrale Ressourcen sind jedoch aus Gründen der *Skalierbarkeit* in verteilten Systemen so weit wie möglich zu vermeiden[24]. Zu diesem Zweck wäre der Einsatz mehrerer Komponenten, die eventuell über eine Organisationsstruktur zueinander in Beziehung stehen, zu untersuchen. Dies muss insbesondere vor dem Hintergrund der zuverlässigen Vermeidung von *Verklemmungen* im systemweiten *Werkabhängigkeitsgraphen* erfolgen.

[21] ASF (2004d).
[22] Bachmann (2003), S. 64.
[23] Vgl. Abschnitt 4.2, Prozessbeschreibungssprache für Asset-Prozesse
[24] Coulouris et al. (1994), S. 17ff.

Die vorstehenden Überlegungen zur Skalierbarkeit treffen auch auf *Namens- und Verzeichnis-* sowie auf den *Nachrichtenverwaltungsdienst* zu. Während beim erstgenannten keine weiteren Schwierigkeiten zu erwarten sind, ist die *zweidimensionale Kooperation* im Nachrichtendienst auf die Frage hin zu untersuchen, inwieweit die Anforderung der korrekten Replikation, die wesentlich auf dem Versand von *Nachrichten* beruht, davon betroffen ist.

- Im Rahmen der Nutzung des *Nachrichtendienstes* für den Versand von *Invalidierungsnachrichten* sind Optimierungsansätze denkbar. Einer davon ist die Möglichkeit „voreilige“ Nachrichten zu verschicken und so den *Werkabhängigkeitsgraphen* von der Wurzel und den Blättern *parallel* abzuarbeiten. Dabei müssen *Werkausführende* ihren Teilwerkausführenden jedoch signalisieren können, das sie eine aktuelle Version der *Werkbeschreibung* benötigen und das hierfür die *Replikate* zu invalidieren sind. Zusätzlich müssen sie sich auf eine *eindeutig* zu identifizierende Nachricht berufen, damit sich beide Handlungsstränge treffen und abbrechen können.

- Über die `Teilwerk/Werk`-Beziehung im Werkverwaltungsdienst können wahlweise flache oder tiefe Werkbeschreibungen von Prototypen abgerufen werden. Diese Option sollte in der Implementation genutzt und deren Nutzen sollte ermittelt werden.

- Wegen der unbegrenzten *Modellevolution* kann sich eine *Komponente* eines Dienstes von dem ursprünglichen *Asset-Modell* der repräsentierten *Domäne* sehr weit entfernen. Im Extremfall haben die Komponenten einer Domäne keine gemeinsamen *Asset-Typen*. Dies wurde in der vorliegenden Arbeit jedoch nicht ausreichend bedacht, weil der Fokus auf den eher statischen *Basisdomänen* und deren *Kooperation* lag. Es genügt demnach nicht, die Komponenten eines *verteilten begriffsorientierten Systems* anhand ihrer Domänen im *Namens- und Verzeichnisverwaltungsdienst* einzutragen. Vielmehr müssen Komponenten ihr gesamtes *Asset-Modell* bekanntgeben, damit gezielt nach Kooperationspartnern gesucht werden kann. Vorhandene Ansätze in dieser Richtung sollten untersucht werden (z.B. *Web-Services*[25]).

[25] ASF (2004d).

