UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik

Verteilte kooperative Wissensräume

Dissertation

Schriftliche Arbeit zur Erlangung
des akademischen Grades
Doktor der Naturwissenschaften

Thomas Bopp

Februar 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die ersten Ideen zur Unterstützung computergestützter kooperativer Arbeit entstanden bereits in den 60er Jahren (vgl. Engelbart und English 1968) mit der folgenden Prägung des Begriffs CSCW (Computer Supported Cooperative Working) als Beschreibung des sich ausbildenden Forschungsfelds (Greif 1988). Bis heute sind CSCW-Systeme (Groupware) lediglich auf einzelne, gemeinsam genutzte Bereiche bezogen. Diese sind im Sinne der Client-Server-Architekturmodelle durch einen Server verwaltet. Damit sind die Handlungsräume der Benutzer ebenfalls auf den Handlungskreis dieses Servers beschränkt.

Ziel ist es eine nächste Stufe kooperativer Zusammenarbeit auf Basis virtueller Wissensräume (Hampel 2002) zu schaffen, die abgelöst von Serverstrukturen einen transparenten Wechsel zwischen verschiedenen Wissensräumen erlaubt. Gleichzeitig sind Handlungen von Benutzern im Idealfall ebenfalls unabhängig von Servern und erfolgen übergreifend virtueller Grenzen. Grundlage einer Kooperation ist auf technischer Ebene ein gemeinsamer Datenraum zwischen verschiedenen Bereichen und eine Kopplung von Servern. Der Fokus dieser Arbeit liegt dabei auf Übergängen zwischen Räumen verschiedener Instanzen eines kollaborativen Systems.[1]

In einem raumbasierten System sind Übergänge zwischen einzelnen Bereichen von zentraler Bedeutung, die einen semantischen Zusammenhang zwischen verschiedenen Räumen anzeigen und dem Benutzer die Möglichkeit geben, sich in diese benachbarten Räume zu bewegen. Das an der Universität Paderborn entwickelte sTeam-System (Hampel 1999; Hampel und Keil-Slawik 2001, 2002) setzt eine solche Raummetapher um, die als natürlicher Kontext einer gemeinsamen Zusammenarbeit dient. Das sTeam-System verfügt als klassisches serverbasiertes CSCW-System über eine Vielzahl von Schnittstellen, die es ermöglichen über unterschiedliche Wege einen Zugang zu den Räumen zu

---

[1] Dieser Ansatz ist nicht zu verwechseln mit frühen Formen replizierter Architekturmodelle für synchrone Werkzeuge, die in der Regel über keine gemeinsame Persistenz der Materialien verfügen.

erlangen und spezifische Sichten auf die Inhalte bereitzustellen. Als Beispiel für eine derartige Sicht gelingt durch die Zuordnung einer Position innerhalb eines Raums eine Darstellung als zweidimensionale Fläche (Whiteboard-Sicht).

Auf technischer Ebene stellt die Grundarchitektur eine Ablage für Objekte dar und bietet umfangreiche Funktionen zur Unterstützung kontinuierlicher kooperativer Arbeit.

Hinter dem Wissensraum steht das theoretische Konzept der Medienfunktionen (Keil-Slawik und Selke 1998), die abstrakte Grundfunktionen für Aktivitäten im Umgang mit Materialien definieren. Damit wird eine Verbindung zwischen Vorstellung und Wirklichkeit für den Anwender ermöglicht und dadurch ein Wahrnehmungsraum geschaffen (Hampel 2002). Der Wissensraum stellt einem Benutzer sämtliche kooperativen Medienfunktionen in Abhängigkeit von Berechtigungen zur Verfügung. Bislang sind Handlungen wie Arrangieren oder Verschieben in der Umsetzung des sTeam-Systems lediglich auf lokale unabhängige Bereiche begrenzt.

Mit einer wachsenden Popularität eines kollaborativen Systems lässt sich feststellen, dass einzelne Institutionen sogar mehrere Installationen von CSCW-Systemen nebeneinander betreiben. Dies ist auf eine notwendige Kontrolle administrativer Funktionen und Skalierbarkeit der Systeme zurückzuführen. Die einzelnen Server existieren dann oftmals vollkommen isoliert voneinander ohne einheitliche Schnittstellen oder Übergänge. Lediglich LDAP stellt eine weit verbreitete, standardisierte Schnittstelle dar, um Benutzer und Gruppen zu verwalten und serverübergreifend zur Verfügung zu stellen. Obwohl damit ein kleiner gemeinsamer Datenraum geschaffen wird, sind für grundlegende Formen kooperativen Arbeitens globale Gruppen- und Rechtestrukturen notwendig, um sich unabhängig von Servern in Gruppen organisieren zu können. Damit können die Mehrzahl aktueller CSCW-Systeme als Insellösungen im Sinne einer serverübergreifenden Zusammenarbeit bezeichnet werden.

Im Kontext verteilter Wissensräume steht ein transparentes Bewegen der Nutzer über Grenzen von Servern hinweg im Vordergrund. Dabei ist das Erzeugen von Übergängen lediglich ein erster Schritt in Richtung einer serverübergreifenden Kooperation von Benutzern. Ein gemeinsamer Datenraum ist notwendig, um Materialien unterschiedlicher Server gegenseitig verfügbar zu machen. Darüber hinaus ist ein einheitliches Benutzermanagement zu gewährleisten, damit ein Benutzer, der auf einem Server zu finden ist, auch auf einem anderen Server gefunden werden kann. Schließlich ist ein Austausch von Daten zwischen den Servern notwendig, um einen vollständig gemeinsamen

Datenraum zu schaffen und damit Benutzern das Übertragen von Materialien zwischen verschiedenen entfernten Räumen zu ermöglichen.

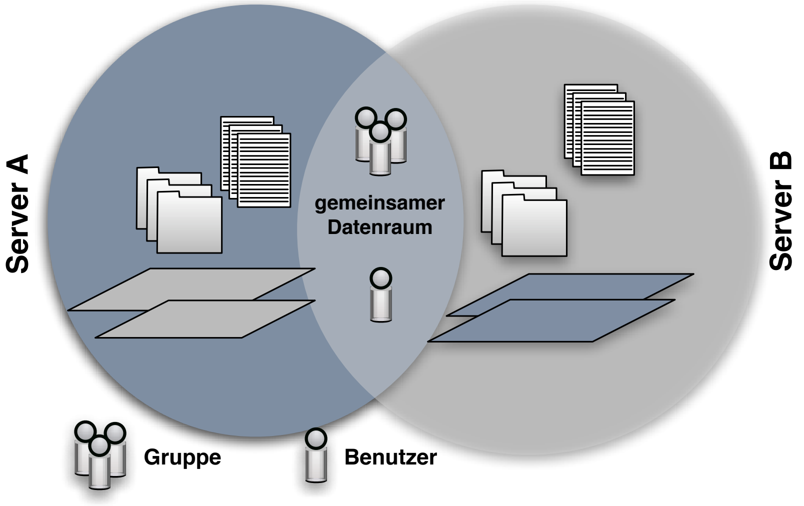


Abbildung 1.1: *Gemeinsamer Datenraum*

In Abbildung 1.1 sind zwei Server abgebildet, bei denen sich nur ein Teil des Datenraums überschneidet. Offensichtlich stehen damit lediglich die Daten der Schnittmenge für beide Server zur Verfügung und Materialien, die sich nicht in dieser Menge befinden, können weder referenziert noch zwischen entfernten Räumen bewegt werden. Als minimale Datenmenge bieten sich Benutzer- und Gruppendaten an, die für serverübergreifende Aktivitäten unbedingt notwendig sind.

Um die vielfältigen Probleme von verteilten kooperativen Wissensräumen mit den Aktivitäten von Benutzern und einer gegenseitigen Wahrnehmbarkeit zu erfassen, wird das Konzept des Handlungsraums[2] eingeführt. Dieser überspannt einen Verbund von Servern und ist eine Erweiterung eines vollständig gemeinsamen Datenraums um Ereignisse.[3] Dadurch sind die Benutzer des Verbunds in der Lage, auf alle Dokumente, unabhängig des physischen Standorts, zurückzugreifen. Der Handlungsraum bietet daher eine technische Basis für kooperative Medienfunktionen und zielt darauf ab, diese Medienfunktionen auch

---

[2] Bei einem Handlungsraum handelt es sich um ein technisches Konzept, das nicht mit anderen Definitionen eines Handlungsraums zu verwechseln ist.

[3] Der Handlungsraum umfasst im Gegensatz zu einem Datenraum immer *sämtliche* Daten.

im serverübergreifenden Betrieb zu ermöglichen. Damit ist die Grundlage kooperativen Arbeitens ohne Abhängigkeit von bestimmten Servern gegeben.

Des Weiteren zeichnet sich ein Handlungsraum durch eine Ereignisorientiertheit aus, die durch eine globale Verarbeitung von Ereignissen im Verbund charakterisiert ist. Diese werden dementsprechend nicht nur lokal in einem Server verarbeitet, sondern wenn nötig auch innerhalb des gesamten Verbunds ausgelöst. Erst dadurch wird ein Austausch von Veränderungen der Serverzustände innerhalb des Verbunds ermöglicht und damit die Wahrnehmbarkeit der Aktivitäten anderer Benutzer gewährleistet. Dies lässt sich auf eine Abfolge von Aktion, Modifikation, Ereignis und Wahrnehmung zurückführen.

Die Schaffung eines gemeinsamen Handlungsraums ist nicht mit einfachen Ansätzen zu realisieren. Vielmehr sind Architekturkonzepte sowohl im Bereich von Online-Spielen (Massive Multiplayer Online Games), Cooperative Virtual Environments (Benford et al. 2001) (CVE) und Multi User Dungeons (Churchill und Bly 1999) (MUD) von Belang (Kapitel 2). In diesen Systemen kooperieren und agieren Benutzer (Spieler) direkt miteinander und bekommen sofortige Rückmeldungen über die Aktion eines anderen Benutzers. Virtuelle Umgebungen bieten erste Ansätze einer Verteilung der Welt in Regionen/Zonen, die möglichst unabhängig voneinander sind. Besonders im Bereich von Online-Spielen existieren praxistaugliche Plattformen für viele Benutzer, die damit auch im Hinblick auf eine Skalierbarkeit von Interesse sind. Im Bereich der Peer-to-Peer-Dateisysteme werden Dateien auf alle beteiligten Knoten verteilt und es wird so ein gemeinsamer Datenraum erzeugt (Androutsellis-Theotokis und Spinellis 2004). Allgemein handelt es sich bei allen Knoten um gleichberechtigte Partner, die jeweils Dienste zur Verfügung stellen und gleichzeitig Dienste anderer Knoten in Anspruch nehmen. Aufgrund der allgemeinen Verfügbarkeit der Daten in diesem Netzwerk handelt es sich um einen gemeinsamen Datenraum, der alle Daten umfasst.

Entsprechend gibt es in den genannten Systemklassen verschiedene Entwurfsqualitäten, die für verteilte Wissensräume und einen serverübergreifenden Handlungsraum von Bedeutung sind. Dazu zählen funktionale Beziehungen und geografische Verteilungen, die für eine Verteilung von Wissensräumen genutzt werden können. Um diese in Beziehung zu setzen, muss dabei zunächst das Konzept des Wissensraums mit den entstehenden Strukturen genau erläutert werden (Kapitel 3).

Zielvorstellung ist eine Architektur für ein System, das für den Benutzer unsichtbar über mehrere Server verteilt arbeitet und dabei schnelle Zugriffe auf alle Bereiche aufweist. Benutzer bewegen sich transparent von einem Wissensraum in einen anderen, der sich auf einem beliebigen Server des Verbunds befinden kann. Dort treffen sie andere Benutzer, mit denen sie kommunizieren und kooperieren können.

Das Schaffen von Übergängen zwischen virtuellen Wissensräumen unterschiedlicher Server bildet daher einen Kernpunkt dieser Arbeit. Folgende Motivationen zur Verteilung von Wissensräumen über verschiedene Server sind identifizierbar:

- Mobilität in Wissensräumen: Benutzer können sich frei innerhalb eines Verbunds und zwischen beliebigen Wissensräumen bewegen.
- Kooperatives Arbeiten mit Benutzern eines anderen Servers: Benutzer können mit Benutzern und Gruppen, die sich auf einem anderen Server befinden, frei interagieren. Es existiert kein Unterschied kooperativen Arbeitens innerhalb eines einzelnen Servers oder eines Serververbunds.
- Verbesserung der Skalierbarkeit eines Systems: In einem Verbund stehen mehrere Server für die Beantwortung von Anfragen zu Verfügung.
- Verteilung von Ressourcen: Neben der Rechenleistung können weitere Ressourcen zwischen verschiedenen Systemen aufgeteilt werden, sodass z.B. umfangreichere Speicherkapazität für Wissensräume zur Verfügung steht.
- Verbesserung der Verfügbarkeit von Servern: Im Verbund kann der Ausfall von einzelnen Servern kompensiert werden. Dadurch wird eine Verfügbarkeit von Wissensräumen im Verbund sichergestellt.

Die genannten Motivationen können je nach Einsatzkontext unterschiedlich stark gewichtet werden, wobei im Rahmen dieser Arbeit besonders das kooperative Arbeiten mit Benutzern unabhängig von Serverstrukturen im Vordergrund steht. Eine Ausnahme bildet der technische Vorteil einer verbesserten Skalierbarkeit, der offensichtlich immer gewünscht ist. Es ist allerdings zu beachten, dass Designkonflikte hier eine Rolle spielen, da der Aufwand einer technischen Umsetzung bei einigen Anforderungen stark anwachsen kann. Die Komplexität verteilter Systeme ist allgemein wesentlich größer als bei zentralisierten Ansätzen. Dies zeigt sich insbesondere darin, dass die Abläufe der verteilten Umgebungen nur schwer nachvollzogen werden können und eine Korrektheit einer Implementierung kaum nachweisbar ist.

Aus diesem Grund sind Forschungen im Bereich von verteilten Systemen als Grundlage verteilter Wissensräume zu beachten (Kapitel 5). Konkurrierende Zugriffe und gemeinsamer Zugriff auf Speicherbereiche beinhalten grundlegende Konzepte, die für ein verteiltes kollaboratives System ebenso von Bedeutung sind wie für beliebige verteilte Anwendungen. Auf der anderen Seiten müssen für kollaborative Systeme zusätzlich die Verteilung von Benutzern, Gruppenstrukturen und Berechtigungen untersucht werden.

Ohne die Einbeziehung existierender Infrastrukturen gestaltet sich der Aufbau einer alltagstauglichen Lösung verteilter Wissensräume sehr aufwändig. In diesem Kontext ist die Umsetzung des Wissensraumkonzepts im sTeam-System berücksichtigt worden. Darauf aufbauend werden in dieser Arbeit unterschiedliche Stufen eines gemeinsamen Daten- und Handlungsraums aufgezeigt. Diese finden sich in den verschiedenen Architekturen verteilter Wissensräume in Kapitel 6. Dazu zählt eine Abschätzung zwischen dem Aufwand einer Umsetzung und dem zu erwartenden Resultat. Ausgangspunkt ist hier ein gemeinsamer Datenraum, der einen ersten Schritt in Richtung von verteilten Wissensräumen darstellt. Der Entwurf der Architekturen erfolgt unter Einbeziehung existierender Systeme und Technologien.

Auf dem Weg hin zu einer Umsetzung verteilter Wissensräume gilt es zunächst Anforderungen zu ermitteln. Aufgrund der technischen Komplexität müssen diese detailliert beschrieben sein. Dazu werden Szenarien aus einer technischen Perspektive verwendet (Kapitel 4), die von einer zunächst zentralisierten Form von Wissensräumen ausgehen, um diese in einem nächsten Schritt zu verteilen. Einbezogen werden hierbei die Entstehung von Übergängen zwischen Wissensräumen und eine Abspaltung von Räumen. Weitere wichtige Elemente sind eine Suche in einem Verbund und das Übertragen von Materialien zwischen verschiedenen Räumen. Weiterhin müssen bei den identifizierten Anforderungen verschiedene Rahmenbedingungen berücksichtigt werden, die aus technischer oder administrativer Perspektive von Bedeutung sein können. Dies beinhaltet den Aufwand einer Umsetzung unter gleichzeitiger Einbeziehung zu erwartender technischer Komplikationen. Um dem Problem einer hohen Komplexität des Entwurfs zu begegnen, bietet sich eine schrittweise Vorgehensweise an, die jeweils zu stabilen Systemzuständen führt. Darüber hinaus sind verschiedene Randbedingungen zu beachten, die unterschiedliche Systeme für verschiedene Einsatzkontexte erfordern.

Aus den verschiedenen Architekturen verteilter Wissensräume werden Entwurfsmuster identifiziert (Kapitel 7), die einen zentralen Teil dieser Arbeit darstellen. Sorgfältig ausgewählte Entwurfsmuster bieten Lösungen für Aspekte einer Umsetzung im Bereich von verteilten Wissensräumen und sind für den Aufbau eines flexiblen Grundsystems von Bedeutung. Zu diesen Mustern zählen sowohl grundlegende Entwurfsmuster wie Remote-Proxy als auch komplexe ereignisbasierte Microkernel und Persistenzmanager.

Im folgenden Kapitel werden zunächst Ansätze verteilter Architekturen vorgestellt. Diese Architekturen umfassen alltagstaugliche Plattformen für viele Benutzer (Massive Multiplayer Online Games) und Peer-to-Peer-Lösungen, die einen Austausch von Dateien zwischen Benutzern bieten.

2 Ansätze verteilter Strukturen

Eine Verteilung von Wissensräumen muss existierende Technologien im Bereich verteilter System berücksichtigen. Aus diesem Grund werden in diesem Kapitel verschiedene Multi-Server-Architekturen und relevante Arbeiten aus verschiedenen Forschungsgebieten detailliert vorgestellt.

Grundsätzlich muss bei kollaborativen Systemen zwischen klassischen serverzentrierten und replizierten Architekturen[1] unterschieden werden, die einen Objekt-Cache auf Seite der Clients verwenden. Es handelt sich in der Regel um Client-Server-Systeme, die Materialien auf einem Server bereitstellen. Benutzer verbinden sich und kooperatives Arbeiten findet statt — z.B. werden Materialien abgerufen, auf einem Server abgelegt oder auch kommentiert. Der Server übernimmt dabei die zentrale Koordination und verteilt Ereignisse an die einzelnen Clients.

Die Vorteile einer Server-zentrierten Architektur ist die einfache Implementierung und Robustheit. Aufgrund der zentralen Datenhaltung kommt es zu keinen Inkonsistenzen. Auf der anderen Seite kann die Serverinfrastruktur einen Engpass darstellen. Abhängig von der Internetanbindung und verfügbarer Hardware kann der Server nur eine begrenzte Anzahl von Clients bedienen. So bietet eine Replizierung von Daten durchaus verschiedene Vorteile: Der Kommunikationsaufwand zum Server wird reduziert und es können auch Verbindungsabbrüche überbrückt werden (Huizinga und Mann 1996). Bei replizierten Architekturen wird dies durch eine Zwischenspeicherung der Daten auf Seite eines Clients realisiert, sodass die Daten zu einem späteren Zeitpunkt im Server synchronisiert werden müssen. Dadurch ist es möglich, dass ein Client (z.B. ein Whiteboard) auf diesen Daten direkt arbeitet und damit auch ein Netzausfall, zumindest kurzfristig, überbrückt werden kann.

Eine Synchronisation der Daten ist auf dem Server nicht immer ohne Konflikte möglich. Es können verschiedene Strategien angewendet werden, um solche aufzulösen oder diese bereits im Vorfeld auszuschließen. So geht die Bandbreite

---

[1] Beispiele für eine serverzentrierte Architektur sind das BSCW-System (Bentley und Appelt 1997) und Cure (Haake et al. 2004b).

der Lösungen von optimistischen Verfahren („Es wird schon nichts passieren.“) bis hin zum vollständigen *Locking* eines Objekts (einem Client wird exklusiver Zugriff eingeräumt).

Neben diesen Client-Server Lösungen existieren verteilte Architekturen (Peer-to-Peer), die durch Napster und Gnutella an Bedeutung gewonnen haben. Diese zeichnen sich dadurch aus, dass Daten verteilt im Netz abgelegt und Direktverbindungen zwischen einzelnen Knoten (Peer)[2] hergestellt werden. Ein Datenaustausch wird durch spezielle, zu diesem Zweck entwickelte Protokolle ermöglicht. Dabei existieren sowohl proprietäre Protokolle (Napster (Shirky 2001), Overnet[3]) als auch Open-Source-Entwicklungen wie JXTA (Gong 2001), Gnutella[4], Freenet (Clarke et al. 2001) und Bram Cohens BitTorrent[5].

Die meisten dieser existierenden Ansätze kommunizieren lediglich innerhalb des eigenen Peer-to-Peer-Netzes, ohne bestehende Server-Infrastrukturen einzubeziehen. Im Bereich der kooperativen Wissensorganisation werden Peer-to-Peer-Lösungen zumeist in mobilen Szenarien eingesetzt (vgl. Wessner et al. 2003). Auch hier erfolgt keine Einbettung in klassische Infrastrukturen und es werden wieder neue Insellösungen kooperativen Arbeitens geschaffen.

Für ein Zusammenwirken der unterschiedlichen Systeme sind offene Schnittstellen notwendig, sodass Daten und Anfragen zwischen den Systemen ausgetauscht werden können.

Dieses Kapitel stellt in den nächsten Abschnitten verschiedene Multiserversysteme vor, die einen Überblick über verschiedene Ansätze von verteilten Architekturen vermitteln sollen. Dabei dient der Abschnitt „Multiserversysteme“ dazu, einen allgemeinen Überblick zu geben und verschiedene Systeme vorzustellen, die in irgendeiner Form einen Multiserveransatz darstellen. So bietet das IRC-System in der ursprünglichen Entwicklung z.B. keine echte Multiserverarchitektur, sondern entwickelten sich aufgrund der wachsenden Anforderungen zu einem verteilten System und ist in dieser Hinsicht einer wachsendenden Struktur von Bedeutung.

Eine weitere wichtige Entwicklung stellen CVE (Collaborative Virtual En-

---

[2] Eine Anwendung, die sowohl Client als auch Server ist. Ein Peer nimmt also Dienste in Anspruch und bietet gleichzeitig eigene Dienste an. Peer-to-Peer ist dabei nicht eindeutig von Client-Server abzugrenzen (vgl. Minar und Hedlung 2001).

[3] Overnet basiert auf dem Kademlia Distributed Hash Table (Maymounkov und Mazieres 2002) und ist verfügbar unter http://www.overnet.com[Stand: 24.11.2005].

[4] Informationen zu der Entwicklung von Gnutella sind verfügbar unter http://groups.yahoo.com/group/the_gdf [Stand: 02.12.2005].

[5] Verfügbar unter http://www.bittorent.org [Stand: 11.12.2005].

vironments) dar, die den Benutzern eine virtuelle Umgebung bieten. Aus dem Bereich dreidimensionaler Umgebung sind besonders die Systeme NPSNET (Zyda et al. 1992), SPLINE (Waters et al. 1997) und MASSIVE (Greenalgh et al. 2000) zu nennen. Die Anforderungen sind hier weitaus umfangreicher als z.B. in einem textbasierten CVE, da eine Vielzahl von Ereignissen entstehen, die an die beteiligten Clients gesendet werden müssen.

In dieselbe Kategorie sind auch die MMOG (Massive Multiplayer Online Games) (Jakobsson und Taylor 2003) einzuordnen. Diese sind durch eine sehr große Anzahl an Benutzern gekennzeichnet und benötigen daher eine verteilte Architektur, um einen reibungslosen Spielablauf zu gewährleisten.

Diese Systeme verwenden jeweils objektorientierte Ansätze, wobei Objekte keine abstrakten Dinge, sondern Gegenstände in der virtuellen Umgebung repräsentieren. Ein Objekt ist dabei durch Attribute und Berechtigungen beschrieben — in grafischen Umgebungen wird jedem Objekt darüber hinaus eine grafische Repräsentation zugeordnet. Benutzer können die Gegenstände manipulieren, übertragen und in festgelegter Weise benutzen.

2.1 Multiserversysteme

Als Multiserversystem wird ein System bezeichnet, das auf verschiedenen Servern verteilt läuft. In dieser Hinsicht sind diese Systeme wichtig in Bezug auf verteilte Wissensräume und ermöglichen erste grundlegende Überlegungen einer Verteilung. Es muss dabei zwischen zwei unterschiedlichen Motivationen der Entstehung solcher Systeme unterschieden werden. Auf der einen Seite spielt die Skalierbarkeit die entscheidende Rolle, da zunächst einzelne Server aufgrund einer hohen Last auf verschiedene Server verteilt worden sind. Daneben stehen Multiserversysteme, die durch ein Zusammenwachsen von Systemen zustande gekommen sind. Dabei sind Server an verschiedenen Stellen in Betrieb genommen worden und erst später sind Übergänge entstanden, sodass aus den einzelnen Teilsystemen ein großes System entstanden ist (vgl. die Entstehung von IRC: IRC 1993).

Multiserversysteme existieren schon seit geraumer Zeit. Bereits das Internet selbst, als Zusammenschluss verschiedener Rechner, kann als ein Multiserversystem angesehen werden. Aufbauend auf diesem Netz entstand das Usenet (Quarterman und Hoskins 1986) als Verbindung zwischen verschiedenen News-

Servern[6], die eine Vielzahl von Foren für Benutzer zum Informationsaustausch bereitstellen. Eine zentrale Kontrolle gibt es zwischen den Servern nicht und damit existiert zumindest eine Parallele zu den Peer-to-Peer Systemen (Minar und Hedlung 2001).

Später entstanden Chat-Systeme wie Internet Relay Chat (IRC) (IRC 1993), die es Benutzern erlauben miteinander in Kanälen direkt zu kommunizieren. Dies stellt zugleich eine erste Form synchroner Kommunikation dar.

Einzelne IRC-Server dienen als Zugangsknoten zu IRC-Netzen wie Undernet oder Dalnet.[7] Dabei kommunizieren Server untereinander und einzelne Kanäle innerhalb des Netzes sind tatsächlich über mehrere Server verteilt. Bei dem Ausfall eines oder mehrerer Server sind die Kanäle dann nur noch den entsprechenden verbleibenden Servern zugeordnet. Dadurch können sich die Personen nur noch innerhalb des verbleibenden Verbundes unterhalten. Gleichzeitig gibt es so statt einem einzelnen Kanal auf jedem Teil des Verbundes diesen Kanal. Ist der Verbindungsausfall überwunden, kann wieder in einem Kanal miteinander kommuniziert werden. Für die Benutzer stellt sich dies durch einen Verbindungsausfall der betroffenen Benutzer dar. Der Ausfall wird also durch einen Benutzer wahrgenommen und es ist offensichtlich, welche Benutzer weiterhin im Kanal kommunizieren können (die Benutzerliste wird entsprechend aktualisiert).

Im Gegensatz zu IRC bieten Instant-Messaging-Systeme (IM-Systeme) hauptsächlich direkte Kommunikation zwischen Benutzern. Anstelle von Kanälen stehen hier einzelne Chat-Fenster, die jeweils der Kommunikation zwischen zwei Teilnehmern zugeordnet sind. Gleichzeitig wird eine Liste mit Freunden und anderen bekannten Benutzern geführt, die Informationen darüber liefert, welche Benutzer gerade erreichbar sind. Der Server eines solchen Systems dient hauptsächlich zur Umverteilung der Nachrichten. Für einen Benutzer wird lediglich die Liste der befreundeten Personen (Buddyliste) auf dem Server hinterlegt.

Insgesamt stellen solche synchronen Kommunikationskanäle wichtige Bestandteile von kollaborativen Systemen dar. So beschreiben Nardi et al. (2000), dass Instant Messaging nicht nur zum Austausch von Informationen benutzt wird, sondern auch zum Aushandeln weiterer Kommunikationskanäle dient.

---

[6] Zu Beginn bestand lediglich eine Verbindung zwischen zwei Rechnern. Einer davon befand sich an der Universität von North Carolina, der andere an der Duke Universität.

[7] Die Webseiten dieser beiden Netze sind unter http://www.under.net und www.dal.net erreichbar.

Darüber hinaus werden reale Treffen vereinbart, indem Zeiten und Orte abgesprochen werden.

Ein ewähnenswertes Instant-Messaging-Protokoll ist die Open-Source-Entwicklung Jabber[8], die im Gegensatz zu anderen Instant-Messaging-Systemen wie ICQ über ein offenes Protokoll verfügt. Daraus folgt eine einfache Verwendbarkeit in eigenen Applikationen.[9] Trotzdem stellt ICQ das am weitesten verbreitete Instant-Messaging-System dar und es existieren eine Vielzahl von Clients und Bibliotheken.

Andere Multiserver Systeme existieren im World Wide Web zur Verbesserung der Beantwortung von Anfragen. So sind beliebte Webseiten häufig so stark frequentiert, dass ein einzelner Server unter dieser Last zusammmenbrechen würde. Bei statischen Inhalten können Replikationsmechanismen leicht Abhilfe schaffen. Dabei werden die gleichen Inhalte auf verschiedenen Servern hinterlegt und durch so genanntes *Load Balancing* werden die Anfragen auf freie Server verteilt (Cardellini et al. 1999).[10] Die einfachste Methode ist dabei einen Benutzer initial auf einen Server umzuleiten, sodass weitere Anfragen direkt an den gewählten Server gesandt werden. Diese Vorgehensweise führt dazu, dass der Benutzer die geänderte URL in der Adresszeile seines Browsers sehen kann.

Bei dynamischen Inhalten ist die Replikation von Inhalten wesentlich komplexer, denn alle Variablen, die den Inhalt bestimmen, müssen in das Caching einfließen. Dies bedeutet eine Aktualisierung der gespeicherten Inhalte auf allen Servern, falls sich eine Variable ändert. Je mehr Variablen diesen Prozess also beeinflussen, umso weniger Vorteile ergeben sich durch die Replikation der Inhalte (Wolfson et al. 1997).

Ein anderer Ansatz in diesem Bereich ist eine Verteilung von Inhalten auf verschiedene Server. Dabei werden mehrere Webserver eingesetzt und HTML-Dokumente so dynamisch angepasst, dass Verweise[11] von einem Server auf einen anderen Server zeigen, der über die gleichen Inhalte verfügt (Li und

---

[8] Informationen sind unter http://www.jabber.org [Stand: 11.12.2005] verfügbar.

[9] Das unterliegende Protokoll ist zur Standardisierung als RFC veröffentlicht (Jabber 2004).

[10] Es sind verschiedene Verfahren zu unterscheiden. Dabei gibt es sowohl Load Balancing auf Seite des Servers, der z.B. die Anfragen an entsprechende freie Server weitergibt, als auch auf Seite eines Clients.

[11] Verweise in HTML-Dateien sind in der Form *Beschreibung. Eine URL (Universal Ressource Identification) beschreibt eine Ressource im WWW.* Es handelt sich dabei um so genannte Hyperlinks, die von einem Startdokument zu einer Zielseite verweisen.

Moon 2001). Alle Server, die diesem Verbund zugeordnet sind, arbeiten also mit den gleichen replizierten Dokumenten. Die Modifikation der Hyperlinks geschieht zur Laufzeit und richtet sich nach der Anzahl der Anfragen[12], die der lokale Server und insgesamt die Server im Verbund zu bearbeiten haben. Ist die Belastung des lokalen Servers sehr hoch und es existieren weitere Server, die nicht stark belastet sind, wird eine Umleitung vorgenommen.[13]

Eine Aufteilung von Anfragen bei einer großen Zahl von Benutzern ist auch im Bereich von Collaborative Virtual Environments sinnvoll. Es handelt sich dabei um Systeme, die eine virtuelle Welt abbilden und in der Regel einen dreidimensionalen Raum verwenden. In dieser Hinsicht handelt es sich um Systeme, die gewisse Parallelen zu einem raumbasierten CSCW-System aufweisen. Aufgrund der hohen Anforderungen an die Kommunikation zwischen Client und Server mit ständigen Übermittlungen von Aktivitäten und Bewegungen existieren viele Ansätze einer Aufteilung der virtuellen Welt auf verschiedene Server. Aus diesem Grund kann eine derartige Aufteilung in verschiedene Zonen als Vorlage für eine Verteilung von Wissensräumen dienen.

2.2 CVE und MMOG

Collaborative Virtual Environments (CVE) sind virtuelle Umgebungen, in denen sich Benutzer treffen, um dort ein gemeinsames Ziel zu verfolgen (Benford et al. 2001). Zu CVE zählen damit sowohl textbasierte Multi User Dungeons (MUD) als auch dreidimensionale virtuelle Umgebungen — an dieser Stelle sollen allerdings die dreidimensionalen Umgebungen im Vordergrund stehen.

Der Zweck eines CVEs kann sehr unterschiedlich sein und reicht von seriösen Anwendungen zu Spielen mit vielen Benutzern. Historisch gesehen können CVE als Zusammenführung der Entwicklungen von virtuellen Realitäten und CSCW-Systemen gesehen werden (Benford et al. 2001).

Die Metapher des dreidimensionalen Raums eines CVEs orientiert sich an der realen Welt. Avatare sind ebenfalls dreidimensional beschrieben und interagieren mit Objekten. Der Grad der Details kann ganz unterschiedlich sein: Es gibt einfache grafische Repräsentationen bis hin zu detaillierten Darstellungen einer Person mit Anzeige von Gebärden und Emotionen (Benford et al. 1995).

Speziell im Bereich von Spielen sind so genannte Massive Multiplayer Online-

---

[12] Diese Größe wird auch als Serverlast bezeichnet.

[13] Ein solches Verfahren wird als Load-Balancing bezeichnet.

Games entstanden, die sich dadurch auszeichnen, dass eine sehr große Anzahl von Spielern diese Systeme gleichzeitig bevölkern. Die Interaktion zwischen Spielern kann je nach Art des Spiels ganz unterschiedlich ausfallen. In Fantasiespielen ist eine Kooperation zum gemeinsamen Bekämpfen von Gegnern und Bestehen von Aufgaben üblich.

Beide Arten von Systemen bieten den Benutzern virtuelle Welten, durch die diese sich bewegen und interagieren können. Der Unterschied liegt im Prinzip lediglich in der Ausrichtung der Anwendung. Während es sich bei CVE eher um seriöse Anwendungen handelt, geht es bei den erfolgreichen Online Rollenspielen (MMORPG) um eine spielerische Interaktion der Benutzer.

Massive Multiplayer Online Games beherbergen eine sehr große Zahl von Spielern. Diese liegen bei kommerziellen Systemen in dem Bereich von mehreren tausend Spielern, die zur gleichen Zeit in der Spielwelt aktiv sind. Offensichtlich sind einzelne Server bei einer solchen großen Zahl von Benutzern vollkommen überfordert, sodass die Spielwelt auf mehrere Server aufgeteilt werden muss. Die Lösung ist also eine Partitionierung der Spielwelt in verschiedene Bereiche (vgl. IBM Gamegrid 2005; Berry 2004; Cai et al. 2002). Dabei werden z.B. einzelne Kontinente so verteilt, dass die Übergänge zwischen den Servern möglichst beschränkt und die Bereiche weitgehend unabhängig sind. Dadurch wird ein Austausch von Ereignissen zwischen den einzelnen Servern weitgehend vermieden.

Ein Beispiel für solche schmalen Übergänge sind Schiffslinien, die einzelne Kontinente miteinander verbinden, und im Laufe der Reise wechselt ein Spieler dann transparent die Verbindung zu einem neuen Server, der für den neuen Kontinent zuständig ist. Eine Landverbindung birgt größere Probleme, da an der Grenze zwischen den Kontinenten z.B. aufgrund der Sichtweiten der Avatare Ereignisse über diese Grenzen ausgetauscht werden müssen. Ohne einen solchen Austausch endet die Wahrnehmung eines Avatars abrupt an der Grenze zwischen den Kontinenten. Diese Sachverhalte sind bereits in den CVE-Systemen als Area of Interest Management berücksichtigt worden (vgl. Benford et al. 2001).

Neben dieser räumlichen Aufteilung einer virtuellen Welt können Objekte auch entsprechend der Häufigkeit ihrer Verwendung (temporal partitioning) und anhand ihrer Funktion gruppiert werden (Cai et al. 2002).

Abbildung 2.1 zeigt beispielhaft eine Aufteilung einer virtuellen Welt in verschiedene Zonen. Zwischen den einzelnen Kontinenten bestehen keine Übergänge, sodass der jeweilige Abschnitt der virtuellen Welt natürlich für die Spieler

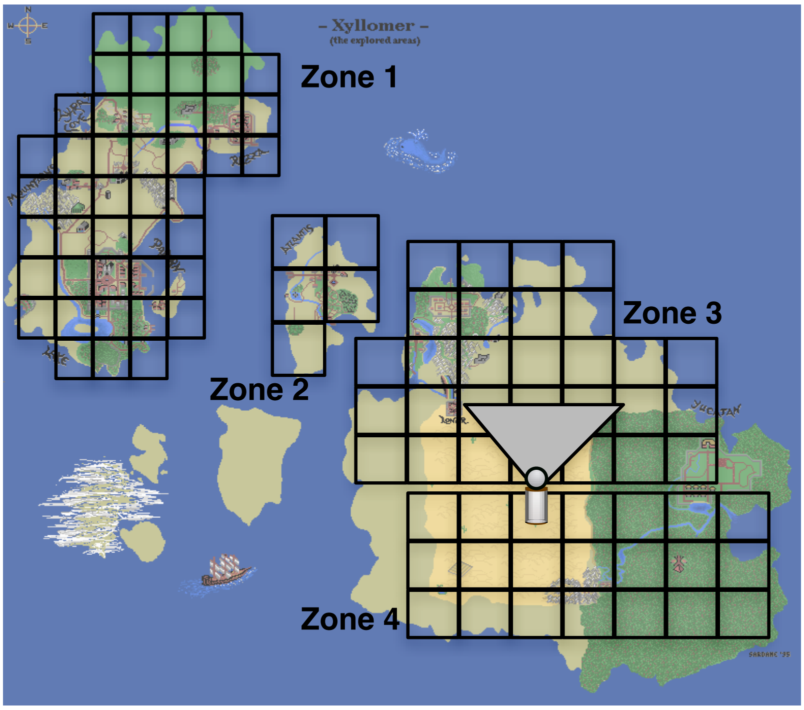


Abbildung 2.1: *Partitionierung einer virtuellen Welt in verschiedene Zonen (Quelle: http://www.xyllomer.de)*

begrenzt ist. Der in der Mitte dargestellte Benutzer befindet sich allerdings zwischen zwei verschiedenen Zonen, die aneinander grenzen. Wie zu sehen ist, müsste der Sichtbereich des Benutzers von der Zone 3 bis in Zone 4 reichen. Bei einer Aufteilung der Zonen auf verschiedene Server ist in diesem Fall eine Kommunikation zwischen den beiden Servern notwendig. Die übrigen unabhängigen Gebiete erfordern hingegen nur geringe Kommunikation bei einem Wechsel von einem Gebiet in ein anderes.

Eine ähnliche Vorgehensweise findet sich bei dem Neteffect-Projekt (Das et al. 1997), wo Gemeinschaften (Online-Communities) über verschiedene Server aufgeteilt werden. Auch hier ist festgestellt worden, dass die Querverbindungen unterschiedlicher Communities nur marginal sind und daher eine Aufteilung problemlos erfolgen kann. Jederzeit kann so auch eine Gemeinschaft von einem Server auf einen anderen verschoben werden. Dabei müssen sämtliche betroffenen Objekte nur von einem Server exportiert und auf einen anderen importiert werden. Falls ein Benutzer in mehr als einer Community aktiv ist, muss er angeben welche er benutzen möchte. Ein Wechsel dieser Community kann später einen Wechsel des Servers beinhalten, wenn sich die Communities auf unterschiedlichen Servern befinden.

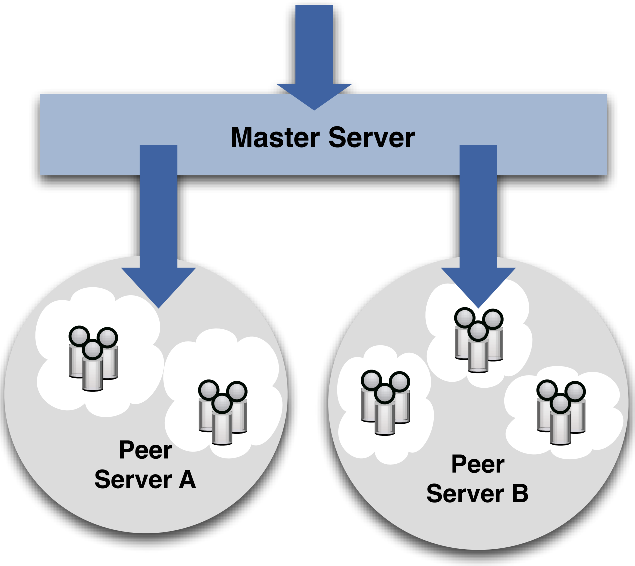


Abbildung 2.2: *Verteilung von Communities über mehrere Server*

Abbildung 2.2 zeigt die Verteilung verschiedener Communities über zwei so genannte Peer-Server. Davor geschaltet ist ein Master-Server, der die Kommunikation mit den Clients regelt und die Anfragen an den entsprechenden Peer-Server weiter leitet. Alle Strukturen, die den jeweiligen Communities zugeordnet sind, befinden sich auf demselben Server.

Croquet (Smith et al. 2003) ist ein globales Multi-User-Netzwerk, das sich an das World Wide Web anlehnt. Allerdings handelt es sich um eine virtuelle, dreidimensionale Umgebung. Benutzer haben ihre eigenen Bereiche (Welten), die untereinander durch Portale verbunden werden können. Durch eine konsequent objektorientierte Architektur werden in diesem System ausschließlich kollaborative Objekte verwendet, die Benutzer gemeinsam nutzen können.

Die Skalierbarkeit solcher virtueller Welten ist ein wichtiges Forschungsfeld, da es bei einer großen Anzahl von Benutzern leicht zu Engpässen kommen kann. Der gesamte virtuelle Raum als Einheit ist durch ein Koordinatensystem beschrieben und jedes Objekt in dem Raum kann durch eine Koordinate beschrieben werden. Wenn sich viele Objekte im Raum befinden und viele

Benutzer interagieren, müssen unter Umständen viele Benachrichtigungen verschickt werden, um die einzelnen Clients auf dem aktuellsten Stand zu halten. Bewegungen von Objekten werden zur Vereinfachung und aufgrund von Leitungsverzögerungen z.B. durch Richtungsvektoren beschrieben.

Eine erste Optimierung, die auch den Bereich von dreidimensionalen Darstellungen betrifft, ist ein *Area of Interest Management*. Dabei wird der Blickwinkel des Benutzers berücksichtigt und nur Benachrichtigungen über Veränderungen im Sichtfeld des Benutzers übertragen. Es handelt sich also um eine Filterung von Ereignissen.

Aufbauend auf dieser Filterung von Ereignissen bietet sich eine Aufteilung der virtuellen Welt in verschiedene Zonen an. Die Grundidee einer solchen Aufteilung ist, dass eine größere Umgebung nicht vollständig von einem einzelnen Benutzer in einem Augenblick erfasst werden kann (Barrus et al. 1996). Daher ist es sinnvoll einzelne Bereiche zu schaffen, die unabhängig voneinander bearbeitet werden können (vgl. Tabelle 2.1).

Das SpaceFusion-System (Sugano et al. 1997) teilt die virtuelle Welt ebenfalls in Bereiche ein. Diese Regionen beschreiben jeweils virtuelle Städte, wobei ein einzelner Server in einem Serververbund für eine oder mehrere Regionen zuständig ist. Bei den Clients wird zwischen *Watcher* und *Deliverer* unterschieden. Während die *Watcher* bestimmte Regionen lediglich überwachen, kontrollieren die *Deliverer* Objekte in einer Region.[14] Durch eine Überwachung mehrerer Regionen, die auf verschiedenen Servern liegen, stellen auch die Grenzen zwischen den Servern kein Problem dar. Dazu muss ein Client mit zwei Servern verbunden sein und dort Regionen beobachten.

Es zeigt sich bei allen Systemen eine einheitliche Vorgehensweise der Aufteilung einer virtuellen Welt in verschiedene Zonen anhand von Sichtweiten und der Umgebung eines Benutzers. Diese Partitionierung erfolgt zur Verbesserung der Performanz eines Servers, da die Ereignisse effektiver gefiltert werden können. Trotzdem gibt es bei den existierenden Massive Multiplayer Online Games eine Grenze der maximalen Anzahl von Benutzern, die sich gleichzeitig an ein Server-Cluster anmelden können. Aus diesem Grund werden mehrere Cluster verwendet, die verschiedene parallele Welten der gleichen virtuellen Umgebung darstellen.