Anhang A

Asset-Modelle der Basisdienste

A.1 Benutzer- und Gruppenverwaltungsdienst

Listing A.1: *Asset-Modell* der Benutzer- und Gruppenverwaltung

```
model UserGroupManagement

from RightRoleMangement import Right, Role
from NameDirectoryManagement import Directory

class OE {

    concept characteristic id   : LocalIdentifier

            relationship superOEs   : OE*
            relationship subOEs     : OE*
            relationship domains    : Domain*
            relationship rights     : Right*
            relationship roles      : Role*
            relationship directory  : Directory

            ; *** uniqueness
            contraint lookfor OE { id = self.id } = { self }

            ; *** inversion
            constraint subOEs.superOEs >= { self }
                onviolation update subOEs { superOEs += self }

            ; *** others
            constraint (superOEs.domains - self.domains) # na

}

class User refines OE {

    concept characteristic userid       : String
            characteristic password     : String
            characteristic firstname    : String
            characteristic lastname     : String

            ; *** uniqueness
            contraint lookfor User { userid = self.userid } = { self }
```

```
}

class Group refines OE {

    concept characteristic name : String

}

class Domain {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship domainOEs  : OE*
            relationship directory  : Directory

            ; *** uniqueness
            constraint lookfor Domain { id = self.id } = { self }

            ; *** inversion
            constraint domainOEs.domains >= { self }
                onviolation update domainOEs { domains += self }

            ; *** cardinality
            constraint directory # na

            constraint lookfor Domain { directory = self.directory } = { self }

}
```

Listing A.2: *Proxy-Asset-Modell* der Rechte- und Rollenverwaltung

```
model RightRolleManagement ; this is only a proxy model

class Right {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Right {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}

class Role {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Role {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}
```

Listing A.3: *Proxy-Asset-Modell* der Namens- und Verzeichnisverwaltung

```
model NameDirectoryManagement ; this is only a proxy model

class Directory {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Directory {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}
```

A.2 Rechte- und Rollenverwaltungsdienst

Listing A.4: *Asset-Modell* der Rechte- und Rollenverwaltung

```
model RightRoleManagement

from AllDomains import Asset

class Right {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship asset  : Asset

            ; *** uniqueness
            constraint lookfor Right { id = self.id } = { self }

            ; *** cardinality
            constraint asset # na

}

class RLookfor refines Right {

    concept
            ; *** cardinality
            constraint lookfor RLookfor { asset = self.asset } = { self }

}

class RUpdate refines Right {

    concept
            ; *** cardinality
            constraint lookfor RUpdate { asset = self.asset } = { self }

}

class RDelete refines Right {
```

```
    concept
            ; *** cardinality
            constraint lookfor RDelete { asset = self.asset } = { self }

}

class Role {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship superroles : Role*
            relationship subroles   : Role*
            relationship rights     : Right*

            ; *** uniqueness
            constraint lookfor Role { id = self.id } = { self }

            ; *** inversion
            constraint subroles.superroles >= { self }
                onviolation update subroles { superroles += self }

}
```

Listing A.5: *Proxy-Asset-Modell* f"ur beliebige Dom"anen

```
model AllDomains ; this is only a proxy model

class Asset {

    concept characteristic nds_path : String
            characteristic type     : AssetType
            characteristic id       : LocalIdentifier

            relationship works : Work*

            ; *** uniqueness
            contraint lookfor Asset {

                nds_path    = self.nds_path
                type        = self.type
                id          = self.id

            } = { self }

}
```

A.3 Werkverwaltungsdienst

Listing A.6: *Asset-Modell* der Werkverwaltung

```
model WorkManagement

from AllDomains import Asset
from NameDirectoryManagement import Directory

class Work {

    content prescription : XML
```

```
    concept characteristic id   : HashValue
            characteristic name : String

            relationship superworks         : Work*
            relationship subworks           : Work*

}

class Prototype refines Work {

    concept relationship exemplars  : Exemplar*

        ; *** uniqueness
        constraint lookfor Prototype { id = self.id } = { self }

        constraint lookfor Prototype { name = self.name } = { self }

        ; *** inversion
        constraint lookfor Prototype { superworks >= self } = subworks
            onviolation update self {

                subworks += lookfor Prototype { superworks >= self }

            }

        constraint exemplars.prototype = { self }
            onviolation update exemplars { prototype := self }

}

class Exemplar refines Work {

    concept relationship prototype  : Prototype
            relationship executor   : Directory

        ; *** uniqueness
        contraint lookfor Exemplar {

            id          = self.id
            executor    = self.executor

        } = { self }

        ; *** inversion
        constraint lookfor Exemplar { superworks >= self } = subworks
            onviolation update self {

                subworks += lookfor Exemplar { superworks >= self }

            }

        constraint prototype.exemplars >= { self }
            onviolation update prototype { exemplars += self }

        ; *** cardinality
        constraint executor # na

}
```

Listing A.7: *Proxy-Asset-Modell* f"ur beliebige Dom"anen

```
model AllDomains ; this is only a proxy model
```

```
class Asset {

    concept characteristic nds_path : String
            characteristic type     : AssetType
            characteristic id       : LocalIdentifier

            relationship works : Work*

            ; *** uniqueness
            contraint lookfor Asset {

                nds_path    = self.nds_path
                type        = self.type
                id          = self.id

            } = { self }

}
```

Listing A.8: *Proxy-Asset-Modell* der Namens- und Verzeichnisverwaltung

```
model NameDirectoryManagement ; this is only a proxy model

class Directory {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Directory {

                nds_path    = self.nds_path
                id          = self.id

            } = { self }

}
```

A.4 Namens- und Verzeichnisverwaltungsdienst

Listing A.9: *Asset-Modell* der Namens- und Verzeichnisverwaltung

```
model NameDirectoryManagement

from MessageManagement import Channel

class Name {

    concept characteristic id       : LocalIdentifier
            characteristic key      : String
            characteristic value    : String

            relationship directory  : Directory

            ; *** uniqueness
            constraint lookfor Name { id = self.id } = { self }

            constraint lookfor Name {
```

```
                directory   = self.directory
                key         = self.key

            } = { self }

            constraint lookfor Directory {

                superdir    = self.directory
                name        = self.key

            } = na

            ; *** cardinality
            constraint directory # na

}

class Directory {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship superdir   : Directory     := Root
            relationship subdirs    : Directory*
            relationship names      : Name*
            relationship channel    : Channel

            ; *** uniqueness
            constraint lookfor Directory { id = self.id } = { self }

            constraint lookfor Name {

                superdir    = self.superdir
                name        = self.name

            } = { self }

            ; *** inversion
            constraint subdirs.superdir = self
                onviolation update subdirs { superdir = self }

            constraint names.directory = self
                onviolation update names { directory = self }

            ; *** others
            constraint superdir # na
                onviolation update self { superdir = Root }

}
```

Listing A.10: *Proxy-Asset-Modell* der Nachrichtenverwaltung

```
model MessageManagement ; this is only a proxy model

class Channel {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Channel {

                nds_path    = self.nds_path
```

```
                id          = self.id

            } = { self }

}
```

A.5 Nachrichtenverwaltungsdienst

Listing A.11: *Asset-Modell* der Nachrichtenverwaltung

```
model MessageManagement

from WorkManagement import Work
from AllDomains import Asset

class Message {

    content content : XML
    concept characteristic id       : LocalIdentifier
            characteristic subject  : String

            relationship asset  : Asset*
            relationship work   : Work*

            ; *** uniqueness
            constraint lookfor Message { id = self.id } = { self }

            ; *** others
            constraint      (asset # na or work # na)
                        and (asset = na or work = na)

}

class Channel {

    concept characteristic id   : LocalIdentifier
            characteristic name : String

            relationship messages   : Message*

            ; *** uniqueness
            constraint lookfor Channel { id = self.id } = { self }

}
```

Listing A.12: *Proxy-Asset-Modell* der Werkverwaltung

```
model WorkManagement ; this is only a proxy model

class Work {

    concept characteristic nds_path : String
            characteristic id       : LocalIdentifier

            ; *** uniqueness
            contraint lookfor Work {

                nds_path    = self.nds_path
                id          = self.id
```

```
            } = { self }

}
```

Listing A.13: *Proxy-Asset-Modell* f"ur beliebige Dom"anen

```
model AllDomains ; this is only a proxy model

class Asset {

    concept characteristic nds_path : String
            characteristic type     : AssetType
            characteristic id       : LocalIdentifier

            relationship works : Work*

            ; *** uniqueness
            contraint lookfor Asset {

                nds_path    = self.nds_path
                type        = self.type
                id          = self.id

            } = { self }

}
```