Im Folgenden werden Peer-to-Peer-Systeme beschrieben, bei denen Ressourcen über ein Netzwerk verteilt werden, wobei ein Austausch von Dateien im Vordergrund steht. Im Gegensatz zu den CVE handelt es sich bei den Peer-to-

---

[14] Sie kontrollieren z.B. Avatare, aber auch beliebige andere Objekte.

NPSNET (Zyda et al. 1992)	Die Aufteilung der virtuellen Welt geschieht in hexagonale Zonen fester Größe. Neben Ereignissen der lokalen Zellen können auch Ereignisse beliebiger anderer Zellen registriert werden (z.B. Bewegungen von Objekten).
SPLINE (Waters et al. 1997)	Die Welt wird in so genannte *locales* aufgeteilt, die unterschiedliche Größen haben und nebeneinander liegen. Benutzer bekommen Rückmeldungen über Ereignisse der *locale*, in der sie sich befinden, und eventuell über angrenzende *locales*.
MASSIVE-2 (Greenalgh et al. 2000)	Die Aufteilung erfolgt in einzelne Regionen, die aneinander grenzen. Jede Grenze kann unterschiedliche Charakteristiken besitzen, die bestimmen, welche Ereignisse in angrenzende Regionen übertragen werden.

Tabelle 2.1: Aufteilung virtueller Welten in verschiedenen Systemen

Peer-Systemen um neue Entwicklungen, die eine verteilte Struktur zugrunde legen. Damit sind sie offensichtlich für ein verteiltes CSCW-System und eine Verteilung von Wissensräumen von großer Bedeutung.

2.3 Peer-to-Peer-Systeme

Peer-to-Peer-Systeme sind interessante Entwicklungen, die besonders aufgrund der Verteilung von Ressourcen innerhalb eines Netzwerks relevant sind. Weiterhin handelt es sich um dynamische Netze, bei denen immer wieder Peers hinzukommen oder das Netz verlassen. In dieser Hinsicht sind diese Systeme von großer Bedeutung für die Umsetzung eines dynamischen Serververbundes.

Minar und Hedlung (2001) sehen das Internet als erstes Peer-to-Peer-Netzwerk. Die Grundidee beim ARPANET (McQuillan und Walden 1977) ist es Rechenleistung zu teilen und Dateien auszutauschen — Wissenschaftlern sollte ein Austausch von Forschungsergebnissen ermöglicht werden. Jeder der Server war gleichberechtigt und ein Packetaustausch zwischen beliebigen Rechnern war möglich.

Auf der anderen Seite ist der Begriff Peer-to-Peer nicht genau definiert und kann unterschiedliche Bedeutungen haben. Einerseits ist er auch auf einen Verbund von Servern anwendbar (einzelne Server, die untereinander kommunizieren) und allgemein auf Programme die Daten von einem Benutzer an einen anderen schicken. So kann selbst E-Mail oder Telefonieren als Peer-to-Peer beschrieben werden (vgl. Shirky 2001). Auf der anderen Seite ist selbst das als erste Peer-to-Peer-Anwendung bekannte Napster streng genommen nicht in den Bereich der Peer-to-Peer-Systeme einzuordnen, da der Datentausch über zentrale Server gesteuert wird.[15]

Aus diesem Grund können die verschiedenen Peer-to-Peer-Systeme in *purist* und *brokered* unterteilt werden (Park und Hwang 2003). Dabei sind Systeme wie Napster wie oben beschrieben keine reinen Peer-to-Peer-Systeme und damit als *brokered* einzustufen. Der Server dient dabei als Vermittler und regelt Suchfunktionalität im Netzwerk. Auf der anderen Seite stehen puristische Peer-to-Peer-Systeme wie Gnutella, in denen sich die Knoten des Netzes nur direkt untereinander austauschen. Dabei kommunizieren immer benachbarte Knoten des Netzes miteinander und eine Suche wird durch die Weiterleitung von Anfragen an die jeweiligen Nachbarknoten realisiert.

Eine erweiterte Unterscheidung der verschiedenen Systeme ordnet die Architekturen in vollkommen dezentralisiert, teilweise zentralisiert und hybrid ein (Androutsellis-Theotokis und Spinellis 2004). Dabei entsprechen vollkommen dezentralisiert und hybrid der obigen Definition von *purist* und *brokered.* Zwischen diesen Einstufungen gibt es zusätzlich teilweise zentralisierte Architekturen, die einigen Knoten eine größere Bedeutung zuordnen. Dabei ist es wichtig festzustellen, dass diese so genannten Supernodes (Mizrak et al. 2003) keine Single Points of Failure darstellen, da das Netzwerk nicht von ihnen abhängig ist.

Im Wesentlichen gilt bei einem Peer immer die Definition einer Mischung aus Client und Server.[16] Dies bedeutet für eine Anwendung sowohl als Client die

---

[15] Napster verwendet Server als Einstiegspunkte in das Netzwerk und zur Realisierung von Suchfunktionalität.

[16] Ein *Servent* ist ebenso definiert, um sich von Peers in hybriden Peer-to-Peer-Systemen mit Servern abzugrenzen. Im Allgemeinen wird der Begriff Peer-to-Peer jedoch weniger auf die technischen Eigenschaften eines Systems bezogen, sondern ergibt sich aus der

Dienste eines Servers anzunehmen als gleichzeitig auch die Rolle eines Servers einzunehmen und für die anderen Teilnehmer Dienste anzubieten. Prinzipiell ist damit lediglich festzustellen, dass der Bergriff Peer-to-Peer eine konzeptuelle Sichtweise darstellt, die nach Möglichkeit auf zentralisierte Dienste verzichtet.

Peer-to-Peer-Systeme formen eigene Netzwerke, die eine neue Ebene über dem darunter liegenden physischen Netzwerk darstellen. Daher spricht man von so genannten Overlay Networks, die eigene Routing-Algorithmen besitzen (Androutsellis-Theotokis und Spinellis 2004). Beispiele dieser Systeme sind Pastry (Rowstron und Druschel 2001a) und Chord (Stoica et al. 2001).

Bei einer Anwendung des Peer-to-Peer-Gedankens auf virtuelle Wissensräume müssen Dienste in Abhängigkeit zum jeweiligen Raum stehen: Server bieten im Wesentlichen Wissensräume mit den zugehörigen Objekten (Dokumente, Objekte zur Strukturierung des Raums, Kommentare), in denen sich andere Benutzer bewegen und mit den Objekten interagieren können. Bei einer Verteilung von Wissensräumen auf mehrere Peers wird ein gemeinsamer Datenraum erzeugt.

Ein Einsatz von Peer-to-Peer-Systemen kann verschiedene Vorteile gegenüber zentralisierten Systemen haben (Milojicic et al. 2002):

- **Verbesserte Skalierbarkeit**: Anstatt an einer zentralen Stelle arbeitet das System verteilt auf den verschiedenen Knoten des Netzwerks. Dadurch kann z.B. die Bandbreite des gesamten Netzes genutzt werden.

- **Zusammenführung von Ressourcen**: Jeder Knoten bringt Ressourcen zu dem gesamten Netz. Dabei kann es sich z.B. um Rechenleistung oder Speicherkapazität handeln, die von allen Knoten gemeinsam genutzt werden.

- **Unabhängigkeit**: Es gibt keine zentrale Instanz, die alles regelt. Stattdessen gibt es unabhängige Organisationen innerhalb des Netzes (z.B. auch ein einzelner Peer).

---

externen Wahrnehmung solcher Systeme (Androutsellis-Theotokis und Spinellis 2004). Eine verbreitete und allgemein anerkannte Definition von Peer-to-Peer ist z.B. „P2P is a class of applications that takes advantage of resources — storage, cycles, content, human presence — available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.“ (Shirky 2000)

- **Anonymität**: In einem Client/Server-System identifiziert der Server typischerweise die Clients durch die übermittelten Authentisierungsdaten. Im Gegensatz dazu werden bei einem Peer-to-Peer-System die Aktivitäten direkt zwischen einzelnen Peers durchgeführt und dadurch Anonymität garantiert. Freenet ist ein Beispiel für ein anonymes System (Clarke et al. 2001).
- **Dynamik**: Es handelt sich um ein dynamisches Netzwerk, bei dem ständig Peers dazukommen und das Netz wieder verlassen. Dies steht besonders im Gegensatz zu statischen Serververbünden.

Es gibt insgesamt eine Vielzahl von Systemen, die sich selbst als Peer-to-Peer definieren oder so klassifiziert werden können. Dazu gehört auch das Grid-Computing, welches die Rechenleistung verschiedener beteiligter Rechner nutzt, um damit umfangreiche Berechnungen durchzuführen.[17] Die Klassifikation ergibt sich teilweise aus der Art der Ressourcen, die in dem System zusammengeführt werden. Während es bei Grid-Computing um Rechenleistung geht, wird bei Peer-to-Peer-Dateisystemen die Speicherkapazität aller Rechner mit einbezogen. Zunächst sollen allerdings die Tauschbörsen Erwähnung finden, durch die der Begriff Peer-to-Peer bekannt geworden ist.

2.3.1 Tauschbörsen

Tauschbörsen haben das Ziel, den Austausch von Dateien zwischen unabhängigen Benutzern zu unterstützen. Dabei liegen die Herausforderungen an die Architektur darin, die Dateien auffindbar zu machen und den Datenaustausch effizient zu gestalten (Saroiu et al. 2002). Die Lösungen für diese Probleme fallen bei den einzelnen Systemen unterschiedlich aus. Während Gnutella einen strikten Peer-to-Peer-Ansatz verfolgt, werden bei Napster Server zur Verwaltung der Dateien eingesetzt. Dazu werden alle Dateien im Netzwerk in einem Index hinterlegt und Peers treten dem Netzwerk über solche Server bei. Im Gegensatz dazu leiten Peers im Gnutella-Netzwerk Suchanfragen an Nachbarn weiter. Dieser Ansatz weist allerdings eine höhere Belastung des Netzes auf (Charwathe et al. 2003).[18]

---

[17] Ein Beispiel für Grid-Computing stellt das bekannte SETI@Home-Projekt dar.

[18] Besonders Peers, die selbst gar keine Dateien zum Download anbieten, spielen hier eine Rolle, da solche Peers trotzdem zur Suche verwendet werden (vgl. Adar und Huberman 2000).

Als Tauschbörsen für Dokumente bieten die Peer-to-Peer-Systeme bereits einen gemeinsamen Datenraum. Dieser beschränkt sich allerdings lediglich auf Inhalte von Dokumenten, die auf Wunsch repliziert werden. Aufgrund dieser Ausrichtung liegt das Konzept eines dezentralen Dateisystems auf der Hand.

Tauschbörsen und verteilte Dateisysteme sind für Wissensräume in Bezug auf eine verteilte Speicherung von Materialien von Bedeutung. Darüber hinaus können die unterliegenden Technologien Hinweise auf eine Umsetzung verteilter Wissensräume geben.

2.3.2 Verteilte kooperative Dateisysteme

CFS (Cooperative File System) (Dabek et al. 2001) stellt ein Dateisystem dar, das den einzelnen Peers einen lesenden Zugriff ermöglicht. Einen ähnlichen Ansatz verfolgt das Ivy System (Muthitacharoen et al. 2002), das jedoch auch schreibenden Zugriff erlaubt. Dies wird durch *Logs* realisiert, die für jeden Teilnehmer alle Änderungen mit führen. Ein lesender Zugriff auf das System erfordert dann eine Auswertung aller *Logs*. Um dieses Verfahren trotzdem effizient zu gestalten, werden von Zeit zu Zeit persönliche Kopien erstellt und nur die aktuellen Änderungen überprüft. Es ist des Weiteren möglich, nur die *Logs* bestimmter Teilnehmer zu verwenden und dadurch ein Vertrauensverhältnis abzubilden.

Auch das PAST-System (Rowstron und Druschel 2001b) erlaubt das Abrufen und Einfügen von Dateien. Dabei sind die Dateien nach einem einmaligen Einfügen persistent und unveränderlich. Es besteht die Möglichkeit einen Zugangsknoten in das Netz zu bilden und optional Speicherkapazität zu bieten. Jeder Knoten wird durch eine ID identifiziert.[19] Die Dateien im PAST-System werden auf verschiedene Knoten kopiert, sodass eine Verfügbarkeit gegeben ist.

Bei Freenet (Clarke et al. 2001) handelt es sich um ein weiteres kooperatives Dateisystem, bei dem eine Publizierung von Inhalten im Vordergrund steht. Dabei unterscheidet sich das System besonders durch eine Anonymität der Benutzer von den oben genannten Systemen. Zur eindeutigen Identifizierung von Dateien werden verschiedene Schlüssel verwendet.[20] Jeder Knoten stellt dem Netz dabei Speicherkapazität zur Verfügung und kann gleichzeitig auf

---

[19] Diese Node-ID wird aus dem öffentlichen Schlüssel des Knotens generiert. Es besteht kein Zusammenhang zwischen dieser ID und der geografischen Position eines Knotens.

[20] Der Keyword Signed Key (KSK) wird z.B. aus einer Kurzbeschreibung der Datei generiert. Der private Teil dieses Keys wird verwendet, um die Datei zu unterzeichnen.

den gemeinsamen Speicher zugreifen. FreeNet arbeitet auf Basis von Dateien und sieht keine Metadaten oder Attribute für diese vor.

Bei Freehaven (Dingledine et al. 2001) handelt es sich um eine anonyme Speicherung von Dateien. Ähnlich wie bei Freenet geht es um Publizieren von Dokumenten. Bei der Anonymität wird zwischen verschiedenen Formen unterschieden. Dazu zählen Anonymität des Autors, des Lesers und des Publizierenden.

OceanStore (Kubiatowicz et al. 2000) stellt ein weiteres kooperatives Dateisystem dar, bei dem Konsumenten eine monatliche Gebühr für die Bereitstellung des persistenten Speichers entrichten sollen. Es handelt sich um ein globales System, dem beliebige Clients beitreten können. Innerhalb des Verbundes wird zwischen den Knoten kein direktes Vertrauensverhältnis aufgebaut, sondern sämtliche Daten werden verschlüsselt übertragen und gespeichert.

Technisch wird bei OceanStore eine Replikation von Objekten verwendet, wobei die Replikate unabhängig von den Servern sind (*floating replicas*). Modifikationen von Objekten basieren auf Updates, die eine neue Version eines Objekts erzeugen. Es wird dabei zwischen aktiven Objekten und archivierten Objekten unterschieden. Die Konsistenz wird durch eine Versionierung der Objekte garantiert.[21] Die Replikation der einzelnen Versionen ist daher auf einfache Weise möglich und erfordert keine Überlegungen im Hinblick auf eine Konsistenz der Daten. Zu jedem Objekt existiert ein so genanntes Primär-Replikat, dem die aktuelle Version des Objekts bekannt ist und welches sämtliche Versionen nachverfolgt. Im OceanStore-Netzwerk werden eng beieinander liegende Knoten zu Pools zusammengefasst, die sehr effizient miteinander kommunizieren können.

Die Sicherheit der Daten wird durch eine Verschlüsselung der Daten garantiert. Bei Zugriffen wird lediglich zwischen Lese- und Schreibberechtigungen unterschieden, die einem Objekt als eine Liste von Berechtigungen (ACL) zugeordnet sind.

OceanStore bietet sowohl Zugriffsmöglichkeiten analog zu einem Dateisystem als auch einen einfachen Transaktions-Zugriff.[22] Im Unterschied zu anderen Peer-to-Peer-Dateisystemen wird ein *introspection*-Verfahren verwendet, bei dem sich das System selbstständig den Gegebenheiten anpasst. Dazu wird ein Zyklus bestehend aus Computation, Observation und Optimization ange-

---

[21] Objekte werden dabei nie gelöscht und nach Modifikationen existieren mehrere Versionen von einem Objekt (vgl. auch Rhea et al. 2003).

[22] Dabei werden die ACID-Eigenschaften erfüllt.

wandt. Zu den Optimierungen gehören die Erkennung von zusammengehörigen Dateien und die Verwaltung der Replikate.[23]

Ein weiterer Ansatz findet sich in dem SAV-System (Stanford Archival Vault). Dabei geht es spezifischer um eine Speicherung von Dokumenten (Cooper et al. 2000). Diese werden einmalig abgelegt und erlauben keine weiteren Modifikationen von Benutzern. Es handelt sich um Systeme zur Archivierung von Dokumenten. Das SAV-System verwendet zu einem Dokument so genannte Set-Objekte, die Zugang zu verschiedenen Darstellungen eines Dokuments bieten. Ein Dokument kann in so einem Archiv in verschiedenen Formaten wie PDF, Postscript oder HTML abgelegt werden.

Das PeerDB-System (Ng et al. 2003) verfolgt einen objektorientierten Ansatz. Jeder Knoten im Netzwerk ist ein vollwertiges Objekt-Management-System. Eine Besonderheit stellt die Suche dar, die ohne ein festes globales Schema abläuft. Dazu werden zu jeder Relation Schlüsselwörter gespeichert, die diese beschreiben. Über diese Schlüsselwörter kann dann ein Abgleich der Relationen erfolgen. Die Suche nach den Inhalten wird durch Agenten unterstützt.

2.3.3 Distributed Hash Tables

Die im vorherigen Abschnitt beschriebenen Peer-to-Peer Dateisysteme sind darauf ausgelegt Dateien möglichst effizient im Netz zu verteilen. Dabei wird von unzuverlässigen Verbindungsinfrastrukturen ausgegangen und die Inhalte werden so repliziert, dass ein Zugriff generell möglich ist. Die Basis dieser Systeme bilden so genannte Distributed Hash Tables (DHT). Diese Systeme verteilen Daten über ein Netzwerk von Knoten und senden Nachrichten auf möglichst kurzen Routen von einem Knoten zu einem anderen (Rowstron und Druschel 2001a). Es handelt sich bei den DHT um eine Abbildung von Hash-Funktionen auf die Knoten in einem Netzwerk. Sowohl Knoten als auch Nachrichten bekommen dabei die ID als Hash-Wert aus demselben Wertebereich zugeordnet. Eine Nachricht wird dann zu dem Knoten gesendet, dessen ID sich am wenigsten von der ID der Nachricht unterscheidet. Das Routing einer Nachricht kann dann mit einer geeigneten Tabelle in wenigen Schritten erfolgen (typisch sind z.B. logarithmisch viele Schritte in Abhängigkeit der Knotenzahl).

Da das Routing von Nachrichten der DHT anstelle der Netzwerkschicht übernimmt, wird von einem Overlay-Netzwerk gesprochen. In dem physikalischen

---

[23] Replica Management passt die Anzahl und den Aufenthaltsort der *floating replicas* an die realen Zugriffe an.

Netzwerk wird die Route durch schrittweise Weiterleitung an die Knoten im DHT-Netzwerk umgesetzt.

Pastry verwendet für die Knoten im Netzwerk einen 128 Bit großen Adressraum. Die Zuordnung der ID für einen Knoten erfolgt über eine Hashfunktion, die für eine gleichmäßige Verteilung der IDs sorgt. Dabei gibt es keinen Zusammenhang zwischen physikalischer Adresse und der ID im Pastry-Netzwerk (vgl. Rowstron und Druschel 2001a).

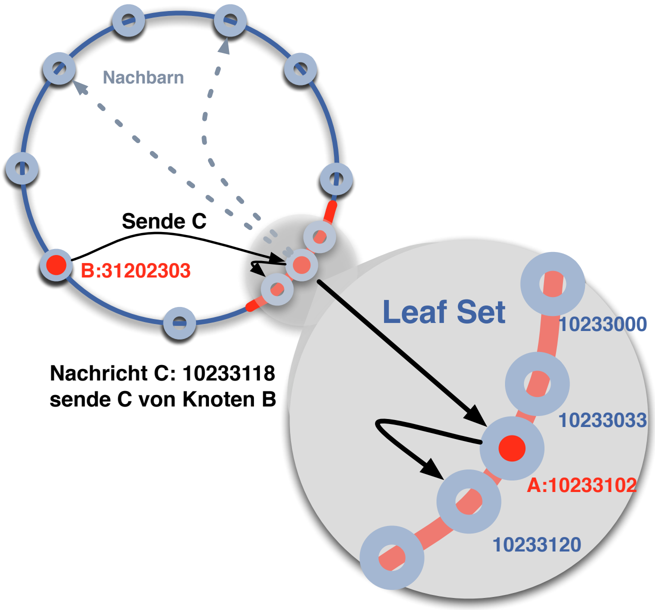


Abbildung 2.3: *Senden einer Nachricht im DHT*

Abbildung 2.3 zeigt den Ablauf einer Nachrichtenübermittlung im DHT-Netzwerk. Zunächst wird die Nachricht mit der ID 10233118 an den Knoten A mit der ID 10233102 geroutet. Von dort kann sie direkt im Leaf Set von A weitergeleitet werden an den Empfänger 10233120, der offensichtlich der Knoten mit der numerisch ähnlichsten ID darstellt. Wenn ein Knoten mit der ID 10233118 existieren würde, müsste die Nachricht diesem Knoten zugestellt werden.

Damit bieten DHTs eine verlässliche Peer-to-Peer-Infrastruktur, um Nachrichten auszutauschen. Dabei sind Themen wie Objekte, Replikation und Ereignisse noch nicht berücksichtigt. Mit Erweiterungen wie Scribe (Rowstron et al. 2001) und PAST (Rowstron und Druschel 2001b) werden auch diese Bereiche abgedeckt, sodass einem vielseitigen Einsatz nichts im Wege steht.

Gleichzeitig handelt es sich um sehr grundlegende Technologien, die für eine Anwendung im Bereich CSCW sehr stark erweitert werden müssen. Im Prinzip kann ein Distributed Hash Table als Peer-to-Peer-Infrastruktur für verteilte Wissensräume genutzt werden.

Mit OpenDHT (Rhea et al. 2005) steht weiterhin ein öffentlicher DHT-Service zur Verfügung, der die Verwendung von DHTs wesentlich vereinfachen soll. Dabei stehen im Wesentlichen die Routinen *get*, *put* und *remove* zur Verfügung. Der Servicecharakter dieser DHT-Implementierung zeigt sich in einer Bereitstellung dieser Schnittstellen über TCP und HTTP.

Neben dieser sehr grundlegenden Entwicklung gibt es bereits Ansätze Kollaboration in Peer-to-Peer-Umgebungen zu ermöglichen.

2.3.4 Kommunikation und Kollaboration

Ein Ansatz der Kommunikation und Kollaboration findet sich im Bereich des verteilten Wissensmanagements, der von autonomen Einheiten ausgeht. Diese verfügen jeweils über einen eigenen Bestand an Wissen, das beliebig organisiert sein kann. Jeder Knoten bietet Dienste für eine Wissens-Community, um lokales Wissen zu stukturieren. Gleichzeitig müssen soziale Strukturen und semantische Bedeutungen koordiniert werden, damit die verschiedenen Knoten zusammenarbeiten können.

Das KEx-System (Bonifacio et al. 2002) stellt eine serviceorientierte Peer-to-Peer-Umgebung zum Wissensmanagement dar. Die einzelnen Einheiten (Peers) sind entweder als *Provider* oder *Seeker* klassifiziert. Verschiedenen Wissensorganisationen soll dabei ein Höchstmaß an Unabhängigkeit gewährt werden. Einzelne Peers werden zu Gruppen, so genannten K-Federations, zusammengefasst, die sich untereinander austauschen. Ein einzelner Knoten verfügt dabei über zwei Depots: Einerseits werden die lokalen Dokumente des Knotens bereitgestellt. Darüber hinaus wird eine kontextuelle Sicht auf die weiteren Knoten angeboten, die eine individuelle Sicht der Gruppe auf das gesamte Wissen aller Knoten bietet. Dokumente werden in dem KEx-System kategorisiert und bereitgestellt. Ein Mechanismus zur synchronen Kollaboration fehlt allerdings.

Das EDUTELLA-Projekt (Nejdl et al. 2002) befasst sich mit dem Austausch von Lehrmaterialien. Hier stehen Suchmechanismen im Vordergrund, da das Auffinden von Materialien in einem Peer-to-Peer-Netz schwierig sein kann. Dazu wird der RDF-Standard (Resource Description Framework) des

W3C[24] verwendet und JXTA (Gong 2001) Peer-to-Peer-Infrastruktur eingesetzt.

Eine kommerzielle Plattform stellt das Groove-System[25] dar. Es ist sowohl eine fertige Software für Windows-Systeme als auch eine Plattform zur Entwicklung von Peer-to-Peer-Systemen. Das System unterstützt Kommunikation, Austausch von Dokumenten und Kollaboration ohne Verwendung von Servern. Aufgrund des Einsatzes von Windows-Komponenten ist das System nicht auf anderen Betriebssystemen verwendbar.

Das Magi-System stellt ebenfalls eine Peer-to-Peer-Infrastruktur dar (Milojicic et al. 2002). Im Gegensatz zu Groove ist Magi plattformunabhängig[26] und verwendet Web-Standards: die einzelnen Magi-Peers kommunizieren über das HTTP-Protokoll. Dadurch besteht der Vorteil, auch über geschützte Netzwerken hinweg zu kommunizieren, da das HTTP-Protokoll von Firewalls in der Regel nicht blockiert wird.

Neuere Entwicklungen gehen dazu über Peer-to-Peer-Ansätze auch auf Massive Mulitplayer Online Games zu übertragen. Dabei muss insbesondere eine Manipulation der Daten von Seiten der Spiele verhindert werden. Wenn die Spielerdaten nicht an einer zentralen Stelle gespeichert sind, sondern über alle Knoten (auch die der Spieler) verteilt werden, ist eine Sicherheit dieser Daten nicht mehr ohne Weiteres gegeben. SimMud (Knutsson et al. 2004) stellt ein verteiltes Massive Multiplayer Online Game dar, das neben Peer-to-Peer-Ansätzen auch die bekannten Ideen existierender MMOG mit einbezieht. So erfolgt eine Aufteilung der virtuellen Welt in verschiedene Zonen unter Berücksichtigung der Area of Interest. Da die Zustände von Objekten aufgrund der Anforderungen an die Synchronizität der Ereignisse unmittelbar abgeglichen werden müssen, wird jedem Objekt ein Koordinator zugewiesen. Alle Aktualisierungen und Ereignisse werden über diesen Koordinator verteilt.

Ein anderes Forschungsfeld betrifft das Grid Computing.[27] Es handelt sich ebenfalls um Peer-to-Peer-Systeme bzw. um Systeme, die diesen sehr stark ähneln. Diese entstanden parallel zu den Tauschbörsen wie Napster und Gnutella und die Knoten beteiligen sich an einer gemeinsam zu lösenden Aufgabe. Ebenso wie in Peer-to-Peer-Systemen existieren beim Grid Computing wesentliche

---

[24] Der RDF-Standard wird auf den Seiten des World Wide Web Consortium beschrieben und ist verfügbar unter http://www.w3.org/RDF [Stand: 01.12.2005].

[25] Verfügbar unter http://www.groove.net [Stand: 01.12.2005].

[26] Das System ist in Java implementiert.

[27] Dieser Bereich wird auch als Peer-to-Peer-Computing bezeichnet.

Konzepte einer Verteilung von Ressourcen, sodass diese Systeme auf Verteilungsstrategien hin untersucht werden, um Analogien zu einer Verteilung von Räumen zu identifizieren.

2.3.5 Grid Computing

Die Bezeichnung Grid Computing ist in den 90er Jahren geprägt worden und bezeichnet eine Verteilung von Ressourcen über ein Netzwerk (Foster und Kesselmann 1999). Seitdem wird der Begriff in vielen unterschiedlichen Kontexten verwendet — im Zentrum steht allerdings immer das Konzept einer virtuellen Organisation. Teilnehmer eines Grids wollen entweder Ressourcen teilen oder zusammen ein Problem bewältigen. Ein Grid kann folgendermaßen charakterisiert werden (vgl. Casanova 2002):

- Bei Ressourcen kann es sich um Daten, Computer, Programme oder andere Dinge handeln.
- Die Verteilung der Ressourcen muss zentral koordiniert sein.
- Die Teilnahme in einem Grid ist dynamisch — es können also Teilnehmer hinzukommen oder das Grid wieder verlassen.

Es existieren bereits Ansätze, ein Grid nicht nur aus statischen Ressourcen zusammenzusetzen, sondern auch mobile Geräte mit einzubeziehen (Phan et al. 2002).

Als Abgrenzung zu Ansätzen wie Peer-to-Peer kann das Grid eher zur Lösung einer gemeinsamen Aufgabe gesehen werden. Besonders Berechnungen stehen bei existierenden Grid-Technologien im Vordergrund.

Für den Einsatz eines Grids im Bereich von CSCW-Systemen ergeben sich eine Reihe von Vorteilen (Bote-Lorenzo et al. 2003):

- Verteilte Administration unterschiedlicher Organisationen: Jeder Knoten des Grids kann selbstständig und unabhängig von dem gesamten Grid administriert werden.
- Zugriff auf unterschiedliche Materialien in einer verteilten Umgebung: Die Dokumente sind über das Grid verteilt und Benutzer können auf alle verfügbaren Materialien zugreifen, falls sie über entsprechende Berechtigungen verfügen.

Das Access-Grid-Projekt (Childers et al. 2000) realisiert virtuelle Räume als Treffpunkte für Kollaborationen. Der Fokus liegt im Bereich von Videokonferenzen, sodass sich Benutzer in realen Räumen treffen können und von dort Videoübertragungen erfolgen. Aufgrund der Komplexität der Kommunikation wird Multicast verwendet, um die Übertragungen an mehrere Teilnehmer gleichzeitig auszuliefern. Es erfolgt hier allerdings keine Verteilung von Räumen und Objekten.

In Caballé et al. (2004) wird eine Basisarchitektur für CSCL-Systeme beschrieben, der eine Grid-Infrastruktur zugrunde liegt. Die Collaborative Learning Purpose Library (CLPL) stellt verschiedene Komponenten für CSCL-Applikationen bereit. Dazu zählen Security Management, Administration Management und Knowledge Management. Ziel ist ein System, das flexibel an die Bedürfnisse angepasst werden kann. Daher besteht die Möglichkeit, verschiedenen CSCL-Komponenten einzusetzen und eine Austauschbarkeit zu gewährleisten. Für verteilte Wissensräume ist dieser Ansatz in Hinsicht auf die unterliegende Infrastruktur von Bedeutung.

Das Globus Toolkit (Foster 2005) stellt eine serviceorientierte Grid-Infrastruktur dar, die aus einer Sammlung von Programmen und Bibliotheken besteht, die Lösungen für Probleme verteilter Applikationen zur Verfügung stellen. Dabei werden hauptsächlich Web Services[28] zur Kommunikation verwendet. Ebenso wie die Distributed Hash Table könnte dieses Toolkit als Basis für verteilte Wissensräume dienen.

2.3.6 Zusammenfassung

Die verschiedenen vorgestellten Architekturen und Systeme beinhalten verschiedene Konzepte und Ideen. Aus den einzelnen Bereichen können bewährte Ideen verwendet werden, um diese auf den Wissensraum zu übertragen. Es stellt sich die Frage, inwieweit bestehende Lösungsansätze für eine Architektur verteilter Wissensräume verwendbar sind. Dabei spielen die verschiedenen Arten von Peer-to-Peer-Systemen eine Rolle, die grundsätzlich eine verteilte Infrastruktur bieten. Insbesondere die Distributed Hash Table und Grid-Infrastrukturen sind wichtige Entwicklungen, die ein erster Schritt in Richtung einer verteilten Ablage von Objekten sind. Bezogen auf bestehende Serversysteme muss hier allerdings bemerkt werden, dass sich eine Integration aufwändig

---

[28] Die Webservice-Aktivitäten und Spezifikationen des W3C sind verfügbar unter http://www.w3.org/2002/ws/ [Stand: 10.12.2005].

gestaltet, da die Systeme bisher auf klassische Infrastrukturen zurückgreifen.

Gleichzeitig sind existierende Peer-to-Peer-Systeme in der Regel stark auf einen gemeinsamen Zugriff von Dateien ausgerichtet und bieten keine allgemeinen Funktionen zur Manipulation von Objekten.[29] Es handelt sich darüber hinaus lediglich um einen gemeinsamen Datenraum ohne Ereignisse. Die Übertragung der grundlegenden Ideen dieser Systeme ist dennoch möglich und kann auf Ebene der Distributed Hash Tables (DHT) durchgeführt werden. Eine Verarbeitung von Ereignissen muss dabei aufbauend auf Systeme wie Pastry und Scribe erfolgen.

Im Gegensatz zu diesen verteilten Infrastrukturen handelt es sich bei den Massive Multiplayer Online Games um Plattformen, die bereits auf grafische 3D-Welten ausgelegt sind und in dieser Hinsicht viele Funktionen der Interaktion mit Objekten der virtuellen Welt bieten. Die Aufteilung dieser Welten in einzelne, voneinander unabhängige Zonen kann auch einen Ansatz für eine Verteilung von Wissensräumen bieten. Neben einer solchen geografischen Aufteilung erfolgt die Aufteilung im Neteffect-System anhand von Communities. Hier ergibt sich ein Bezug zu kollaborativen Systemen mit Gruppen- und Benutzerstrukturen, die möglicherweise auf ähnliche Weise verteilt werden können. Im Gegensatz zu einer willkürlichen Verteilung zu Verbesserung von verteilten Zugriffen liegt der Vorteil in der semantischen Aufteilung mit einer Autonomie der einzelnen Systeme. Damit wird neben einer Kontrolle über lokal gespeicherte Objekte auch eine garantierte Verfügbarkeit zugesichert.

Im folgenden Kapitel wird das Kernkonzept des virtuellen Wissensraums, mit den Strukturen von Räumen und Verbindungen in einem CSCW-System, vorgestellt. Unter Einbeziehung der vorgestellten Peer-to-Peer- und Multiserversysteme werden verschiedene Konzepte einer Verteilung von Wissensräumen erarbeitet.

---

[29] Die Protokolle stellen keine Mechanismen zur Verfügung, um Methoden in Objekten aufzurufen. Eugster und Baehni (2002) beschreiben, wie diese Mechanismen auch auf Peer-to-Peer-Systeme übertragen werden können.

3 Konzepte kooperativer Wissensräume

In diesem Kapitel wird das Konzept des Wissensraums erläutert und verschiedene Strategien einer Verteilung von Wissensräumen vorgeschlagen.

Die Idee eine virtuelle Welt zu schaffen, in der sich Benutzer bewegen und treffen können, ist nicht neu. Eine Umsetzung findet sich sowohl in Multi User Dungeons (MUD) als auch in dreidimensionalen Collaborative Virtual Environments (CVE) (vgl. Benford et al. 2001) oder auch in Massive Multiplayer Online Games (MMOG). Während die grafischen Umgebungen die Welt durch ein Raster mit Koordinaten beschreiben und die jeweilige Sicht eines Avatars durch Sichtweite und Blickrichtung gefiltert wird, sind textbasierte MUDs durch untereinander verbundene Räume ausgeprägt. In beiden Fällen bewegen sich Repräsentanten der Benutzer (grafische oder textuelle Avatare) durch die virtuelle Umgebung.

In der Science-Fiction-Literatur finden sich Ideen, die über diese existierenden Systeme weit hinausgehen: Der so genannte Cyperspace (Gibson 1984) bietet eine virtuelle Umgebung, in der sich Benutzer mit gleichen Wahrnehmungen wie in der realen Welt bewegen können. So stellt sich der virtuelle Raum sehr ähnlich der realen Welt dar und die Benutzer können auf ihre Vorerfahrungen zurückgreifen, um mit den Gegenständen in der virtuellen Umgebung zu interagieren.

Dies ist eine entscheidende Qualität virtueller Umgebungen, die zu einer Unterscheidung zwischen Raum (Space) und Ort (Place) geführt hat (Harrison und Dourish 1996). Als Ort ist in diesem Kontext die virtuelle Umgebung beschrieben, die einen realen Raum als Vorbild hat und sich dadurch die Erfahrungen der Benutzer zunutze macht. Eine direkte räumliche Anordnung ist allerdings nicht unbedingt nötig, um dem Benutzer die Interaktion mit dem System so einfach wie möglich zu gestalten. Räume in einem MUD genügen der Anforderung eines Ortes ebenfalls, obwohl es hier im Gegensatz zu einem realen Raum keine räumliche Nähe zwischen Objekten gibt. Diese Überlegungen lassen sich auch auf CSCW-Systeme übertragen, insofern diese eine virtuelle Umgebung darstellen. So stellt TeamRooms (Roseman und Greenberg 1996) ein CSCW-System auf der Basis von Räumen dar. Dabei wird der Raum ebenfalls als Ort beschrieben und dadurch von sitzungsorientierten Systemen abgegrenzt. Der Raum bleibt mitsamt des Inhalts auch über eine Kollaboration von Benutzern hinaus erhalten. Eine solche Form der Persistenz ist ausschlaggebend für eine kontinuierliche Kooperation der Benutzer.

Aktuelle Entwicklungen im Bereich raumbasierter Umgebungen sind sTeam (Hampel 1999) und CURE[1] (Haake et al. 2004b). Neben einer Raum-Metapher findet bei CURE eine Schlüssel-Metapher Verwendung, sodass jedem Raum Schlüssel zugeordnet werden können, die bestimmen, welche Benutzer über Berechtigungen innerhalb eines Raums verfügen (Haake et al. 2004a).

Die Idee verteilter Räume ist technisch geprägt und geht von zunächst unabhängigen Strukturen aus. Für ein Zusammenwachsen dieser verteilten Strukturen müssen Übergänge zwischen Räumen geschaffen werden, die sich an unterschiedlichen Orten befinden. Dies bedeutet, dass Wissensräume an verschiedenen Orten entstehen, ohne zu Beginn einen Bezug zueinander zu haben (vgl. Szenario 1 in Abschnitt 4.1).

Zum Verständnis der verschiedenen Konzepte wird zunächst der Begriff des Wissensraums eingeführt. Aus der Strukturierung von Räumen ergibt sich dabei eine Struktur von Räumen und enthaltenen Objekten. Der Raum selbst dient als Kontext für die darin befindlichen Objekte und als Treffpunkt für Benutzer.

Danach werden in Abschnitt 3.2 bis 3.4 die verschiedenen Konzepte eines Verbundes mit einem gemeinsamen Datenraum vorgestellt.

3.1 Wissensraum

Der Wissensraum (Hampel 2002) stellt ein Konzept für ein kollaboratives System dar und dient als Kontext und zum Arrangieren von Materialien. Andere bekannte Anwendungen einer Raum-Metapher finden sich in dem Rooms-System von Henderson und Card (Henderson und Card 1986)[2] und bei dem TeamRooms-System.

---

[1] CURE ist die Lernplattform der Fernuniversität Hagen und erreichbar unter http://teamwork.fernuni-hagen.de/CURE [Stand: 09.12.2005].

[2] Es handelt sich bei dem Rooms-System allerdings um keine virtuelle Umgebung, sondern um eine Übertragung der Raum-Metapher auf ein Desktop-System.

Im Gegensatz zu diesen Systemen basiert die Idee des Wissensraums wesentlich auf einer Kopplung von Hypertext-System mit einem Multi User Dungeon (MUD) (vgl. Bollmeyer 1997; Pfister et al. 1998). Die ersten virtuellen Räume im Internet sind in diesen MUDs (auch Multi User Domain) zu finden. Der Wissensraum leitet sich aus dem Raumkonzept eines Multi User Dungeons ab, wobei im Gegensatz zu einem MUD innerhalb von Räumen in der Regel Dokumente angeordnet werden. Im folgenden Abschnitt wird die Verwendung von Räumen in MUDs vorgestellt. Im Anschluss daran wird der Wissensraum mit zusätzlichen Konzepten beschrieben.

3.1.1 MUD

MUDs sind Online-Treffpunkte zum Spielen oder Kommunizieren, die sich einer räumlichen Metapher bedienen. Alle Aktivitäten finden in Räumen statt, die untereinander verbunden sind, sodass sich Benutzer von einem Raum in einen benachbarten bewegen können. Dazu müssen textuell Kommandos eingegeben werden, um die Repräsentation der eigenen Person, den Avatar, zu steuern. Innerhalb von Räumen befinden sich Gegenstände, die die Spieler aufnehmen oder durchsuchen können. Alle diese Aktivitäten werden innerhalb von Räumen synchronisiert und für andere Benutzer in einem Raum wahrnehmbar gemacht.

Die ersten MUDs sind Spiele, bei denen es um eine Verbesserung der Eigenschaften des eigenen Avatars geht. Jeder Spieler hat verschiedene Werte, die bestimmen, wie gut dieser in Kämpfen gegen computergesteuerte Nicht-Spieler-Figuren (NPC: None Player Character) oder auch andere Spieler, abschneidet. Bei solchen Kämpfen sammeln die Spieler Erfahrungspunkte, die wieder zu einer Verbesserung der Eigenschaften eingesetzt werden können. In diesen Spielen findet nur teilweise Interaktion zwischen Spielern statt – zum Teil bewegen sich die Spieler unabhängig durch die Spielwelt, um z.B. fest einprogrammierte Rätsel zu lösen, und interagieren so lediglich mit der vorgegebenen Umgebung.

Gleichzeitig sind MUDs als soziale Treffpunkte entdeckt worden, bei denen die Kommunikation zwischen den Benutzern im Vordergrund steht. Vertreter dieses Typs sind besonders TinyMUD und später als technisch verbesserte Version das MOO (Mud Object Oriented), bei dem auch die Möglichkeit der Erweiterung der virtuellen Welt zur Laufzeit des Systems besteht.

Neben technischen Details eines MUDs stellt sich besonders die Frage, wie das Zusammenleben der Benutzer funktionieren kann. Dazu existieren eine Vielzahl von Veröffentlichungen, die sich mit sozialen Aspekten befassen, und der Gemeinschaften, die sich in solchen Systemen bilden. Der Begriff der virtuellen Community ist von Rheingold geprägt worden (Rheingold 1993) und beschreibt solche Gemeinschaften. Untersuchungen ergaben allerdings, dass diese nur dann auf Dauer Bestand hat, wenn sie durch reale Treffen weitergeführt wird.

Nachdem die Konzepte eines MUDs vorgestellt worden sind, soll auch die technische Umsetzung solcher Systeme Erwähnung finden. Es handelt sich um Server, die Netzwerkverbindungen verschiedener Benutzer akzeptieren und eine Datenbank von Objekten bereitstellen, die die virtuelle Welt beschreiben (Curtis und Nichols 1994). MUDs sind die ersten Systeme, die es Benutzern erlauben synchron auf Objekte[3] im Internet zuzugreifen und diese zu manipulieren. Churchill und Bly (1999) beschreiben MUDs als Mehrbenutzersysteme, die erweiterbare, eine geringe Bandbreite benutzende und verteilte virtuelle Umgebungen darstellen.

Die Ereignisverarbeitung ist synchron und Benutzer können direkt miteinander kommunizieren und interagieren. Der Befehlssatz umfasst neben Kommunikations- und Bewegungskommandos auch Emotionen (lächeln, grinsen, usw.) und Systembefehle. Im Gegensatz zu IRC werden alle Eingaben direkt als Befehl interpretiert und ausgeführt.[4]

Eine weitere wichtige Eigenschaften bestimmter MUD-Systeme ist die Erweiterbarkeit zur Laufzeit (Bruckman 1994). – Benutzer können selbstständig aus dem MUD heraus Erweiterungen vornehmen und neue Räume und Objekte hinzufügen. Der Grad dieser Einflussnahme hängt von dem Typ des MUDs ab (vgl. Bopp 2000, S. 8ff.).

Nachdem die ursprüngliche Idee eines Raums beschrieben worden ist, wird im nächsten Abschnitt auf die Verwendung von Räumen im Rahmen eines kollaborativen Systems eingegangen.

3.1.2 Raum und Wissensraum

Das Konzept von Räumen ist in Multi User Dungeons bzw. Textabenteuern (Textadventures) eingeführt worden, um den Aufenthaltsort eines Spielers tex-

---

[3] Objekte sind Gegenstände in einer virtuellen Umgebung.

[4] Im IRC sind alle Eingaben zunächst Kommunikationsbeiträge im aktuellen Kanal. Befehle sind durch einen „/“ am Anfang einer Zeile gekennzeichnet.

tuell zu beschreiben. Benachbarte Räume sind durch Himmelsrichtungen verbunden, die es einem Benutzer ermöglichen durch Kommandos wie „Gehe nach Westen“ (typischerweise durch Eingaben wie „west“ abzukürzen), von einem Raum in einen anderen zu gelangen. Solche Verbindungen sind als so genannte *Exits* bekannt, da sie ein Verlassen des Raums ermöglichen. Dieser Begriff lässt auf Räume mit umgebenden Wänden schließen, allerdings sind MUDs später von den Dungeons (Höhlen) auch auf Außenräume erweitert worden.

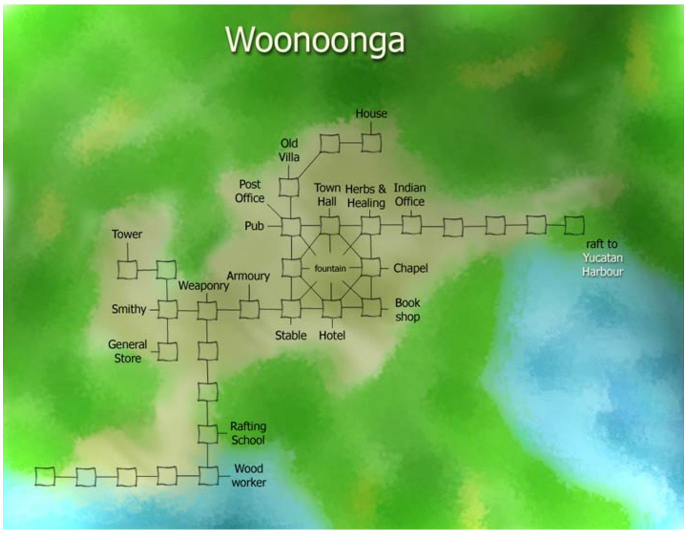


Abbildung 3.1: *Karte eines MUDs (Quelle: http://www.xyllomer.de)*

Abbildung 3.1 zeigt eine Übersichtskarte einer Region eines MUDs. Es handelt sich dabei um die Stadt Woonoonga, die Teil der Spielwelt Xyllomer[5] ist. Die einzelnen Räume der dargestellten Stadt sind gemäß der Himmelsrichtungen untereinander verbunden, sodass eine Orientierung in der Umgebung leicht möglich ist.

Der Raum ist die zentrale Komponente eines MUDs und erlaubt dem Benutzer mit anderen Objekten, die sich im selben Raum befinden, zu interagieren. Als Kernpunkt dieses Konzepts ist der Raum als Treffpunkt für die Benutzer charakterisierbar. Abhängig von der Beschreibung des Raums existieren Räume zu unterschiedlichen Zwecken. So ist ein Raum mit verschiedenen Gegenständen und Rätseln eher zur Interaktion mit dem System gedacht. Andere Räume, die schon von der Beschreibung her einen eher gemütlichen Charakter

---

[5] Die Homepage von Xyllomer ist verfügbar unter: http://www.xyllomer.de [Stand: 21.12.2005].

aufweisen, dienen als soziale Treffpunkte.

In grafischen Umgebungen (CVE) existiert das Raumkonzept in der Regel nicht. Stattdessen wird der Aufenthaltsort durch eine Koordinate festgelegt. Andere Objekte und Benutzer befinden sich an anderen Punkten und durch Berechnungen des sichtbaren Bereichs wird entschieden, welche Gegenstände und Personen einem Benutzer angezeigt werden. So sind die Übergänge dort vollkommen fließend. Eine Ausnahme stellt das Croquet-System dar, das unabhängige Bereiche von Benutzern mit Portalen verbindet. Dabei wird eine Space Metapher verwendet, die Räume, aber auch Gebiete (Landschaften) einschließt (vgl. Smith et al. 2003).

Viele klassische CSCW- und CSCL-Systeme stellen Ablagen für Dokumente als Ordner bereit, wie sie in Betriebssystemen wie Unix und Windows zu finden sind. Synchrone Werkzeuge arbeiten neben den Dokumentenablagen in einer separaten Sitzung ab.

Das TeamRooms-System stellt eine Groupware dar, die auf einem Raumkonzept in ähnlicher Art eines MUDs aufbaut. Im Gegensatz zu sitzungsorientierten Systemen wird der Raum in Anlehnung an physikalische Räume beschrieben. Es gilt dabei den Raum ähnlich eines realen Besprechungsraums zu modellieren. Die folgenden Kriterien an einen Raum ergeben sich daraus (Roseman und Greenberg 1996):

- Räume haben eine lange Lebensdauer.
- Räume sind mitsamt ihrem Inhalt persistent.
- Es besteht die Möglichkeit mit anderen Benutzern zu kommunizieren.
- Jeder Raum verfügt über Kollaborationswerkzeuge.
- Räume können um zusätzliche Werkzeuge erweitert werden.

Der Wissensraum basiert ebenfalls auf diesen grundlegenden Überlegungen und erweitert diese durch zusätzliche Konzepte.

Virtuelle Wissensräume (vgl. Hampel 2002) sind semantisch verknüpfte virtuelle Räume, in denen Nutzer sich aufhalten und kommunizieren können. Räume enthalten Materialien und Dokumente und können dabei auf verschiedene Weise gemeinsam genutzt und strukturiert werden. Wesentlich für die Idee kooperativer Wissensräume ist hierbei, dass Werkzeuge und Materialien nicht isoliert nebeneinander stehen, sondern ein konsequent objektorientierter und integrativer Ansatz verfolgt wird. Damit ist der Wissensraum konzeptuell als eine Sicht auf zusammengeführte unterschiedliche Protokolle und Standards zu sehen. Darüber hinaus steht eine kontinuierliche Kooperation der Benutzer im Vordergrund, die durch eine Persistenz der Daten und die Verwendung von Sichten geprägt ist. Im Gegensatz zu einem sitzungsbasierten Ansatz werden alle Aktivitäten, unabhängig von dem Werkzeug/der Sicht, auf den gleichen Wissensraum bezogen.

Eine Implementierung des Konzepts virtueller Wissensraum findet sich in der Open-Source Lehr- und Arbeitsumgebung sTeam. sTeam verfolgt entsprechend nicht das Ziel einer monolithischen, geschlossenen Umgebung, sondern integriert verschiedene Dienste und Möglichkeiten in das Konzept des virtuellen Wissensraums. Dabei legt sich die Art eines Raums nicht über die vorhandenen Dienste fest, sondern durch den Inhalt des Raums. Im Prinzip stehen also sämtliche Funktionen zu jedem Zeitpunkt zur Verfügung und die jeweilige Sicht legt die Verwendung dieser Dienste fest.

Im Gegensatz zu einem Ablagebereich für Dokumente stehen also semantische Verknüpfungen im Vordergrund. Ein Raum beschreibt darüber hinaus den Aufenthaltsort eines Benutzers und vereint alle Technologien und Protokolle in sich. Zu nennen sind hier die Integration von E-Mail (Schmidt et al. 2004), Annotationen (Hampel und Bopp 2001), Chat-Protokolle wie IRC und Jabber, Wiki-Funktionalität (Bopp et al. 2005b), RSS-Technologie (Bopp und Hampel 2005a) und allgemein Services.

Das Ziel dieser Arbeit ist es, den Benutzern einen gemeinsamen Handlungsraum zu bieten. Dieser zeichnet sich durch Aktivitäten und kooperatives Arbeiten der Benutzer aus und verfügt daher über eine einheitliche Ereignisverarbeitung ohne Grenzen zwischen den Servern. Dadurch werden alle Aktivitäten überall übertragen und können verarbeitet werden. In dieser Hinsicht unterscheidet sich der Handlungsraum von einem gemeinsamen Datenraum.

Es existiert eine enge Verknüpfung zwischen Räumen, Benutzern und Gruppen. Diese ist durch verschiedene Arbeitsräume charakterisiert, die die Wurzeln einer hierarchischen Raumstruktur bilden. Dabei muss zwischen zwei unterschiedlichen Arten von Arbeitsräumen unterschieden werden:[6]

- **Arbeitsraum einer Gruppe**: Jeder Gruppe ist ein eigener Arbeits-

---

[6] Es handelt sich dabei um eine grundsätzliche Unterscheidung. Darüber hinaus sind verschiedene Arten von Räumen in Abhängigkeit von den Inhalten und verfügbaren Sichten identifizierbar. Dazu zählen Vorlesungsraum, Übungsraum, Galerie und Lerngruppenraum (vgl. Hampel 2002).

raum zugeordnet, der allen Mitgliedern der Gruppe eine Umgebung zur Kooperation bietet. Es handelt sich um offene Bereiche, sodass alle Dokumente dort zugänglich sind.

- **Privater Arbeitsraum**: Im Gegensatz zu diesen offenen Bereichen sind die persönlichen Arbeitsräume nur für den Benutzer selbst einsehbar (vgl. auch Pfister et al. 1998). Sie dienen zur Ablage von privaten Dokumenten oder auch zur Vorbereitung von Strukturen, die erst später in einen Gruppenbereich eingefügt werden sollen.

Zur weiteren Strukturierung von Räumen können Container dienen, die ebenso wie Räume Objekte beinhalten. So können in die Container Objekte eingefügt und wieder entnommen werden. Der Container unterscheidet sich lediglich durch die Metapher eines Behälters von einem Verzeichnis in einem Dateisystem. In einem Wissensraum dienen sie analog zu der Verwendung in einem MUD als Hilfsmittel zur Strukturierung von Räumen. Damit kann der Raum auch mit vielen Objekten noch übersichtlich gehalten werden und Objekte, die semantisch dem Raum zugeordnet sind, werden in einen Container bewegt.

Eine solche Ablage von Objekten in Container-Strukturen unterhalb des Raums macht in vielen Situationen Sinn. Dies wird deutlich, wenn z.B. HTML-Dokumente benutzt werden, die neben dem eigentlichen Hauptdokument aus verschiedenen weiteren Dokumenten wie eingebetteten Bildern und Stylesheets bestehen können.

Des Weiteren ist es möglich einen weiteren Raum in einen Raum zu bewegen. Dadurch entsteht eine Struktur von Räumen, die eine zentrale Bedeutung für eine Verteilung von Wissensräumen spielen kann, da sich dadurch komplexere Strukturen unterhalb eines einzelnen Wurzelraums bilden können.

3.1.3 Raumstruktur, hierarchische Wissensräume

Die Verbindungen zwischen Räumen sind ein weiteres Merkmal von Wissensräumen. Sie dienen dazu, semantisch benachbarte Räume miteinander zu verknüpfen und die Übergänge für Benutzer wahrnehmbar zu machen. Damit kann ein Benutzer durch die Verwendung eines Übergangs von einem Raum in einen anderen wechseln.

Eine weitere Möglichkeit der Strukturierung stellt die *Ist-enthalten*-Beziehung von Räumen dar. Diese ist dadurch gekennzeichnet, dass ein Raum aus verschiedenen Objekten besteht. In der Regel handelt es sich um Dokumente, Container und Benutzer. Darunter könnten sich aber auch weitere Räume befinden, die eine weitere Struktur unterhalb des Raums darstellen. Die Bedeutung dieser Räume kann analog zu den Verbindungen zwischen Räumen als thematisch *Ist-enthalten* beschrieben werden.

Aufgrund der verschiedenen Verbindungen ist ein Raum innerhalb eines Servers nicht als isoliert anzusehen, sondern es ist die gesamte Struktur von Wissensräumen zu betrachten. Mit den Räumen und Verbindungen prägt sich ein Graph aus (vgl. Haake et al. 2004b). Dabei gibt es sowohl Verbindungen, die nur in eine Richtung gehen, als auch bidirektionale Verbindungen zwischen Räumen.

Unter Einbeziehung der *Ist-enthalten* Relation ergibt sich daraus ein Graph, der aus einzelnen Bäumen besteht. Jeder der Bäume ist durch das *Ist-enhalten* ausgeprägt mit Wurzeln, die sich dadurch auszeichnen, dass sie keine Umgebung besitzen. Durch die Hinzunahme von Querverbindungen[7] werden unterschiedliche Bäume verbunden und es ergibt sich ein Graph (vgl. Abbildung 3.2).

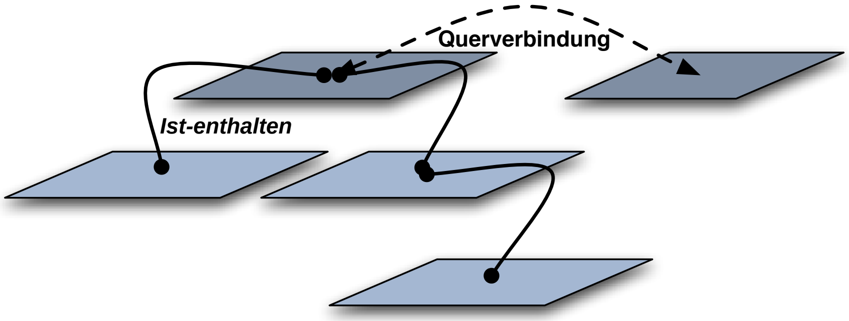


Abbildung 3.2: *Raumstruktur*

Bei den Räumen ohne Umgebung handelt es sich um die oben beschriebenen Arbeitsräume von Gruppen oder Benutzern. Ausgehend von diesen Strukturen erzeugen die Benutzer neue Räume, Container und Verbindungen, die dann jeweils in die entsprechenden Arbeitsräume oder die darunter liegenden Räume eingefügt werden.

---

[7] Die Verbindungen sind mit den *Exits* eines MUDs vergleichbar.

3.1.4 Raum als Kontext

Wie bereits beschrieben, kann der Wissensraum den zentralen Punkt kooperativen Arbeitens angesehen werden. Jeder Gruppe und jedem Benutzer sind Räume zugeordnet, die weiter strukturiert werden können. Dies drückt sich durch die Positionierung und Verbindung von Dokumenten, Containern und anderen Objekten der Umgebung aus und durch die Struktur von Räumen innerhalb des Wissensraums sowie Querverbindungen zu anderen Räumen.

Es muss zudem möglich sein, zu jedem Wissensraum eine oder mehrere Personen mit administrativer Funktion zuzuordnen. Bei einem Gruppenarbeitsraum besteht dieser Kreis aus den Administratoren der Gruppe und bei einem persönlichen Arbeitsraum handelt es sich um den Besitzer des Raums.

Die Raumadministratoren stellen die Gegebenheiten der Arbeitsumgebung nach ihren Bedürfnissen ein. So werden bestimmte Gruppen zur Mitarbeit berechtigt oder bestimmte Personen der lesende Zugriff erlaubt. Eine Einstellung dieser Berechtigungen für jedes einzelne Objekt der Umgebung gestaltet sich offensichtlich zu aufwändig. Das CURE-System realisiert eine Zugriffskontrolle daher durch die Vergabe von Schlüsseln, die jeweils einem bestimmten Raum zugeordnet sind (Haake et al. 2004a). Die Objekte innerhalb des Raums sind dabei vollständig der Umgebung unterworfen und können keine eigenen Berechtigungen vergeben. Ein Schlüssel kann weiterhin verschiedene Rechte innerhalb des Raums umfassen.

Das sTeam System erreicht eine Vereinfachung durch die Verwendung von Gruppenarbeitsräumen. Dazu werden bestimmte Rechte direkt dem Raum zugeordnet und auf die Objekte, die sich in diesem Raum befinden, vererbt. Dadurch bildet der Raum einen Kontext für die enthaltenen Objekte. In dieser Hinsicht ähneln sich die gewählten Ansätze (vgl. auch Pfister et al. 1998). Es ist dann allerdings weiterhin möglich zusätzliche Berechtigungen an einzelne Objekte zu binden, während die Berechtigungen des Arbeitsraums weiterhin Gültigkeit besitzen. Eine Erweiterung dieses Konzepts auf alle Räume des Systems führt zu einer mehrstufigen Vererbung von Rechten.

Noch deutlicher wird dieses Konzept wenn die Verschiebung eines Objekts von einer Umgebung in eine andere betrachtet wird. Ein Benutzer trägt z.B. ein eigenes Dokument von seinem persönlichen Arbeitsraum in den gemeinsamen Gruppenraum. Während im lokalen Bereich noch die dort geltenden privaten Rechte gültig waren, die in der Regel nur dem Benutzer selbst Erlaubnis zur Manipulation eines Objekts geben, wird nach der Verschiebung die gesamte

Gruppe zur kooperativen Arbeit an dem Objekt berechtigt (vgl. Hampel 2002).

Der Wissensraum stellt auf ähnliche Art einen Kontext im Bereich von Ereignissen und Benachrichtigungen dar. So ist die Ereignisverarbeitung nicht auf ein Objekt lokal beschränkt[8], sondern leitet Ereignisse auch an den jeweiligen Kontext (der Umgebung) weiter. Ein Benutzer kann also auf einfache Weise über alle Ereignisse innerhalb eines Raums benachrichtigt werden.

3.1.5 Raum als Treffpunkt

Ein einzelner Raum wird innerhalb der virtuellen Welt zu einem Treffpunkt für Benutzer. Die Benutzer bewegen sich durch die Verbindungen von einem Raum in einen anderen. An den verschiedenen Orten können sich die verschiedenen Benutzer treffen und dort interagieren. Die Formen einer solchen Interaktion fallen dabei ganz unterschiedlich aus und beinhalten Chat und den Austausch von Dokumenten.

Über den Raum werden synchrone und asynchrone Nachrichten zwischen den einzelnen beteiligten Personen ausgetauscht. Diese können in einem Raum hinterlegt oder an andere Objekte gebunden werden. Generell findet eine Interaktion mit den Gegenständen des Raums und den anderen Benutzern statt.

3.1.6 Rucksack

Das Konzept eines Rucksacks leitet sich analog zum Wissensraum aus virtuellen Umgebungen ab. Ein Benutzer kann Objekte aufnehmen und mit sich tragen. Damit dient dieser virtuelle Rucksack als Ablage für die Objekte eines Avatars, der die Objekte im Rucksack ständig mit sich führt. Gleichzeitig werden Objekte von einem Raum übertragbar in einen anderen Bereich, wo die Objekte fallen gelassen werden können. Außerdem sind diese Objekte durch den Rucksack mit dem Benutzer verknüpft.

3.1.7 Objektmodell

Eine virtuelle Umgebung lässt sich durch Objekte abbilden. Dabei handelt es sich um Gegenstände, die sich in der Umgebung befinden. Das Modell dieser Objekte bildet die Grundlage für ein objektorientiertes Design und fördert das

---

[8] Eine lokale Liste zu benachrichtigender Objekte ist in einer objektorientierten Umgebung üblich.

Verständnis der daraus entstehenden Strukturen. Außerdem wird das Zusammenspiel aller Objekte innerhalb eines Raums durch die Abhängigkeiten der Klassen deutlich. Der Wissensraum bildet in diesem Modell die Kernkomponente zur Strukturierung von Objekten. Das CURE-System der Fernuniversität Hagen verwendet ebenfalls ein objektorientiertes Design mit dem Raum als zentralem Element (Haake et al. 2004b).

Das Objektmodell des Wissensraums basiert als objektorientiertes Modell auf der Klasse *Objekt*, die eine einheitliche Schnittstelle zur Manipulation[9] von allen Objekten der virtuellen Umgebung bietet.

Das vorliegende Modell stellt die Basis einer raumbasierten Architektur dar. Dabei sind als wesentliche Klassen *Dokument*, *Container* und *Benutzer* anzusehen. Ein Raum ist dann ein *Container*, der auch Benutzer enthalten darf. Ein *Benutzer* ist als Modellierung des Rucksackkonzepts (vgl. Abschnitt 3.1.6) ebenfalls von der Klasse *Container* abgeleitet.

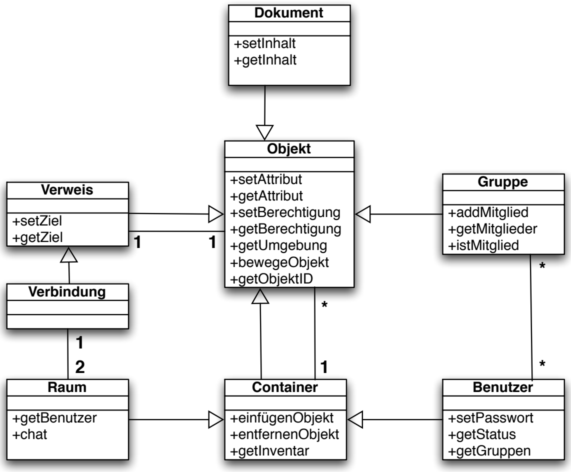


Abbildung 3.3: *Objektmodell*

Abbildung 3.3 stellt ein Objektmodell für ein raumbasiertes CSCW-System dar. Dieses ist durch die Raum- und Verbindungsklasse charakterisiert, die einen Übergang zwischen zwei Räumen ermöglicht. Obwohl zwischen Gruppen, Benutzern und Räumen keine Verbindungen dargestellt sind, schafft die Verwendung von Arbeitsräumen allerdings eine enge Verknüpfung zwischen diesen Klassen.

---

[9] Dazu zählt eine Vergabe von Berechtigungen, das Anfügen von Kommentaren, die Modifikation von Attributen und damit verbunden eine Auslösung von Ereignissen.

Die anderen Objektklassen stellen eine typische Objektmenge zur Abbildung der Anforderungen an ein kollaboratives System dar. Dabei gibt es Verweise, die auf beliebige Objekte zeigen können, Container verfügen ebenso wie Räume über einen Inhalt von Objekten und Dokumente sind Objekte mit Daten. Die wesentlichen Elemente einer Architektur mit Wissensraum sind neben dem Raum die Verbindungen, Benutzer und Gruppen.

Nachdem der Raum als Kernkonzept einer objektorientierten Architektur vorgestellt worden ist, werden nun verschiedene Konzepte einer Verteilung von Wissensräumen vorgestellt.

3.2 Gemeinsamer Datenraum

Die direkte Verbindung verschiedener zunächst unabhängiger Bereiche kann durch einen gemeinsamen Datenraum erfolgen (vgl. Bopp und Hampel 2004). Dieser stellt lediglich eine gewisse Schnittmenge dar und dient zum Datenaustausch unabhängiger Systeme. Ein typisches Beispiel für einen solchen Austausch stellen die Benutzerdaten dar, die dadurch nicht nur isoliert für ein System Gültigkeit besitzen, sondern von verschiedenen Seiten gemeinsam genutzt werden können. Damit ist eine einheitliche Authentifizierung innerhalb eines Verbundes möglich.

Zur Realisierung eines transparenten Wechsels von einem Raum in einen anderen bilden die Benutzerdaten die *minimale Datenmenge* des gemeinsamen Datenraums. Sie bilden die Grundlage einer Identifikation eines Benutzers an einem anderen Ort und sind gleichzeitig zur gegenseitigen Wahrnehmung von Benutzern innerhalb eines Raums notwendig.

Es kann bei der Speicherung der Benutzerdaten in einen gemeinsamen Bereich zwischen zwei Verfahren unterschieden werden:

- Partielle Speicherung: Nur die unbedingt notwendigen Daten werden gespeichert. Dabei handelt es sich in der Regel um die Benutzerkennung und ein Passwort. Mit diesen Daten kann eine Identifizierung erfolgen. Die partielle Speicherung muss gewählt werden, falls z.B. aus technischen Gründen nur ein Teil der Attribute in dem gemeinsamen Datenraum gespeichert werden kann und weitere Daten zum korrekten Betrieb der CSCW-Umgebung erforderlich sind.

- Vollständige Speicherung: Alle Benutzerdaten werden gespeichert. Dazu zählt besonders die Mitgliedschaft in Gruppen. Die Verwendung dieser

> Methode setzt voraus, dass im Datenraum beliebige Daten gespeichert werden können. Dies wird z.B. bei einem LDAP-Verzeichnis durch selbstdefinierte Schemata ermöglicht.

Die Verwendung eines gemeinsamen Datenraums zur Speicherung der Benutzerdaten stellt kein neues Konzept dar. Viele unterschiedliche Systeme bieten z.B. durch LDAP (Yeong et al. 1995) einheitliche Schnittstellen für den Zugriff auf Benutzerdaten. Es handelt sich dabei um zentrale Ablagen für die Daten, die von allen Systemen gleichzeitig benutzt werden. Teilweise wird auf die Daten nur lesend zugegriffen, während die Administration von anderer Stelle aus erfolgt.

Die Benutzer- und Gruppenobjekte bilden zudem lediglich einen kleinen Teil der Menge der Objekte eines kollaborativen Systems. Alle anderen Objekte sind jedoch in dem beschriebenen Datenraum nicht gespeichert. Es kommt also zu keiner vollständigen Überschneidung. Daraus ergeben sich in jedem Fall starke Einschränkungen bei dem Wechsel in einen anderen Bereich. Während z.B. die Benutzerkennung weiterhin Gültigkeit besitzt, ist es nicht möglich Dokumente zwischen den Servern zu übertragen. Der eigene private Arbeitsbereich und die Arbeitsbereiche der Gruppen sind jeweils nur lokal verfügbar.

Der gemeinsame Datenraum stellt daher nur ein sehr grundlegendes Konzept zur Koppelung von kollaborativen Systemen dar. Er bietet lediglich den Vorteil, dass sich eine Benutzerverwaltung auch aus bestehenden Systemen mehr oder weniger leicht ausgliedern lässt. Daher stellt das Konzept eines gemeinsamen Datenraums auch eine Lösung für existierende CSCW-Systeme dar und erfordert keine vollständig neue Implementierung. Der Aufwand richtet sich nach dem Aufbau eines Systems. Monolithische Systeme sind dabei schwieriger zu erweitern als komponentenbasierte Software.

Des Weiteren setzt der oben beschriebene Datenraum einen statischen Verbund von Servern voraus, da sich die Daten an einer zentralen Stelle befinden. Diese müssen von überall zugänglich sein, sodass ein entsprechendes Vertrauensverhältnis im Verbund bestehen muss.

Im folgenden Abschnitt wird das Konzept des gemeinsamen Datenraums auf einen Handlungsraum ausgedehnt.

3.3 Statischer Verbund mit Handlungsraum

Ein statischer Verbund besteht aus einer festen Menge von Servern. Das Hinzufügen neuer Bereiche erfolgt niemals automatisch, sondern wird manuell von den Administratoren des Verbundes konfiguriert. Ebenso verlassen Server den Verbund nur bei einer manuellen Abspaltung (vgl. Bopp et al. 2004).

Aufgrund der statischen Ausrichtung besteht zwischen den einzelnen Servern ein Vertrauensverhältnis, sodass jeder Server auf den gesamten Datenraum zugreifen kann. Im Gegensatz zu einem gemeinsamen Datenraum, der lediglich eine kleine Schnittmenge darstellt, kann in einem statischen Verbund mit Handlungsraum jeder Server auf beliebige Daten eines anderen Servers zugreifen. Der Handlungsraum charakterisiert sich insbesondere durch eine Ereignisverarbeitung und bildet dadurch die Grundlage für einen transparenten Zugriff auf Objekte eines entfernten Servers. Aus einem Datenraum der nur eine partielle Schnittmenge der Daten beinhaltet, folgen Einschränkungen in der Transparenz für die Benutzer. Weiterhin bildet der Handlungsraum die Grundlage für eine Bereitstellung von kooperativen Medienfunktionen.[10]

Ein Handlungsraum entsteht in einem statischen Verbund durch eine generelle Verfügbarkeit der Daten. Dazu existieren zwei Konzepte:

- **Zentrale Speicherung**: Die Daten befinden sich für alle Server zugänglich an einem zentralen Ort. Da es sich um einen gemeinsamen Handlungsraum handelt, befinden sich im Gegensatz zu einem partiellen Datenraum sämtliche Daten an dieser Stelle.
- **Dezentrale Speicherung**: Jeder Server verfügt über einen eigenen Bestand an Daten. Diese werden z.B. durch Kanäle zwischen den Servern ausgetauscht.

Der direkte Austausch von Daten zwischen den Servern erfordert die Identifikation von Benutzern, um die Berechtigungen zu überprüfen. Dazu muss entweder der Benutzer direkt gegenüber dem entfernten Server authentifiziert werden oder es besteht ein unmittelbares Vertrauensverhältnis zwischen den Servern, sodass die Benutzer lokal identifiziert werden können und kein Zweifel an den übermittelten Kennungen besteht.

Die beiden obigen Verfahren der Speicherung eines Handlungsraums haben verschiedene Vor- und Nachteile. Bei einer dezentralen Ausrichtung ist die

---

[10] Das theoretische Konzept kooperativer Medienfunktionen wird von Hampel (2002) detailliert dargestellt.

Autonomie der einzelnen Server stärker gegeben. So könnte ein Server den Verbund sogar verlassen und mit dem eigenen Datenbestand weiterarbeiten. Auch bei einer Zusammenführung von verschiedenen Servern zu einem Verbund ist die dezentrale Speicherung zu bevorzugen. Eine zentrale Speicherung würde bedeuten, dass sämtliche Daten eines Servers an den zentralen Speicherort überführt werden müssten.

Erweitert man diese generellen Überlegungen auf das Konzept von Wissensräumen und Objekten, so speichert jeder Server einen eigenen Bestand an Räumen. Jeder dieser Räume enthält Gegenstände (Objekte) und kann sich in weitere Strukturen aufteilen (vgl. Abschnitt 3.1.3).

Die Aufteilung der verschiedenen Räume über einen festen Verbund erinnert an grafische MMOG, die die virtuelle Welt zwischen verschiedenen Server aufteilen (Jakobsson und Taylor 2003). Eine offensichtliche Lösung stellt dabei eine geografische Aufteilung nach Kontinenten dar. Zur Vermeidung zusätzlicher Kommunikation sollten diese Zonen so wenig wie möglich miteinander verbunden sein.

Bei der Entstehung eines Verbundes aus mehreren, vorher autarken Servern verändert sich die Sicht eines Benutzers zunächst nicht: Der Zugriff auf die Daten verläuft über den gewohnten Zugang. Ein veränderter Datenbestand wird erst durch eine serverübergreifende Suche oder Verbindungen zwischen Räumen unterschiedlicher Server deutlich. Der gemeinsame Datenraum ermöglicht den Benutzern außerdem eine direkte Authentisierung gegenüber beliebigen Servern des Verbundes. In diesem Kontext sind auch Web-Lösungen zu sehen, die die Anfragen auf verschiedene Server verteilen (Cardellini et al. 1999). Dabei kann die Kommunikation insgesamt durch einen so genannten Gateway geregelt werden, der den zentralen Zugang bietet.

Diese Idee kann auf kollaborative Systeme erweitert werden. Ein zentraler Server bietet einen Zugang zu einem dahinter liegenden statischen Serververbund und nimmt die Rolle eines Vermittlers ein. Die Kommunikation wird vollständig durch den Gateway gekapselt[11]. Es muss dabei zwischen zwei Lösungen unterschieden werden:

- **Replizierte Server**: Der Gateway leitet eintreffende Anfragen an einen Server weiter, der die geringste Auslastung besitzt. Die Auswahl des Servers ist dabei beliebig, da der Bestand an Objekten jeweils identisch

---

[11] Eine ähnliche Lösung findet sich auch bei Neteffect durch eine Aufteilung von verschiedenen Communities über mehrere Server (Das et al. 1997).

ist. Aufgrund der dynamischen Ausrichtung eines kollaborativen Systems muss der Datenbestand ständig zwischen den Servern aktualisiert werden. Insgesamt werden die Daten damit also zentral gehalten.

- **Verbund von Servern**: Es handelt sich um verschiedene Server, die sich zu einem Verbund zusammenschliessen. Daher verfügt jeder einzelne Server über Objekte, Benutzer und Gruppen, die nur diesem Server zugeordnet sind (dezentrale Speicherung). Der Gateway regelt die Zugriffe, kennt zu jedem Objekt den entsprechenden Server und dient als Vermittler.

Offensichtlich muss ein CSCW-Gateway viele technische Probleme lösen. Das Resultat einer solchen Architektur ist ein Gateway, der von außen wie ein einzelnes CSCW-System erscheint. Der Verbund, der sich so bildet, ist statisch und die dazugehörigen Server müssen dem Gateway einzeln hinzugefügt werden.

Neben der Ausrichtung des Verbundes mit zentraler oder Server-Server-Kommunikation ist für ein kollaboratives System auf der Basis von Räumen die Verteilung der Objekte von Bedeutung. Der Raum bildet für die darin befindlichen Objekte einen Kontext und es besteht die Möglichkeit ein Objekt von einem Raum in einen anderen zu verschieben. Daraus ergeben sich Fragestellungen der Verteilung von Objekten im Serververbund.

3.3.1 Verteilungsstrategien von Wissensräumen

Die Art der Verteilung von Wissensräumen über verschiedene Server stellt ein wichtiges Konzept dar. Dabei gilt es Anforderungen an die Kommunikation zwischen Servern, Verfügbarkeit von Objekten und Aspekte der Datensicherheit zu berücksichtigen:

- **Kommunikation**: Lokale Objekte lassen sich schneller abfragen, da lokale Zugriffe wesentlich schneller durchführbar sind als eine Verwendung von entfernten Objekten.

- **Verfügbarkeit**: Objekte stehen generell für alle Server zur Verfügung und im Idealfall besteht auch die Möglichkeit auf Objekte zuzugreifen, wenn ein Server des Verbundes nicht erreichbar ist.

- **Datensicherung**: Es sollte möglich sein, alle voneinander abhängigen Objekte zusammen zu sichern. Dies bedeutet z.B., dass ein Wissensraum als Einheit gesichert werden kann mit den darin enthaltenen Objekten. Aus der Datensicherung kann damit der Wissensraum wieder hergestellt werden.

- **Datenschutz**: Die Zuordnung zwischen Servern und Objekten kann eine Rolle spielen, da eine beliebige Verteilung unter Umständen problematisch sein kann. So ist es möglich, dass die Replikation von wichtigen Inhalten an nicht vertrauenswürdige Stellen unterbleiben soll. OceanStore (Kubiatowicz et al. 2000) verwendet z.B. zur Gewährleistung einer Datensicherheit eine Verschlüsselung sämtlicher Inhalte. Damit kann sichergestellt werden, dass die Berechtigungen für den Zugriff auf Objekte überall Gültigkeit besitzen.

Die Art der Verteilung der Räume über den Serververbund spielt also eine Rolle bei der Performanz der Zugriffe und der Verfügbarkeit von Ressourcen. Neben diesen allgemeinen Aspekten müssen die Strukturen von Wissensräumen innerhalb einzelner Server berücksichtigt werden.

Es gibt verschiedene Strategien der Verteilung, die sich aus der Struktur von Wissensräumen und aus verschiedenen existierenden Systemen und Ansätzen ableiten lassen. Diese sind nicht spezifisch einem Gateway-Ansatz zugeordnet.

- **Räumlich/Zonen**: Die Aufteilung der Wissensräume über verschiedene Knoten des Netzes erfolgt im Prinzip nach einer Aufteilung einer virtuellen Welt. Diese Vorgehensweise entspricht der Bildung von Zonen in MMOG (Jakobsson und Taylor 2003) und CVE-Systemen wie NPSNET, MASSIVE und SPLINE (Zyda et al. 1992; Greenalgh et al. 2000; Waters et al. 1997). Die Idee bei einer Übertragung dieser Konzepte auf den Wissensraum ist den Raum als Basis einer Verteilung zu verwenden. Die Übergänge zwischen den Bereichen ergeben sich dann aus den Verbindungen zwischen den Räumen. Alle Objekte, die sich in einem Raum befinden, sind zusammen mit dem Raum gespeichert. Diese Verteilung stellt die Bewegung eines Benutzers bzw. den Avatar in den Vordergrund.
- **Gruppen**: Wie bereits beschrieben, können Gruppen und Räume eng verwoben sein und die Gruppenarbeitsräume enthalten weitere Strukturen, die von den Mitgliedern der Gruppe kreiert worden sind. Dabei handelt es sich um Räume, Container, Dokumente und andere Objekte. Insgesamt kann einer Gruppe damit eine Vielzahl von Objekten zugeordnet werden. So bietet sich eine Aufteilung der Strukturen gemäß der

Gruppen an. Alle Räume, die einer bestimmten Gruppe zugeordnet sind, befinden sich auf dem gleichen Server. Die Überführung von Strukturen von einer Gruppe in eine andere beinhaltet dann möglicherweise eine Verschiebung in einen anderen Server. Diese Vorgehensweise leitet sich aus der Verteilung von Communities über verschiedene Server ab (vgl. Das et al. 1997).

- **Funktionale Verknüpfung**: Die Verteilung unterliegt keiner semantischen Struktur, sondern ist beliebig oder bestimmte Formeln liegen zugrunde, die z.B. für eine gleichmäßige Aufteilung von Objekten sorgen. In diesem Bereich sind besonders die DHT (Rowstron und Druschel 2001a; Stoica et al. 2001) einzuordnen, die die Objekte mit Hilfe einer Hashfunktion den einzelnen Knoten zuweisen.

Die Verteilung der Räume und Objekte muss sowohl bei statischen als auch bei dynamischen Verbünden berücksichtigt werden. Die bisherigen Betrachtungen haben sich ausschließlich auf statische Verbünde bezogen. Im folgenden Abschnitt werden die Überlegungen auf dynamische Situationen ausgeweitet, sodass zu jedem Zeitpunkt Knoten hinzukommen können.

3.4 Dynamischer Verbund

Ein dynamischer Verbund stellt eine größere Herausforderung als die bisherigen Konzepte dar. So kann ein Vertrauensverhältnis zwischen den einzelnen Knoten hier nicht vorausgesetzt werden. Unter Einbeziehung von mobilen Knoten gehen die Konzepte über einfache Client/Server-Architekturen hinaus. Jeder Knoten des Netzes bietet selbst Dienste an und verfügt über einen eigenen Bestand an Wissensräumen, Benutzern und Gruppen. Eine Autonomie der einzelnen Knoten ist daher von besonderer Bedeutung. Die Systeme können also sowohl autark als auch im Verbund zusammenarbeiten. Dies kann durch eine Verfügbarkeit von Objekten im gesamten Netzwerk realisiert werden.

Weiterhin existiert bei diesem Ansatz keine Abhängigkeit zu klassischen Serverinfrastrukturen, sondern es besteht auch die Möglichkeit der direkten Kooperation zwischen Knoten. Es handelt sich also um ein Peer-to-Peer-System, bei dem jeder Knoten des Netzes über unterschiedliche Funktionen verfügen kann. Auf der einen Seite stehen Peers, die fast ausschließlich Dienste in Anspruch nehmen. Andererseits existieren Knoten, die sehr stark einem klassischen Server ähneln und viele Dienste zur Verfügung stellen. Im Gegensatz dazu geht der Begriff Peer-to-Peer im ursprünglichen Sinne von der gleichen Funktionalität aller Knoten aus.

Trotzdem sind hybride Systeme, die unterschiedliche Knoten enthalten, ebenfalls dem Bereich Peer-to-Peer zuzuordnen. So wurden bei Napster bereits Server für spezielle Aufgaben eingesetzt (Suche, Zugangsknoten ins P2P-Netzwerk). Dies ist später zu der Idee so genannter Superpeers (Mizrak et al. 2003) erweitert worden. Dabei entspricht die Grundfunktionalität eines Superpeers der anderer Knoten eines Netzwerks.

Um einen gemeinsamen Datenraum für einen dynamischen Serververbund zu schaffen, sind Forschungen im Bereich von Peer-to-Peer-Dateisystemen von Belang (Dabek et al. 2001; Muthitacharoen et al. 2002; Clarke et al. 2001; Kubiatowicz et al. 2000). Diese ordnen jedem Server einen eigenen Bestand an Daten zu und alle beteiligten Peers können gegenseitig auf diese Daten zugreifen. Im Gegensatz zu diesen Dateisystemen müssen allerdings Objekte und insbesondere Wissensräume gespeichert werden. Systeme wie OceanStore (Kubiatowicz et al. 2000) gehen von einer Speicherung von Dateien aus und bieten daher lediglich ein verteiltes Dateisystem.

Eine Möglichkeit, trotzdem auf ein solches System zurückzugreifen, ist eine Speicherung der serialisierten Daten eines Objekts im Dateisystem.[12] Ein Nachteil dieses Verfahrens ist der fehlende Zugriff auf einzelne Werte eines Objekts. Auch müssen zusätzlich Verfahren realisiert werden, damit Suchmöglichkeiten nach bestimmten Attributen verfügbar sind. Schließlich muss auf die Dateiebene eine weitere Schicht aufgesetzt werden, um die Zugriffe zu kapseln und um auf der Basis von Objekten zu arbeiten.

Offensichtlich sind daher solche Peer-to-Peer-Dateisysteme nicht direkt als Basis eines verteilten CSCW-Systems geeignet. Da die obigen Systeme auf Distributed Hash Tables (DHT) (Karger et al. 1997) basieren, ist es sinnvoller direkt auf diese Ebene aufzubauen. Damit steht direkt eine Verbindungsinfrastruktur und eine Speicherung von Objekten zur Verfügung. Ein entsprechender Ansatz einer Verteilung von Objekten über mehrere Knoten findet sich in dem SimMud-Projekt (Knutsson et al. 2004). Dabei baut das System auf Pastry (Rowstron und Druschel 2001a) und Scribe (Rowstron et al. 2001)

---

[12] Dies ist z.B. durch die Ablage einer XML-Beschreibungsdatei möglich. Die Serialisierung wandelt alle Daten eines Objekts in Zeichenketten um und speichert diese. Die Daten werden so abgelegt, dass sie zu einem späteren Zeitpunkt wieder deserialisiert werden können. Gleichzeitig werden solche Serialisierungen zur Kommunikation zwischen verschiedenen Servern benötigt (Birrel und Nelson 1984).

als Multicast-Infrastruktur auf, damit insbesondere Positionswechsel innerhalb der Region an alle beteiligten Knoten effizient umverteilt werden können.

Insgesamt besteht der Serververbund aus verschiedenen Peers, die Dienste anbieten und in Anspruch nehmen. Die lokale Datenmenge jedes Peers umfasst zumindest einen Benutzer und einen lokalen Arbeitsbereich. In dem Netzwerk gibt es dedizierte Knoten, die über wesentlich mehr Kapazität verfügen als andere Peers und eine Vielzahl von Diensten anbieten. Besonders die Schnittstellen solcher Superpeers sind wesentlich umfangreicher und es werden verschiedene Protokolle als Zugangsmöglichkeiten angeboten wie beispielsweise eine Webschnittstelle, die den Benutzern einen gewohnten Zugang zu Dokumenten durch ihren Browser bietet. Außerdem können Webservices bereitgestellt und damit Dienste auch außerhalb des Netzes verfügbar gemacht werden (Hampel et al. 2004).

Die Gesamtheit der Knoten des Netzes bildet einen gemeinsamen Datenraum, sodass jeder Peer auf die gesamten Daten zugreifen kann. In jedem Knoten werden Objekte soweit wie nötig zwischengespeichert, damit auch Verbindungsabbrüche überbrückt werden können. Eine Suche im Serververbund kann z.B. durch einfache Weiterleitung einer Anfrage an alle Server realisiert werden. Die Ergebnisse werden wieder lokal gesammelt und dem Benutzer präsentiert.

Der dynamische Verbund kann beliebig erweitert werden, wobei die Aufnahme eines Knotens mit zusätzlichen Gruppenstrukturen nicht die Sicherheit des Verbundes beeinträchtigen darf. Die lokalen Berechtigungen eines Raums können z.B. aus Einträgen von Gruppen bestehen, die bestimmten Servern zugeordnet sind. Dann gilt es lediglich die Gruppen bzw. Server eindeutig zu identifizieren. Eine neu hinzukommende Gruppe (die sich auch auf einem anderen Server befinden kann) verfügt nirgendwo über Berechtigungen und hat damit keine besonderen Privilegien.

Die initiale Vernetzung eines Servers zu einem Verbund ordnet diesem Server (bzw. den Gruppen und Benutzern des Servers) daher eine eindeutige Identität zu und vergibt Zertifikate, die eine Verifikation dieser Identität zu einem späteren Zeitpunkt ermöglichen. Erst durch die Vergabe von Berechtigungen an Gruppen und Benutzer des Servers entstehen Vertrauensverhältnisse.

In Abbildung 3.4 ist ein Netzwerk mit drei einfachen Peers und zwei Hauptknoten (Superpeers) zu sehen. Die Hauptknoten bieten durch die bereitgestellten Protokolle viele Zugangsmöglichkeiten in das Netz. Damit kann auf die Räumen und Materialen im gemeinsamen Datenraum des Verbundes von

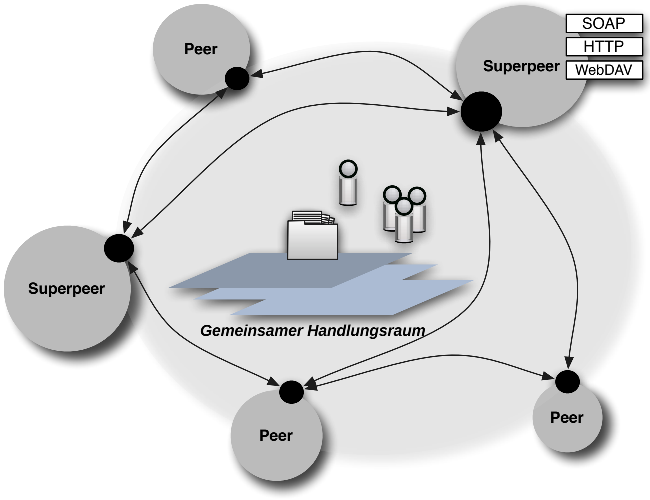


Abbildung 3.4: *Peer-to-Peer-Verbund*

außen zugegriffen werden.