Anhang B

Erklärungen

„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.“

Hamburg, den 9. Juni 2004

Jörn Fornfeist

„Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereichs einverstanden.“

Hamburg, den 9. Juni 2004

Jörn Fornfeist

Literaturverzeichnis

ANSA: The Advanced Network Systems Architecture (ANSA) Reference Manual. Castle Hill, 1989, Architecture Project Management

Axel Wienberg, Matthias Ernst, Andreas Gawecki, Olaf Kummer, Frank Wienberg, Joachim W. Schmidt: Content Schema Evolution in the CoreMedia Content Application Platform CAP. In Lecture Notes in Computer Science. Band 2287, 8th International Conference on Extending Database Technology (EDBT 2002) 2002

Bernd Page, Andreas Häuslein, Klaus Greve: Das Hamburger Umweltinformationssystem HUIS. Umweltbehörde Hamburg, Projektgruppe HUIS, 1993 – Druckschrift

Charles S. Peirce: Collected Papers of Charles Sanders Peirce. Harvard University Press, 1931

Christiane Schmitz-Rigal: Die Kunst offenen Wissens, Ernst Cassirers Epistemologie und Deutung der modernen Physik. Band 7, Cassirer-Forschungen. Felix Meiner Verlag, 2002

Claudia Niedereé, Claudia Hattendorf, Sven Müssig, Martin Warnke, Joachim W. Schmidt: Warburg Electronic Library. uni hh Forschung XXXI 1996, 6 – 16

Claudia Niedereé: Personalisierung, Kooperation und Evolution in digitalen Bibliotheken. Dissertation, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2002

CoreMedia AG: CoreMedia Smart Content Technology. http://www.coremedia.de, 2004

CoreMedia AG: Customers - CoreMedia. http://www.coremedia.com/de/Customers/index.html, 2004

Cornelia Jacobsen: Personalisierung: Konzepte und Techniken. Diplomarbeit, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2002

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns. Addison-Wesley Publishing Company, 1995

Ernst Cassirer: Philosophie der symbolischen Formen. Erster Teil: Die Sprache. Band 11, Gesammelte Werke. Hamburger Ausgabe. Felix Meiner Verlag, 2001, Herausgegeben von Birgit Recki, Text und Anmerkungen bearbeitet von Claus Rosenkranz

Ernst Cassirer: Philosophie der symbolischen Formen. Dritter Teil: Pänomenologie der Erkenntnis. Band 13, Gesammelte Werke. Hamburger Ausgabe. Felix Meiner Verlag, 2002, Herausgegeben von Birgit Recki, Text und Anmerkungen bearbeitet von Julia Clemens

Ernst Cassirer: Philosophie der symbolischen Formen. Zweiter Teil: Das mythische Denken. Band 12, Gesammelte Werke. Hamburger Ausgabe. Felix Meiner Verlag, 2002, Herausgegeben von Birgit Recki, Text und Anmerkungen bearbeitet von Claus Rosenkranz

Florian Matthes: Mobile Processes in Cooperative Information Systems. In STJA'97, Smalltalk und Java in Industrie und Ausbildung. Springer-Verlag, 1997

George Coulouris, Jean Dollimore, Tim Kindberg: Distributed Systems - Concepts and Design. Zweite Auflage. Addison-Wesley Publishing Company, 1994

Gerard Tel: Introduction to Distributed Algorithms. Zweite Auflage. Cambridge University Press, 2000

Gunther Bachmann: Eine Architektur für kooperatives begriffsorientiertes Arbeiten. Diplomarbeit, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2003

Gustavo Rossi, Daniel Schwabe, Robson Guimaraes: Designing Personalized Web Applications. In Proceedings of the tenth international conference on World Wide Web. 8th International Conference on Extending Database Technology (EDBT 2002) ACM Press, 2001

Hans-Werner Sehring: Konzeptorientierte Inhaltsverwaltung - Modell, Systemarchitektur und Prototypen. Dissertation, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2004

Holm Wegner: Objektorientierter Entwurf und Realisierung eines Agentensystems für kooperierende Internet-Informationssysteme. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 1998