Unter Einbeziehung von mobilen Knoten kann eine Erreichbarkeit der Daten dieser Knoten nicht garantiert werden. Im Gegensatz zu Serversystemen, die klassische Verbindungsinfrastrukturen nutzen, ist bei mobilen Knoten davon auszugehen, dass die Daten nicht mehr erreichbar sein werden. Auf das Konzept der Wissensräume übertragen können daher zwei Arten von Räumen identifiziert werden:

- **Persistenter Wissensraum**: Der Raum ist auf einem Knoten gespeichert, der über eine feste Verbindungsinfrastruktur verfügt.

- **Temporärer Wissensraum**: Der Raum ist auf einem temporären Knoten gespeichert. Eine Verfügbarkeit kann nicht garantiert werden.

3.4.1 Temporärer Wissensraum

Im Verlauf von Ad-hoc-Sitzungen stellt der Wissensraum ebenfalls die Basis einer Kollaboration verschiedener Teilnehmer dar. Ohne feste Verbindungsinfrastrukturen entstehen im Zuge solcher Sitzungen jedoch keine persistenten Wissensräume, die überall verfügbar sind, sondern um Räume die einen temporären Charakter besitzen (vgl. Eßmann et al. 2004). Ohne eine feste Verbindung zu einem statischen Server kann keine persistente Speicherung eines Raums gewährleistet werden. Aufgrund der schlechten Verfügbarkeit ist es denkbar, Wissensräume von mobilen Knoten in einen statischen Server zu überführen und damit die Erreichbarkeit zu erhöhen. Ziel des Konzepts eines temporären Wissensraums ist es diesen auch in einen persistenten Zustand überführen zu können. Deshalb ist es notwendig Räume zwischen Servern verschieben zu können.

3.5 Zusammenfassung

Der Wissensraum ist als zentrales Konzept eines raumbasierten CSCW-Systems vorgestellt worden. Dabei dienen die Räume als Bezugspunkte für die Benutzer, für Aktionen, Ereignisse und Berechtigungen. Die Grundlage für selbstorganisiertes kooperatives Arbeiten bilden die kooperativen Medienfunktionen. Auf technischer Ebene muss diesen ein Handlungsraum zugrunde gelegt werden, der insbesondere für eine Verteilung von Wissensräumen notwendig ist. Die möglichen Architekturen unterscheiden sich dabei in der Verwendung eines Handlungsraums oder eines Datenraums, der nicht ereignisorientiert ist. Auf der einen Seite stehen dabei statische Verbünde mit gemeinsamem Datenraum, der nur einen Teil der Objekte abbildet. Die Realisierung eines Handlungsraums ist darüber hinaus für statische und dynamische Verbünde möglich.

Im folgenden Kapitel werden verschiedene Szenarien vorgestellt, die die Entstehung eines Verbundes von Wissensräumen beschreiben und die Aktivitäten in Wissensräumen einbeziehen. Auf diese Szenarien aufbauend können verschiedene Anforderungen für eine Architektur verteilter Wissensräume abgeleitet werden.

4 Szenarien verteilter kooperativer Wissensräume

Mit dem Ziel Anforderungen für die Verteilung von Wissensräumen zu finden müssen zunächst die Struktur und das Arbeiten mit Wissensräumen berücksichtigt werden. Räume können keinesfalls isoliert betrachtet werden, sondern sind immer im Kontext der Umgebung und der Verbindungen zu weiteren Räumen zu sehen. Der Wissensraum steht als zentrales Konzept dieser Arbeit ebenso im Mittelpunkt aller Szenarien eines Verbundes. Dabei ist es notwendig einen Raum nicht als reine Dokumentenablage zu betrachten, sondern als Treffpunkt von Benutzern für synchrone und kontinuierliche Zusammenarbeit (vgl. Abschnitt 3.1.2).

Weiterhin steht in diesem Kapitel der Server als zentraler Speicherort von Objekten und Räumen im Vordergrund. Alle Szenarien gehen zunächst von einem serverorientierten Verbund aus. Dies steht im Zusammenhang mit der Zielrichtung, bestehende Client/Server-Architekturen zu erweitern. Im Verlauf der Arbeit relativiert sich diese Sichtweise hin zu einer möglichen Peer-to-Peer-Architektur, die mit einer vollständigen Neuentwicklung eines CSCW-Systems einhergeht.

Es gilt im Folgenden verschiedene Szenarien zu finden, die eine technische Perspektive darstellen (vgl. auch Hampel und Bopp 2004). Dabei sind sowohl die Entstehung von Servern als auch die unterschiedlichen Eigenschaften verschiedener Server zu berücksichtigen. In der Regel wird ein Verbund von Servern nicht direkt als solcher entstehen, sondern die Strukturen werden mit der Zeit zusammenwachsen. Auf der anderen Seite steht eine Auflösung existierender Verbünde mit der Abspaltung von Wissensräumen.

Auch der Zusammenhang von Gruppe und Wissensraum, der durch die Gruppenarbeitsräume gekennzeichnet ist (vgl. Abschnitt 3.1.4), spielt eine Rolle. Diese Räume bilden immer die Startpunkte für eine Kooperation von verschiedenen Benutzern.[1] Darunter können weitere Raum- und Containerstruk-

---

[1] Im Gegensatz dazu sind die Arbeitsräume von Benutzern lediglich private Bereiche.

turen entstehen, die jeweils im Kontext ihrer Umgebung stehen und damit ebenfalls der Gruppe zugeordnet sind. Ein Aspekt ist z.B. die Art der Zusammenarbeit innerhalb eines Raums, die es auf einen Serververbund auszudehnen gilt. Die folgenden Handlungen könnten in einem Verbund für Benutzer von Bedeutung sein:

- Materialien/Dokumente in Wissensräumen suchen,
- verteilte Wissensräume verbinden,
- von einem Server auf einen anderen wechseln,
- Materialien eines Wissensraums referenzieren,
- Dokumente zwischen Wissensräumen übertragen,
- mit Benutzern in Wissensräumen kooperieren.

Eine Verbindung zwischen Räumen unterschiedlicher Server bildet eine Grundlage für die meisten der genannten Aktionen. Damit entsteht die Möglichkeit für Benutzer, sich durch diese Verbindung zu bewegen und damit einen Wechsel des Servers durchzuführen.

Die Realisierung verteilter Wissensräume erfordert also die Berücksichtigung ganz unterschiedlicher Szenarien kooperativen Arbeitens. Die Struktur und der Aufbau von Wissensräumen müssen bekannt sein, um die nachfolgenden Szenarien und die daraus resultierenden Probleme zu erarbeiten (vgl. Abschnitt 3.1.3).

Die Auswahl der Szenarien basiert auf einer bestehenden Verteilung von Servern und Überlegungen zu der Entstehung eines Verbundes. Dabei erfolgt eine schrittweise Erweiterung ausgehend von einem einfachen Szenario, das aus dem Einsatz von CSCW-Systemen bekannt ist. An verschiedenen Stellen werden Systeme eingesetzt und zu einem späteren Zeitpunkt wird die Notwendigkeit von Übergängen deutlich.

Das erste Szenario befasst sich mit dem Aufbau eines Verbundes aus zunächst unabhängigen Strukturen. Daraus entstehen Überlegungen, wie solche zusammenwachsenden Strukturen später wieder gelöst werden können. Im dritten Szenario wird dann die Suche in einem Serververbund aufgegriffen, die eine entscheidende Rolle bei der Verknüpfung von Räumen, Dokumenten und Benutzern spielt.

4.1 Szenario 1: Zusammenschluss von Wissensräumen zu einem gemeinsamen Verbund

Die Ausgangssituation dieses Szenarios sind getrennt voneinander entstandene Wissensräume, die zunächst vollkommen unabhängig voneinander sind. Es handelt sich dabei um vollständig unabhängige Server, die über einen eigenen Bestand an Benutzern, Gruppen und verschiedenen Materialien verfügen. Benutzer bewegen sich innerhalb der Strukturen von Wissensräumen und greifen auf die dort vorhandenen Dokumente zu. Dabei erfordert ein Zugriff auf geschützte (nicht publizierte) Bereiche eines Servers entsprechende Berechtigungen und damit eine Authentifizierung des Benutzers.

Ein konkretes Beispiel für so eine Verteilung von lokalen Wissensräumen existiert an der Universität Paderborn. Verschiedene Arbeitsgruppen betreiben ihre eigenen Server zur Verwaltung von Materialien und zur Abwicklung von Vorlesungen und Übungsgruppen. Es kommt dabei schnell zu Situationen, in denen Studenten auf mehrere solcher Systeme zugreifen müssen. Hier bietet sich eine einheitliche Benutzerkennung an, die überall Gültigkeit besitzt. Gleichzeitig ergeben sich Anforderungen für Übergänge zwischen unterschiedlichen Wissensräumen, da die Benutzer von überall auf ihre Bereiche zugreifen möchten.

Es lassen sich nun verschiedene Motivationen ermitteln, Übergänge zwischen diesen Räumen zu schaffen:

- **Verweise auf Materialien eines entfernten Wissensraums anlegen**: In einem Wissensraum befinden sich Materialien, die für eine Gruppe von Benutzern eines anderen Raums von Interesse sind. Generell muss die Möglichkeit bestehen Materialien eines entfernten Wissensraums in eine Suche mit einzubeziehen.

- **Einheitliche Benutzerkennung im Verbund**: Ein Benutzer nimmt an verschiedenen Veranstaltungen auf unterschiedlichen, verteilten Wissensräumen teil und sollte eine einheitliche Benutzerkennung im Verbund besitzen.

- **Serverübergreifende Kooperation und Gruppenstrukturen**: Zur gemeinsamen Arbeit sollen auch Benutzer eines entfernten Wissensraums eingeladen werden können.

Analyse

Da der Wissensraum das zentrale Konzept ist und sich Benutzer zwischen den Wissensräumen bewegen, müssen Verbindungen zwischen verteilten Räumen geschaffen werden. Daraus folgt, dass es innerhalb eines Verbundes unumgänglich ist, Benutzern von anderen Bereichen Rechte einzuräumen. Nur dann sind diese Benutzer in der Lage sich uneingeschränkt im Verbund zu bewegen.

Die Frage stellt sich, in welcher Form eine gemeinsame Benutzer- und Gruppenverwaltung aufgebaut werden kann. Wie oben beschrieben, zählen zu den Motivationen eines Verbundes sowohl eine einheitliche Benutzerkennung als auch serverübergreifende Gruppenstrukturen. Die Server müssen also einen gemeinsamen Datenraum aufbauen, sodass ein Austausch von Daten innerhalb des Verbundes möglich ist.

Ein weiteres Kriterium des Verbundes ist, dass sich die Interaktion von Benutzern verschiedener Server nicht von der Interaktion von Benutzern, die sich auf dem gleichen Server befinden, unterscheiden darf. Daher kann die Beweglichkeit von Benutzern nicht eingeschränkt werden und muss für den gesamten Serververbund gelten. Die Verbindungen zwischen Räumen sind so zu erweitern, dass es möglich ist von einem Raum des lokalen Servers zu einem Raum eines entfernten Servers zu wechseln.

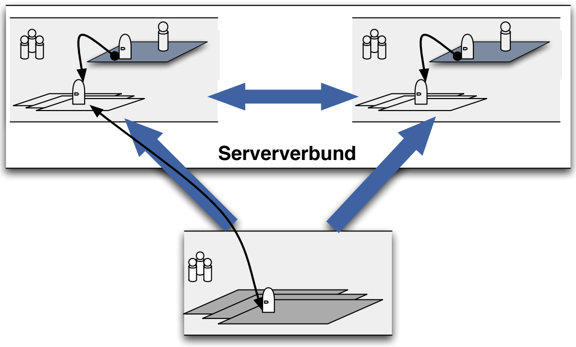


Abbildung 4.1: *Ein Server wird in einen Verbund aufgenommen*

Dieses Szenario ist nicht auf zusammenwachsende Einzelsysteme beschränkt, sondern es kann so erweitert werden, dass zu einem bestehenden Verbund ein neuer Server hinzukommt (vgl. Abbildung 4.1). Genauso können auch zwei oder mehr Serververbünde zusammengeführt werden. Ausgehend von einer Transparenz einzelner Verbünde unterscheidet sich dies nicht von dem Zusammenschließen von Einzelsystemen zu einem Verbund. Schließlich ist ein

Verbund keineswegs als statisch anzusehen, sondern es ist immer möglich, dass ein neuer Server hinzukommen soll.

4.2 Szenario 2: Abspalten von Wissensräumen

Neben einer Erweiterung eines Verbundes muss auch das Ausscheiden eines Servers berücksichtigt werden. Dabei ist zwischen einem temporären Ausfall und einer endgültigen Verkleinerung des Verbundes zu unterscheiden. Obwohl Serversysteme mit permanenter Verbindung zum Internet eine hohe Erreichbarkeit besitzen, kann generell nicht von einer solchen Voraussetzung ausgegangen werden. Ein kurzfristig nicht verfügbarer Server führt dazu, dass die entsprechenden Querverweise nur temporär nicht aufgelöst werden können. Die Dauer und Häufigkeit der Ausfälle spielt hier eine entscheidende Rolle, denn bei längeren Zeiträumen müssen adäquate Strategien gefunden werden, die Materialien weiterhin bereitzustellen.

Das Szenario setzt einen zunächst intakten Verbund voraus. Dieser Verbund wird dann verkleinert, indem sich ein Server ablöst. Auf der Ebene der im Server vorhandenen Strukturen beinhaltet dies eine Abspaltung von Wissensräumen, die nicht mehr erreicht werden können.

Analyse

Grundsätzlich muss zwischen festen Infrastrukturen und temporären Knoten unterschieden werden. Dabei kann in beiden Fällen keine generelle Erreichbarkeit von Knoten garantiert werden, da es auch bei festen Vernetzungen zu Verbindungsabbrüchen kommen kann. Bei temporären Knoten ist jederzeit mit einem solchen Ausfall zu rechnen. Ein temporäres Ausscheiden bedeutet, dass der Server zu einem späteren Zeitpunkt wieder verfügbar sein wird. Die Frage stellt sich, ob überhaupt ermittelt werden kann, wann und ob ein Server wieder zu dem Verbund zurückkehrt. Die Dauer eines Netzausfalls kann sehr unterschiedlich sein und damit unter Umständen schon große Beeinträchtigungen des Arbeitsablaufs zur Folge haben.

Des Weiteren ist bei einer Wiederaufnahme eines Servers die Wiedererkennung ein wichtiger Punkt, denn die Eindeutigkeit von Objekten muss gewährleistet sein. Besonders in Bezug auf Benutzer und Gruppen ist dies von großer Bedeutung, da diese in lokalen Strukturen eingetragen sein können und damit unter Umständen auch Berechtigungen verbunden sind. Es muss daher ausgeschlossen sein, dass sich ein Server als ein anderer Server ausgeben kann. Nur so kann die Authentizität von Gruppen und Benutzern sichergestellt werden.

Das endgültige Abspalten eines Servers von dem Verbund bedeutet, dass ein Server und damit verbunden alle Referenzen nicht mehr verfügbar sind. Jede Verbindung zwischen Wissensräumen wird für Benutzer zu Sackgassen und die Verweise auf Dokumente sind nicht mehr auflösbar. Auf dem verbleibenden Serververbund bleiben Spuren in Form von Referenzen an vielen Stellen zurück, die auf den abgespaltenen Server verweisen. Dabei handelt es sich um Übergänge zu Räumen eines entfernten Servers, Referenzen auf Gruppen zur Vergabe von Rechten oder andere Abhängigkeiten.

Auf jeden Fall muss gewährleistet sein, dass der Serververbund nach der Abspaltung einzelner Knoten weiterhin korrekt arbeitet. Im Idealfall muss die Transparenz für den Benutzer ebenfalls weiterhin gegeben sein — dieser hat sich, wie im vorherigen Szenario beschrieben, transparent innerhalb des Serververbundes bewegt. Falls der Serververbund als solcher für den Benutzer unsichtbar ist, stellt sich die Frage, inwieweit die fehlenden Bereiche eines Servers zu Verwirrungen führen können.

Mit einem einzelnen Server, der sich vom Verbund ablöst, wird genauso verfahren wie mit dem übrigen Serververbund. Ebenso ist es möglich, dass sich ein Serververbund in zwei Verbünde aufteilt. Inwieweit die einzelnen Server und Serververbünde funktionsfähig sind, richtet sich nach dem Umfang der Integration und der Anzahl der Querverbindungen. Umso mehr die Knoten untereinander verbunden waren, desto häufiger wird ein Benutzer mit toten Verbindungen konfrontiert. Wieder stellt sich die Frage, inwieweit es dem Benutzer transparent ist, dass bestimmte Verbindungen nicht mehr existieren oder diese als nicht funktionsfähig gekennzeichnet sind.

Weiterhin ist die Verschiebung von Objekten innerhalb von Serververbünden für dieses Szenario relevant. Wenn die Systemarchitektur eine Verschiebung von Objekten vorsieht, kann es passieren, dass ein ehemals lokales Dokument nach einem Zerfall des Verbundes nicht mehr auf dem eigenen Server existiert.

Abbildung 4.2 zeigt die Absplittung eines einzelnen Servers von einem Verbund. Dieser bleibt weiterhin in Form von zwei Servern bestehen. Referenzen zwischen dem Einzelserver und dem verbleibenden Verbund sind wie gekennzeichnet aufzulösen.

Insgesamt stellt sich die Frage, inwieweit solche verwobenen Strukturen auflösbar sind. Diese Frage ist in technischer und konzeptioneller Hinsicht zu beantworten.

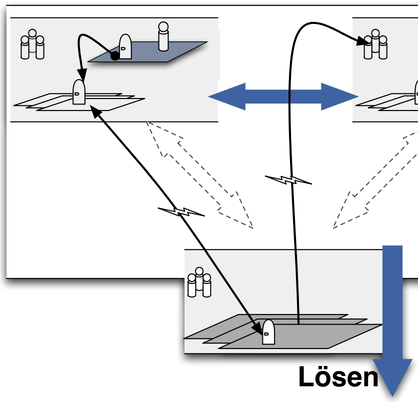


Abbildung 4.2: *Auflösen eines Verbundes*

4.3 Szenario 3: Suche in einem Verbund von Wissensräumen

Eine Suche wird auf einem Server durch einen Benutzer ausgelöst. In der Regel bedeutet dies, dass der Benutzer angemeldet sein muss und lediglich Dokumente findet, für die er auch Zugriffsrechte besitzt. Damit sind alle nicht-öffentlichen Bereiche eines Servers für Suchdienste wie z.B. Google unsichtbar. Es ist also nicht möglich Inhalte eines kollaborativen Systems von außen abzufragen.

Die meisten CSCW-Systeme verfügen über interne Suchfunktionen, die jedoch nicht der Außenwelt zur Verfügung gestellt werden. Dabei gibt es sowohl einfache Suchen nach Namen von Objekten als auch komplexere Suchen mit vielen Parametern oder die Suche nach Inhalten von Dateien (Volltextsuche).

Dieses Szenario setzt einen bestehenden Verbund voraus. Ein Benutzer gibt auf einem Server des Verbundes ein Schlüsselwort ein und wartet auf ein Suchergebnis. Ein Benutzer, der eine Verbindung zu einem Wissensraum schaffen möchte, muss zum Beispiel ausgehend von seiner aktuellen Position einen Zielraum bestimmen. Dies kann über die Suchfunktion durch die Suche nach dem Zielraum unterstützt werden, indem alle Räume zurückgegeben werden, die dem Schlüsselwort entsprechen.

Analyse

Allgemein ist es in einem Verbund von Servern notwendig, dass nicht nur die Dokumente des lokalen Servers, sondern des Verbundes insgesamt gefunden werden können. Erst dadurch ergeben sich Anknüpfungspunkte, die Server zusammenwachsen zu lassen.

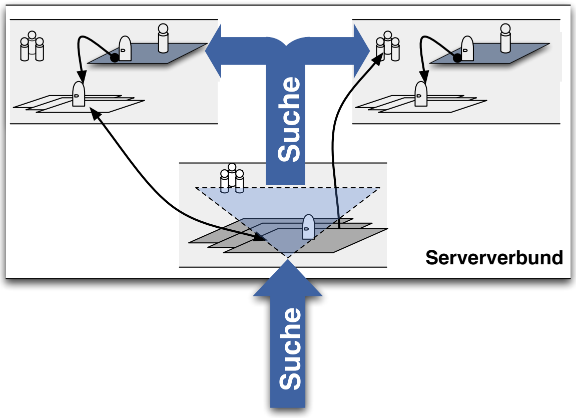


Abbildung 4.3: *Suche in einem Verbund*

Abbildung 4.3 zeigt eine Suche auf einem einzelnen Server, die dann innerhalb des Verbundes an die anderen Server weitergeleitet wird. Im Folgenden wird der Datenbestand des lokalen Servers durchsucht und zu diesen Ergebnissen wird das Ergebnis der weitergeleiteten Suchen hinzugefügt. Das Auffinden von Räumen über ihren Namen ist z.B. essentiell für ein Verbinden von Räumen unterschiedlicher Server. Ebenso muss es möglich sein Benutzer und Gruppen anhand ihres Namens zu finden. Erst dann kann ein kollaboratives Arbeiten über Servergrenzen hinweg eingeleitet werden.

4.4 Objekte zwischen Wissensräumen übertragen

Ein Benutzer gelangt über einen Übergang von einem lokalen Wissensraum zu einem entfernten Wissensraum, der auf einem anderen Server gespeichert ist als das Benutzerobjekt. In diesem Raum stehen einem Benutzer Materialien zur Verfügung und es besteht die Möglichkeit in tiefere Strukturen einzutauchen. Insbesondere wenn sich ein Benutzer des Aufenthaltsorts nicht bewusst ist muss der gleiche Aktionsradius wie auf lokalen Strukturen erhalten bleiben.

Dazu zählen das Abrufen und die Modifikation von Dokumenten, aber auch das Übertragen von Objekten.[2] Damit ist das Aufnehmen eines Objekts z.B. in den Rucksack eines Benutzers gemeint (vgl. Abschnitt 3.1.6).

Analyse

Auf technischer Ebene ist das Benutzerobjekt auf einem anderen Server gespeichert als das zu übertragende Objekt. Damit muss das Objekt auf den Speicherort des Benutzers verschoben werden oder für das Aufnehmen ein lokales Duplikat erzeugt werden. In jedem Fall kommt es dabei zu einer Übertragung der Objektdaten zwischen den Servern (vgl. Abbildung 4.4).

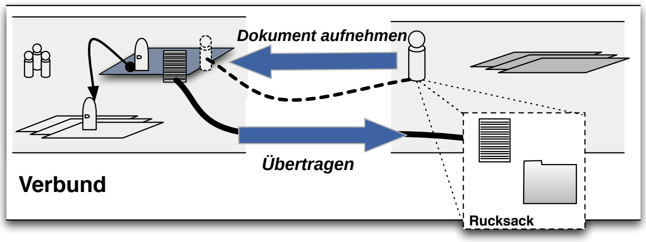


Abbildung 4.4: *Übertragen von Objekten zwischen verteilten Wissensräumen*

4.5 Ereignisse in verteilten Wissensräumen

Die Benutzung einer Verbindung von einem Wissensraum in einen entfernten Raum erfordert Berechtigungen, um dort auf Materialien zugreifen zu können. Gleichzeitig ist hier die Basis für eine Kooperation gegeben, wenn der Aktionsradius des Benutzers nicht eingeschränkt wird. Damit neben einer asynchronen Kooperation auch ein synchrones Zusammenarbeiten gelingen kann, müssen die Ereignisse an alle Beteiligten übertragen werden. Die Zustände aller Objekte werden in diesem Fall direkt synchronisiert.

Die folgenden Aktionen finden in Wissensräumen statt:

- Lesen eines Dokumentes,
- Erzeugen eines Objekts,

---

[2] Die Bereitstellung von kooperativen Medienfunktionen (Hampel 2002) ist im Idealfall auch für verteilte Raumstrukturen zu gewährleisten.

- Löschen eines Objekts,
- Modifikation eines Dokuments,
- Arrangieren/Anordnen von Objekten,
- Aufnehmen/Verschieben von Objekten,
- Kopieren von Objekten,
- Benutzer und Gruppen zur Mitarbeit einladen,
- Kommunikation mit anderen Benutzern.

Diese Aktionen stellen typische Aktivitäten in Wissensräumen dar. Das Arrangieren bezieht sich dabei auf eine Anordnung innerhalb eines Raums. Im Gegensatz dazu wird ein Objekt durch Verschieben in einen anderen Kontext bewegt. Eine genaue Beschreibung der einzelnen Aktionen wird in Abschnitt 5.6 gegeben. Die Wahrnehmung von Aktionen erfolgt in der Regel über eine Ereignisverarbeitung (vgl. Fuchs et al. 1995).

Analyse

Aus den oben genannten Gründen ist ein Austausch von Ereignissen[3] über Servergrenzen hinweg notwendig, damit diese für alle Benutzer des Verbundes wahrnehmbar werden. Konzeptuell dient dies zur Erhaltung kooperativer Medienfunktionen im Serververbund (vgl. Abschnitt 3.3). Der Verbund wird im Idealfall nicht von den Benutzern wahrgenommen und unterscheidet sich für diese nicht von einem einzelnen Server. In so einer Situation ist davon auszugehen, dass sich stark verwobene Strukturen bilden mit vielen Übergängen zwischen den Systemen.

4.6 Zusammenfassung

Alle Szenarien gehen von einem Verbund von Servern aus, der durch Übergänge zwischen zunächst unabhängigen Bereichen gebildet wird. Von den in Kapitel 2 vorgestellten Architekturen und Systemen scheint zunächst keine ohne Weiteres den Szenarien zu entsprechen. Aus dem Blickwinkel der Client/Server-Architekturen ergeben sich die meisten Überschneidungen bei den MMORPG

---

[3] Ereignisse beschreiben Zustandsänderungen von Objekten

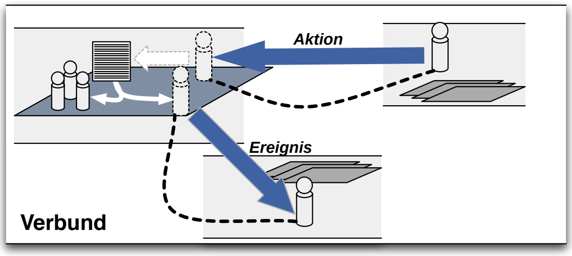


Abbildung 4.5: *Austausch von Ereignissen zwischen verteilten Wissensräumen*

(vgl. Abschnitt 2.2). Die Systeme wagen keine neuartigen Architekturen, sondern beschränken sich darauf die bekannten und erprobten Ansätze lediglich zu erweitern, um eine Verbesserung der Skalierbarkeit auf wenige tausend Spieler zu erreichen.

Im Bereich der Peer-to-Peer-Systeme ist eine Ausrichtung auf den Austausch von Dateien festzustellen. Vorherrschend ist dabei ein einmaliges Veröffentlichen von Dateien ohne die Möglichkeit später Modifikationen vorzunehmen. Lediglich die DHT-Systeme bieten hier eine grundlegende Systemklasse, die für weitere Anwendungen geeignet erscheint (vgl. Abschnitt 2.3.3).

Auf der anderen Seite bieten alle Systeme verschiedene Aspekte, die für einen Verbund von CSCW-Systemen relevant sind. Bei einer Umsetzung können erfolgreiche Architekturen als Muster dienen, um ein alltagstaugliches kollaboratives System zu schaffen.

Die zu erarbeitende Architektur soll darüber hinaus den Wissensraum in den Vordergrund stellen und eine Verteilung von Räumen über verschiedene CSCW-Server ermöglichen. Dazu sind Konzepte nötig, die sich zum Teil bereits in existierenden Systemen finden lassen. Dabei enthält der Bereich der Peer-to-Peer-Systeme viele Aspekte, die für einen dynamischen Verbund von CSCW-Servern relevant sind. Auf der anderen Seite unterstützen CVE-Systeme lediglich den Aufbau von statischen Verbünden. Allerdings findet bei diesen Systemen eine Aufteilung einer virtuellen Welt statt, die sich auch auf Wissensräume anwenden lässt. Räume bilden einen Kontext für die Verarbeitung von Ereignissen und gegenseitige Wahrnehmung von Benutzern und beschreiben daher im Sinne des AOIM eine Area of Interest (Hosseini et al. 2002).

Bei den Anforderungen sind zwei verschiedene Ebenen zu unterscheiden: Auf der einen Seite ergeben sich Unterschiede in der Verbindungsinfrastruktur zwischen den Servern. So können sowohl statische als auch dynamische Verbünde entstehen. Statische Verbünde können direkt als solche entstehen oder aus Einzelservern zusammengefügt werden, die einen Bestand an Räumen, Benutzern und Objekten in den Verbund mitbringen.

Des Weiteren sind Unterschiede in der Verteilung von Objekten zu bedenken. Damit die Möglichkeit für einen Server besteht einen Verbund später wieder zu verlassen, muss es weiterhin lokale Objekte und Benutzer auf diesem Server geben. Übergänge zu anderen Servern sind nicht mehr verfügbar und entfernte Benutzer und Gruppen verlieren ihre Berechtigungen und können nicht mitarbeiten.

Zusammenfassend lassen sich die Anforderungen aus den Szenarien wie folgt feststellen:

- **Gemeinsamer Datenraum**: Die Objekte der Server stehen jeweils anderen Servern im Verbund in Abhängigkeit von Berechtigungen zur Verfügung (vgl. Abschnitt 3.2).
- **Dezentrale Datenhaltung**: Die Objekte sind nicht an einer zentralen Stelle gespeichert, sondern jeder Server verfügt über einen eigenen Bestand an Benutzern, Gruppen, Räumen und Dokumenten.
- **Autonomie**: Die einzelnen Server sind möglichst unabhängig von dem Verbund und können auch alleine bestehen. Jeder Server wird lokal administriert.
- **Transparenz**: Es ist für Benutzer möglich, sich transparent durch den Serververbund zu bewegen und Objekte von einem Raum in einen Raum eines anderen Servers zu überführen.
- **Dynamischer Verbund**: Es können beliebig Server zu dem Verbund hinzukommen und den Verbund wieder verlassen.

Neben einem dynamischen Verbund von Servern ist es sinnvoll gleichzeitig auch spontane Vernetzungen mit einzubeziehen. Insgesamt bildet der Verbund unter Berücksichtigung der obigen Kriterien einen gemeinsamen Handlungsraum, der die gesamten Daten des Verbundes jedem einzelnen Knoten und damit den Benutzern transparent bereitstellt.

Aus den obigen Anforderungen folgen weitere Kriterien, die sich auf die Verwendung von Wissensräumen beziehen. Benutzer treffen sich in einem Wissensraum, um dort gemeinsam zu arbeiten. Damit dies auch in einem Serververbund für alle Benutzer möglich ist, müssen Übergänge zwischen den Servern existieren.

- **Verbindungen zwischen Wissensräumen unterschiedlicher Server**: Es ist unbedingt notwendig, Räume von unterschiedlichen Servern miteinander zu verbinden. Diese Verbindungen unterscheiden sich für den Benutzer im Idealfall nicht von anderen Verbindungen zu Räumen des lokalen Servers.

- **Serverübergreifende Benutzer und Gruppen**: Auf jedem Server des Verbundes gibt es Benutzer- und Gruppenstrukturen. Die Eindeutigkeit dieser Strukturen muss gewährleistet sein, sodass Gruppen und Benutzer überall identifiziert werden können. So ist auch eine Authentisierung von Benutzern gegenüber beliebigen Servern des Verbundes möglich.

- **Serverübergreifende Suche**: Es muss gewährleistet sein Objekte auf beliebigen Servern aufzufinden. Eine Kooperation mit Benutzern von anderen Servern erfordert z.B., diese Benutzer zur Zusammenarbeit in einen lokalen Wissensraum einzuladen.

- **Austausch von Ereignissen**: Zur Schaffung eines gemeinsamen Handlungsraums müssen neben den Objekten auch Ereignisse ausgetauscht werden. Die Aktivitäten innerhalb eines Wissensraums sollen auch für Benutzer eines entfernten Raums wahrnehmbar sein, damit eine Transparenz für den Benutzer gegeben ist. Insbesondere für synchrone Zusammenarbeit ist dies unbedingt notwendig.

Sowohl die allgemeinen Kriterien eines verteilten CSCW-Systems als auch die spezifischen Anforderungen für verteilte Wissensräume sind bei einer Architektur verteilter Wissensräume zu berücksichtigen. Dies bedeutet, dass sich die technischen Anforderungen an Wissensräume so verallgemeinern lassen, dass sie den Anforderungen eines CSCW-Systems entsprechen.

Im nächsten Kapitel werden zunächst weitere technische Grundlagen für einen Verbund von Servern gelegt. Dabei werden sowohl die Szenarien einbezogen als auch die bereits beschriebenen Multiserversysteme. Insbesondere erfolgt eine detaillierte Vorstellung der technischen Aspekte der Systemklassen.

5 Verteilte Objekte

In diesem Kapitel wird auf verschiedene Ansätze verteilter Architekturen eingegangen. Ziel ist es, einen konzeptionellen Rahmen für verteilte kooperative Wissensräume zu schaffen. Dabei werden insbesondere die unterschiedlichen Komponenten eines Servers berücksichtigt. Diese ergeben sich aus einem Kern mit einigen Modulen, der Kommunikation zwischen Server und Client (Server und Server), Persistenz von Objekten, Benutzern, Gruppen und Rechten.

Bei den Architekturen muss zwischen zwei Ansätzen unterschieden werden: Auf der einen Seite existieren einfache Architekturen, die direkt auf einer Datenbank aufbauen und als technische Grundlage z.B. ein LAMP-System (Linux, Apache, Mysql, PHP) verwenden. Komplexere Architekturen sind in der Regel objektorientierte Systeme, die ebenfalls Datenbanken nutzen können, wobei Zugriffe vollständig abstrahiert werden und in der Regel direkt auf Objekten gearbeitet wird.

- Objektorientiert: Als Basis der verteilten Architektur sind Objekte anzusehen. Alle Operationen werden auf Objekten durchgeführt. So genannte Object Repositorys sind verteilte Speicherorte von Objekten, die dort abgelegt und abgerufen werden können. Um die Integrität zu gewährleisten, kommt es zu einer Synchronisation des Zustands eines Objekts.

- Datenbankorientiert: Die Architektur arbeitet direkt auf einer Datenbank und es werden typische Verfahren von Datenbanken eingesetzt wie z.B. Transaktionsmanagement. (Ritter 1995)

Für eine verteilte Architektur existieren verschiedene Motivationen. Im Allgemeinen spielt bei fast allen Client-/Server-Systemen die Performanz eine bedeutende Rolle, da auf Seite des Servers schnell ein Engpass entstehen kann. Neben der Verbesserung der Hardware eines einzelnen Servers ist die Erweiterung auf mehrere Server eine mögliche Lösung, um solchen Engpässen entgegenzuwirken. Darüber hinaus stehen bei verteilten Wissensräumen die Übergänge zwischen Räumen von zunächst unabhängigen Bereichen im Vordergrund (z.B. Server, die einzeln entstanden sind und später zusammenwachsen). Weiterhin können grundsätzlich beliebige Ressourcen in einer verteilten Umgebung zwischen allen Knoten aufgeteilt werden.

Eine Architektur der Verteilung auf mehrere Server kann als *m:n*-Architektur beschrieben werden, da *m* Server mit *n* Clients kommunizieren (wobei in der Regel *m* wesentlich kleiner als *n* sein wird). Dies steht im Gegensatz zu einem klassischen Client/Server Ansatz mit einem Server und einer Vielzahl von Clients und damit einer *1:n*-Situation. Im Bereich von Peer-to-Peer-Vernetzungen ist sogar jeder Client ebenfalls Server und in der oben eingeführten Notation ist dies als *n:n*-Situation anzusehen — für jeden Client steht in so einem Netzwerk also im Prinzip ein Server zur Verfügung. Offensichtlich ist die Leistung eines einzelnen Servers in einer Client/Server-Situation nicht mit einem Peer in einer Peer-to-Peer-Vernetzung gleichzusetzen. Hier spiegelt sich auch eine mögliche Verteilung von Wissensräumen wider. Entweder sind die Räume über mehrere Server verteilt (*m* Server und *n* Clients) oder es befinden sich Wissensräume auf jedem Knoten eines Peer-to-Peer-Netzwerks.

Insgesamt wird in diesem Kapitel auf sämtliche Komponenten einer Plattform für verteilte Wissensräume eingegangen. Dabei müssen allgemeine Anforderungen an ein raumbasiertes System Berücksichtigung finden. Neben einer Ereignisverarbeitung und einem Rechtemanagement, die ebenso auf einzelnen Servern zur Verfügung stehen müssen, sind im Wesentlichen die verschiedenen Teile des Objektmodells (vgl. Abschnitt 3.1.7) bei der Umsetzung von Belang. Dazu zählen, neben dem Wissensraum, besonders Gruppen und Benutzer.

Als Ausgangspunkt werden zunächst der Begriff Objekt eingeführt und verschiedene Technologien aufgezeigt, die direkt im Zusammenhang mit objektorientiertem Design stehen (Persistenz, Ereignisverarbeitung). Wie aus dem Objektmodell ersichtlich ist, handelt es sich bei Wissensräumen ebenfalls um Objekte (vgl. Abschnitt 3.1.7). Aus diesem Grund muss eine Verteilung von Räumen (Benutzer, Gruppen und anderen abgeleiteten Klassen) auf Basis einer Verteilung von Objekten erfolgen.

5.1 Objekte

Objekte sind Instanzen von Klassen in einer objektorientierten Programmiersprache. Es handelt sich dabei um Abbilder von physischen oder konzeptionellen Objekten, die von dem Programmierer identifiziert worden sind. Die Objekte in der Applikation können daher als reale Objekte aufgefasst werden.

Dies führt zu intuitiven und verständlichen Systemen (vgl. Pugh et al. 1987).

Virtuelle Umgebungen sind durch Objekte beschrieben, die Gegenstände darstellen und von den Benutzern manipulierbar sind. Von der eigentlichen Klasse *Objekt* existieren dabei keine Instanzen — sie definiert nur die generelle Schnittstelle zur Manipulation eines Objekts.

Objekte verfügen im Allgemeinen über verschiedene Attribute, die den Zustand beschreiben. Die Identifikation eines Objekts erfolgt über eine eindeutige ID (kurz OID), die normalerweise durch einen Zahlenwert abgebildet wird. Weiterhin können Objekten Berechtigungen zugeordnet werden, die regulieren, welcher Benutzer welche Aktionen mit einem Objekt ausführen darf. Neben Benutzern existieren noch Gruppen, die eine Menge von Benutzern zusammenfassen. Diese können ebenfalls für eine Vergabe von Berechtigungen verwendet werden, wobei solche Rechte jedes Mitglied der Gruppe betreffen. Gruppenstrukturen sind darüber hinaus hilfreich, um Gruppen von Benutzern zu definieren, die innerhalb von Wissensräumen miteinander kooperieren können.

5.2 Benutzer, Gruppen und Rechte

Gruppenstrukturen und Zugriffsrechte haben eine entscheidende Bedeutung für kooperatives Arbeiten. Obwohl es in vielen Situationen ausreicht allen Benutzern einer Gruppe umfassende Berechtigungen einzuräumen, kann es oftmals trotzdem sinnvoll sein einzelne Zugriffsrechte zu beschränken. So sind Aufgaben wie das Passwort eines Benutzers zu ändern nur Administratoren erlaubt.

Weiterhin ist es üblich, anderen Gruppen lesenden Zugriff auf eigene Strukturen zu gewähren oder, falls eine vollständige Zusammenarbeit erwünscht ist, umfangreichere Rechte einzuräumen. Dabei existieren in kooperativen Systemen neben einfachen Rechten, die den Lese- und Schreibzugriff regeln, erweiterte Berechtigungen, die sich besonders auf den Wissensraum beziehen. Zu nennen sind hier das Recht neue Dokumente in einen Raum einbringen zu können (*insert*-Recht) und die Möglichkeit Dokumente anordnen zu können (*move*-Recht).

Ein Sicherheitssystem sollte also auf der einen Seite umfassende Möglichkeiten bieten, auf der anderen Seite allerdings dem Benutzer möglichst verständlich sein. Dies kann z.B. durch unterschiedliche Rechtedialoge realisiert werden. Im Idealfall muss sich ein Benutzer allerdings so wenig wie möglich mit Rechten befassen.

Es kann zwischen zwei grundsätzlichen Ansätzen der Vergabe von Berechtigungen unterschieden werden:

- Speicherung an jedem Objekt/Dokument (Access Control Listen (ACL) (Lampson 1974)): Die Daten sind zentral an jedem Objekt gespeichert. Jeder Eintrag weist einem Benutzer oder einer Gruppe bestimmte Rechte zu.
- Zentrale Datenhaltung an jedem Benutzer/jeder Gruppe (Rollenbasierte Rechte (Ferraiolo und Kuhn 1992)): Die Rechte sind direkt einem Benutzer oder einer Gruppe zugeordnet.[1] Aus einem Objekt/Dokument ist direkt nicht ersichtlich, welcher Benutzer es manipulieren darf.

Beide Ansätze haben verschiedene Vor- und Nachteile. Bei einer zentralen Datenhaltung ist vom Benutzer/von der Gruppe ausgehend direkter ersichtlich, welche Benutzer an welchen Objekten über Berechtigungen verfügen. Analog dazu ist bei dem Ansatz mit einer lokalen Datenhaltung in jedem Objekt dort direkt ersichtlich, wer über Berechtigungen verfügt. Insgesamt lassen sich daraus zwei verschiedene Sichten (aus Sicht eines Benutzers oder Administrators) identifizieren:

- ACLs bieten eine Sicht, die zu einem Objekt sofort die Berechtigungen aufzeigen kann. Diese Sicht ist nützlich, um Berechtigungen für ein Objekt zu manipulieren (einen Benutzer oder eine Gruppe in eine ACL eintragen). Der Nachteil dieser Vorgehensweise ist, dass es nicht ohne Weiteres möglich ist alle Rechte eines Benutzers darzustellen.
- Rollen bieten eine Sicht, die die Rechtevergabe an einen Benutzer bevorzugt. Bei Betrachtung eines Benutzers kann dort also sofort ermittelt werden, an welchen Objekten/Wissensräumen dieser Berechtigungen besitzt. Auf der anderen Seite ist nicht direkt an einem Objekt ersichtlich, welche Benutzer Zugriffsrechte haben.

Das Konzept der Rollen ist eng mit dem Konzept der Gruppen verwandt und leitet sich aus realen Strukturen ab. Eine Rolle stellt eine Klassifizierung der Personen dar (Wang 1999). Eine Gruppe ist ein sehr allgemeiner Begriff für

---

[1] Als Vorläufer von Rollen sind Systeme anzusehen, die den Benutzern direkt Rechte an Objekten zuschreiben, ohne den Umweg über Rollen zu gehen. Diese Vorgehensweise hat den Nachteil, dass diese Rechte für jeden Benutzer einzeln vergeben werden müssen.

eine Menge von Personen. Gruppe und Rolle fassen mehrere Benutzer zusammen, die für Berechtigungen an einem Objekt (oder Strukturen) verwendet werden können.

Es ist möglich mit Gruppen Rollen auszudrücken und umgekehrt. Die Mitgliedschaft in einer Gruppe gibt einem Benutzer bestimme Privilegien, was der Zuordnung einer Rolle für einen Benutzer entspricht. Daher ergibt sich durch die Verwendung von ACLs mit Gruppen eine ähnliche Struktur. Wang hält dies jedoch für ungünstig, da aus organisatorischer Sicht eine Vermischung der verschiedenen Konzepte nicht intuitiv sein könnte.

Bevor eine Prüfung von Zugriffsrechten stattfinden kann, muss zunächst die Identität eines Benutzers ermittelt werden. Dazu ist eine Authentisierung notwendig, die in der Regel durch Eingabe eines Namens mit dazugehörigem Kennwort gekennzeichnet ist.

5.2.1 Authentisierung

Eine Identifizierung eines Benutzers ist in einem CSCW-System unbedingt notwendig. Nur dann kann das System entscheiden, zu welchen Aktionen ein Benutzer autorisiert ist (vgl. Lampson et al. 1992). Darüber hinaus können so persönliche Daten gespeichert und auch anderen Benutzern Informationen über die Aktivitäten eines Benutzers präsentiert werden. Dies wird bereits bei der Kommunikation zweier Benutzer deutlich, denn ohne Identifizierung bleibt die Identität eines Benutzers verborgen.

Die Identifizierung selbst erfolgt in der Regel durch die Eingabe einer Benutzerkennung und Verifikation durch ein Passwort. Ein sicheres Passwort und eine Verschlüsselung der Kommunikation muss vorausgesetzt werden, um ein Minimum an Sicherheit für die Benutzer und das System zu gewährleisten.