Holm Wegner: Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement. Dissertation, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2002

infoAsset AG: infoAsset AG: Knowledge Management Software infoAsset Broker. http://www.infoasset.de, 2004

International Standards Organistaion: Overview and guide to use. Band 1, Basic Reference Model of Open Distributed Processing. International Standards Organistaion, 1992

Joachim W. Schmidt, Hans-Werner Sehring, Michael Skusa, Axel Wienberg: Subject-Oriented Work: Lessons Learned from an Interdisciplinary Content Management Project. In Lecture Notes in Computer Science. Band 2151, Fifth East-European Conference on Advances in Databases and Information Systems Springer-Verlag, 2001

Joachim W. Schmidt, Hans-Werner Sehring: Conceptual Content Modeling and Management - The Rationale of an Asset Language. In Proc. Perspectives of System Informatics. Akademgorodok Springer-Verlag, 2003

Jörn Fornfeist: Webinterfaces für multimediale Digitale Bibliotheken. Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2003 – Studienarbeit

Martin Fowler, Kendall Scott: UML Distilled. Applying the Standard Object Modelling Language. Addison-Wesley Publishing Company, 1997

Martin Raulf, Rainer Müller, Ulrike Steffens, Florian Matthes, Klaus J. Scheunert, Joachim W. Schmidt: Begriffsorientierte Dokumentenverwaltung für das internetgestützte Projektmanagement. In Tagungsband 4. GI-Fachgruppentagung „Management und Controlling von IT-Projekten“ dpunkt.verlag, 2001

Matthias Bruhn: The Warburg Electronic Library in Hamburg: A Digital Index of Political Iconography. Visual Resources XV 1999, 405 – 423

Michael Hoffmann: Peirces Zeichenbegriff: seine Funktionen, seine phänomenologische Grundlegung und seine Differenzierung. http://www.uni-bielefeld.de/idm/semiotik/peirces_zeichen.pdf, 2004

Patrick Hupe: Eine daten- und prozessorientierte Architektur zur Integration kooperierender Informationssysteme. Diplomarbeit, Technische Universität Hamburg-Harburg, Arbeitsbereich Softwaresysteme, 2000

Peter Bunemann, Sanjeev Khanaa, Wang-Chiew Tan: Why and Where: A Characterization of Data Provenance. In Lecture Notes in Computer Science. Band 1973, Database Theory - Proceedings of the ICDT 2001, 8th International Conference Springer-Verlag, 2001, S. 316 – 330

Peter C. Lockemann, Joachim W. Schmidt: Datenbank-Handbuch. Springer-Verlag, 1987

P. Dourish, V. Bellotti: Awareness and Coordination in Shared Workspaces. In Proceedings of ACM CSCW 92 Conference on Computer-Supported Work. 1992, S. 107 – 114

Sun Microsystems, Inc.: Java Technology., http://java.sun.com

The Apache Software Foundation: The Jakarta Site - Apache Tomcat. http://jakarta.apache.org/tomcat/index.html, 2004

The Apache Software Foundation: mod_rewrite - Apache HTTP Server. http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html, 2004

The Apache Software Foundation: WebServices - SOAP. http://ws.apache.org/soap, 2004

The Apache Software Foundation: Welcome to Web Services Project @ Apache. http://ws.apache.org, 2004

The Apache Software Foundation: Welcome - The Apache HTTP Server Project. http://httpd.apache.org, 2004

Warburg-Haus Hamburg: Warburg-Haus Hamburg. http://www.warburg-haus.hamburg.de, 2004

webdav.org: WebDAV Resources. http://www.webdav.org, 2004

Wil van der Aalst, Kees van Hee: Workflow Management - Models, Methods, and Systems. Zweite Auflage. MIT Press, 2002

World Wide Web Consortium: Extensible Markup Language (XML). http://www.w3.org/XML/, 2004

World Wide Web Consortium: HTTP - Hypertext Transfer Protocol Overview. http://www.w3.org/Protocols/, 2004