Im Gegensatz zu der Identifikation von Objekten über die OID (vgl. Abschnitt 5.1) erfolgt die Zuordnung bei Gruppen und Benutzern durch den Namen. Dies ist bei Benutzern auf den Vorgang der Authentisierung zurückzuführen, der in der Regel mit Name und Kennwort arbeitet. Daher muss das zugehörige Benutzerobjekt zu einem Namen ermittelt werden. Gleichzeitig identifizieren sich Benutzer gegenseitig anhand eines Namens (Nickname, E-Mail, Vor- und Nachname). Ebenso verhält es sich bei den Gruppen im Kontext einer Vergabe von Berechtigungen. Ein numerischer Wert bietet dem Benutzer keinen sinnvollen Zugang, sodass eine Unterscheidung zwischen den verschiedenen Gruppen anhand von eindeutigen Namen erfolgen muss.

Die Authentisierung erfolgt im Idealfall schrittweise durch verschiedene Ebenen, die jeweils darüber entscheiden können, ob dem Benutzer Zugriff zum System gewährt wird (vgl. PAM – Pluggable Authentication Modules: Samar 1996). Dabei ist ein modularer Aufbau vorzuziehen, um verschiedene Komponenten realisieren und austauschen zu können. So ist es denkbar, dass auch Sperrinformationen für einen Benutzer durch ein spezielles Modul zu diesem Zweck vorliegen.

Es kann sinnvoll sein, die Authentisierung von Benutzern im gesamten Verbund zu erlauben. Von einem Betrieb mit einem einzelnen Server ausgehend verfügt ein solcher Server jeweils über einen lokalen Bestand an Benutzern. Diese identifizieren sich gegenüber ihrem Server und können sich dann in dem Verbund bewegen. Aufgrund des gemeinsamen Datenraums ist es technisch genauso möglich, einen beliebigen Server für den initialen Verbindungsaufbau zu verwenden. Prinzipiell kann dann zwischen zwei Möglichkeiten der Authentisierung unterschieden werden: Die erste Strategie ist die Identität eines Benutzers in dem Server zu überprüfen, der die Benutzerdaten speichert. Diese Methode ist vorzuziehen, falls kein Vertrauensverhältnis zwischen den Servern besteht. Eine Übertragung des Kennworts eines Benutzers kommt dann nicht in Frage.

Falls so ein Vertrauensverhältnis besteht, kann allerdings eine alternative Methode verwendet werden, die eine lokale Prüfung der Identität vornimmt und vertrauliche Benutzerdaten von einem Server in einen anderen überträgt. Diese Vorgehensweise hat Vorteile, falls die Erreichbarkeit von Servern nicht immer vorausgesetzt werden kann. Dann sind die replizierten Daten zur Authentisierung verwendbar und ein Benutzer kann auch in diesem Fall auf den restlichen Verbund zugreifen. Dieser Vorteil kann von Benutzern auch aktiv verwendet werden, falls der Server des Benutzers nicht erreichbar ist. Dann kann sich dieser Benutzer an einen anderen Server des Verbunds wenden, um diesen Ausfall zu kompensieren.

Neben einzelnen Benutzern und Berechtigungen in ACLs sind die Gruppenstrukturen zur Ermittlung von Rechten relevant. Sie bieten eine einfache Möglichkeit, eine ganze Gruppe von Benutzern gleichzeitig zur Mitarbeit in einen Wissensraum einzuladen.

5.2.2 Gruppen

Im Bereich der Gruppenorganisation in CSCW-Systemen existieren unterschiedliche Vorgehensweisen. Während einige Systeme nur flache Strukturen erlauben, bieten hierarchische Gruppen mit Ober- und Untergruppen flexiblere Möglichkeiten. Dadurch wird eine indirekte Mitgliedschaft der Obergruppe von Mitgliedern einer Teilgruppe realisiert und damit die Möglichkeit, entweder die gesamte Gruppe zu berechtigen oder aber die Zugriffsrechte auf Teilmengen dieser Gruppe einzugrenzen. Eine Architektur verteilter Wissensräume muss daher eine hierarchische Struktur von Gruppen berücksichtigen.

Eine Gruppe besteht aus einer Menge von Mitgliedern. Jedes Mitglied ist entweder ein Benutzer oder eine Untergruppe dieser Gruppe. Weiterhin verfügt jede Gruppe über einen gemeinsamen Arbeitsbereich, sodass jedes Mitglied volle Zugriffsrechte über diesen Bereich und alle Unterstrukturen besitzt (vgl. Abschnitt 6.1.6).

Jede Gruppe besteht des Weiteren aus einer Menge von Administratoren. Initial kann der Benutzer, der die Gruppe angelegt hat, als erster Administrator dieser Gruppe eingesetzt werden. Administratoren können weitere Mitglieder in die Gruppe einladen und generell über den Beitritt von weiteren Benutzern entscheiden. Im Bezug auf den gemeinsamen Arbeitsbereich sind Administratoren der Gruppe gleichzeitig Administratoren des Wissensraums und in der Lage Berechtigungen an andere Benutzer weiterzugeben.

Die gesamte Struktur einer Gruppe mit den dazugehörigen Untergruppen kann als eine Gemeinschaft mit gemeinsamen Interessen verstanden werden. Für jede Untergruppe gilt genauso die Enthalten-Relation wie für Benutzer der Gruppe (vgl. auch Abschnitt 3.1.3). Also sind auch Mitglieder von Untergruppen indirekt Mitglieder der Obergruppe. Dies drückt sich auch in der Auflösung von Zugriffsrechten aus: Ein Objekt, welches einer Gruppe Rechte einräumt, berechtigt damit auch direkt die Untergruppen dieser Gruppe. Es können also sowohl Benutzer der Gruppe als auch die Benutzer sämtlicher Untergruppen auf gleiche Weise auf dieses Objekt zugreifen.

Aufgrund der zentralen Bedeutung von Gruppen ist die eindeutige Identifikation von großer Bedeutung. Daher müssen Gruppennamen so vergeben werden, dass es zu keinen Missverständnissen kommen kann (zu den Namen von Benutzern und anderen Gruppen). Den Benutzern werden die Gruppen durch ihre Namen präsentiert und anhand dessen wird entschieden, welcher Gruppe oder welchem Benutzer Berechtigungen einzuräumen sind. Eine Verwechslung kann hier zu Sicherheitsrisiken führen, denn eine andere Gruppe von Benutzern könnte so fälschlicherweise berechtigt werden.

5.2.3 Access Control List

Eine Access Control Liste ist eine Datenstruktur, die bestimmt, welcher Benutzer/welche Gruppe über Rechte an einem Objekt verfügt (Lampson 1974). Die ACL ist somit direkt einem Objekt zugeordnet und kann lokal modifiziert werden. Dazu wird einfach eine Methode aufgerufen, die einem Objekt (Benutzer oder Gruppe) Rechte (Lesen, Schreiben usw.) zuordnet. Der Aufruf dieser Methode unterliegt einer Überprüfung der Berechtigungen, sodass nur Benutzer, die Administratoren für dieses Objekt sind, die ACL verändern können. Initial, also nachdem das Objekt angelegt worden ist, verfügt nur der Benutzer, der das Objekt erzeugt hat , über diese Berechtigungen.

Analog zur Manipulation einer ACL existieren auch Methoden, die berechtigten Benutzer eines Objekts abzufragen. Wenn es zu einer Prüfung der Rechte im Kontext einer Aktion kommt, muss die lokale ACL überprüft werden. Falls der aktive Benutzer über Berechtigungen verfügt, so wird die Aktion zugelassen. Andernfalls wird die gesamte Aktion blockiert und nicht durchgeführt. Als Ergebnis wird der Benutzer über die fehlenden Berechtigungen aufgeklärt.

Wird beispielsweise ein Attribut eines Objekts abgefragt, so entspricht dies einem lesenden Zugriff. Daraus folgt eine Überprüfung der Berechtigungen des aktiven Benutzers — d.h., ob der Benutzer (oder eine Gruppe, in der der Benutzer Mitglied ist) in der ACL des Objekts eingetragen ist. Diese Abfrage berücksichtigt damit den aktiven Benutzer, die Ressource und die Art des Zugriffs. Lampson beschreibt die Teile dieses Modells als Source, Request, Guard und Resource (Lampson et al. 1992). Dabei dient der Guard dazu die Abfrage von Berechtigungen vorzunehmen.

Diese Methode der Überprüfung von Zugriffsrechten bringt offensichtlich administrative Probleme mit sich, denn es ist sehr aufwändig, die Berechtigungen für jeden einzelnen Benutzer korrekt zu setzen. Abhilfe schaffen dabei Zugriffsrechte aus dem Kontext, die besonders im Bezug auf Wissensräume sinnvoll sind (vgl. Abschnitt 3.1.4).

5.2.4 Weitergabe von Rechten/Meta-Rechte

Die Weitergabe von Berechtigungen ist ein weiteres wichtiges Konzept, da dies ein entscheidendes Kriterium zur Selbstadministration darstellt. Jeder Benutzer muss in der Lage sein die Zugriffsrechte eigener Objekt selbst festzulegen und vertrauenswürdige Benutzer dadurch zur Kollaboration einzuladen (vgl. Hampel 2002, S. 123ff.). In dieser Hinsicht gibt es zwei Möglichkeiten:

- Explizit durch das Eintragen eines Benutzers in die ACL eines Objekts.
- Implizit durch die Mitgliedschaft in Gruppen und das Bewegen eines Objekts in einen gemeinsamen Arbeitsbereich (vgl. Abschnitt 6.1.6).

Zur Regelung der Weitergabe von Berechtigungen im Kontext von ACL besteht eine Vorgehensweise in einer Erweiterung jedes einzelnen Rechts um ein Meta-Recht. Dieses gibt an, ob ein Benutzer nicht nur über eine Berechtigung verfügt, sondern auch in der Lage ist diese an Dritte weiterzugeben. Ein gemeinsames Besitzrecht eines Objekts ist damit gegeben und die von Dewan und Shen (1998) aufgestellten Bedingungen eines kollaborativen Systems sind erfüllt.

5.2.5 Rollen

Rollenbasierte Systeme sind in den 90er Jahren entstanden, um Rechte und Sicherheit in großen Organisationen abzubilden (Ferraiolo und Kuhn 1992). Diese Systeme ordnen einem Benutzer Rollen zu, die direkt mit entsprechenden Rechte verbunden sind. Im Bereich von Schulen und Universitäten können z.B. Rollen wie Schüler, Student, Lehrer und Tutor identifiziert werden.

Barkley zeigt, dass einfache rollenbasierte und ACL-basierte Systeme gleichwertig sind (Barkley 1997). Vererbungskonzepte und Meta-Rechte gehen über diese einfachen Systeme hinaus und bieten umfangreiche Möglichkeiten. Sie können sowohl rollenbasierte, als auch ACL-basierte Systeme erweitern.

Rollenbasierte Systeme beschreiben eine organisatorische Sicht, sodass Administratoren den Benutzern in Abhängigkeit der Position innerhalb einer Organisation Berechtigungen zuschreiben (vgl. auch Abschnitt 5.2).

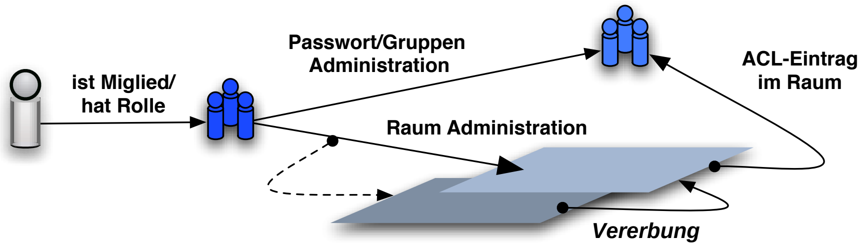


Abbildung 5.1: *Rollen und Rechte*

Abbildung 5.1 zeigt den Ablauf einer Überprüfung von Berechtigungen unter Verwendung der Mitgliedschaft in Gruppen. Auf der einen Seite wird der

Zugriff im Bereich von Räumen über Raumadministratoren geregelt. Auf der anderen Seite steht die Administration einer Gruppe, die von einem Benutzer übernommen werden kann. Beide Zugriffe zeigen die Anwendung eines Rollenkonzepts auf Gruppen. Der Benutzer ist Mitglied in einer Gruppe und erhält damit administrative Rollen auf einen Raum und eine Gruppe. Die Raumverwaltung bezieht sich durch die Vererbung von Rechten auf Raumstrukturen mit enthaltenen Objekten und Räumen.

5.2.6 Verteilte Benutzer

Wie im Szenario der Verschmelzung von Wissensräumen (vgl. Abschnitt 4.1) beschrieben, verfügt jeder Server zunächst über einen eigenen Bestand an Benutzern, sofern das Gesamtsystem nicht direkt als Verbund konzipiert worden ist. Daraus folgt eine initiale Verteilung von Benutzern über den Verbund. Diese können im weiteren Verlauf in einem zentralen Verzeichnisdienst zusammengeführt werden oder weiterhin verteilt vorliegen. Als wichtiges Kriterium ist eine Autonomie der einzelnen Server anzusehen und die Berücksichtigung einer Auflösung des Serververbunds. Aus diesem Grund ist im Idealfall eine verteilte Datenhaltung zu bevorzugen.

Insgesamt lassen sich die folgenden Kriterien für eine Authentisierung von Benutzern identifizieren (vgl. Foster et al. 1998).

- **Single Sign-On**: Ein Benutzer muss sich nur einmal gegenüber dem Verbund authentisieren und ist dann auf allen Servern automatisch angemeldet. Dieses Kriterium folgt technisch aus dem Konzept der Transparenz des Verbunds für die Benutzer.

- **Schutz der Authentisierung**: Daten wie Passwörter und private Schlüssel müssen geschützt werden. Es darf für Dritte nicht möglich sein solche sensiblen Daten auszuspähen. Auch zwischen den Servern dürfen keine Passwörter ausgetauscht werden. Für einen Server reicht es aus, dass ein Benutzer auf dem entfernten Server erfolgreich authentisiert worden ist.

- **Unterschiedliche Sicherheitsmechanismen**: Verschiedene Server können über unterschiedliche Mechanismen verfügen, um Zugriffe zu schützen. Jeder Server und die damit verbundene Organisation können so eigenständige Strategien zur Absicherung des Servers definieren.

- **Einheitliche Infrastruktur**: Im Verbund muss ein einheitliches Verfahren zur Authentisierung existieren. Die Schnittstelle zur Identifizierung von Benutzern muss zwischen den einzelnen Servern des Verbunds fest definiert sein.

- **Sichere Kommunikation**: Die Kommunikation muss abgesichert sein, sodass nicht autorisierte Personen die Verbindung nicht abhören können. Eine Verschlüsselung der verschiedenen Kommunikationskanäle muss gewährleistet sein.

Die Anforderung für Benutzer, sich an jedem Server des Verbunds anzumelden und dort Aktionen durchzuführen, stellt eine wichtige Grundlage dar. Damit die verteilten Benutzer jedoch auf die Materialien zugreifen können, muss es möglich sein Benutzer und Gruppen, unabhängig von ihrem Heimatserver, zu berechtigen.

5.2.7 Verteilte Gruppen

In einem Serververbund muss immer von einer verteilten Gruppenstruktur ausgegangen werden. Dies gilt sowohl für einzelne Gruppen, deren Benutzer über verschiedene Server verteilt sind als auch für die gesamte Gruppenstruktur des Servers. Ausgehend vom Szenario des Aufbaus eines Serververbunds (vgl. Abschnitt 4.1) verfügt jeder Server über einen eigenen Bestand an Gruppen und Benutzern. Dies muss insbesondere gewährleistet sein, damit das Kriterium einen Verbund wieder lösen zu können (vgl. Szenario 4.2) erfüllt werden kann: Löst sich ein einzelner Server ab, muss dieser wieder unabhängig von dem Verbund existieren können.

Der gemeinsame Datenraum der Server bildet sich dann entweder durch einen gegenseitigen Datenaustausch mit Hilfe von direkten Kommunikationskanälen oder durch ein zentrales Verzeichnis, in dem alle Gruppen gespeichert sind. Während eine zentrale Speicherung eine Synchronisation der Daten zwischen Verzeichnis und den unterschiedlichen Servern mit sich bringt, ergeben sich bei einer verteilten Speicherung verschiedene Anforderungen.

- Die Daten müssen generell verfügbar sein. Zur Verbesserung der Verfügbarkeit und Effizienz kann Caching oder Replikation verwendet werden. Dabei muss wiederum eine Synchronisation der Daten Berücksichtigung finden.

- Eine eindeutige Identifizierung von Gruppen muss gewährleistet werden, da die Mitgliedschaft in Gruppen gleichzeitig mit Privilegien verbunden ist. So können alle Mitglieder potentiell Daten im Gruppenarbeitsraum manipulieren.

- Die Mitgliedschaft in Gruppen muss synchronisiert sein zwischen den Servern. Eine gegenseitige Speicherung der Daten ist sinnvoll, um Zugriffe zu optimieren. So führt die Gruppe die Mitglieder als eine Liste von Benutzern und weiteren Gruppen und am Benutzer sind alle Gruppen ebenso als Liste gespeichert.

Bei einem Verbund von Servern, der ein transparentes Bewegen von Benutzern erlaubt, müssen weiterhin serverübergreifende Gruppenstrukturen möglich sein. Ohne Beschränkung müssen Benutzer zur Mitarbeit in einer Gruppe bzw. dem Gruppenarbeitsraum eingeladen werden können. Eine Gruppe dient zur Kooperation zwischen Benutzern verschiedener Server, die sich in einem gemeinsamen Arbeitsraum dieser Gruppe treffen können. Die Mitglieder einer solchen Gruppe sind über den Verbund verteilt.

5.2.8 Verteilte Rechte

Nachdem verschiedene Strategien zur Vergabe und Auflösung von Rechten vorgestellt worden sind, ist die Situation eines einzelnen Servers nun auf eine verteilte Umgebung auszudehnen. Dabei sind die folgenden Anforderungen aus den vorherigen Kapiteln zu berücksichtigen:

- **Gemeinsamer Datenraum**: Objekte sind über Servergrenzen hinweg verfügbar. Daher muss es möglich sein die Zugriffe zu regulieren. Dies bedeutet sowohl eine Zugriffskontrolle von Servern, als auch die Möglichkeit Berechtigungen zu vergeben, indem Rechte an Objekte oder Benutzer gebunden werden, die im gemeinsamen Datenraum adressierbar sein müssen.

- **Autonomie**: Jeder Server wird weitgehend unabhängig vom Verbund administriert und definiert eigene Verfahren zur Überprüfung von Berechtigungen. Für administrative Rollen bedeutet dies, dass diese nur auf bestimmte Bereiche beschränkt sind. Eine übergeordnete Instanz, die den gesamten Verbund administriert, ist nur durch eine jeweilige lokale Zuordnung auf jedem einzelnen Knoten zu realisieren.

- **Dezentrale Datenhaltung**: Anstelle eines zentralen Speicherorts für alle beteiligen Knoten verfügt jeder Knoten über einen eigenen Datenbestand. Aufgrund der Autonomie der Server umfasst dieser Datenbestand auch die Berechtigungen für Zugriffe auf die Objekte in Form von ACLs oder Rollen.

- **Transparenz**: Benutzer bewegen sich unabhängig der Servergrenzen von Raum zu Raum. Dazu ist es notwendig dem Benutzer entsprechende Berechtigungen einzuräumen. Das Berechtigen von entfernten Benutzern darf sich nicht von einer Rechtevergabe für lokale Benutzer unterscheiden.

- **Dynamischer Verbund**: Es existiert kein Vertrauensverhältnis zwischen einzelnen Servern. Die Identität von Benutzern und Gruppen muss durch eindeutige Identifizierung gewährleistet sein.

Die Kriterien einer Autonomie und eines Datenraums sind auch Bestandteil einer Definition des GRID (vgl. Bote-Lorenzo et al. 2003).

Sowohl ACL als auch rollenbasierte Systeme sind in einem Verbund verwendbar (vgl. Abschnitt 5.2). Aufgrund des zweiten Kriteriums von autarken Servern, die unabhängig administriert werden können, müssen beide Arten von Systemen in einem Verbund berücksichtigt werden: Es ist denkbar, dass ein Server über ein rollenbasiertes Rechtesystem verfügt, während andere Server auf Basis von ACL arbeiten.

Bei einer Zuordnung von Berechtigungen in einer verteilten Umgebung ist Folgendes zu beachten:

- Bei der Verwendung einer ACL wird ein entfernter Benutzer lokal in die ACL eines Objekts eingetragen und auf diese Weise berechtigt.

- Rollenbasierte Systeme definieren Rollen für Benutzer und damit, auf welche Ressourcen ein Benutzer zugreifen darf. Rollen können auch auf bestimmte Bereiche eingeschränkt werden.

Die Verwendung von Rollen erfordert weitere Überlegungen, da den Rollen eines anderen Servers auf dem lokalen Server nicht unbedingt Bedeutung zugeordnet werden kann. Es gilt hier zwischen zwei Möglichkeiten zu unterscheiden:

- Einer nicht lokalen Rolle wird Bedeutung zugeschrieben, indem die Eigenschaften der Rolle importiert werden. Dieses Vorgehen widerspricht allerdings der Autonomie der einzelnen Server.

- Für jeden Benutzer werden lediglich die lokalen Rollen verwendet. Dies bedeutet ein Benutzer hat auf jedem Server unterschiedliche Rollen und um einen Benutzer zu berechtigen, müssen diesem zunächst Rollen zugeordnet werden.

Obwohl Rollen und ACL gleichwertige Alternativen darstellen, muss das Konzept der Selbstadministration[2] Berücksichtigung finden. Da rollenbasierte Systeme die Sichtweise von Organisationen besonders gut widerspiegeln, ist eine ACL für selbst administrierte Systeme zu bevorzugen. Diese geht von Objekten[3] aus und ermöglicht den Besitzern eine Freigabe der eigenen Objekte. Weiterhin ist es in verteilten Umgebungen von Vorteil die Berechtigungen direkt den Objekten zuzuschreiben, um damit am Speicherort des Objekts auch den Zugriff prüfen zu können.

Nachdem Objekte, Räume, Benutzer und Gruppen als zentrale Elemente eines CSCW-Systems beschrieben worden sind, stellt sich direkt die Frage, wie und wo Objekte gespeichert und abgerufen werden können. Dazu muss grundsätzlich die Möglichkeit bestehen, Referenzen auf entfernte Objekte zu erzeugen. Der folgende Abschnitt zeigt verschiedene Möglichkeiten einer persistenten Speicherung von Objekten auf.

5.3 Persistenz

Die persistente Speicherung von Daten und Objekten eines CSCW-Systems ist eine notwendige Voraussetzung kooperativen Arbeitens. Sitzungen können so zu jedem beliebigen Zeitpunkt abgebrochen und später wieder fortgesetzt werden. Alle Objekte des Systems sind auch nach einem Neustart des Servers vorhanden, ohne dass sich ihr Zustand verändert hat. Für Wissensräume bedeutet dies eine Speicherung des Raums inklusive des gesamten Inhalts.[4]

Während die Art der Speicherung für den Benutzer zunächst verborgen bleibt, sind technisch doch wesentliche Unterschiede zu verzeichnen. Auf Basis

---

[2] Unter Selbstadministration ist das Konzept der Ausübung aller Medienfunktionen für sämtliche Benutzer zu verstehen (vgl. Hampel 2002, S. 48f.).

[3] Bei raumbasierten Systemen erfolgt die Vergabe von Berechtigungen vornehmlich auf Basis von Räumen, z.B. aufgrund der kontextuellen Berechtigungen (vgl. Abschnitt 3.1.4).

[4] Der Inhalt eines Raums besteht aus den sich dort befindlichen Objekten. Im Prinzip müssen sämtliche abhängige Objekte gespeichert werden. Dies bezeichnet man als Persistence by Reachability (vgl. Moss und Hosking 1996; Crawley und Oudshoorn 1994).

von Objekten sind reflexive Eigenschaften der Programmiersprache von Bedeutung, um eine Persistenz der Daten zu ermöglichen (vgl. Lunney und McCaughey 2003). Nur so ist die so genannte orthogonale Persistenz erreichbar, die alle Daten eines Objekts betrifft und keine speziellen Persistenzmechanismen für bestimmte Objekte erfordert. Ziel ist es, die Daten für den Nutzer transparent auf Persistenzebenen zu speichern. Die zentrale Ebene ist dabei sicherlich eine Datenbank, die Suchfunktionalität anbietet und Funktionen zur Datensicherung bereitstellt — auf dedizierten Servern kann dies immer vorausgesetzt werden.

Im Gegensatz dazu muss bei mobilen Geräten von sehr beschränkten Ressourcen ausgegangen werden (vgl. Meyer 1995). Deshalb stehen herkömmliche Datenbanken nicht unbedingt zur Verfügung. Eine Speicherung der Daten im Dateisystem, die generell immer möglich ist, kann Abhilfe schaffen. Umso mehr Funktionalität in eine solche Speicherung integriert wird (Suchfunktion, Transaktionen), desto stärker nähert sich diese wieder einer Datenbank an. Aus diesem Grund ist es sinnvoller, auf bestehende Software aufzusetzen anstatt diese neu zu implementieren. Steht jedoch keine minimale Datenbank mit einem SQL-Befehlssatz zur Verfügung, so ist eine Speicherung im Dateisystem unumgänglich.

Aufgrund der unterschiedlichen Gegebenheiten ist daher eine Austauschbarkeit der Persistenzebene von Vorteil. Dadurch kann das gleiche System mit unterschiedlichen Konfigurationen auf Servern und Clients eingesetzt werden und passt sich den verschiedenen Gegebenheiten an.

Die Speicherung der Objektdaten kann dann ganz unterschiedlich erfolgen. In den meisten Fällen handelt es sich um Datenbanken, die mehr oder weniger direkt angebunden sind. So kann man zwischen datenbankorientierten Systemen[5] und Datenbanken zur Speicherung von Objekten bzw. objektorientierten Datenbanken unterscheiden. Die erstgenannten Systeme arbeiten direkt mit einer Datenbank unter Verwendung von Tabellen und Datenstrukturen. Im Gegensatz dazu sind bei objektorientierten Systemen die Datenbanken vollständig als Persistenzschicht gekapselt. Dabei können entweder direkt objektorientierte Datenbanken verwendet werden oder die Persistenzschicht realisiert die Speicherung von Objekten mit den Mitteln einer normalen Datenbank (Atkinson et al. 1989). Auch die Verwendung von nicht lokalen Speicherorten ist denkbar (vgl. Lunney und McCaughey 2003). Darüber hinaus erlaubt die Ver-

---

[5] Die Persistenz der Daten wird unter Verwendung einer Query-Sprache (z.B. SQL) für eine Datenbank realisiert (vgl. Crawley und Oudshoorn 1994).

wendung eines Persistenzmanagers auch eine hybride Speicherung der Daten, in Abhängigkeit von bestimmten Kriterien. Im einfachsten Fall handelt es sich dabei um die Klasse eines Objekts, aber auch andere Zusammenhänge sind möglich. Selbst die hybride Speicherung einzelner Objekte ist denkbar, wobei eine konsistente Speicherung der Daten dadurch erschwert wird.

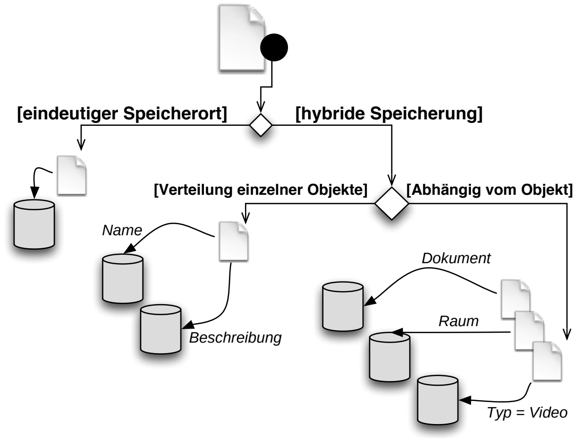


Abbildung 5.2: *Verwendung verschiedener Speicherorte*

Abbildung 5.2 gibt eine Übersicht über die verschiedenen Möglichkeiten der Speicherung von Objekten. Bei einer hybriden Speicherung ist ein Beispiel für ein Kriterium der Typ eines Dokuments. Auf diese Weise können bestimmte Repository angebunden werden und so z.B. ein Videoserver in einem Gesamtsystem Berücksichtigung finden.

Durch die Kapselung der Persistenzebene kann für einen Systementwickler die Erweiterung des Systems im Idealfall ohne Kenntnis der Speicherung von Objektdaten erfolgen.[6] Diese arbeitet transparent im Hintergrund und explizite Datenbankzugriffe müssen nur für spezielle Funktionen, wie z.B. die Suche nach Objekten, durchgeführt werden.

Bei einer Speicherung ist das zugrunde liegende Modell belanglos. Die Speicherungsschicht muss lediglich Daten eines Objekts speichern können. Ein Datum ist dabei die Klasse des Objekts, die später von dem CSCW-System verwendet wird, um wieder eine Instanz der entsprechenden Klasse aus den Daten zu erzeugen. Des Weiteren muss die Speicherung von Verweisen auf

---

[6] Dies stellt auch den Kern der Java Data Objects dar (JSR-12 spezifiziert unter http://jcp.org/aboutJava/communityprocess/maintenance/jsr012/index2.html [Stand: 24.11.2005]).

andere Objekte ermöglicht werden. Dies spiegelt sich in dem Aufbau eines Wissensraums wider, der mit den enthaltenen Objekten in einen persistenten Zustand überführt werden muss. Diese Vorgehensweise entspricht einer transitiven Persistenz (Persistence by Reachability), die von einem Root-Knoten aus alle erreichbaren Daten speichert (vgl. Atkinson und Morrison 1995; Moss und Hosking 1996).

5.3.1 Identifizierung von Objekten

Die Identifizierung von Objekten erfolgt allgemein über eine eindeutige Kennung des Objekts. Bei Datenbanken und vielen weiteren Systemen werden solche Indizes als numerische Werte erzeugt, die bei neuen Daten entsprechend inkrementiert werden. Eine Sicherstellung der Eindeutigkeit innerhalb eines Servers stellt kein Problem dar — schwieriger wird die Situation bei verteilten Serververbünden. Die Identifizierung muss hier global innerhalb des Verbunds möglich sein, da eine eindeutige Unterscheidung zweier Objekte mit gleicher Kennung sonst nicht möglich ist. Aus diesem Grund sieht Hyperwave (Kappe 1995) für jedes Objekt eine eindeutige ID vor. Dabei teilt sich diese in 32 Bit zur Identifizierung des Objekts[7] und 32 Bit zur Identifizierung des Servers auf. Damit ist es möglich neben den lokalen Objekten auch Objekte eines anderen Servers zu referenzieren. Eine solche Aufteilung findet sich auch in dem TODS System (Jin et al. 2002), wobei zusätzlich eine ID für den Namensraum verwendet wird, die so genannte Object Space ID (OSID). Insgesamt werden die Informationen zusammen in eine 128 Bit OID kodiert. Dabei entfallen 48 Bit auf die eigentliche ID des Objekts (in TODS als Object Serial Number bezeichnet) und jeweils 32 Bit auf die ID des Knotens (Server ID) und auf die OSID.

PAST (Rowstron und Druschel 2001b) verwendet einen 160 Bit SHA-1-Hash, der eine Datei eindeutig identifiziert und sich aus dem Dateinamen, einem öffentlichen Schlüssel des Besitzers und einem Zufallswert zusammensetzt.

Andere Ansätze einer eindeutigen Identifizierung sind im Bereich von Peer-to-Peer-Systemen zu finden. OceanStore (Kubiatowicz et al. 2000) verwendet eine global eindeutige Identifizierung (GUID: Globally Unique Identifier), die aus einem Namen und einem Schlüssel des Besitzers zusammengesetzt ist. Die GUID des Objekts wird dann durch einen Hashwert bestimmt.

---

[7] Mit einer 32-Bit-Kennung sind über 4 Milliarden Objekte identifizierbar.

Das JXTA-Projekt nutzt lokale IDs zur Identifikation von Entities. Diese UUIDs sind durch 128 Bit beschriebene Werte, die lediglich im Rahmen eines einzelnen Peers eindeutig sind und denen keine globale Bedeutung zugeschrieben wird. Erst durch Hinzunahme einer weiteren Kennung, die den Namensraum beschreibt (z.B. die Netzwerkadresse eines Peers) kann eine Eindeutigkeit für das gesamte Netzwerk erreicht werden. Das Adressmodell von JXTA verwendet dazu eine Peer-ID, die jedem Peer eine eindeutige Kennung zuweist. Aufgrund von dynamischen Netzwerkadressen (z.B. unter Verwendung von DHCP) wird diese ID zufällig generiert und ist dadurch fest und unveränderlich.

Es zeigt sich also, dass alle diese Systeme wesentliche Gemeinsamkeiten bei der Identifizierung von Objekten haben. Für eine Architektur verteilter Wissensräume bietet sich daher eine ähnliche Vorgehensweise an.

5.3.2 Hybride Datenhaltung

Neben den oben genannten Gründen für eine austauschbare Persistenzschicht kann eine hybride Datenhaltung das System noch flexibler gestalten. So wird eine Trennung von Objekt- und Benutzerdaten möglich und die interne Benutzerverwaltung kann isoliert gegen eine externe Benutzerverwaltung ausgetauscht werden. Eine LDAP-Anbindung ist z.B. eine gängige Schnittstelle vieler Softwaresysteme.

Die Verwendung einer hybriden Speicherung führt dazu, dass die Daten sich nicht nur in einer Persistenzschicht befinden, sondern auf mehrere Schichten verteilt sind. Im Allgemeinen sind viele Speicherorte möglich — die Zahl beschränkt sich z.B. durch die unterschiedlichen Objektklassen, die im System vorliegen. So könnten Räume in einer MySQL-Datenbank gespeichert werden, Dokumente im Dateisystem und Benutzer in einem LDAP-Verzeichnis.

Darüber hinaus ist auch eine Aufteilung der Daten denkbar, sodass etwa nur ein Teil der Benutzerdaten aus dem LDAP-Verzeichnis gelesen werden und die restlichen Daten in einer lokalen Datenbank gespeichert sind. Offensichtlich ergeben sich dadurch Probleme bei der Konsistenz und dem Anlegen von Sicherheitskopien, da es nicht mehr möglich ist eine einzelne Datenbank zu sichern, sondern sich Daten über ganz unterschiedliche Orte verteilen. Selbst ein einzelnes Objekt kann so nicht ohne Weiteres gesichert werden — notwendig wäre Exportfunktionalität aus dem System heraus, die den Zustand eines Objekts sichert.

Bei der Datensicherung müsste ein Systemadministrator in diesem Fall auch über alle Speicherungsorte informiert sein und alle sichern, damit sich die Objekte bei einem Systemabsturz wieder rekonstruieren lassen. So würde eine Speicherung von Attributen in eine Datenbank und Dokumentinhalten in das Dateisystem erfordern, dass sowohl die Datenbank als auch das Dateisystem gesichert werden. Ebenso sind bei einem Datenverlust beide Quellen separat wieder zu rekonstruieren.

Offensichtlich bietet eine solche hybride Datenhaltung sowohl Vorteile als auch Nachteile. Während eine Aufteilung zwischen Benutzerdaten und Objektdaten in bestimmten Situationen sinnvoll sein kann, ist doch eine einheitliche Speicherung an einen einzelnen Ort in der Regel als die bessere Alternative anzusehen.

Falls grundsätzlich eine Aufteilung der Daten erfolgen soll, kommt dem Proxy-Pattern eine entscheidende Bedeutung zu. Der Proxy bietet eine Möglichkeit den Verweis von den eigentlichen Daten zu trennen.

5.3.3 Proxy-Pattern

Das Proxy-Pattern (vgl. Gamma et al. 1995, S. 207f.) beinhaltet verschiedene Vorteile, die für CSCW-Systeme von Bedeutung sind. Es handelt sich dabei um eine Speicherung von Verweisen auf Objekte anstelle einer direkten Speicherung eines Objekts. Ein Proxy ist ein Platzhalter für das eigentliche Objekt, der über einen einfachen Verweis im Speicher hinausgeht und zusätzliche Funktionalität enthalten kann. Besonders im Bereich von verteilten Systemen ist das Proxy-Pattern von Bedeutung, da sich Objekte so indirekt und unabhängig vom Speicherort verwalten lassen (vgl. Coulouris et al. 2002, S. 178-179).

Ein Proxy kann auch als Schnittstelle eines Objekts benutzt werden, da alle Methodenaufrufe zunächst über den Proxy erfolgen. Dadurch sind das Interface und die Implementierung eines Objekts getrennt und es lassen sich damit Methodenaufrufe realisieren, bei denen der Aufruf unabhängig von der Ausführung erfolgt. Darüber hinaus ist ein einfacher konkurrierender Zugriff auf ein Objekt möglich und Methodenaufruf und Ausführung finden in verschiedenen Programmabschnitten statt (vgl. Active-Object-Pattern: Lavender und Schmidt 1997).

Typischerweise werden mit Hilfe eines Proxy die Referenzen eines Objekts gezählt oder Swapping-Mechanismen realisiert. Die eigentlichen Daten des Proxys sind entweder im Speicher geladen oder befinden sich in der Persistenzschicht. Bei einer Referenzierung des Proxys werden die Daten automatisch von der Persistenzebene geladen (vgl. Abbildung 5.3).

Sobald es eine Instanz des Objekts im Speicher gibt, kommuniziert diese selbst mit der Persistenzebene. Dies erlaubt eine automatische Aktualisierung von Programmen: Der Proxy bleibt als Verweis bestehen und die eigentliche Instanz eines Objekts wird verworfen — die Daten befinden sich weiterhin in der Persistenzschicht und der Ausgangszustand ist wieder hergestellt (wie bei einem Neustart des Systems). Danach können alle Instanzen neu aus dem geänderten Programmcode erzeugt werden. Diese Vorgehensweise hat sich in der Praxis als sehr flexibel erwiesen — um eine verteilte Persistenz zu ermöglichen, ist es allerdings von Vorteil den Mechanismus, wie später beschrieben, zu modifizieren (vgl. Abschnitt 6.4.1).

Offensichtlich hat die Benutzung des Proxy-Patterns große Bedeutung für ein verteiltes CSCW-System. Neben den Vorteilen im laufenden Betrieb sind zusätzlich noch Vorteile in der Entwicklung zu nennen. Über Proxys lassen sich so z.B. auch Abläufe regulieren wie eine Kontrolle der Nebenläufigkeit und das Locking von Objekten.

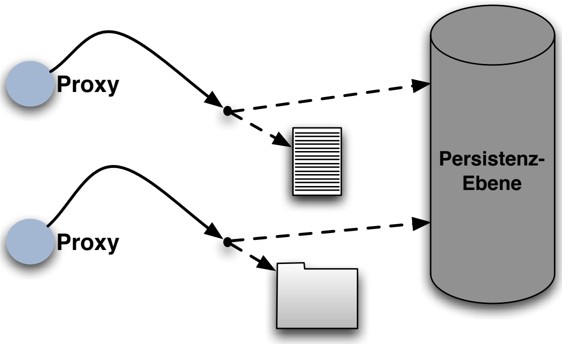


Abbildung 5.3: *Proxy-Pattern*

Nachdem der Proxy als Platzhalter des eigentlichen Objekts beschrieben worden ist, wird im nächsten Abschnitt auf die Aufbewahrung der Objekte eingegangen.

5.3.4 Object Repositorys

Object Repositorys sind Aufbewahrungsorte für Objekte und ähneln objektorientierten Datenbanken. Sie unterscheiden sich durch das Informationsmodell des Repositorys, das die Struktur und Bedeutung der zu speichernden Objekte definiert und von Anwendungen genutzt wird, um die Daten des Repositorys zu interpretieren (vgl. Bernstein 1998). Dies geschieht durch ein Objektmodell und ein Schema für eine unterliegende Datenbank. Obwohl zu diesem Zweck eine objektorientierte Datenbank genutzt werden könnte, werden häufig relationale Datenbanken eingesetzt.

Das Informationsmodell ist Teil der *Repository Engine*, welche das eigentliche Repository kapselt und alle Zugriffe regelt. So wird als Repository oftmals eine Datenbank verwendet, die durch eine zusätzliche Softwareschicht erweitert wird.

Ein Repository verfügt über die folgenden Funktionen (vgl. Bernstein 1998):

- **Objektmanagement**: Speicherung der Daten eines Objekts (z.B. Attribute). Die Daten werden dabei in serialisierter Form abgelegt und können von verschiedener Stelle abgerufen werden. Zu einem Management der Objekte zählen weiterhin eine Identifikation von Objekten und Suchfunktionalität.

- **Erweiterbarkeit**: Das Informationsmodell enthält Typinformationen zu einem Objekt, sodass auf einfache Weise neue Objektklassen hinzugefügt werden können. Das Repository muss in der Lage sein beliebige Objektmodelle abzubilden und diese auch nachträglich zu erweitern.

- **Relationsmanagement**: Abhängigkeiten zwischen Objekten können abgespeichert und wieder aufgelöst werden. Dies beinhaltet z.B. eine Speicherung von Verweisen auf andere Objekte und die Konsistenz solcher Verweise.

- **Benachrichtigungen**: Eine Operation in einem Objekt kann Benachrichtigungen in anderen Objekten auslösen. Es handelt sich also um eine objektorientierte Ereignisverarbeitung.

- **Versionsmanagement**: Es existiert Funktionalität, um verschiedene Versionen eines Objekts zu speichern. Jede Änderung eines Objekts spiegelt sich in einer neuen Version wider. Auch bei einer Modifikation des Inhalts einer Datei kommt es zu einer Änderung der Version.

- **Konfigurationsmanagement**: Verschiedene Objekte können in Konfigurationen gruppiert werden. Das Konfigurationsmanagement beschreibt

> die Administration und Kontrolle einer solchen Konfiguration. Eine Vergabe von Releases in Versionierungssystemen ist z.B. diesem Bereich zuzuordnen. Anwendungen können Konfigurationen verwenden, um die zugehörigen Objekte zu beschreiben.

Es gibt eine Reihe von Projekten, die sich mit der Speicherung und Bereitstellung von Objekten beschäftigen. Dazu zählen z.B. TODS (Tsinghua Object Data Store) (Jin et al. 2002) und THOR (Liskov et al. 1996), die sich mit dem Entwurf einer verteilten Datenbank zur Speicherung von Objekten befassen. Das zuerst genannte Projekt geht von einem statischen Verbund von Servern (Cluster) im LAN (Local Area Network) aus und erzielt dabei eine sehr gute Effizienz. Ziel ist eine Entwicklung von skalierbaren Services, die sich innerhalb desselben Clusters befinden. Jedes Objekt verfügt über eine 128 Bit ID, die sich folgendermaßen aufteilt: Object Space ID (OSID), Class ID (CLID), Node ID (NID) und die lokale ID des Objekts. Ein Bit ist als Flag für temporäre Objekt-IDs gedacht. Die OSID beschreibt den Namensraum des Objekts und die NID zeigt den aktuellen Aufenthaltsort eines Objekts an.

TODS bietet einen transparenten Zugriff innerhalb des Repositorys und überführt Objekte explizit in einen persistenten Zustand.[8] Die Speicherung erfolgt als ein Tupel mit Schlüssel und Wert, wobei der Schlüssel die Objekt-ID darstellt und als Wert die Daten des Objekts übergeben werden. Metaserver verwalten weitere Informationen wie z.B. Klassendefinitionen.

Die Speicherung der Objekte führt gleichzeitig dazu, dass alle abhängigen Objekte ebenfalls persistent werden.[9] Obwohl die Server innerhalb des Clusters als Peer-Server bezeichnet werden und eine verteilte Umgebung geboten wird, handelt es sich um ein geschlossenes System. Eine Umsetzung von Wissensräumen in einer solchen Umgebung bietet eine feste Speicherung von Räumen ohne die Flexibilität einer austauschbaren Persistenzschicht oder hybriden Datenhaltung (vgl. Abschnitt 5.3). Die Aktualisierung von Objekten erfolgt auf Basis von Notifikationen, wobei keine Angaben zu einer Verteilung von Ereignissen auf Applikationsebene gemacht werden. Diese ist allerdings für die Umsetzung eines Handlungsraums unbedingt erforderlich.

Im Bereich der Peer-to-Peer-Systeme stellt das SAV-System (vgl. Cooper et al. 2000) ein Objektarchiv dar. Objekte werden dort im Gegensatz zu den Anforderungen eines kollaborativen Systems nur einmalig abgelegt und später nicht verändert (*write-once*). Die Architektur des Systems gliedert sich in

---

[8] TODS ist in Java implementiert und kompatibel zu JDO (Java Data Objects).

[9] Es handelt sich auch hier um das *Persistence by Reachability*-Verfahren.

verschiedene Komponenten und der Zugriff auf Objekte wird durch mehrere Schichten gekapselt. Dabei werden die Objekte in einem Objektstore gespeichert und in dem darüber liegenden *Reliability Layer* werden Proxys auf die Objekte verwaltet. Gleichzeitig ist diese Schicht für eine Replikation von Objekten zuständig. In der obersten Schicht können zusätzliche Funktionen implementiert werden wie z.B. eine Sicherheitsebene.

Im Bereich der Repositorys sind auch Systeme wie Corba (vgl. Abschnitt 5.7.3) einzuordnen, da eine persistente Speicherung von Objekten hier ebenso zu finden ist. Die Funktionalität der Middleware ist daher auch in den meisten Fällen Bestandteil eines Repositorys.

5.3.5 Datenbanken

Datenbanken spielen eine wichtige Rolle bei der Implementierung von CSCW-Systemen. Sie dienen zur persistenten Speicherung aller wichtigen Daten und bieten gleichzeitig effiziente Zugriffsmöglichkeiten. So wird eine Suche nach bestimmten Attributen eines Objekts direkt durch die Indizierung von Tabellen unterstützt.

Insgesamt kann zwischen zwei Arten der Nutzung einer Datenbank für ein CSCW-System unterschieden werden. Auf der einen Seite steht eine indirekte Nutzung der Datenbank, die durch eine Zwischenschicht gekapselt ist und eine Speicherung von Objekten in der Datenbank realisiert (es handelt sich also um eine objektorientierte Datenbank). Darüber hinaus kann eine Datenbank auf klassische Weise auch direkt für ein CSCW-System genutzt werden. Dazu muss ein entsprechendes Informationsmodell für das CSCW-System aufgestellt werden (Schemadefinitionen und Zugriffskontrolle) (Mariani und Rodden 1991). Aufgrund des dynamischen Charakters eines CSCW-Systems stellt sich die Frage, inwieweit ein statisches Schema ausreichend ist.

In objektorientierten Systemen erfolgt die Art der Speicherung von Objekten in die Datenbank durch unterschiedliche Schemata, die jeweils in einzelnen Tabelle die Attribute eines Objekts speichern. Komplexere Datentypen können in serialisierter Form gespeichert werden (z.B. Arrays) oder direkt im Schema durch entsprechende Abhängigkeiten hinterlegt werden.

Ein besonders wichtiges Konzept einer Datenbank sind Transaktionen, die eine Gruppe von einzelnen Zugriffen kapseln. Nach einer Transaktion ist so immer ein gültiger Zustand erreicht. Bei einer objektorientierten Umgebung erfordern verteilte Zugriffe eine Verwendung von Transaktionen. Dabei sind verschiedene Situationen identifizierbar, die ohne Transaktionen zu ungültigen Zuständen führen:

- Während ein Server bereits die Daten eines Objekts verändert, versucht ein zweiter Server z.B. lesend auf das gleiche Objekt zuzugreifen und liest einen ungültigen Zustand, da das Schreiben zu diesem Zeitpunkt noch nicht abgeschlossen worden ist.

- Mehrere Server versuchen gleichzeitig die Daten eines Objekts zu verändern und überschreiben dabei die Daten gegenseitig. Der resultierende Zustand kann je nach Art der Speicherung eines Objekts z.B. eine Mischung aus den verschiedenen Schreiboperationen sein.

- Abhängigkeiten zwischen verschiedenen Objekten können weitere Situationen erzeugen, die die Notwendigkeit von Transaktionen mit mehreren Objekten aufzeigen. Ein Beispiel ist die beidseitige Enthalten-Beziehung von Containern und Objekten.[10] Während ein Objekt in einen Container bewegt wird, könnte gleichzeitig ein weiterer Schreibzugriff auf denselben Container erfolgen. Das Resultat ist dann unter Umständen eine veränderte Umgebung für das bewegte Objekt, ohne dass sich das Objekt im Inhalt des Containers befindet.

Die obigen Situationen beschreiben die konkreten Probleme einer objektorientierten Umgebung, die sich so ähnlich auch in anderen Systemen identifizieren lassen. Aus diesen Gründen sind Transaktionen unbedingt zu verwenden, falls die Daten von unterschiedlichen Stellen verändert werden. Gleichzeitig müssen die Transaktionen ACID-Eigenschaften erfüllen, um gültige Zustände von Objekten zu gewährleisten. Diese bestehen aus 4 grundlegenden Kriterien (vgl. Gray und Reuter 1993):

- **Atomicity**: Bei Transaktionen handelt es sich um atomare Operationen, sodass alle Operationen einer Transaktion vollständig ausgeführt werden oder aber gar keine Operation durchgeführt wird.

- **Consistency**: Es wird durch die Transaktion eine korrekte Änderung des Zustands vorgenommen, die keine bestehende Bedingung verletzt.

---

[10] In einem Objekt ist ein Verweis auf die Umgebung gespeichert und in dem Container (der Umgebung) ist das Objekt Teil des Inhalts (vgl. Abschnitt 6.1.5).

- **Isolation**: Die Abarbeitung von einzelnen Transaktionen erfolgt isoliert, sodass es zu keinerlei Beeinträchtigungen zwischen verschiedenen Transaktionen kommt.

- **Durability**: Der Zustand nach einer Transaktion muss persistent sein. Durch Verwendung von Sicherungskopien und Sicherung aller Transaktionen kann diese Eigenschaft gewährleistet werden.

Diese Eigenschaften sind auch auf Transaktionen in anderen Softwaresystemen übertragbar und müssen auch für die Objektablage gültig sein. Falls bei einer Transaktion Probleme auftreten, ist es möglich einen *Rollback* durchzuführen und damit alle Änderungen rückgängig zu machen. Es wird damit also ein vorheriger gültiger Zustand wieder hergestellt.

Weiterhin ist zu berücksichtigen, dass ein Fehler während eines schreibenden Zugriffs einen ungültigen Zustand eines Objekts zur Folge haben kann. Um diesem vorzubeugen, existieren Ansätze, die Objekte nur einmalig schreiben und danach jeweils vollständig neue Versionen eines Objekts anlegen (vgl. Wiebe 1986).

5.3.6 Replikation

Ein weiterer wichtiger Aspekt der Datenhaltung ist die Replikation (vgl. Tanenbaum und van Steen 2003, S. 333ff.). Bei verteilten Systemen verbessert dies die Verfügbarkeit der Daten, die nicht nur lokal in einem Server eines Verbunds gespeichert sind, sondern an verschiedenen Stellen gespeichert werden. In einem objektorientierten System bezieht sich eine Replikation offensichtlich auf Objekte als kleinste Einheit, sodass insgesamt durch eine Replikation die Erreichbarkeit und Zugriffszeiten von Objekten verbessert werden. Gleichzeitig schützt eine Replikation allgemein vor einem Verlust von Daten.

Dieser Mechanismus erfordert auf der anderen Seite eine Synchronisierung der verschiedenen Replikate, damit ein aktueller (konsistenter) Zustand aller Objekte gegeben ist. Durch Modifikationen der Objekte werden Ereignisse ausgelöst, die zu einer Anpassung des Zustands der Replikate führen. Die Ereignisse können bei einem Verbund von Servern jedoch in unterschiedlichen Reihenfolgen bei den verschiedenen Servern eintreffen, sodass sich die Zustände der Replikate wieder unterscheiden (vgl. Lamport 1978). Daher kann die Korrektheit der Replikation durch eine Reihe von Kriterien ausgedrückt werden, die die Reihenfolge der Aktionen berücksichtigen (vgl. Amir et al. 1994)[11]:

- **Konsistenz**: Wenn das i-te Ereignis in einem Objekt und Replikat verarbeitet wird, so handelt es sich um ein und dasselbe Ereignis.
- **Vollständigkeit**: Ein Objekt, welches ein Ereignis verarbeitet, hat alle vorherigen Ereignisse bereits bearbeitet.
- **Replikation**: Ein Ereignis, welches auf einem Objekt ausgeführt wird, wird auch auf allen Replikaten ausgelöst.
- **Aktualität**: Sobald eine Verbindung zwischen Objekt und Replikat besteht, wird ein Ereignis auch auf dem Replikat verarbeitet.

Es gibt verschiedene Synchronisationsmechanismen, um die obigen Kriterien zu erfüllen und einen gültigen Zustand der Replikate sicherzustellen. Wie streng eine solche Konsistenz der Daten sein muss, richtet sich dabei nach der Art der Anwendung. Dies kann sich beispielsweise in der Aktualität widerspiegeln. Insgesamt stellt diese Konsistenz den größten Nachteil einer Replikation von Daten dar, da inkonsistente Daten die Funktion des Systems beeinträchtigen können (vgl. Tanenbaum und van Steen 2003, S. 333ff.). Weiterhin sind Konsistenzmodelle teilweise schwierig umzusetzen.

Bei der Time-Warp-Synchronisierung (Steinman et al. 1998) wird bei jeder Änderung ein *Snapshot* durchgeführt und falls ein Ereignis zu einer Modifikation eintrifft, das vor dem letzten Ereignis liegt, so wird ein *Rollback* zu dem letzten gültigen Zeitpunkt vorgenommen. Danach werden alle folgenden Ereignisse wieder ausgeführt, sodass ein gültiger Zustand entsteht.

In einem kollaborativen System mit dem bereits beschriebenen Objekt als Basis (vgl. Abschnitt 5.1) muss grundsätzlich zwischen Objekten und Dokumenten unterschieden werden. Bei den meisten Objekten kann von einer Modifikation von Attributen und zugehörigen Ereignissen ausgegangen werden. Da diese Attribute in der Regel unabhängig voneinander sind, kann die Konsistenz der Daten auf der Basis einzelner Objekte betrachtet werden. So macht es sicherlich nur wenig Sinn den Zustand eines Objekts zurückzusetzen, nur um eine vorherige Änderung des Namens durchzuführen, die schließlich wieder durch die letzte Namensänderung überschrieben wird. In der Regel kann also die letzte Änderung für den aktuellen Zustand des Objekts verwendet werden.

---

[11] Amir et al. beschreiben Aktionen bezogen auf replizierte Datenbestände. Die Kriterien sind jedoch direkt auf Objekte übertragbar.

Alternativ ist es auch denkbar, die Modifikationen mit unterschiedlichen Prioritäten zu versehen, sodass ein Gruppenadministrator über einen stärkeren Einfluss verfügt als ein normales Gruppenmitglied. Letztendlich muss insbesondere sichergestellt sein, dass alle Replikate den gleichen Zustand besitzen. Dazu muss mit einbezogen werden, dass die Ereignisse zu unterschiedlichen Zeitpunkten eintreffen. Da Änderungen von Objekten zur gleichen Zeit sowieso unwahrscheinlich sind, kann die letzte Modifikation eines Attributs verwendet werden. Weiterhin muss der asynchrone Fall mit einbezogen werden, da zu irgendeinem Zeitpunkt ein Objekt geändert werden kann, ohne dass eine Verbindung zu dem Serververbund besteht. Insgesamt müssen die Zustände der Replikate nach der Synchronisierung einheitlich sein.

Die Synchronisierung von Dokumenten gestaltet sich wesentlich komplizierter. Eine analoge Vorgehensweise zu der Modifikation von Attributen ist dennoch möglich. Es wird also immer die letzte Änderung des Inhaltes eines Dokumentes verwendet, wobei andere Änderungen einfach überschrieben würden, sofern die letzte Änderung sich auf einen älteren Zustand bezieht. Eine Abhilfe können hier Locking-Mechanismen schaffen, sodass Dokumente explizit markiert werden und danach für eine gewisse Zeit keine Änderungen mehr möglich sind oder aber das Dokument wieder freigegeben wird. (Das WebDAV-Protokoll stellt diese Möglichkeiten bereit (Goland et al. 1999).)

Offensichtlich ist es jedoch nicht möglich ein Lock auf Replikate zu übertragen, die nicht erreichbar sind. In so einer Situation ist es denkbar, dass ein Benutzer eine Änderung an einem Replikat vornimmt und sich danach wieder mit dem Verbund verbindet. Dann erst wird das Lock, das schon vorher aktiviert worden ist, für den Benutzer sichtbar und die Änderung zurückgewiesen.

Das Globe-System (van Steen et al. 1997) bietet unterschiedliche Replikationsstrategien auf Basis von Objekten. Bei dem System handelt es sich um eine verteilte Middleware, bei der jedes einzelne Objekt einen eigenen Mechanismus verwenden kann. Die Objekte verfügen dadurch über eine gewisse Unabhängigkeit. In dieser Hinsicht ist dieses System auch für unabhängige Server von Belang, sodass jeder Server in dem Netzwerk eigene Verfahren für Objekte definieren kann.

Die Synchronisation der Daten wirft also verschiedene Probleme auf, die nicht ohne Weiteres lösbar sind. Insbesondere die Zusammenführung verschiedener Versionen von Dokumenten gestaltet sich kompliziert und ist im Allgemeinen nicht auflösbar. Allgemein bietet die Replikation von Daten jedoch die einzige Möglichkeit, um Verbindungsabbrüche zu kompensieren. Im Wesentlichen ergibt sich aus einer Zwischenspeicherung von Objekten kein Problem. Erst ein schreibender Zugriff kann zu Inkonsistenzen führen.

Neben der Speicherung und Replikation von Objekten erfolgt der Austausch der Objekte zwischen den einzelnen Servern durch Protokolle. Dies geschieht im Hintergrund zwischen den einzelnen Persistenzschichten, die sich dadurch abgleichen und gegenseitig Zugriffe ermöglichen.

5.3.7 Persistenz in kollaborativen Systemen

Die Persistenzschicht ist eine wichtige Komponente eines kollaborativen Systems. Sie dient bei einer Speicherung der Daten eines objektorientierten Systems als Object Repository und regelt alle Zugriffe auf Objekte, um eine Konsistenz der Daten zu gewährleisten. Dabei sind in den vorherigen Abschnitten folgende Kriterien identifiziert worden:

- **Eindeutige Identifizierung von Objekten**: Objekte verfügen über weltweit eindeutige IDs, die sich aus einzelnen IDs für Knoten und Speicherort zusammensetzen. Proxy-Objekte dienen dabei als Platzhalter für die eigentlichen Objekte.

- **Transparente Speicherung von Objekten**: Ohne Kenntnis über die Persistenz der Daten werden Objekte im Hintergrund gespeichert und geladen.

- **Austauschbare Persistenzschicht**: Die Persistenzschicht kann ausgetauscht werden und sich damit den Voraussetzungen unterschiedlicher Systeme anpassen.

- **Hybride Datenhaltung**: Die Daten können an verschiedenen Stellen, z.B. in Abhängigkeit des Objekttyps, gespeichert werden.

- **Suchfunktionalität**: Die Persistenzschicht verfügt über Suchfunktionen, um ein Auffinden von Objekten zu ermöglichen.

- **Robustheit**: Das Schreiben von Daten muss so erfolgen, dass die Sicherheit der Daten zu jedem Zeitpunkt sichergestellt ist (vgl. ACID-Eigenschaften in Abschnitt 5.3.5).

- **Datensicherung**: Eine einheitliche Datensicherung über alle Objekte eines Servers muss möglich sein, sodass der vollständige Zustand des Servers wieder hergestellt werden kann.

Objekte und deren Persistenz bilden die grundlegenden Eigenschaften eines kollaborativen Systems. Aus diesem Grund bietet jeder Server mindestens diese Dienste an — sie bilden den Kern einer solchen Plattform.

5.4 Kernel

Der Begriff Kernel kommt eigentlich aus dem Bereich der Betriebssysteme. Dabei handelt es sich um den Kern des Systems, der alle grundlegenden Aufgaben übernimmt. Es wird zwischen monolithischen Kerneln, die alle Aufgaben des Betriebssystems direkt enthalten, und modularen Microkerneln (Rashid et al. 1989) unterschieden. Diese zeichnen sich durch eine Konfigurierbarkeit aus, sodass sich die Funktionalität des Kernels nach den Modulen richtet.

Bei einem CSCW-System lässt sich ebenfalls ein Kernel als Kern des Systems identifizieren. Dieser ist bei jedem System allerdings unterschiedlich und lässt sich durch die Komponenten, die nicht unbedingt verwendet werden, beschreiben. Ähnliche Probleme und Anforderungen sind auch aus anderen Bereichen bekannt. Ein Beispiel findet sich in der Entwicklung von grafischen virtuellen Umgebungen. Das Konzept des Microkernels ist auf diese Systeme übertragen worden und findet bei dem Maverik-System (Hubbold et al. 1999) und dem JADE-System Anwendung. Während das Maverik-System aus Gründen der Performanz in C implementiert ist, ist JADE eine Java-Applikation. Daher ist die Plattformunabhängigkeit hier weitgehend gegeben. Auch bei Maverik besteht zumindest eine gewisse Portabilität, da sich der in C programmierte Microkernel gut auf andere Systeme anpassen lässt.

Der Microkernel als Entwurfsmuster ist in (Buschmann et al. 1996, S. 171-191) beschrieben worden. Dabei ist eine Übertragbarkeit auf die oben genannte Klasse von Systemen bereits vorhergesagt worden: Betriebssysteme und grafische Umgebungen haben oft eine lange Lebensdauer und müssen sich daher an die aktuellen Gegebenheiten anpassen (Hardware und Software).

Bamboo (Watsen und Zyda 1998) stellt einen weiteren Microkernel dar. Dieser bietet ebenfalls eine gewisse Plattformunabhängigkeit durch die Verwendung des ACE-Toolkits (Schmidt 1994). Dabei handelt es sich um eine Bibliothek, die für die C++ Programmiersprache entwickelt worden ist und viele Funktionen für Netzwerkzugriffe und Objekte bereitstellt. Aufgrund der Portabilität dieser Bibliothek ist auch der Microkernel gut auf andere Plattformen umzusetzen.

Als Multi-Agent-Plattform bietet das MadKit-Projekt (Gutknecht et al. 1996) einen minimalen Kern, um darauf Agentensysteme zu entwickeln. Dazu sind verschiedene Agenten in Gruppen zusammengefasst und nehmen spezielle Rollen an. Agenten sind lediglich als aktive Kommunikationseinheiten definiert — die Rolle repräsentiert die Funktion eines Agenten. Der Microkernel kontrolliert die Agenten, Gruppen und Rollen und den Austausch von Nachrichten. Über einen Publish-/Subscribe-Mechanismus können so genannte Kernel-Hooks abonniert werden, die es ermöglichen über Ereignisse informiert zu werden. Unterschieden wird zwischen Monitoring-Hooks und Interceptor-Hooks, die lediglich ein einzelner Agent verwenden darf.

Die Collaborative Learning Purpose Library (LCPL) (Caballé et al. 2004) verwendet ebenfalls Ansätze eines Microkernels. So wird auf eine Erweiterbarkeit des Systems und eine Austauschbarkeit von Komponenten besonderer Wert gelegt. Die LCPL soll als Basis für unterschiedliche CSCL-Applikationen dienen. Dabei werden als wichtige Komponenten z.B. Benutzermanagement und Sicherheitsmanagement angeführt. Weiterhin wird auf die Wichtigkeit von Persistenz und Ereignissen hingewiesen.

Neuere Entwicklungen setzen Microkernels im Bereich der Online -Spiele ein. Diese so genannten Massive Multiplayer Online Games (MMOG) (Zyda et al. 1992; Oliveira et al. 1999) müssen in der Lage sein, eine sehr große Anzahl von Spielern zu bewältigen.

Im Bereich von kollaborativen Applikationen stellt der Microkernel ein neues Konzept dar, dem im Bereich der Vernetzung verschiedener Systeme eine wichtige Bedeutung zukommt (Bopp und Hampel 2005c). Die unterliegende Infrastruktur kann durch die Verwendung eines minimalen Kerns vereinheitlicht werden. Die Applikationen können sich dennoch unterscheiden und verschiedene Sichten auf ein gemeinsames Repository bieten.

Für einen Microkernel sind die folgenden Anforderungen von besonderer Bedeutung:

- Das Serversystem wird auf dedizierten Serversystemen mit umfangreichen Ressourcen eingesetzt und ermöglicht die Bildung von Verbünden.
- Es finden auch Systeme mit weniger Ressourcen, wie Desktop-Systeme, bis hin zu mobilen Applikationen Berücksichtigung.

Unter Berücksichtigung dieser Aspekte müssen viele Komponenten austauschbar sein. Speziell im Bereich mobiler Geräte ist mit ganz unterschiedlichen

Konfigurationen und Plattformen zu rechnen. Bei einem PDA oder Mobiltelefon muss z.B. sparsam mit den Ressourcen umgegangen werden und so verfügen solche Geräte über Komponenten, die eher auf besondere Stromsparsamkeit als Leistungsfähigkeit ausgelegt sind. Einen Peer-to-Peer-Verbund mit unterschiedlichen Teilnehmern beschreiben Cugola und Picco (2002). Hier wird zwischen *backbone peers*, *non-backbone peers* und *light peers* unterschieden.

Gleichzeitig ist es nicht unbedingt sinnvoll, mit unterschiedlicher Software für einzelne Geräte zu arbeiten, denn die Wartbarkeit und Entwicklung einer einzelnen Plattform ist wesentlich einfacher, als unterschiedliche Systeme zu pflegen. Es kann insgesamt zwischen monolithischen und modularen Systemen unterschieden werden.

5.4.1 Monolithischer Kernel

Ein monolithischer Kernel verfügt über keine Module und hat dadurch eine starre Funktionalität. Daher sind alle notwendigen Funktionen direkt in dem Kern einprogrammiert und können nur dort direkt erweitert und modifiziert werden. Auf die geänderte Funktionalität kann also nur nach einem Neustart des gesamten Systems zugegriffen werden.

Der Aufbau einer modularen Architektur unterscheidet sich von einem monolithischen System durch die fest definierten Schnittstellen. Während bei einem modularen Aufbau sämtliche Bausteine voneinander getrennt sind und nur Schnittstellen für den Zugriff definieren, sind in einem monolithischen System alle Komponenten zumeist sehr stark untereinander verwoben. Dadurch ist eine Aufteilung in verschiedene Komponenten nicht möglich. Programmteile können so an anderer Stelle nicht wiederverwendet werden.

Monolithische Systeme sind also starre Programme, die einen fest definierten Ablauf haben. Sie integrieren sämtliche Funktionalität in dem System, die dabei so starke Abhängigkeiten aufweisen, dass es nur schwer möglich ist Teile auszutauschen.

5.4.2 Microkernel

In modularen Systemen ist es jederzeit möglich Komponenten hinzuzufügen oder auszutauschen. Jeder Baustein des Systems verfügt über fest definierte Schnittstellen, sodass auch auf bestimmte Komponenten verzichtet werden kann.

Bei einem Microkernel handelt es sich um einen besonders kleinen Kernel, der nur über die grundlegendsten Funktionen verfügt (vgl. Buschmann et al. 1996, S. 171-191). Dazu gehört lediglich ein Modulmanagement, das die Verwaltungsschicht für die Module bildet. Im Laufe der Lebensdauer einer Anwendung werden Module geladen, aktiviert, abgefragt und ausgetauscht. Jedes Modul wird dabei separat konfiguriert und die Gesamtheit der verwendeten Module wird als Konfiguration des Microkernels bezeichnet.

Der Microkernel kann also durch unterschiedliche Konfigurationen in verschiedenen Einsatzkontexten betrieben werden (vgl. Effert und Wunderlich 2004). Dabei kann jeweils auf Module verzichtet werden, die nicht verwendet werden. Daher ist ein Einsatz sowohl auf Geräten, die nur über eingeschränkte Ressourcen verfügen, als auch auf großen Serversystemen denkbar.

Bei dem Microkernel handelt es sich um ein architektonisches Entwurfsmuster, das besondere Relevanz für Systeme mit langer Entwicklungs-/Einsatzzeit besitzt. Aufgrund der Flexibilität des Kernels ist das Gesamtsystem in der Lage sich Änderungen des Umfelds und der Anforderungen anzupassen. Darüber hinaus lassen sich weitere Gründe für die Verwendung eines Microkernels identifizieren (vgl. Bopp und Hampel 2005b):

- **Erweiterbarkeit**: Das modulare Konzept eines Microkernels erlaubt das Einbringen neuer Funktionalität durch ein Hinzufügen zusätzlicher Module.

- **Skalierbarkeit der Verwendung von Ressourcen**: Ein System, das auf verschiedenen Plattformen eingesetzt wird, muss sich den jeweiligen Bedingungen anpassen. Ein mobiles Gerät verfügt über ganz andere Ressourcen (Speicher, Prozessor) als ein dedizierter Server.

- **Gute Wartbarkeit**: Die Wartbarkeit eines komplexen Systems spielt eine wichtige Rolle. Einzelne Module können von den jeweiligen Besitzern gewartet werden. Zwischen den Modulen des Kernels bestehen fest definierte Schnittstellen, sodass Fehlerquellen auf einfache Weise identifizierbar sind.

- **Konfigurierbarkeit**: Jedes einzelne Modul besitzt eine separate Konfiguration und Abhängigkeiten zwischen Modulen können aufgelöst werden. Der Server selbst ist ebenfalls anpassbar und lässt sich mit verschiedenen Modulkonfigurationen betreiben. Dadurch ist es möglich sich unterschiedlichen Bedingungen anzupassen (vgl. Skalierbarkeit der Verwendung von Ressourcen).

- **Alternative Verbindungsinfrastrukturen**: Ein Microkernel legt noch nicht die Art der Kommunikation fest und kann sowohl die Basis eines Client/Server-Systems als auch eines Peer-to-Peer-Systems bilden. Es sind grundsätzlich mehrere Protokollmodule in dem Microkernel nutzbar.

- **Wiederverwertbarkeit**: Module lassen sich in verschiedenen Applikationen betreiben, falls diese über einen identischen Kern verfügen. Dabei müssen die Abhängigkeiten verschiedener Module berücksichtigt werden.

Für kollaborative Systeme lassen sich bestimmte Funktionen identifizieren, die überall benötigt werden. Daraus lässt sich eine Reihe von Modulen ableiten, die in verschiedenen kollaborativen Applikationen verwendet werden können. Insbesondere lassen sich aber auch unterschiedliche Konfigurationen einer bestimmten Applikation mit einer einheitlichen unterliegenden Infrastruktur betreiben.

Die wichtigsten Module für ein kollaboratives System sind (vgl. Bopp und Hampel 2005c):

- Persistenz: Objekte werden in einer Persistenzschicht abgelegt, sodass die Daten nach Beendigung einer Sitzung oder Applikation Bestand haben. Ein typisches Beispiel für eine Ablage von Objekten ist eine Datenbank (vgl. 5.3). Jede Persistenzschicht muss Suchfunktionen zum Auffinden der gespeicherten Objekte unterstützen.

- Management von Benutzern: Die Benutzer des Systems werden durch ein Benutzermodul verwaltet. Die Identifizierung von Benutzern erfolgt im Gegensatz zu Objekten über den Namen des Benutzers.

- Management von Gruppen: Ein Modul für Gruppenstrukturen muss ebenfalls geladen werden. Ein CSCW-System muss die Strukturen einer Organisation wiedergeben können (vgl. Benford et al. 1993). Dazu bieten sich Gruppenstrukturen, mit Unter- und Obergruppen an. Die Vergabe von Rechten oder Rollen kann ebenfalls komfortabler auf Basis von Gruppen erfolgen.

- Protokollmodul: Der Zugang zu einem zentralen Serversystem muss über ein Netzwerkprotokoll erfolgen. Aufgrund der großen Verbreitung des

HTTP-Protokolls verfügen die meisten CSCW-Systeme über einen Web-Zugang. Weitere Protokolle schaffen zusätzliche Zugriffsmöglichkeiten auf die Objekte.

- Sicherheitsmodul: Der Zugriff auf Bereiche von Benutzern und Gruppen muss geschützt werden. Dieser Schutz ist dabei von der Verwendung des Systems abhängig und aus diesem Grund ist die Verwendung von unterschiedlichen Sicherheitsmodulen und einer Austauschbarkeit dieser vorzusehen. So kann das System an die Anforderungen angepasst werden (vgl. auch Caballé et al. 2004).

Nachdem alle Module vom Kernel geladen worden sind, steht die Funktionalität über verschiedene Methoden zur Verfügung. Für eine parallele Beantwortung mehrerer Anfragen müssen der Kern des Systems und die Module eine Nebenläufigkeit der Verarbeitung unterstütztén (vgl. Coulouris et al. 2002, S. 285f.).

5.5 Skalierbarkeit, Nebenläufigkeit und gleichzeitige Zugriffe

Die Skalierbarkeit beschreibt das Verhalten des Systems im Hinblick auf den Ressourcenbedarf bei anwachsender Anzahl von Anfragen. Daraus kann z.B. die Anzahl der Benutzer abgeleitet werden, die mit dem System zur gleichen Zeit arbeiten können. Dies erfolgt unter Einbeziehung der Art der Nutzung des Systems, da eine starke Abhängigkeit zwischen der Häufigkeit von Anfragen und unterschiedlichen Szenarien der Nutzung besteht. Eine Skalierbarkeit ist gegeben, falls sich die Ressourcen an die Anzahl der Benutzer anpassen lassen, sodass das System reibungslos arbeitet. Im Kontext von Client/Server-Applikationen betrifft eine solche Aufwertung des Systems im Wesentlichen die Hardware des Servers.

- Verbesserung des Prozessors: Die einfachste Methode einer Anpassung an eine steigende Anzahl von Anfragen ist die Verarbeitungsgeschwindigkeit auf Seite des Servers zu erhöhen. Je nach aktuellem Stand der Entwicklung der Prozessoren existiert eine Obergrenze, die nur geringe Verbesserungen um einen konstanten Faktor erlaubt.

- Erweiterung des Hauptspeichers: Analog zu der Verwendung eines besseren Prozessors kann der Speicher bis zu einem gewissen Grad erweitert werden.
- Vergrößerung der Festplattenkapazitäten: Der Erweiterung des Festplattenspeichers sind kaum Grenzen gesetzt. Mit externen Lösungen und geeigneten Schnittstellen können große Plattensysteme angebunden und verwendet werden.
- Betrieb mit mehreren Prozessoren: Moderne Serversysteme verfügen in der Regel über mehrere Prozessoren und können in vielen Fällen mit weiteren Prozessoren ausgebaut werden. Eine Einschränkung in der Verwendung mehrerer Prozessoren besteht auf Seite der Software, da diese über mehrere nebenläufige Programmabschnitte verfügen muss, die sich aufteilen lassen, und gleichzeitig konkurrierende Zugriffe abfangen muss. Für die Anzahl der Prozessoren und den Leistungsgewinn existieren ebenfalls Obergrenzen.
- Verteilung auf mehrere Server: Anstelle eines einzelnen Servers ist die Last in einem Verbund von Servern auf mehrere Knoten verteilt. Durch die Hinzunahme weiterer Knoten kann die Leistung hier auf einfache Weise verbessert werden. Die Anforderungen an die Software sind jedoch wesentlich größer, da die einzelnen Knoten miteinander kommunizieren müssen, um ihre Zustände zu synchronisieren.

Eine Skalierbarkeit ist bei Servern, die nur einen einzelnen Verarbeitungsprozess besitzen, im Bezug auf eine Hinzunahme von weiteren Servern oder Prozessoren nicht gegeben. Der Server muss also aus mehreren Prozessen bestehen, um sich effektiv auf verschiedene Prozessoren aufteilen zu lassen. Ein solcher Prozess ist ein nebenläufiger Programmabschnitt, der innerhalb eines anderen Prozesses auf einem Server läuft (Thread). Mit Hilfe von mehreren Threads kann eine Nebenläufigkeit realisiert werden, sodass der Server nicht von Aufgaben blockiert wird und bestimmte Aufgaben im Hintergrund abgearbeitet werden können.

5.5.1 Thread-Management und konkurrierende Zugriffe

Die Abläufe in einem Server müssen genau geregelt werden. Entweder werden Anfragen streng hintereinander abgearbeitet oder es kann zu konkurrierenden

Zugriffen kommen, die abgesichert werden müssen (vgl. Tanenbaum und van Steen 2003, S. 162f.). Die Verwendung von Threads vereinfacht dabei die Entwicklung von Serversystemen, die eine Parallelität ausnutzen (z.B. die Nutzung mehrerer Prozessoren).

Von außen betrachtet melden sich Benutzer an dem Server an und senden Anfragen, die bearbeitet werden müssen. Danach sendet der Server das Ergebnis zurück an den Client. Die einfachste Methode der Bearbeitung ist in der gleichen Reihenfolge, wie die Anfragen den Server erreicht haben. Dies erfordert zwar keine besonderen Überlegungen zur Programmierung und Absicherung von Objekten, hat aber gravierende Nachteile im Verhalten des Serversystems. So kann es leicht passieren, dass der Server zur Bearbeitung einer Anfrage eine längere Zeit braucht. In dieser Zeit ist das System für weitere Anfragen blockiert und andere Benutzer müssen längere Wartezeiten in Kauf nehmen (vgl. Tanenbaum und van Steen 2003, S. 163). Aus diesem Grund ist eine gleichzeitige Bearbeitung mehrerer Anfragen vorzuziehen. Bei einer Applikation mit mehreren Threads ist allerdings wiederum der zusätzliche Aufwand mit einzubeziehen, der nötig ist, um den Kontext zu wechseln.

Weiterhin gibt es Aufgaben, die intern im Server ablaufen ohne direkt von einem Benutzer ausgelöst zu werden. Ein Beispiel ist eine Datensicherung, die eine längere Zeit braucht und damit den Server blockieren würde. Eine Verarbeitung solcher Aufgaben ist im Hintergrund durchführbar, sodass der Server weiterhin uneingeschränkt zur Verfügung steht.

Grundsätzlich kann zwischen den folgenden Arten von Servern unterschieden werden:

- Single-Server, Single-Threaded: Ein einzelner Server verfügt über einen Verarbeitungsprozess, der eintreffende Anfragen nacheinander abarbeitet.
- Single-Server, Multi-Threaded: Ein Server mit mehreren Prozessen, die auf Anfragen warten und diese gleichzeitig bearbeiten.
- Multi-Server, Multi-Threaded: Ein Serververbund bestehend aus unterschiedlichen Servern, die jeweils über mehrere Prozesse zur Bearbeitung von Anfragen verfügen.

Ein Multiserver-System kann nur dann funktionieren, wenn ein einzelner Server im Verbund nicht bei Anfragen blockiert.[12] Dies ist auf das Kommunikations-

---

[12] Auch ohne die Verwendung von Threads ist es möglich eine Parallelität zu erreichen, indem

verhalten und ein mögliches Blockieren eines Servers zurückzuführen.[13] Eine Nachverfolgung der folgenden Kette macht dies sofort deutlich (vgl. Abbildung 5.4): Server A sendet eine Anfrage an Server B. Dieser muss zur Bearbeitung der Anfrage eine Rückfrage an Server A stellen. Da beide Server nun vor der Weiterverarbeitung auf eine Antwort warten müssen, befinden sie sich in einem Deadlock und können keine weiteren Anfragen mehr entgegennehmen. Eine Unterbrechung der Verarbeitung (Timeout) schafft hier Abhilfe und stellt wieder ein funktionstüchtiges System her.

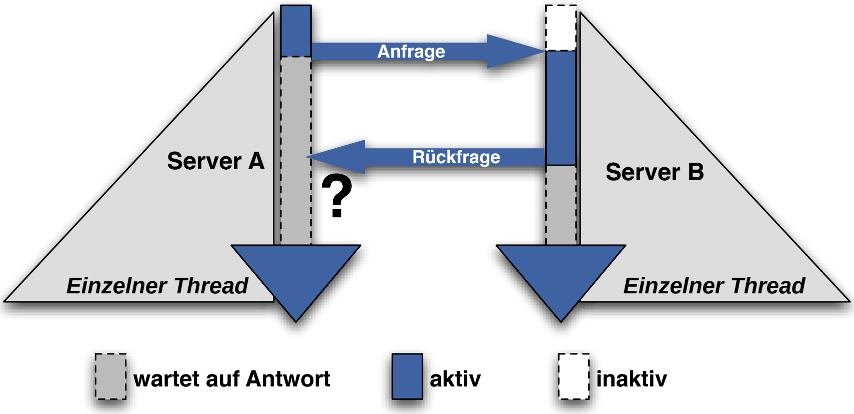


Abbildung 5.4: *Deadlock mit einzelnem Thread*

Daher muss ein Multiserver-System in der Lage sein Anfragen zwischenzeitlich zu unterbrechen, um auf Antworten zu warten und gleichzeitig muss sichergestellt werden, dass das System weiterhin in der Lage ist auf weitere Anfragen zu reagieren. Üblicherweise wird so bei einem Multi-Threaded-System eine Menge von Threads (Pool) bereitgestellt (so genannte Worker-Threads), die auf Anfragen warten und diese beantworten. Eine andere Lösung stellt die Unterstützung von Nebenläufigkeit direkt in einem Objekt dar (vgl. Tanenbaum und van Steen 2003, S. 336)

Die Größe dieses Pools sollte flexibel gehalten werden und je nach Last des Systems skalieren (vgl. Welsh et al. 2001). Es darf niemals passieren, dass alle Anfragen auf Antworten warten und keine Anfrage mehr beantwortet werden

---

der Server als Automat implementiert wird (vgl. Tanenbaum und van Steen 2003, S. 169-171).

[13] Coulouris et al. (2002, S. 615ff.) beschreiben Deadlocks bei Transaktionen in verteilten Systemen.

kann, da dann ein so genannter Deadlock entstanden ist.

Wenn Ereignisse synchronisiert werden müssen, kann auch eine Zugriffskontrolle von Bedeutung sein (Concurrency Control). Sie koordiniert Prozessabläufe, die sich potenziell überschneiden können und parallel ablaufen. Dabei gibt es zwei Arten der Konfliktlösung: Entweder werden Zugriffe so in Transaktionen aufgeteilt, dass es zu keinen Überschneidungen kommen kann, oder sie werden so neu angeordnet, dass sie wie hintereinander ausgeführt erscheinen (Bernstein et al. 1987).

Auf Datenebene der Programmiersprache müssen Transaktionen berücksichtigt sein, dass eine Variable nicht gleichzeitig von zwei Zugriffen manipuliert wird. Es wird zwischen kooperativer und konkurrierender Zugriffskontrolle unterschieden. Dabei werden bei der kooperativen Kontrolle unterschiedliche Zugriffe berücksichtigt und atomare Zugriffe verwendet, um eine Absicherung vorzunehmen. Bei der konkurrierenden Zugriffskontrolle wird die Zugriffsebene vollständig geschachtelt und der Programmierer muss sich nicht darum kümmern (Lee und Anderson 1990).

5.5.2 Staged Event-Driven Architecture

Die Synchronisation und das Verhalten einer Applikation kann durch die Verwendung einer Staged Event-Driven Architecture (SEDA) (vgl. Welsh et al. 2001) verbessert werden. Dabei werden die Daten zwischen verschiedenen Ebenen einer Architektur (Stages) durch Warteschlangen ausgetauscht. Die einzelnen Stages sind dadurch weitgehend unabhängig und agieren autonom im Gesamtsystem.

Die Ressourcen jeder einzelnen Ebene können dynamisch an die Situation angepasst werden. Im Gegensatz dazu arbeiten viele Systeme mit einem Thread pro Anfrage, wobei zum Teil ein Pool von Threads auf höchster Ebene zur Bearbeitung der Anfragen eingesetzt wird (vgl. Tanenbaum und van Steen 2003, S. 588f.). Dadurch können jedoch intern weitere Engpässe bei der Verarbeitung entstehen.

Abbildung 5.5 zeigt eine einzelne Ebene einer SEDA Architektur. Ein Thread-Pool wird bereitgestellt, der Ereignisse aus der Warteschlange (Event Queue) bearbeitet und die Resultate an weitere Ebenen weiterleitet, die ebenso aufgebaut sind. Innerhalb jeder Stage sind ein oder mehrere Threads aktiv, die die jeweiligen Aufgaben bearbeiten. Je nach Last kann die Zahl dieser Threads dynamisch angepasst werden.

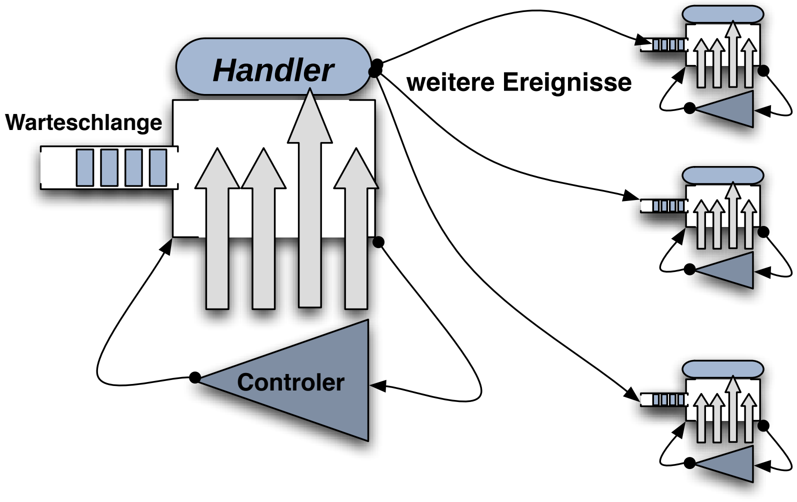


Abbildung 5.5: *Stage einer Staged Event-Driven Architecture*

Eine Ereignisverarbeitung ist Grundlage einer SEDA Architektur und kann helfen, die starre Kommunikation zwischen den einzelnen Komponenten zu verbessern. Dabei können Module auf Ereignisse reagieren, die im System ablaufen.

5.6 Ereignisverarbeitung

Ereignisse sind spezielle Arten von Nachrichten, die in einem System verschickt werden (vgl. Cugola et al. 1998). Sie dienen der Zusammenarbeit verschiedener Komponenten einer ereignisbasierten Architektur, wobei der Sender das Ereignis an einen *Event Dispatcher* übermittelt, der weitere Komponenten über dieses Ereignis informiert. Im Gegensatz zu einer herkömmlichen Nachricht werden die Empfänger eines Ereignisses nicht direkt in der Nachricht festgelegt, sondern über den *Dispatcher* dynamisch ermittelt.

In kollaborativen Systemen existiert eine direkte Verknüpfung zwischen den Aktionen von Benutzern und Ereignissen. In der Regel führt ein Benutzer eine Aktion durch und löst dadurch ein oder mehrere Ereignisse aus. Daraus ergibt sich eine unmittelbare Verknüpfung zu den Awareness-Funktionen des Systems (vgl. Fuchs et al. 1995). Dabei werden Benutzern Informationen über aktuelle und vergangene Ereignisse zur Verfügung gestellt.

Eine Ausnahme der Kopplung zwischen Aktion und Ereignis bilden Aktionen, die zeitverzögert ausgelöst werden.[14] Sowohl bei zeitverzögerten als auch bei unmittelbar ausgeführten Aktionen kann einem Ereignis ein Benutzer zugeordnet werden. Das entsprechende Ereignis kann dann im Kontext dieses Benutzers ausgeführt werden, sodass die Information des aktiven Benutzers zur Verfügung steht. Bei zeitverzögerten Aktionen ist es darüber hinaus von Bedeutung die Aktion im Kontext des korrekten Benutzers ablaufen zu lassen, damit die Berechtigungen für die Ausführung überprüft werden können.

Die Ereignisverarbeitung ist ein zentrales Konzept eines kollaborativen Systems. Zum synchronen Arbeiten ist es essentiell Ereignisse auszutauschen, um über die Aktivitäten anderer Benutzer informiert zu werden (vgl. auch Hauswirth et al. 2005). Im Wesentlichen verursachen Aktionen eines Benutzers Zustandsänderungen von Objekten, die in einer synchronen Sitzung allen Teilnehmern mitgeteilt werden müssen. So werden alle Sichten aktualisiert und das Resultat ist ein einheitlicher Zustand aller Objekte.

Neben diesen direkten Benachrichtigungen spielt auch eine asynchrone Verarbeitung von Ereignissen eine wichtige Rolle. Dazu können Monitorobjekte andere Objekte überwachen und Benutzer über Änderungen z.B. per E-Mail informieren. In diesem Fall geht es nicht darum den Zustand verteilter Objekte zu aktualisieren, sondern lediglich eine entsprechende Überwachungsfunktion zu bieten. Analog zu der synchronen Ereignisverarbeitung stellt die Wahrnehmung von Aktionen anderer Benutzer einen wichtigen Aspekt dar. Gleichzeitig dienen Ereignisse generell dazu die Kommunikation zwischen verschiedenen Komponenten des Systems intuitiv zu regeln.

In dem JADE-System (Oliveira et al. 1999) wird ein Publish-/Subscribe-Mechanismus für die Verarbeitung von Ereignissen verwandt. Dies bedeutet, dass Ereignisse von beliebigen Objekten im System abonniert werden können. Danach wird das Objekt bei jedem Auftreten eines Ereignisses über dieses informiert. Das System verfügt darüber hinaus über zentrale Instanzen, die Ereignisse global verwalten. Diese Instanzen können ebenfalls angesprochen werden, um ein Ereignis global zu abonnieren. Anstatt ein Objekt als Kontext für das Ereignis anzugeben, werden dann alle Ereignisse, unabhängig von dem Objekt, in dem sie auftreten, erfasst.

Zum besseren Verständnis dieser Ereignisverarbeitung kann die Verschiebung eines Objekts dienen. Die lokale Verarbeitung bezieht sich lediglich auf ein einzelnes Objekt, sodass alle Subscriber (Objekte, die Ereignisse abonnie-

---

[14] Ein Beispiel stellen Unix-Cronjobs dar, die zu einem definierten Zeitpunkt oder in definierten Intervallen ablaufen.

ren) über die Bewegung des Objekts informiert werden. Im Gegensatz dazu werden alle Subscriber der globalen Ereignisverarbeitung informiert, falls irgendein Objekt bewegt wird.

Aufgrund des Aufbaus eines Objekts ist es ausreichend eine allgemeine Zustandsänderung mit dem Attribut mitzuteilen, welches sich geändert hat. In (Fuchs et al. 1995) wird zwischen zwei Arten von Ereignissen unterschieden: Aktivitäten und Modifikationen. Dabei werden Modifikationen durch Aktionen von Benutzern ausgelöst und beschreiben die Zustandsänderung. Die Aktivitäten sind die Aktionen der Benutzer und können im Zuge der Verarbeitung verschiedene Modifikationen von Objekten auslösen. Die folgenden Aktivitäten beschreiben typische Aktionen von Benutzern in kollaborativen Systemen (vgl. auch Abschnitt 4.5):

- **Bewegen**: Ein Objekt wird aus einer Umgebung in eine andere bewegt. Entweder ein Benutzer bewegt sich selbst[15] oder er bewegt andere Objekte.

- **Arrangieren**: Objekte werden auf einer Fläche arrangiert (Modifikation der Positionsattribute). Diese Aktion findet in Räumen statt und ordnet den unterschiedlichen Objekten eine Position auf einer zweidimensionalen oder dreidimensionalen Fläche zu.

- **Kommentieren**: Ein Kommentar wird einem Objekt hinzugefügt. Es ist sinnvoll technisch keine Einschränkungen vorzunehmen und die Kommentierbarkeit als grundlegende Eigenschaft jedes Objekts vorzusehen. In objektorientierten Systemen werden im Idealfall Objekte als Kommentare verwendet. Diese Vorgehensweise hat den Vorteil, dass sich die Kommentarobjekte wiederum wie andere Objekte verwenden und dadurch auch mit Berechtigungen versehen lassen. Darüber hinaus ist es möglich weitere Kommentare an diese Objekte anzufügen. Dies spiegelt sich auch in dem Objektmodell des Wissensraums wider (vgl. Abschnitt 3.1.7).

- **Ausführen/Benutzen**: Im Kontext von Objekten handelt es sich um eine Benutzung von Gegenständen in der virtuellen Umgebung. Technisch bezieht sich diese Aktion hauptsächlich auf eine Ausführung von Programmen, Skripten oder Funktionen des kollaborativen Systems.

---

[15] Der Benutzer bewegt das Objekt, durch das er in der virtuellen Welt repräsentiert wird — den Avatar.

- **Kreieren**: Neue Objekte werden von einem Benutzer angelegt. Es kann sich dabei um beliebige Gegenstände handeln, die neu in die virtuelle Umgebung eingefügt werden.
- **Kopieren**: Ein Objekt wird mit Attributen und Inhalt dupliziert. Diese Aktion ist abhängig vom Typ des Objekts: Bei einem Raum werden die enthaltenen Objekte ebenfalls kopiert und bei Dokumenten wird der Inhalt dupliziert.
- **Download**: Ein Dokument wird heruntergeladen. Diese Aktion beschreibt den Zugriff auf Inhalte von Objekten.
- **Upload**: Der Inhalt eines Dokuments wird aktualisiert. Diese Aktion kann sowohl das Kreieren neuer Dokumente beinhalten als auch die Änderung des Inhalts eines bereits bestehenden Dokuments.
- **Sprechen**: Es handelt sich um eine Kommunikation zwischen mehreren Teilnehmern. In einem raumbasierten System findet diese Kommunikation innerhalb eines Raums statt. Ein Benutzer sendet also durch die Aktion „Sprechen“ eine Nachricht an einen Raum, die dann an die anderen Benutzern in derselben Umgebung zugestellt wird.
- **Löschen**: Ein Objekt wird von einem Benutzer gelöscht und das entsprechende Objekt verlässt die Umgebung.

Jede dieser Aktionen führt wie bereits beschrieben zu Modifikationen von Objekten. Entsprechend können viele der Aktionen auch aus der Art einer Modifikation abgeleitet werden. So ist eine Änderung des Umgebungsattributes eines Objekts offensichtlich eine Bewegung des Objekts. Für Aktionen wie „Sprechen“ oder „Ausführen“ ist diese Umsetzung jedoch nicht möglich, da diese nicht unbedingt mit Änderungen von Attributen verbunden sind.

Abbildung 5.6 beschreibt den Zusammenhang zwischen Aktion und Modifikation. Die Zustandsänderung eines Objekts wird dem Benutzer B durch ein Ereignis mitgeteilt, da dieser Benutzer zu den Subscribern des Objekts zählt. Mit Hilfe dieser Benachrichtigung wird eine Kopie des Objekts synchronisiert, die Teil einer Benutzersicht ist (z.B. innerhalb einer Client-Anwendung).

Die Ereignisverarbeitung spielt eine wichtige Rolle bei einer Verteilung von Objekten. Dabei werden Abbilder von Objekten auf Clients oder Servern erzeugt. Es handelt sich entweder um Objekthüllen, die nur den Zugriff auf das eigentliche Objekt kapseln, oder um replizierte Objekte mit allen Daten.

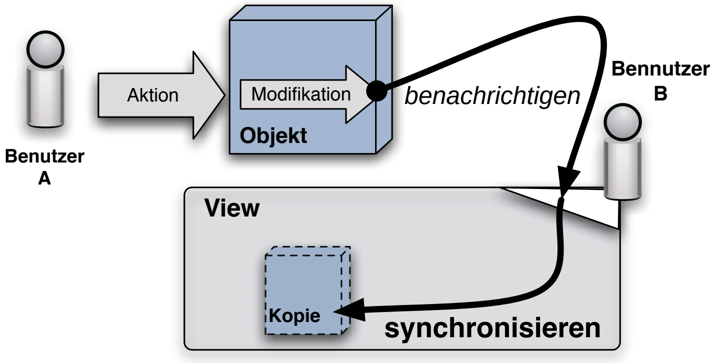


Abbildung 5.6: *Synchronisieren durch Austausch von Ereignissen*

Die Verarbeitung von Ereignissen unterscheidet sich im Einzelserver- und Multiserverbetrieb. Bei einem einzelnen Server und vielen Clients geht es hauptsächlich um eine direkte Benachrichtigung der Teilnehmer über Zustandsänderungen. Im Multiserverbetrieb verhält es sich ähnlich, wobei Ereignisse von einem Server zu einem anderen gesandt werden. Hier liegt ein wesentlicher Unterschied in der Übertragung der Benachrichtigungen, da Ereignisse, die ihr Ziel nicht erreichen, bei Clients lediglich die aktuelle Sitzung betreffen. Im Gegensatz dazu speichern Replikate auf anderen Servern die Objekte persistent, sodass der Zustand dieser Replikate nicht mehr aktuell ist.

In den folgenden Abschnitten werden Ereignisse im Client/Server- und im Multiserver-Betrieb dargestellt.

5.6.1 Client/Server

In einer Client/Server-Umgebung werden die Ereignisse wie im vorangegangenen Abschnitt beschrieben ausgetauscht. Der Server übermittelt Zustandsänderungen von Objekten an die beteiligten Clients, die die entsprechenden Ereignisse abonniert haben. Es handelt sich dabei um Aktualisierungen der Objekte, die bereits von dem Client angefordert worden sind.

Die Clients senden Modifikationen von Objekten direkt an den Server, wobei Attribute in der Regel atomare Daten darstellen. Dies hat zur Folge, dass selbst ungültige Zustände von Objekten die Zustände der Objekte auf Seite des Servers nicht betreffen, da der vorherige Zustand eines Attributs keinen Einfluss hat.

Bei der Ereignisverarbeitung ergeben sich ebenfalls Unterschiede zwischen

Client/Server- und Multiserverbetrieb. Bei einem einzelnen Server kann diese in zwei Phasen ablaufen, sodass sowohl der Beginn als auch das Ende eines Ereignisses bekannt gegeben wird (vgl. Fuchs et al. 1995). Dieser Ablauf ermöglicht eine Blockierung von Aktionen, bevor diese stattfinden. In der zweiten Phase am Ende einer Aktion ist gewährleistet, dass die Benachrichtigungen nur über Ereignisse erfolgen, die auch stattgefunden haben. Diese Vorgehensweise ist bei einem Multiserversystem nicht möglich.

5.6.2 Multiserver

Eine serverübergreifende Verarbeitung von Ereignissen stellt eine große Herausforderung dar. Neben der gegenseitigen Wahrnehmung von Benutzern dient ein Ereignissystem der Kommunikation und Zusammenarbeit einzelner Komponenten des Systems. Diese bilden autonome Einheiten in einer ereignisbasierten Architektur (vgl. Cugola et al. 1998). Im Gegensatz zu geschlossenen Systemen werfen verteilte Umgebungen einige Schwierigkeiten in dem Austausch und der Verarbeitung der Ereignisse auf:

- **Großes Datenaufkommen**: Viele Ereignisse müssen über Netzwerkverbindungen ausgetauscht werden. Daher ist es notwendig die Verteilung der Ereignisse soweit wie möglich zu optimieren.

- **Übermittlung von Ereignissen**: Nur wenn ein anderer Server erreichbar ist, kann ein Ereignis direkt übermittelt werden. Insgesamt muss sichergestellt werden, dass ein Ereignis an andere Server weitergeleitet wird, falls diese darüber informiert werden möchten.

- **Reihenfolge der Ereignisse**: Die Ereignisse treffen in verteilten Umgebungen nicht unbedingt in der gleichen Reihenfolge ein, in der sie ausgelöst worden sind.

Darüber hinaus gilt im Multiserverbetrieb im Prinzip das gleiche wie bei einem einzelnen Server. Die größte Herausforderung liegt in dem Austausch der Ereignisse über die Grenzen der Server. Das Objekt ist in beiden Situationen die Basis für die Verarbeitung. Für Ereignisse, die in nicht lokalen Objekten ablaufen, kann eine lokale Repräsentation geschaffen werden (Proxy oder Replikat des Objekts).

Eine Einschränkung bei der Verarbeitung von Ereignissen muss in Kauf genommen werden. Die Abwicklung von Ereignissen in zwei Phasen ist im Multiserverbetrieb lediglich auf jedem Server isoliert möglich. Blockierende Ereignisse widersprechen dem Konzept einer asynchronen Kommunikation verschiedener verteilter Komponenten. Nur Benachrichtigungen können so effizient an andere Server gesendet werden, denn es wäre zu aufwändig auf die Antwort eines anderen Servers zu warten und damit eine Blockierung der Verarbeitung zu riskieren. Ein Server müsste also zunächst einem anderen Server ein Ereignis mitteilen und dann abwarten, ob der andere Server dieses Ereignis eventuell blockieren möchte.

Ohne die direkte Kenntnis bestimmter Objekte sind zentrale Ereignisse dennoch von Bedeutung. Durch bestimmte Patterns können Ereignisse in Abhängigkeit der Parameter selektiert werden (Carzaniga et al. 2000). Diese Vorgehensweise verbessert die Flexibilität des Ereignissystems weiter und kann auch im Betrieb mit mehreren Servern angewendet werden. Dabei müssen entweder die Ereignisse an alle Knoten weitergeleitet oder das Ereignismuster im Netz verteilt werden.

Im Gegensatz zu einem einzelnen Server spielt die Kommunikation und Verteilung der Ereignisse im Verbund eine wichtige Rolle. Die Vernetzung der Knoten bietet eine Infrastruktur zum Versenden von Nachrichten von einem Knoten an einen anderen. Bei größeren Netzwerken ist das Routing von Bedeutung, um eine Nachricht auf dem schnellsten Weg zu verschicken. Hier bieten Distributed Hash Tables (Rowstron und Druschel 2001a; Stoica et al. 2001) eine Grundlage, um eine Kommunikation effizient zu gestalten. Darauf aufbauend sind für Ereignisse, die unter Umständen mehrere Knoten gleichzeitig erreichen müssen, Multicast-Algorithmen relevant. Scribe (Rowstron et al. 2001) stellt eine Infrastruktur dar, die eine große Zahl von Multicast-Gruppen unterstützt und damit eine viel versprechende Basis zum Austausch von Ereignissen zwischen Knoten bildet (vgl. Knutsson et al. 2004).

Für einen Austausch globaler Ereignisse müssen alle Knoten des Netzwerks benachrichtigt werden. Ein einfaches Verfahren dies zu erreichen ist eine direkte Weiterleitung an alle Nachbarn eines Knotens. Offensichtlich wirft dieses Verfahren ähnliche Probleme auf wie die so genannte Flood-Suche in Gnutella-Netzwerken. Daher sind Algorithmen zur Verbesserung einer Suche in Gnutella-Netzwerken (Charwathe et al. 2003) auch im Bereich eines Austauschs von Ereignissen anwendbar. Ein Unterschied ergibt sich dadurch, dass die Ereignisse nur verteilt werden müssen, aber keine Antwort zurück geschickt wird. Im Gegensatz dazu erfordert das Zurücksenden der Suchergebnisse ebenfalls einen hohen Kommunikationsaufwand.

Nachdem die Ereignisverarbeitung detailliert beschrieben worden ist, soll als nächstes der Austausch von Ereignissen und Daten generell zwischen verschiedenen Servern beschrieben werden. Dies kann z.B. auf Basis von Web Services oder aber allgemeiner mit Hilfe verschiedener Protokolle geschehen.

5.7 Protokolle und Middleware

Protokolle dienen der Kommunikation zwischen Client und Server, zwischen Peers oder generell zwischen verschiedenen Anwendungen. Die einfachste Variante der Kommunikation ist der Remote Procedure Call (RPC) (vgl. Tanenbaum und van Steen 2003, S.90ff.). Dabei wird eine Funktion mit Parametern verschlüsselt und über das Netzwerk zu einem Server verschickt. Dort wird der Funktionsaufruf wieder entschlüsselt und danach die Funktion aufgerufen. Schließlich wird das Resultat verschlüsselt und wieder an den Client zurückgegeben.

Der Begriff Middleware beschreibt eine Kommunikationsschicht für verteilte Applikationen. In objektorientierten Umgebungen werden im Gegensatz zu den RPC entfernte Methoden in Objekten aufgerufen (Remote Method Invocation). Dabei ist die Middleware dafür zuständig den Aufruf inklusive der Parameter zu entschlüsseln und das Ergebnis wieder zu dem Client zurückzuschicken.

Nachfolgend wird eine Übersicht über verschiedene gängige Protokolle und Middleware-Technologien gegeben.

5.7.1 Remote Method Invocation

RMI (Remote Method Invocation)[16] ist die Java-Variante des RPC und stellt die objektorientierte Variante des RPC dar. Aus diesem Grund ist es möglich vollständige Objekte als Parameter anzugeben. Bei klassischem RPC sind lediglich einfache Datentypen möglich.

Die Nachteile von RMI liegen darin, dass es nur zwischen Java-Anwendungen verwendbar ist und die Performanz von Java-RMI durch die Serialisierung kompletter Objekte schlechter ist als bei reinem RPC.

---

[16] Remote Method Invocation wird beschrieben unter: http://java.sun.com/products/jdk/rmi/reference/whitepapers/javarmi.html [Stand: 28.11.2005].

(Nester et al. 1999) zeigen wie die Performanz von Java-RMI signifikant verbessert werden kann. Dabei wird die Serialisierung von Objekten durch ein neues Verfahren ersetzt. Insgesamt ist die Performanz jedoch immer noch deutlich schlechter als bei einfachen RPC-Verfahren.

5.7.2 SOAP

SOAP[17] ist ein anerkannter und viel benutzter Standard im Bereich von Web-Services und setzt als XML-basiertes Protokoll auf einen Transport über das HTTP-Protokoll (vgl. Newcomer 2002, S. 111ff.). Dies bietet verschiedene Vorteile, da das HTTP-Protokoll weit verbreitet ist und sich so Pakete durch Firewalls und über sichere Verbindungen senden lassen. Darüber hinaus ist SOAP ein lesbares Protokoll, sodass es auf einfache Art möglich ist Fehler zu finden und einen Überblick über den Verlauf einer Kommunikation zu bekommen.

SOAP wird als Basis für Dienste im Web verwendet. Diese Webservices bieten einen standardisierten Zugriff auf die bereitgestellten Funktionen. Dies zeigt sich in einer Beschreibung des Services in WSDL (Web Services Description Language) und einem Verzeichnisdienst[18] für vorhandene Dienste (vgl. Newcomer 2002, S. 82ff.).

Gleichzeitig ist die Performanz von SOAP schlechter als bei anderen RMI-Protokollen wie Java-RMI oder Corba (Govindaraju et al. 2000). Aus diesem Grund ist es sinnvoll komplexe Dienste anzubieten, sodass der Aufwand beim Serialisieren der Daten weniger schwer ins Gewicht fällt.

5.7.3 CORBA

CORBA (Common Object Request Broker Architecture)[19] ist eine Middleware und definiert einen offenen Standard. Eine zentrale Idee dabei ist das Interface eines Objekts von der Implementierung zu trennen. Dazu ist eine Interface Definition Language (IDL) Teil von CORBA.

---

[17] Die Spezifikationen von SOAP befinden sich auf der Seite der XML Protocol Working Group unter: http://www.w3.org/2000/xp/Group/ [Stand: 09.01.2006].

[18] Oasis: Universal Description, Discovery and Integration (UDDI) of Web-Services – About UDDI, 2003. Verfügbar unter http://www.uddi.org/about.html [Stand: 11.12.2005]

[19] Die Spezifikation ist verfügbar unter: http://www.omg.org/technology/documents/corba_spec_catalog.htm [Stand: 10.12.2005].

Die Methodenaufrufe werden durch den ORB (Object Request Broker) gesteuert. Wenn ein Client eine Operation auf einem Objekt ausführt, findet der ORB die entsprechende Methode und sendet das Resultat an den Client zurück.

Es handelt sich um eine sehr umfassende, aber auch aufwändig zu erlernende Middleware. CORBA stellt weiterhin ein zentrales Repository dar. Daher kann lediglich ein gemeinsamer Datenraum geboten werden, aber keine vollständige Verteilung der Daten. Im Kontext eines Persistenzmanagers (vgl. Abschnitt 5.3) ist die Anbindung eines Corba-Repositorys als eine Persistenzebene möglich. Dazu muss das Objektmodell des Wissensraums entsprechend abgebildet werden.

5.7.4 JXTA

JXTA (Gong 2001)[20] ist eine Technologie, die eine Grundlage zur Schaffung neuer Peer-to-Peer-Anwendungen bildet. Die zentrale Komponente des Systems sind die Peer-Groups. Eine Gruppe ist dabei eine Menge von Peers und dient als Organisationseinheit im Netzwerk. JXTA besteht aus verschiedenen Protokollen zur Unterstützung der wichtigsten Eigenschaften eines Peer-to-Peer-Netzwerks (Seigneur et al. 2003):

- Auffinden von anderen Peers und deren Services.
- Bekanntgabe der Services eines Peers.
- Datenaustausch mit anderen Peers.
- Routing von Nachrichten zwischen Peers.
- Abfrage des Status eines anderen Peers.
- Gruppierung von Peers in Peer-Gruppen.

Die Protokolle sind dabei unabhängig von einem Transportprotokoll, sodass sowohl TCP/IP, HTTP oder auch Bluetooth genutzt werden können. Nachrichten werden über so genannte Pipes im XML-Format übermittelt.

Damit stellt JXTA eine Peer-to-Peer-Infrastruktur dar, die grundlegende Mechanismen für ein Overlay-Netzwerk zur Verfügung stellt. Im Gegensatz zu DHTs (vgl. Abschnitt 2.3.3) steht bereits eine Plattform zur Entwicklung von Applikationen zur Verfügung.

---

[20] Der Name JXTA ist von *juxtapose* (nebeneinander stellen) abgeleitet.

5.7.5 COAL

Bei COAL (Common Object Access Layer) handelt es sich um eine Middleware, die speziell für das sTeam-System entwickelt worden ist. Es existieren eine Reihe von Implementierungen: für Pike[21], Java und C++. Die Gründe kein standardisiertes Protokoll einzusetzen sind vielschichtig. Zum einen ist eine Middleware wie Corba sehr schwer zu erlernen und beinhaltet einen sehr großen Overhead für Applikationen (Gokhale und Schmidt 1996). Dies ist auf eine Serialisierung der Objekte zurückzuführen.

Auf der anderen Seite existieren für die Programmiersprache Pike weder Implementierungen von Corba noch RMI. Daher ist mit COAL eine Neuentwicklung erfolgt, die sich besonders durch die starke Ereignisorientierung auszeichnet. Als wesentlicher Nachteil gegenüber Corba ist die fehlende Beschreibung der Objekte auf Seite des Clients anzusehen. Gleichzeitig wird dies durch eine umfangreiche Java-API ausgeglichen, die sich insbesondere als für Studenten einfach zu erlernen erwiesen hat und damit für studentische Projekte eine geeignete Basis darstellt.

COAL ist ein objektorientiertes Protokoll, das unmittelbar auf das Ergebnis des Methodenaufrufs wartet. Dadurch ist eine starke Synchronizität gegeben und ein gültiger Zustand von Client und Server ist gewährleistet.

Objekte werden lediglich als Hüllen auf Seite des Clients erzeugt (also ohne Objekt Cache) und es können isoliert Attribute zwischengespeichert werden. Eine Aktualisierung dieser Attribute erfolgt dann auf Basis der Ereignisse. Dadurch ist ein gültiger Zustand der Attribute zu jedem Zeitpunkt gewährleistet.

Die Identifizierung von Objekten erfolgt anhand von IDs, die Teil jedes COAL-Pakets sind, das von einem Client an den Server oder umgekehrt übermittelt wird. Das Protokoll ist in der aktuellen Version so angepasst worden, dass diese IDs nicht wie bisher mit einer festen Länge kodiert, sondern mit einer Längenkodierung vollkommen variabel sind (vgl. Abschnitt 5.3.1).

5.8 Anforderungen

Benutzer und Gruppen bilden die Grundlage eines kollaborativen Systems. In einem raumbasierten System stellen sie außerdem die Basis der Raumstrukturen dar. Von Gruppenarbeitsräumen ausgehend bilden sich Unterstrukturen,

---

[21] Pike ist eine Interpretersprache und verfügbar unter http://pike.ida.liu.se [Stand: 12.12.2005].

die ebenfalls dieser Gruppe zugeordnet werden können.

Auch im Bereich von Berechtigungen bieten diese Strukturen Kontexte, um eine einfache Verwaltung von Rechten zu ermöglichen. Diese unterscheidet sich bei einer Verteilung von Objekten im Netzwerk nicht von einer zentralen Speicherung der Objekte.

Objekte werden durch ihre OID eindeutig identifiziert. Diese Kennung wird durch eine ID für den Server erweitert, sodass sich ein Objekt auf unterschiedlichen Servern befinden kann. Der Ort der Speicherung kann also aus dieser SID ermittelt werden und das Objekt dort schließlich durch die OID abgefragt werden. Mit Hilfe des Proxy-Patterns lassen sich weiterhin indirekte Zugriffe auf ein Objekt realisieren.

In diesem Kapitel sind verschiedene Ansätze der Datenspeicherung vorgestellt worden. Dazu zählen Datenbanken, Dateisysteme und Object Repository. Im Allgemeinen müssen sämtliche Knoten ihre Objekte dem Verbund zur Verfügung stellen. Im Prinzip stellt damit jeder Knoten ein Objekt Repository dar. Diese Anforderung ist in klassischen Systemen nicht umgesetzt. Dazu zählen die verteilten Peer-to-Peer-Dateisysteme, die sich auf eine Verteilung von Dateien beschränken als auch objektorientierte Datenbanken bzw. Object Repositorys wie THOR und TODS (vgl. Abschnitt 5.3.4).

Aus den vorgestellten Architekturansätzen lassen sich eine Vielzahl von Anforderungen ableiten. Diese werden zunächst beschrieben, bevor im nächsten Kapitel auf resultierende Architekturen eingegangen wird. Zur Ermittlung der Anforderungen müssen die aufgestellten Szenarien mit herangezogen werden.

- **Microkernel**: Der Microkernel stellt ein Entwurfsmuster für kollaborative Systeme dar, das eine maximale Unabhängigkeit der einzelnen Komponenten verspricht (vgl. auch Bopp und Hampel 2005b). Der minimale Kern wird durch einen Modul- und Ereignismanager gebildet und lässt sich durch die Hinzunahme von Modulen beliebig erweitern. Zwischen verschiedenen Applikationen lassen sich Module austauschen, die auf dem gleichen Microkernel beruhen. Dies wird durch eine strikte Spezifikation von Abhängigkeiten zwischen den Komponenten erleichtert. Des Weiteren verbessert eine Konfigurierbarkeit der Module die Wiederverwertbarkeit.

- **Persistenz**: Objekte sind persistent und können an verschiedenen Orten gespeichert sein. Dies wird durch einen Persistenzmanager ermöglicht, der als Verwaltungsschicht mehrerer, austauschbarer Persistenzebenen dient. Die Persistenz wird dadurch nicht fest einprogrammiert, sondern sehr flexibel eingebunden. Damit ist sowohl die Verwendung herkömmlicher Datenbanken und Dateisysteme, als auch die Benutzung von Verzeichnisdiensten wie LDAP vorgesehen. Des Weiteren ist die Persistenzschicht des kollaborativen Systems damit auch auf verteilte Persistenzen vorbereitet, sodass die Objekte entfernter Server eingebunden werden können. Dazu muss jedes Objekt über einen gesamten Serververbund eindeutig identifiziert werden können. Dies gelingt durch eine Aufteilung der Objektidentifizierung in lokale ID (OID) und Server-ID (SID). Bei der Server-ID handelt es sich um eine eindeutige Zuordnung zu dem Server, auf dem das Objekt angelegt worden ist. Zusätzlich ist durch den Persistenzmanager eine Namespace-ID (NID) vorgesehen, die die Identifikation der Persistenzebene erlaubt, die das Objekt speichert.

- **Datenraum und Handlungsraum**: Die Persistenzschichten einzelner Server ergeben im Serververbund einen gemeinsamen Datenraum. Daher ist es möglich innerhalb des gesamten Verbunds auf Objekte anderer Server zuzugreifen. Die Einbindung einer Ereignisverarbeitung über alle Knoten des Netzwerks macht aus dem Datenraum einen Handlungsraum und gewährleistet kooperatives Arbeiten im gesamten Verbund.

- **Ereignisverarbeitung**: Änderungen des Zustands eines Objekts werden durch entsprechende Ereignisse innerhalb eines Servers und zwischen Servern ausgetauscht. Dazu wird das betroffene Objekt und das Attribut, das sich verändert hat, übertragen. Die Ereignisverarbeitung stellt den Kern der Kommunikation zwischen einzelnen Komponenten eines Servers dar. Aus diesem Grund wird eine lokale Ereignisverarbeitung und eine globale Verarbeitung ermöglicht. Ereignisse sind also an einem Objekt direkt oder aber allgemein für jedes Auftreten einer bestimmten Aktion zu abonnieren. Die Verarbeitung der Ereignisse ist dabei nicht isoliert für einen Server zu betrachten, sondern verhält sich genauso im Serververbund. Gleichzeitig dienen Ereignisse dazu Benutzer über Zustandsänderungen zu informieren. Damit wird eine gegenseitige Wahrnehmung verschiedener Benutzer ermöglicht.

- **Sicherheit**: Die Sicherheit von Benutzern und ihrer privaten Daten muss zu jedem Zeitpunkt gewährleistet sein. Die einzelnen Aspekte eines sicheren Systems lassen sich wie folgt beschreiben:

  - Authentisierung: Die Identität jedes Benutzern, jeder Gruppe und jedes Knotens eines Verbunds muss zuverlässig ermittelt werden.
  - Verschlüsselung: Der Datenaustausch einzelner Knoten ist verschlüsselt, sodass es Dritten unmöglich ist diesen abzuhören. Dies gilt insbesondere bereits für die Authentisierungsdaten der Benutzer.
  - Autorisierung: Es ist möglich Berechtigungen an einzelne Benutzer oder an Benutzergruppen zu vergeben oder diese zu entfernen. Dadurch wird eine Zugriffskontrolle realisiert. Jeder Knoten des Netzwerks bildet eine autonome Einheit und kann diese Zugriffskontrolle selbst definieren.

- **Autonomität**: Jeder Knoten eines Verbunds wird eigenständig administriert und verfügt über einen eigenen Bestand von Objekten, Benutzern und Gruppen. Weiteren Benutzern des Verbunds können in dem Knoten lokale Rechte zugeordnet werden. Diese Autonomität eines Knotens wird im Microkernel durch eine unterschiedliche Konfiguration von Modulen wiedergegeben. Entscheidend für eine Vernetzung und die Bildung eines Verbunds ist lediglich die unterliegende Infrastruktur.

Aus diesen Anforderungen werden im folgenden Kapitel verschiedene Architekturen erarbeitet. Dabei gilt es zwischen Ansätzen zu unterscheiden, die nur Teilaspekte der Szenarien umsetzen, und einer Architektur, die auf Basis der hier vorgestellten technischen Grundlagen alle Szenarien berücksichtigt.

6 Architekturen verteilter Wissensräume

Bei der Umsetzung einer Architektur verteilter Wissensräume sind eine Reihe unterschiedlicher Kriterien zu erfüllen. In den vorherigen Kapiteln sind aus Szenarien kooperativer Arbeit Anforderungen einer Architektur verteilter Wissensräume ermittelt worden.

Statt einer einzelnen Architektur, die alle Anforderungen umsetzt, werden im Folgenden verschiedene Architekturen vorgestellt, die als Teillösungen dienen können. Während eine umfassende Lösung verteilter Wissensräume eine von Grund auf neue Implementierung erfordert, sind einzelne Komponenten bereits bei bestehenden Systemen austauschbar. Eine grundsätzliche Unterscheidung besteht zwischen einem geteilten Datenraum und einem Handlungsraum als aktiven Bereich zum Austausch von Objekten. Bestehende Systeme sind so erweiterbar, dass diese durch standardisierte Schnittstellen ebenfalls einen gemeinsamen Datenraum bieten oder auf einen solchen zugreifen können. Aufgrund der Autonomie einzelner Server ist zu beachten, dass die Knoten gegenseitig Objekte austauschen, ohne dabei auf einen eigenen Bestand an Objekten zu verzichten.

Alle folgenden Architekturen bilden einen gemeinsamen Datenraum zwischen allen beteiligten Servern. Die verschiedenen Architekturen unterscheiden sich in ihrer Komplexität. Die Auswahl einer geeigneten Umsetzung richtet sich nach den jeweiligen Rahmenbedigungen. So steigert sich der Aufwand einer Implementierung schrittweise zwischen der ersten und letzten vorgestellten Architektur. Insbesondere bei einer Erweiterung bestehender Systeme ist es sinnvoll diese stufenweise auszubauen und auf diese Art immer wieder stabile Entwicklungszustände zu erreichen.

Die erste Architektur verbindet dabei ein CSCW-System mit einem LDAP-Verzeichnis und ermöglicht dadurch Übergänge zwischen allen Servern, die dem Verzeichnis angeschlossen sind. Offensichtlich stellt dieser Ansatz lediglich eine minimale Lösung eines verteilten Datenraums dar und wird den verschiedenen

Szenarien (vgl. Abschnitt 4) nur zum Teil gerecht. Es werden als minimale Datenmenge lediglich Benutzerdaten ausgetauscht, sodass sich Benutzer an beliebigen Servern des Verbunds anmelden können. Damit gelingt auch der Zugriff auf entfernte Daten, wobei es nicht möglich ist die Objekte zwischen Servern zu verschieben und damit von einem entfernten Wissensraum in den Rucksack eines Benutzers aufzunehmen. Ausgehend von dieser ersten Umsetzung eines gemeinsamen Datenraums werden schrittweise weitere Architekturen vorgestellt die zunächst Übergänge zwischen Wissensräumen realisieren (Architektur 2) und schließlich durch einen Austausch von Ereignissen einen Handlungsraum mehrerer Server umsetzen. Dadurch ist es möglich sich transparent in einem Serververbund zu bewegen. Für Benutzer, die sich transparent zwischen den Wissensräumen bewegen können, erscheint der gesamte Verbund als Einheit. Als letzte Architektur wird die Realisierung eines dynamischen Peer-to-Peer Overlay-Netzwerks von Wissensräumen vorgestellt. Dieser setzt im Gegensatz zu den vorherigen Architekturen auf eine vollständig neue Infrastruktur.

Bevor die Architekturen vorgestellt werden, müssen Grundlagen einer objektorientierten Architektur für ein System mit Wissensräumen eingeführt werden. Danach ergeben sich die Anknüpfungspunkte für eine Verteilung von Wissensräumen zwischen mehreren Knoten eines Kollaborationsnetzwerks.

6.1 Grundlagen

Die Grundlagen von verteilten Wissensräumen stellen Objekte, Attribute und Rechte dar (vgl. Hampel 2002, S. 107ff.). Diese sind unabhängig der resultierenden verteilten Architektur zu berücksichtigen, da einige allgemeine Voraussetzungen für raumbasierte kollaborative Systeme existieren.

Mit den Erkenntnissen aus den vorherigen Kapiteln ist eine Microkernel-Architektur mit umfangreicher Ereignisverarbeitung zu bevorzugen. Dadurch ist aufgrund der Eigenschaften des Microkernels bereits eine Plattformunabhängigkeit gegeben. Der Kernel dient dazu eine einheitliche Infrastruktur für alle Knoten eines Verbunds zu schaffen. Des Weiteren lässt sich das System beliebig mit Modulen erweitern und Ereignisse dienen zur Kommunikation der verschiedenen Komponenten.

Da allerdings auch bestehende Systeme Berücksichtigung finden sollen, muss von unterschiedlichen Gegebenheiten ausgegangen werden. Das Resultat sind verschiedene Architekturen, die einige der Kernanforderungen umsetzen. Es ergeben sich dabei einige Bezüge zu den in Kapitel 4 beschriebenen Szenarien.

Zunächst wird die Klasse des Objekts als zentrale Klasse einer raumbasierten Architektur vorgestellt. Das Objektmodell des Wissensraums dient dabei als Grundlage von zentralisierten und verteilten Wissensräumen.

6.1.1 Objekt

Als Basis einer objektorientierten Architektur von Wissensräumen wird eine grundlegende Klasse verwendet, von der sich alle anderen Klassen des Objektmodells (vgl. Abschnitt 3.1.7) ableiten. Dieses Basisobjekt definiert eine einheitliche Schnittstelle für den Zugriff auf jeden Gegenstand in der virtuellen Umgebung. Eine Umsetzung des Objektmodells von Wissensräumen findet sich in dem sTeam-System (vgl. Hampel und Keil-Slawik 2002).

Im Vordergrund stehen bei einem Objekt die Attribute (engl. Property), die abgefragt und verändert werden können. Typische Attribute sind z.B. der Name eines Objekts oder eine Beschreibung. Es handelt sich also um Eigenschaften, die das Objekt und dessen Zustand beschreiben. Jede Zustandsänderung löst korrespondierende Ereignisse aus, die an alle Teilnehmer umverteilt werden, sodass verschiedene Anwendungen immer auf einem aktuellen Stand sind (vgl. Abschnitt 5.6).

Abgeleitet von dieser Klasse sind Klassen, die reale Objekte beschreiben, wie *Dokument* oder *Container*. In Abhängigkeit von der Klasse verfügen diese Objekte über eine Reihe von besondern Attributen oder Eigenschaften. Bei Containern existiert so ein Attribut für den Inhalt des Containers, der durch eine Liste von enthaltenen Objekten beschrieben wird. In Gegensatz dazu besteht der Inhalt eines Dokuments, der gesondert verwaltet und nicht zu den Attributen gezählt wird, aus Daten. Im Kontext eines Persistenzmanagers müssen daher einzelne Persistenzschichten über gesonderte Schnittstellen verfügen, um diese Inhalte von Dateien zu verwalten: Das Lesen und Schreiben von Inhalten erfordert besondere Verfahren, da die Länge von Dateien sehr unterschiedlich sein kann. Grundsätzlich ist eine Speicherung in einzelnen kleineren Paketen sinnvoll, damit der Speicherverbrauch nicht unnötig anwächst (streaming). Mit dieser Aufgabe können spezielle Klassen betraut oder aber für Dateisysteme herkömmliche I/O-Klassen verwendet werden. Darüber hinaus verfügt die Dokument-Klasse über einige spezifische Attribute, die bei anderen Klassen nicht zu finden sind. Zu nennen sind hier Daten der letzten Modifikation des Inhalts und der Typ des Dokuments.

Abbildung 6.1 zeigt den Aufbau eines Objekts mit allen persistenten Da-

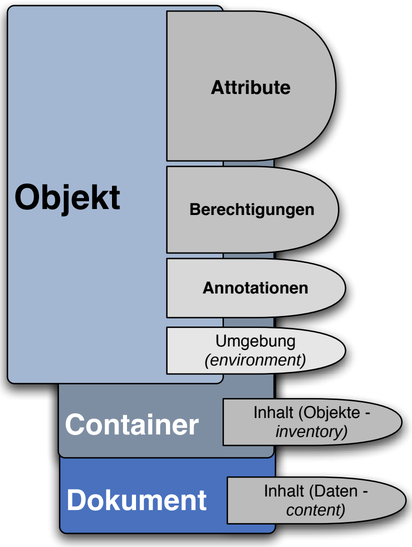


Abbildung 6.1: *Aufbau eines Objekts*

ten. Die Umgebung (*environment*) ist einzeln dargestellt, kann aber ebenfalls als ein Attribut aufgefasst werden. Ebenso können die Berechtigungen eines Objekts als Attribute beschrieben werden. Aufgrund der besonderen Funktion dieser Zugriffsrechte sind diese hier separat dargestellt. Die Änderung von Berechtigungen erfordert darüber hinaus auch spezielle Abfragen. Nicht jeder Benutzer darf andere Benutzer an einem Objekt berechtigen. Eine Kontrolle der Delegation von Berechtigung ist für eine Selbstadministration unbedingt notwendig. Dadurch können auch Benutzer ohne die Hilfe von Administratoren eigene Bereiche kontrollieren und verwalten.

Annotationen sind in der Abbildung dargestellt, um die Annotierbarkeit jedes Objekts aufzuzeigen. Technisch wird dabei genau wie bei den Inhalten von Containern eine Liste von Objekten verwendet, die das Objekt kommentieren. In der Regel wird es sich bei diesen Objekten um Dokumente handeln, die Kommentare in Form von Texten enthalten. Aufgrund dieser allgemeinen Ausrichtung existieren kaum Beschränkungen in der Annotierbarkeit von Objekten.

Die Liste der Klassen mit speziellen Attributen kann beliebig fortgesetzt werden. So zeichnen sich Verweisobjekte durch ein Attribut aus, das das Ziel des Verweises (also ein anderes Objekt) speichert. Neben diesen besonderen Attributen kann eine Reihe von Basisattributen für Objekte in Wissensräumen identifiziert werden, die unabhängig von der Klasse eines Objekts sind:

Attribut	Beschreibung
OID	Die Objekt-ID, die es ermöglicht ein Objekt eindeutig zu identifizieren und effizient zu finden.
Name	Der Name des Objekts. Dieser wird von allen Clients zur Darstellung eines Objekts angezeigt und dient zur Identifikation innerhalb eines Raums.
Anzeigename	Der Name eines Objekts, der dargestellt wird. Dieser kann sich von dem Namen des Objekts unterscheiden und dient nicht zur eindeutigen Identifizierung, sondern dazu den Benutzern einen beschreibenden Namen darzustellen.
Beschreibung	Die Beschreibung eines Objekts ist typischerweise ein kurzer Text, der optional angegeben werden kann. Es wird damit in den meisten Fällen die Funktion eines Objekts umschrieben.
Schlüsselwörter	Zusätzliche Begriffe werden verwendet, um die Suche nach dem Objekt zu erleichtern.
Letzte Änderung	Zeitpunkt der letzten Änderung von Attributen des Objekts.
Besitzer	Der Benutzer, der das Objekt erzeugt hat und automatisch sämtliche Rechte daran besitzt.
Typ	Der Typ eines Dokuments (Mimetype[1]) legt fest, um welche Art von Inhalt es sich handelt.

Ein Name muss generell für jedes Objekt angegeben werden. Die Eindeutigkeit kann dabei innerhalb der Umgebung des Objekts durch das System

---

[1] Der Mimetype eines Objekts hat die Form *globaler/genauer Typ*. Jeder Art von Bild wird der Typ *image* zugeordnet und einem PNG-Bild (Portable Network Graphics) z.B. *image/png*.

garantiert werden, indem der Name in Konfliktsituationen verändert wird. Zur Darstellung wird für die Benutzer der *Anzeigename* verwendet. Die restlichen Attribute müssen nicht unbedingt angegeben werden, um einen gültigen Zustand des Objekts herzustellen. Eine Vorbelegung mit sinnvollen Werten kann hier Abhilfe schaffen.

Das Attribut *Schlüsselwörter* dient zur aktiven Unterstützung der Suche nach Objekten. So ist ein Objekt nicht nur unter dem eigenen Namen auffindbar, sondern die Schlüsselwörter sind ebenfalls gültige Suchbegriffe.

Besonders die Suche nach Dokumenten ist von großer Bedeutung. Da das manuelle Festlegen von Schlüsselbegriffen oft nicht vorgenommen wird, ist eine automatische Suche nach Begriffen hilfreich. Dazu kann bei Dokumenten in dem Inhalt nach Schlüsselwörtern gesucht und damit eine Volltextsuche für Benutzer angeboten werden.

6.1.2 Attribute

Der Zustand eines Objekts wird durch die Attribute beschrieben. Ein Attribut wird durch ein Tupel (Schlüssel, Wert) beschrieben und kann nicht nur einfach Daten wie den Namen und eine Beschreibung enthalten, sondern gestattet die Verwendung beliebiger Datentypen. Es ist also sinnvoll neben Basistypen wie String und Integer auch Float, Array und assoziatives Array vorzusehen. Nur für registrierte Attribute kann der Typ eines Werts überprüft werden und es ist möglich, eine Liste der bekannten Attribute abzufragen. Je nach Klasse des Objekts variieren die Attribute. Ein Dokument verfügt so z.B. über Attribute, die den Zeitpunkt der letzten Änderung anzeigen.

Für die Attribute spielen des Weiteren die Factorys (vgl. Abschnitt 6.1.3 und Abschnitt 7.2) eine wichtige Rolle. Factorys sind besondere Objekte, die die Erzeugung von anderen Objekten kontrollieren. So bieten diese sich dadurch als zentrale Punkte zur Registrierung von Attributen an, da es sich um so genannte Singletons handelt (vgl. Gamma et al. 1995, S. 127ff.).

Eine Kontrolle von Attributen kann nützlich sein, um Fehler zu vermeiden. Ein Austausch von Attributen zwischen verschiedenen Programmen (Clients) erscheint sinnvoll und bei registrierten Attributen ist sowohl der Typ sichergestellt als auch die korrekte Verwendung von Attributen durch entsprechende Beschreibungen. Insbesondere die Sicherstellung von gültigen Kodierungen wie z.B. bei Strings spielt hier eine Rolle. So kann die Factory auch garantieren, dass der Name eines Objekts eine gültige Zeichenfolge[2] enthält.

Neben solchen registrierten Attributen können trotzdem Attribute genutzt werden, um beliebige Daten zu speichern. Falls ein Austausch zwischen unterschiedlichen Anwendungen nicht erwünscht ist, kann ein einzelnes Programm die Werte selbst verwalten und nach Belieben die eigenen Attribute verwenden.[3]

Neben diesem Zusammenhang zwischen Factorys und Attributen zeichnen sich Factorys durch eine enge Verbindung zum Objektmodell des Wissensraums aus (vgl. Abschnitt 3.1.7) und sind für die Erzeugung der einzelnen Klassen des Objektmodells zuständig. Darüber hinaus gilt es eine Korrektheit der Verknüpfungen zwischen den einzelnen Klassen zu garantieren.

6.1.3 Factorys: Erzeugung von Objekten

Das Factory-Pattern (Gamma et al. 1995, S. 107f.) ist ein relevantes Entwurfsmuster zur Erzeugung von Objekten (vgl. auch Abschnitt 7.2). Zu jeder Klasse des Objektmodells (vgl. Abschnitt 3.1.7) kann eine Factory zugeordnet werden, die die Erzeugung von Instanzen der Klasse kontrolliert. Zu einem einzelnen Objekt können darüber hinaus weitere abhängige Objekte angelegt werden, sodass ein gültiger Zustand für die Objekte gegeben ist. Die Factory kann gleichzeitig für eine Registrierung von Attributen genutzt werden und dient dabei als zentrale Verwaltungsinstanz.

Falls Factorys von Objekten abgeleitet sind, besteht außerdem die Möglichkeit das Anlegen von Objekten zu beschränken und mit Berechtigungen zu arbeiten. Nur Benutzer, die über entsprechende Berechtigungen verfügen, dürfen eine Factory ausführen und dadurch ein Objekt anlegen.

Die folgende Tabelle bietet eine Übersicht über die möglichen Funktionen der *Factorys* eines raumbasierten kollaborativen Systems:

---

[2] Eine interne Kodierung im UTF-8 Format ist wichtig, um verschiedene Zeichensätze darzustellen. UTF-8 bedeutet 8 Bit Unicode Transformation Format und ist eine populäre Unicode-Kodierung. Die Kodierung verwendet die normalen ASCII-Zeichen und damit 1 Byte für die ersten 128 Zeichen und 2-4 Byte für weitere Zeichen.

[3] Es ist in diesem Fall sinnvoll einen entsprechenden Namensraum für jede Anwendung zu verwalten. Attribute könnten in der Form *anwendung:attribut* benannt werden.

Factory	Aufgabe
Benutzer	Erzeugt ein Benutzerobjekt und einen Wissensraum als privater Bereich des Benutzers. Darüber hinaus sind verschiedene weitere Objekte denkbar wie ein Container zum Empfang von E-Mails oder Systemnachrichten.
Gruppe	Es wird ein Gruppenobjekt mit einer Liste von Mitgliedern erzeugt. Der aktive Benutzer wird mit administrativen Rechten für diese Gruppe ausgestattet und kann weitere Benutzer einladen. Weiterhin wird zu der Gruppe ein Wissensraum angelegt, der durch eine entsprechende Zuordnung von Berechtigungen für alle Mitglieder der Gruppe zur Verfügung steht.
Raum	Erzeugt einen Raum und weitere abhängige Objekte wie z.B. einen Container, der die Funktion eines „Mülleimers“ übernimmt (Sammeln von zu löschenden Objekten).
Verbindung	Es werden in Abhängigkeit des Typs der Verbindung (unidirektional oder bidirektional) ein oder zwei Objekte erzeugt. Die Verbindungen verweisen dann auf die entsprechenden Zielräume und werden von der *Factory* in diese bewegt.
Dokument	Erzeugt verschiedene Dokumente und setzt einen zugehörigen Typ (Mimetype) für jedes Dokument (z.B. in Abhängigkeit von der Dateierweiterung). Bei Dokumenten kann eine Factory auch im klassischen Kontext des Abstract-Factory-Entwurfsmusters eingesetzt werden und dazu dienen unterschiedliche Klassen in Abhängigkeit einer Dateiendung oder eines Mimetypes zu erzeugen.

Obwohl das Objekt als zentrales Element die Basis des CSCW-Systems bildet, steht aus Sicht des Benutzers der Wissensraum als Umgebung von Benutzern im Vordergrund. Wie bereits beschrieben worden ist, handelt es sich bei Räumen ebenfalls um Objekte (sie verfügen über die gleichen Schnittstellen wie ein Objekt). Zusätzlich verfügt der Raum über weitere Eigenschaften, die diesen zu einem Treffpunkt von Benutzern machen.

6.1.4 Wissensraum

Der Wissensraum bildet die Umgebung und einen Treffpunkt für Benutzer. Im Gegensatz zu Containern können sich in Räumen auch Benutzer aufhalten und es findet eine aktive Gestaltung der Umgebung durch diese Benutzer statt.

Der virtuelle Wissensraum ist ein Treffpunkt für Benutzer zum kontinuierlichen Zusammenarbeiten (vgl. Abschnitt 3.1.2). Durch die Verwendung von Applikationen wie Webbrowser und Whiteboard ergeben sich unterschiedliche Sichten auf einen Raum. Während in einem Browser die Dokumente mit ihren Attributen in einer Listenansicht untereinander dargestellt werden, zeigt ein Whiteboard dieselben Dokumente in einer zweidimensionalen Ansicht. Dazu werden zusätzliche Positionsattribute für die in einem Wissensraum enthaltenen Objekte verwendet (X- und Y-Position). Gleichzeitig können für den Raum Größenangaben für Höhe und Breite vorgesehen werden.

Die Liste der enthaltenen Objekte wird nicht über jede Sicht vollständig dargestellt. In einem Whiteboard werden so Grafikobjekte (z.B. Linien) verwendet, die in einem Webbrowser nicht ohne weiteres sinnvoll dargestellt werden können. Ähnlich verhält es sich mit FTP-Programmen[4] und anderen Werkzeugen zur Übertragung von Dateien, die lediglich Ordner und Dokumente darstellen. Alle Ansichten sind synchronisiert, da es sich um ein und denselben Raum handelt. Wird also beispielsweise ein neues Dokument angelegt, so erscheint dies sofort in einem Whiteboard und wird in der Webschnittstelle bei einer Aktualisierung der Raumdarstellung sichtbar. Der Unterschied ergibt sich durch die Ereignisverarbeitung (vgl. Abschnitt 5.6), die in der Webschnittstelle und anderen asynchronen Clients nicht direkt stattfinden kann.

Technisch gesehen ist der Raum von der Klasse *Container* abgeleitet (vgl. Abschnitt 3.1.7) und unterscheidet sich dabei kaum von einem Container. Das zentrale Merkmal stellt die Liste der enthaltenen Objekte dar, die den Inhalt eines Containers oder Raums bilden. Der Unterschied liegt in der Nutzung des Raums, der im Vergleich zu Ordnern in Dateisystemen Benutzer und Verbin-

---

[4] Das File Transfer Protokoll stellt eine sehr alte Entwicklung dar, die immer noch stark verwendet wird.

dungen zu anderen Räumen beinhalten kann. Räume sind entweder Gruppen- oder Benutzer-Arbeitsräume ohne eigene Umgebung[5]. Eine weitere Möglichkeit besteht darin Räume innerhalb von Räumen anzuordnen. Damit ergibt sich aus der Umgebung und dem Enthalten-Sein von Räumen eine Struktur von Räumen.

6.1.5 Raumstruktur

Jedes Objekt verfügt über ein eindeutiges Attribut für die Umgebung, welches die Position des Objekts im System beschreibt. Damit sind die meisten Objekte Teil eines Raums oder eines Containers (*inventory*), der die Umgebung für das Objekt bildet. Aufgrund der Eindeutigkeit dieser Umgebung kann sich ein Objekt zu einem Zeitpunkt nicht gleichzeitig in zwei Räumen befinden. Dies hat insbesondere Auswirkung auf Avatare, die sich dadurch nicht an zwei Orten gleichzeitig aufhalten können. Es ergeben sich dennoch keine Einschränkungen, da Verweisobjekte genutzt werden können, um einem Objekt indirekt mehrere Umgebungen zuzuordnen.

Die Struktur von Räumen in Räumen ergibt sich über den Inhalt (*inventory*) und die Umgebung (*environment*) (vgl. Abbildung 6.2). Es besteht grundsätzlich die Möglichkeit, Räume innerhalb von Räumen zu platzieren. Ohne ein solches *environment* wird einem Raum kein bestimmter Ort zugeschrieben. Besteht dann eine Zuordnung zu einer Gruppe, so wird bei diesem Raum von einem Gruppenarbeitsraum gesprochen (vgl. Abschnitt 3.1.2). Sie bilden die Wurzel eines Baums, der sich durch die *Ist-enthalten-Relation* (*inventory*) ausprägt. Werden die Querverbindungen zu anderen Räumen hinzugenommen, so bildet sich ein Graph, der aus einzelnen Bäumen besteht, die untereinander verbunden sind. In einem Baum können aufgrund der Struktur wiederum alle Räume derselben Gruppe zugeordnet werden (vgl. Abbildung 3.2).

Im Gegensatz zu einem Container zeichnen sich Räume als Treffpunkte für Benutzer aus. In jedem Raum existieren Dokumente und andere Objekte, mit denen die Benutzer interagieren können. Diese Strukturierung von Räumen ist ein Kriterium der kooperativen Zusammenarbeit der Benutzer. Durch die Ablage von Dokumenten in bestimmten Räumen und durch die Positionierung der Objekte werden semantische Zusammenhänge abgebildet.

Darüber hinaus sind synchrone Formen der Zusammenarbeit möglich. Es ist

---

[5] Diese Räume werden durch die entsprechende Factory des Benutzers oder der Gruppe angelegt und sind durch ein Attribut zugeordnet.

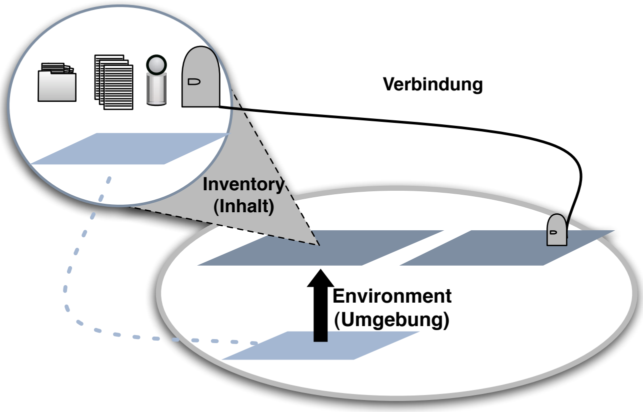


Abbildung 6.2: *Ist-enthalten-Relation*

sinnvoll solche Aktivitäten innerhalb eines Raums zu regulieren, sodass nicht jeder Benutzer beliebige Manipulationen vornehmen kann.

6.1.6 Kontextuelle Rechte

Neben den Berechtigungen eines Objekts ist es sinnvoll den Kontext zu berücksichtigen, der durch den Aufenthaltsort des Objekts[6] bestimmt wird (vgl. Hampel 2002, S. 137ff.). So gelten die Rechte der Umgebung (*environment*) auch für ein enthaltenes Objekt. Dadurch reicht es aus, Benutzern Berechtigungen für Räume zuzuordnen und auf dieser Basis eine starke Vereinfachung zu erreichen. So beinhaltet das Verschieben eines Objekts immer auch eine veränderte Situation der Berechtigungen. Während Benutzer, die direkt Zugriffsrechte an einem Objekt besitzen, weiterhin wie gewohnt zugreifen können, werden darüber hinaus auch Benutzer und Gruppen über die Umgebung berechtigt. Diese Vorgehensweise weist Räumen eine zentrale Bedeutung für die Vergabe von Berechtigungen zu und führt die Rolle des Raumadministrators für Benutzer zur Verwaltung von Teilbereichen ein. An dieser Stelle wird keine Unterscheidung zwischen rollenbasierten[7] und ACL-basierten[8] Systemen vorgenommen, da in beiden Fällen verschiedene Rollen identifiziert werden können:

---

[6] Die Umgebung eines Objekts ist eindeutig und damit ist der Kontext ebenso eindeutig.

[7] Rollenbasierte Systeme bilden Strukturen von Organisationen ab (vgl. Abschnitt 5.2.5).

[8] Eine ACL ist eine Liste, die mehrere Einträge mit Zuordnungen von Benutzer oder Gruppe mit einem Zugriffsrecht enthält und direkt einem Objekt zugeordnet ist (vgl. Abschnitt 5.2.3).

Bei einer Verwendung von ACL ergibt sich die Rolle eines Raumadministrators implizit durch die Vergabe von Administrationsrechten an einem Raum. Bei rollenbasierten Systemen wird die Rolle eines Administrators systemweit verwendet oder auf bestimmte Bereiche oder Räume beschränkt.

Im Kontext von Gruppenarbeitsräumen sind die Vorteile leicht ersichtlich: Benutzer bringen ihre Dokumente in den gemeinsamen Raum und legen zunächst keine Berechtigungen fest, da die Dokumente automatisch von anderen Mitgliedern der Gruppe benutzt werden können.

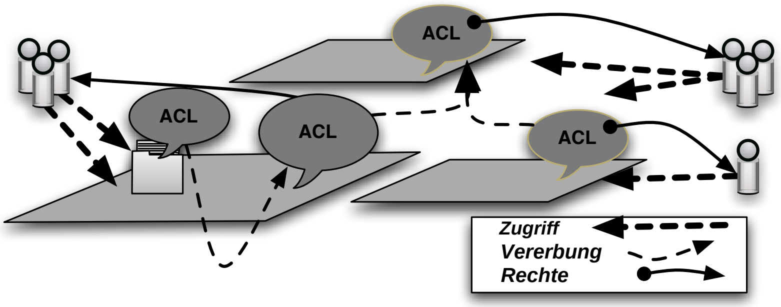


Abbildung 6.3: *Access Control List*

Abbildung 6.3 zeigt ACLs verschiedener Objekte. Die dargestellten Räume und Dokumente verfügen jeweils über eine eigene Liste der berechtigten Benutzer und Gruppen. Zwischen den einzelnen Räumen ist die Vererbung der Rechte dargestellt und damit die an Räume gebundenen kontextuellen Rechte.

Für rollenbasierte Berechtigungen kann dieses System so angepasst werden, dass die Rollen mit entsprechenden Kontexten versehen werden. Für solche Kontexte bieten sich Gruppen oder Räume an, die jeweils mit bestimmten Strukturen im Zusammenhang stehen. So enthalten Räume verschiedene Arten von Objekten und sogar wiederum Räume. In diesem Fall kann eine Rolle auf diese gesamte Struktur von Räumen bezogen werden, sodass die mit der Rolle verbundenen Berechtigungen für diese Struktur wirken. Es handelt sich somit um eine rekursive Vererbung von Berechtigungen. Analog dazu sind auch Gruppen als Kontexte zuzuordnen. Hier steht beispielsweise die Liste der Mitglieder einer Gruppe als Bezug der Rolle zur Verfügung.

Das oben genannte Verhalten kann in einigen Situationen auch unerwünscht sein. Um ein Höchstmaß an Flexibilität bereitzustellen, ist es notwendig die kontextuelle Rechtevererbung in bestimmten Situationen auszuschalten. Ohne eine Vererbung von Rechten sind an einem Objekt lediglich die explizit vergebenen Berechtigungen gültig. Wenn bestimmte Berechtigungen zu einem übergeordneten Objekt nicht verwendet werden sollen, so kann entweder die Vererbung abgeschaltet werden oder es werden so genannte Negativ-Berechtigungen vergeben. Diese speziellen Zugriffsrechte verhindern eine Vererbung der entsprechenden Berechtigungen, indem die nicht erwünschten Rechte ausgeschaltet werden.

Technisch gesehen kann das Konzept kontextueller Rechte so umgesetzt werden, dass ein Verweis auf ein Zielobjekt gespeichert wird, der die aktive Vererbung angibt. Dadurch ist es nicht nur möglich den jeweiligen Kontext anzugeben, sondern auf beliebige Objekt zu verweisen. Ein Verweis ohne Zielobjekt stellt entsprechend eine ausgeschaltete Vererbung dar.

Es stellt sich die Frage, wer, neben dem Besitzer eines Objekts, in der Lage ist Rechte von Objekten zu vergeben. Daher ist die Delegation von Rechten von Bedeutung, um einen größeren Kreis von Benutzern mit der Administration eines Raums zu betrauen.

6.1.7 Überprüfung von Berechtigungen

Bei der Überprüfung der Berechtigungen für einen Zugriff auf ein Objekt wird geprüft, ob ein Benutzer in der ACL des Objekts eingetragen ist und dort berechtigt wird. Bei jedem einzelnen Objekt werden dabei erst der Benutzer und dann die Gruppen des Benutzers überprüft. Wenn der Benutzer nicht direkt über eine Berechtigung verfügt, kann es dennoch sein, dass eine Gruppe des Benutzers berechtigt worden ist. In der Praxis kommt dieser Fall sogar wesentlich häufiger vor, da es komfortabler ist auf Basis von Gruppen zu arbeiten. Damit kann gleich einer größeren Menge von Personen direkt Zugriff eingeräumt werden anstatt jeden Benutzer einzeln zu berechtigen.

Die Prüfung der Zugriffsrechte erfolgt zunächst am betroffenen Objekt und kann dann auf den Kontext (die Umgebung, vgl. Abschnitt 3.1.4) erweitert werden. Für ein beliebiges Objekt innerhalb eines Raums gelten dann nicht nur die lokalen Berechtigungen, sondern die Umgebung wird ebenfalls mit einbezogen. Dadurch tritt der Wissensraum, als Treffpunkt für Benutzer zum gemeinsamen Strukturieren von Dokumenten, wiederum in den Vordergrund.

Berechtigungen sind direkt mit Ereignissen gekoppelt, die im Zuge von Aktionen ausgelöst werden. Dabei greift ein Benutzer auf das System zu und führt bestimmte Aktionen durch. Mit diesen Aktionen (z.B. lesen und schreiben) können dann die entsprechenden Berechtigungen überprüft werden (vgl. Abschnitt 5.6 für eine Liste typischer Aktionen).

6.1.8 Ereignisverarbeitung

Für die Ereignisverarbeitung bietet sich ein Publish-/Subscribe-Mechanismus an: Es ist für ein Objekt möglich, Ereignisse zu abonnieren und bei einem Auftreten des Ereignisses durch den Aufruf einer Methode im Subscriber-Objekt benachrichtigt zu werden (vgl. Abschnitt 5.6). In Fuchs et al. (1995) wird den Ereignissen ein *Interest Context* zugeordnet, der durch Objekte, Ereignisse und Benutzer beschrieben ist, die über ein Ereignis benachrichtigt werden möchten. Mit dem Wissensraum als Basis kann so ein Kontext direkt durch die Umgebung eines Objekts beschrieben werden. Alle Aktionen, die Objekte innerhalb eines Raums betreffen, sind direkt mit diesem Raum gekoppelt und lösen dort entsprechende Ereignisse aus. Damit stellt der Raum einen Kontext für die enthaltenen Objekte dar, wie bereits in Abschnitt 3.1.4 beschrieben worden ist.

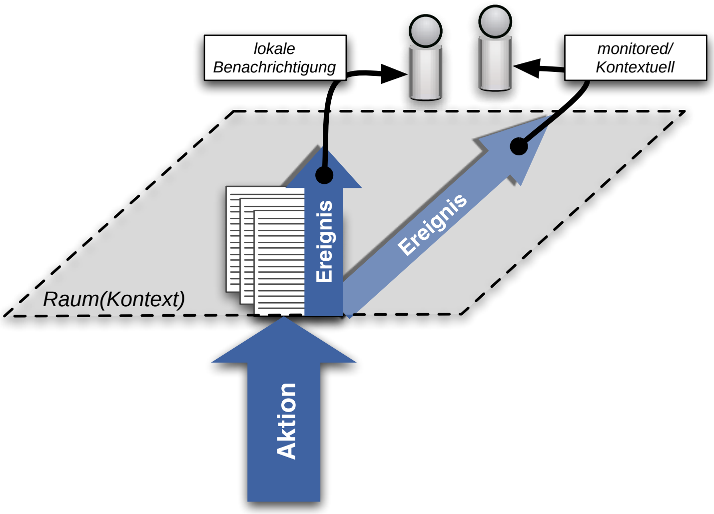


Abbildung 6.4: *Ereignisverarbeitung im Wissensraum*

Abbildung 6.4 zeigt die lokale Verarbeitung von einem Ereignis und die Weiterleitung in den umgebenden Raum. Die Aktion eines Benutzers löst das Ereignis aus und ein Benutzer wird benachrichtigt, der in der lokalen Benachrichtigungsliste des Objekts geführt wird. Analog dazu wird das Ereignis auch in der Umgebung ausgelöst und kann dort ebenfalls abonniert werden. Dazu muss ein *Observed*-Flag mit angegeben werden, welches zur Unterscheidung zwischen dem lokalen Ereignis und dem Ereignis in der Umgebung dient.

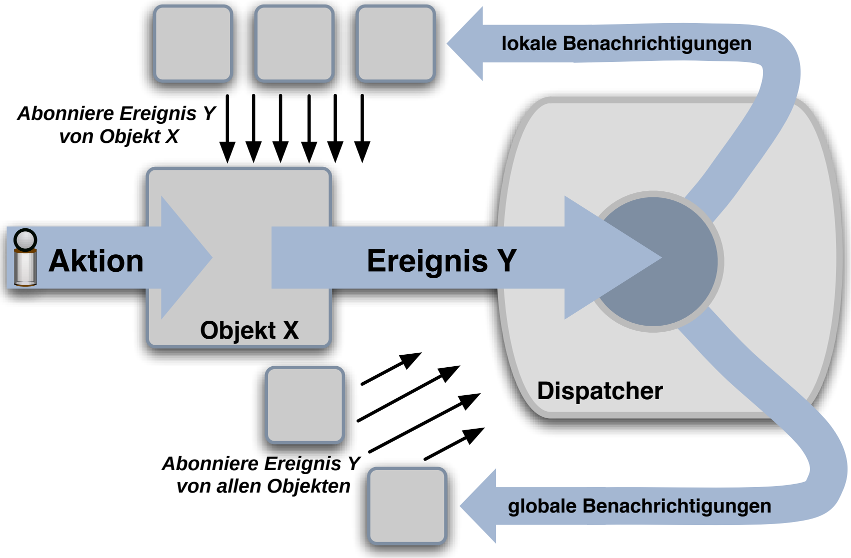


Abbildung 6.5: *Ereignisverarbeitung*

Abbildung 6.5 zeigt eine Verarbeitung von Ereignissen. Eine Aktion löst ein Ereignis aus, das dann über einen Dispatcher weiterverarbeitet wird. Dabei wird zwischen lokalen und globalen Ereignissen unterschieden. Lokale Ereignisse beschreiben Ereignisse, die in einem einzelnen ausgewählten Objekten auftreten. Dagegen beziehen sich globale Ereignisse auf alle Ereignisse eines bestimmten Typs (einer bestimmten Aktion unabhängig vom Kontext) und sind über den Dispatcher zu beziehen. Die Benachrichtigungen werden dann an die Objekte verschickt, die das jeweilige Ereignis abonniert haben. Dieser Mechanismus kann insbesondere von Modulen verwendet werden, um Abläufe im Server zu überwachen und zu kontrollieren.

Auf Basis dieser grundlegenden Architekturelemente werden im Folgenden verschiedene Architekturen vorgestellt, die einen verteilten Datenraum darstellen. Offensichtlich sind einige der im vorigen Kapitel aufgestellten Anforderungen an eine Architektur verteilter Wissensräume bereits durch eine geeignete Basisarchitektur gegeben. Dazu gehören die Erweiterbarkeit und Konfigurierbarkeit, die durch die Verwendung eines Microkernels bereitgestellt wird.

6.2 Architektur 1: Serververbund durch globalen Verzeichnisdienst

Die einfachste Architektur eines Serververbunds stellt die Bereitstellung eines gemeinsamen Datenraums in Form eines globalen Verzeichnisdienstes dar. Dazu wird die Benutzer- und Gruppenverwaltung eines kollaborativen Systems durch ein externes, serverübergreifendes Verzeichnis ausgetauscht. Damit wird ein minimaler Datenraum zwischen den Systemen erzeugt (vgl. Abschnitt 3.2), der eine Anmeldung von Benutzern im gesamten Verbund ermöglicht. Als offene, standardisierte Lösung bietet sich ein LDAP-Verzeichnis an. Offensichtlich handelt es sich bei einer solchen Schnittstelle um eine Lösung, die bereits vielfach verwendet wird. Dabei können verschiedene Systeme innerhalb einer Organisation auf ein zentrales Verzeichnis zurückgreifen, sodass eine einzelne Kennung für einen Benutzer ausreichend ist.

Eine Koppelung des Verzeichnisses an ein kollaboratives Systems kann erfolgen, sofern das System über entsprechend offene Schnittstellen verfügt. Einzelne Server können ihre interne Gruppenverwaltung verwenden und diese zu einem späteren Zeitpunkt durch ein LDAP-Verzeichnis ersetzen. Im Gegensatz zu einem einzelnen Server, der ein LDAP-Verzeichnis verwendet, wird die Situation wesentlich komplexer, wenn sich viele Systeme dieses Verzeichnis teilen und modifizieren. Hier müssen Synchronisationsmethoden berücksichtigt werden, sodass die Integrität der Benutzerdaten zu jedem Zeitpunkt sichergestellt werden kann (vgl. auch ACID-Eigenschaften in Abschnitt 5.3.5).

Neben einem LDAP-Verzeichnis sind andere Systeme für einen gemeinsamen Datenraum verwendbar. So bietet das Globus Toolkit (Foster 2005) z.B. eine Grid-Infrastruktur für Applikationen, die auf Basis von Web Services (vgl. Abschnitt 7.7) kommunizieren. Dadurch kann ein gemeinsamer verteilter Datenraum umgesetzt werden, der über Dienste bereitgestellt wird.

Abbildung 6.6 zeigt einen gemeinsamen LDAP-Datenraum, den sich drei Server teilen. Neben den Daten, die im LDAP-Verzeichnis bereitstehen, verfügt jeder Server über weitere Objekte in einer eigenen Datenbank. Diese Objekte sind lediglich lokal für den jeweiligen Server verfügbar und nicht Bestandteil des geteilten Datenraums.

Grundsätzlich muss zwischen zwei Situationen unterschieden werden: Im einfachsten Fall wird ein statischer Serververbund gebildet, sodass immer die gleichen Server auf das Verzeichnis zugreifen. Im Gegensatz dazu stellt sich die Situation weitaus schwieriger dar, wenn Server dynamisch zum Verbund hin-

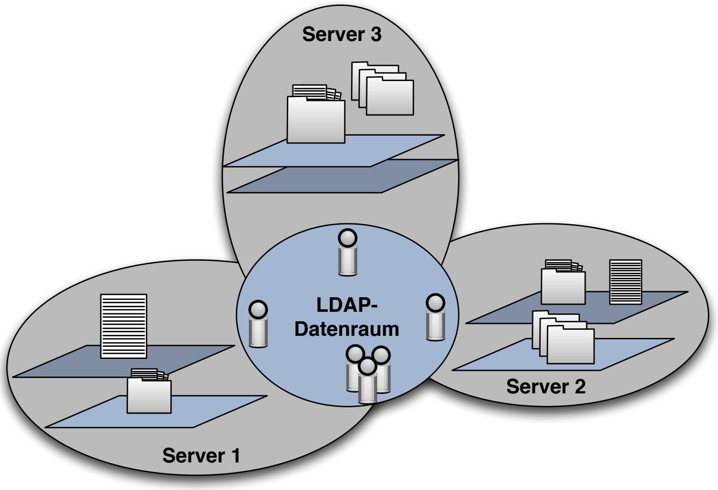


Abbildung 6.6: *Gemeinsamer LDAP-Datenraum*

zukommen oder diesen verlassen. Bei den Überlegungen müssen die Szenarien (vgl. Kapitel 4) einbezogen werden, um alle möglichen Situationen zu identifizieren.

6.2.1 Szenario 1: Statischer Serververbund

Bei einem statischen Serververbund kann ein gemeinsamer Datenraum in Form eines LDAP-Verzeichnisses verwendet werden. Ein solcher Verbund zeichnet sich dadurch aus, dass zu keinem Zeitpunkt ein Server hinzukommt oder den Verbund verlässt.

Bei der Anmeldung eines Benutzers werden die Benutzerdaten aus dem LDAP-Verzeichnis gelesen und mit den übermittelten Authentisierungsdaten verglichen. Stimmen die Daten überein, so ist der Anmeldevorgang erfolgreich. An dieser Stelle besteht darüber hinaus die Möglichkeit die Benutzerdaten lokal zwischenzuspeichern. In diesem Fall muss außerdem überprüft werden, ob der Benutzer bereits auf diesem Server existiert. Falls dies nicht der Fall ist, wird ein Benutzerobjekt mit den aus dem LDAP-Verzeichnis gelesenen Daten erzeugt. Hier gibt es zwei unterschiedliche Vorgehensweisen: Entweder es wird ein Replikat des Benutzers erzeugt, das mit dem Verzeichnis fortwährend synchronisiert werden muss (vgl. Abschnitt 5.3.6), oder es wird ein Platzhalter für den Benutzer erzeugt.

Bei Modifikationen von Daten eines Servers kann dieser die Änderungen direkt in das LDAP-Verzeichnis schreiben. Dieses Szenario erfordert keine besonderen Überlegungen für die Synchronisation der Daten. Lediglich bei der initialen Zusammenführung des Verbunds muss eine Synchronisation erfolgen, falls die Server bereits über lokal gespeicherte Benutzer verfügen. Im Allgemeinen kann allerdings nicht von einem statischen Verbund ausgegangen werden, da in der Regel nicht auszuschließen ist, dass es doch zu einer Erweiterung kommen könnte.

6.2.2 Szenario 2: Dynamischer Serververbund

Bei einem dynamischen Serververbund können Server hinzukommen oder den Verbund verlassen. Zur Vereinfachung kann angenommen werden, dass Benutzerdaten von Servern, die den Verbund verlassen, trotzdem weiterhin im LDAP-Verzeichnis geführt werden. Es ist möglich, dass sich solche Benutzer bereits bei anderen Servern des Verbunds angemeldet haben, sodass dort bereits Referenzen auf diesen Benutzer und weitere Daten des Benutzers existieren.

Bei der Aufnahme eines neuen Servers sind viele Details zu berücksichtigen. Zunächst müssen alle Server des Verbunds dem aufzunehmenden Server vertrauen und als Teil des Verbunds akzeptieren. Ein Vertrauensverhältnis ist unbedingt erforderlich, da der Server auf die Benutzerdaten des Verbunds zugreifen wird. Die Identität eines Servers muss daher ebenfalls eindeutig sein — eine Identifikation durch den Austausch von Zertifikaten ermöglicht beispielsweise eine Bestimmung dieser Identität.

Grundsätzlich gilt es Benutzerdaten im LDAP-Verzeichnis mit dem neuen Server zu synchronisieren. Dabei kann es zu Konflikten kommen und die Identität eines Benutzers muss eindeutig sichergestellt werden, damit es zu keiner Synchronisation von unterschiedlichen Benutzern kommt. Allerdings kann die Identität nicht aus den gespeicherten persönlichen Daten abgeleitet werden, da ein Angreifer Daten gezielt fälschen könnte und damit unauthorisierten Zugriff erlangen würde. Als Sicherheitskriterium ist deshalb lediglich das Passwort des Benutzers geeignet. Gleichzeitig genügt damit zur Feststellung der Identität eine Kombination aus Benutzername und Passwort. Der Nachteil dieser Methode liegt allerdings darin, dass Benutzer mit unterschiedlichen Passwörtern nicht korrekt zugeordnet werden.

Neben diesen einfachen Szenarien existieren noch andere Situationen, die eine flexible Konfiguration des Servers erfordern. So kann auch ein LDAP-Verzeichnis zum Einsatz kommen, aus dem ausschließlich Daten gelesen werden soll. Diese Situation entsteht, wenn Benutzerdaten genutzt werden sollen, die hauptsächlich von einer anderen Applikation verwaltet werden. Damit wird es diesen Benutzern ermöglicht, sich mit der gleichen Kennung am kollaborativen System anzumelden. Die Administration der Daten erfolgt dann extern mit anderen Werkzeugen.

Wird dieser Gedanke weitergeführt, so ist es vorstellbar, dass neben den Benutzern, die von einer anderen Applikation verwaltet werden, zusätzlich das lokale Benutzermanagement des Servers aktiv bleibt. Es handelt sich also um eine hybride Lösung, die das Benutzermanagement nicht vollständig dem LDAP-Verzeichnis überträgt. Das zugrunde liegende Szenario ist eine Einbeziehung eines zentralen Verzeichnisdienstes. Im Kontext einer Universität kann es sich dabei um die Daten aller Studenten und Mitarbeiter handeln, die dadurch Zugriff zum CSCW-System erhalten. Gleichzeitig ist es weiterhin möglich direkt im CSCW-System zusätzliche Benutzer anzulegen und zu verwalten. Damit gibt es also Benutzer des CSCW-Systems, die aufgrund der lokalen Datenhaltung nicht für andere Applikationen gültig sind, die auf den LDAP-Datenraum zugreifen. Insgesamt erschwert eine solche hybride Datenhaltung das Benutzermanagement, da eine Synchronisation und Administration der Daten dadurch aufwändiger wird. Eine flexible Speicherung von Objektdaten in einem zentralen Datenraum gelingt mit Hilfe eines Persistenzmanagers, der den jeweiligen Speicherort eines Objekts auswertet und die Daten lädt (vgl. Abschnitt 5.3).

Folgende notwendigen Attribute sollten vorgesehen werden:

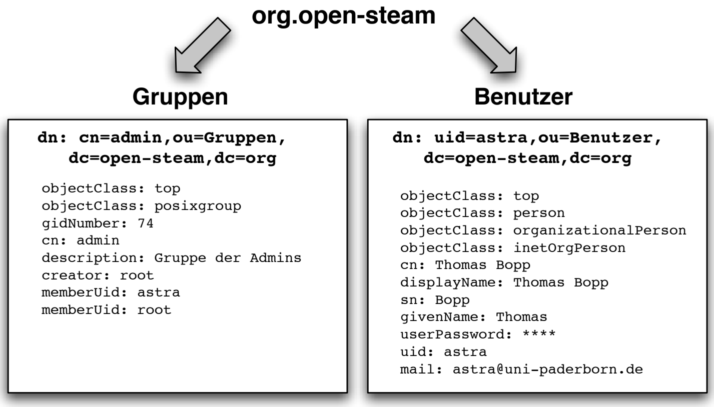
`Vorname, Nachname, Kennung, Passwort und E-Mail.`

Darüber hinaus können jedem Benutzer ein Bild und eine Beschreibung zugeordnet sein. Diese Attribute finden sich in typischen LDAP-Schemata wie *InetOrgPerson*[9] wieder und können von dort problemlos importiert werden. Je nach Art der Verwendung des LDAP-Datenraums kann es auch möglich sein ein eigenes Schema zu integrieren und damit die Beschränkung in der Speicherung der Daten grundsätzlich aufzuheben. Dies ist natürlich nur dann durchführbar, wenn Zugriff auf die LDAP-Installation besteht und diese angepasst werden kann.

Eine weitere Strategie der Benutzung eines LDAP-Verzeichnisses ist die zusätzliche Speicherung der Benutzergruppen eines kollaborativen Systems. Dazu muss der LDAP-Datenraum zweigeteilt werden, sodass neben den Benutzern ein Bereich für Gruppen verfügbar ist. Abbildung 6.7 zeigt die Aufteilung des

---

[9] Dieses Schema ist Bestandteil jeder LDAP-Installation.



```
dn: cn=admin,ou=Gruppen,
   dc=open-steam,dc=org

objectClass: top
objectClass: posixgroup
gidNumber: 74
cn: admin
description: Gruppe der Admins
creator: root
memberUid: astra
memberUid: root
```

```
dn: uid=astra,ou=Benutzer,
   dc=open-steam,dc=org

objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Thomas Bopp
displayName: Thomas Bopp
sn: Bopp
givenName: Thomas
userPassword: ****
uid: astra
mail: astra@uni-paderborn.de
```

Abbildung 6.7: *Aufteilung des LDAP-Verzeichnisses*

LDAP-Verzeichnisses zur Ablage von Benutzern und Gruppen in entsprechenden Unterbereichen. Beispielhaft sind die Einträge für den Benutzer „astra“ und die Gruppe „admin“ angegeben. Der Zugriffspfad ist zu jedem Eintrag explizit angegeben (dn). Mit *objectClass* sind die einzelnen Schemata, die für einen Eintrag verwendet werden, aufgeführt. Aus der Abbildung ist ersichtlich, dass die notwendigen Attribute in das LDAP-Schema überführt werden können, wobei die jeweiligen Schemata über wesentlich mehr Felder verfügen. Für die Gruppe ist die Liste der Mitglieder durch den Eintrag *memberUid* zugeordnet.

Zur Speicherung kann ein besonderes LDAP-Schema Verwendung finden, das eine Hierarchie von Gruppen berücksichtigt. Ein Eintrag, der auf die jeweilige Obergruppe (Vatergruppe, *ist-Mitglied* Relation) verweist, ist wünschenswert, aber nicht unbedingt notwendig. Eine Struktur kann bereits mit Hilfe des *groupOfUniqueNames*- oder *posixGroup*-Schemas realisiert werden. Da Gruppen- und Benutzernamen disjunkte Mengen sind (vgl. Abschnitt 5.2.2), können neben den Benutzerkennungen auch Gruppeneinträgen in den Mitgliedsfeldern gespeichert werden. Die Vergabe von Namen wird durch die Applikation reguliert und dadurch sichergestellt, dass es zu keinen Überschneidungen zwischen Benutzern und Gruppen kommt.

Gruppen der höchsten Ebene, so genannte Top-Level-Gruppen, sind dadurch identifizierbar, dass sie kein Mitglied einer anderen Gruppe sind. Die Überprüfung dieses Sachverhalts ist unter Umständen komplizierter (falls kein Eintrag für die Vatergruppe existiert), da alle Gruppen im LDAP daraufhin untersucht werden müssen.

Beim Aufbau eines Serververbunds sind die im Folgenden beschriebenen Namenskonflikte auch zwischen Benutzer und Gruppen zu berücksichtigen.

6.2.3 Namenskonflikte

Bei der Synchronisation der Benutzer- und Gruppendaten kann es zu Namenskonflikten kommen. Falls eine Benutzerkennung auf mehreren Servern existiert, kann zwischen zwei Situationen unterschieden werden:

- Es handelt sich um ein und denselben Benutzer, der auf unterschiedlichen Servern die gleiche Benutzerkennung gewählt hat — eine Synchronisation der Benutzerdaten kann erfolgen.
- Es handelt sich um unterschiedliche Benutzer, die zufällig die gleiche Kennung gewählt haben. Dieser Sachverhalt stellt einen Konflikt dar und es kann keine Synchronisation erfolgen.

Es gilt nun die Identität eines Benutzers auf irgendeine Weise zu ermitteln. Dabei kann im Allgemeinen angenommen werden, dass Benutzer mit gleicher Kennung und E-Mail-Adresse auch dieselbe Person sind. Gleichzeitig kann es vorkommen, dass ein Benutzer unterschiedliche E-Mail-Adressen verwendet und aus diesem Grund nicht zugeordnet werden kann.

Allerdings spielen hier Sicherheitsaspekte eine Rolle, denn es ist für Dritte ohne weiteres möglich Kennung und E-Mail zu fälschen. Als eindeutiges Merkmal eines Benutzers kann also ausschließlich die Kombination Kennung-Passwort dienen. Gleichzeitig führt dies dazu, dass einige identische Benutzer nicht synchronisiert werden können.

Ein Konflikt bei der Benutzerkennung kann durch die Verwendung eines Serverpräfix (z.B. der Name des Servers) vermieden werden oder aber die Benutzer werden weiterhin nur lokal auf den jeweiligen Servern verwaltet. Trotz des gemeinsamen Datenraums ist es dann für solche Benutzer nicht möglich, auf einen anderen Server des Verbunds zu wechseln (falls der Benutzer mit Serverpräfix gespeichert ist, wäre es möglich sich mit dem um das Präfix erweiterten Namen anzumelden). Weiterhin ist eine manuelle Lösung von Konflikten umsetzbar, indem die Benutzer ihre Benutzerkennungen selbstständig in Bezug bringen.

6.2.4 Verteilte Benutzer, Gruppen und Berechtigungen

Eine Speicherung von Berechtigungen erfolgt auf Basis der Benutzer- und Gruppenobjekte eines Servers. In dieser Hinsicht unterscheidet sich die Situation aufgrund der Einbindung eines zentralen Datenraums für Benutzerobjekte nicht von einem Betrieb mit einem einzelnen Server. Die Zugriffsrechte sind unter Verwendung von ACLs direkt an die Objekte gebunden, die weiterhin lokal gespeichert werden. Bei rollenbasierten Berechtigungen verhält es sich genauso. Die Rollen müssen dabei lokal auf jedem Server verwaltet und lokalen oder entfernten Benutzern zugeschrieben werden.

Aufgrund des Stellenwerts von Gruppen bei der Vergabe von Berechtigungen ist eine Speicherung der Gruppen im gemeinsamen Datenraum von Vorteil. Nur dann kann die Mitgliedschaft von Benutzern in den jeweiligen Gruppen zugeordnet und damit die entsprechenden Berechtigungen in entfernten Arbeitsräumen von Gruppen verbunden werden.

6.2.5 Bewertung der Architektur

Eine Architektur mit einem zentralen Verzeichnis stellt eine minimale Lösung für einen Datenraum zwischen Servern dar. Eine Speicherung von Benutzerdaten an einer zentralen Stelle ermöglicht eine Anmeldung an beliebige Server des Verbunds. Gleichzeitig handelt es sich bei dem LDAP-Server um einen *Single Point of Failure*. Der Ausfall des Servers könnte also den gesamten Verbund betreffen. Diese Probleme können durch Replikation der Daten auf jeden Server des Verbunds bzw. replizierte LDAP-Verzeichnisse gelöst werden.

Insgesamt bleibt der Datenraum zwischen den Servern durch diese Architektur sehr beschränkt. Es ist nicht möglich Objekte auf einem Server aufzunehmen und diese auf einen Raum zu übertragen, der sich auf einem anderen Server befindet. Auch eine verbundweite Suche nach Objekten ist lediglich für die Objekte im gemeinsamen Datenraum möglich (vgl. Szenario der Suche in Wissensräumen in Abschnitt 4.3).

Trotz dieser Beschränkungen bietet der Datenraum neben einem einheitlichen applikationsübergreifenden Benutzermanagement die Grundlage für einen automatischen Wechsel zwischen Applikationen ohne Eingabe von Benutzerkennung und Passwort (Single Sign-On). Dabei wird die Authentisierung direkt zwischen den Applikationen vorgenommen, die z.B. Tickets zur Verifikation der Identität austauschen. Damit erfüllt diese Architektur die Anforderung, Verbindungen zwischen Wissensräumen unterschiedlicher Server zu schaffen (vgl.

Abschnitt 4.1). Auf technischer Ebene beinhaltet dies die Authentisierung von Benutzern auf allen Servern des Verbunds. Auf der anderen Seite sind die meisten Anforderungen aus Abschnitt 5.8 nicht erfüllt. So wird durch den zentralen Speicherort die Autonomität einzelner Server verletzt. Darüber hinaus findet auch keine einheitliche Ereignisverarbeitung im Verbund statt.

Auf die Fragestellung einer Auflösung von verwobenen Strukturen (vgl. Abschnitt 4.2) kann hier auf einfache Weise beantwortet werden, da kaum Abhängigkeiten zwischen den Servern bestehen. Diese ergeben sich hauptsächlich durch Bezüge zu entfernten Benutzern, die jedoch keine technischen Probleme beinhalten, da in jedem Fall Platzhalter für die Benutzerobjekte erstellt werden müssen. Aus diesem Grund unterscheidet sich die Situation bei einer Ablösung eines Servers nicht von einer Inaktivität von lokalen Benutzern.

Statt einem LDAP-Verzeichnis können auch andere Repositorys als ein gemeinsamer Datenraum genutzt werden. Während LDAP insbesondere zur Speicherung von Benutzerdaten dient, kann ein DHT-Service allgemeiner eingesetzt werden (vgl. Rhea et al. 2005). Dabei handelt es sich um eine spezielle Art eines Distributed Hash Tables (vgl. Abschnitt 2.3.3), der sich auf sehr einfache Art einbinden lässt und nur über wenige Basisfunktionen verfügt. Mit Funktionen zur Speicherung und zum Abruf von Daten (get, put) steht eine sehr allgemeine Schnittstelle zur Verfügung, die keine spezielle Art von Objekten oder Daten voraussetzt.

Die im Folgenden dargestellte Architektur stellt eine Erweiterung des passiven Datenraums dar und basiert auf Verbindungen zwischen den einzelnen Servern des Verbunds, um auch einen Austausch von Ereignissen im Verbund zu ermöglichen.

6.3 Architektur 2: Inter-Serververbindungen

Mit dem Ziel, kooperatives Arbeiten über Servergrenzen hinweg zu ermöglichen, ist der nächste logische Schritt, eine Verbindung zwischen Wissensräumen unterschiedlicher Server zu schaffen. Dies kann mit einem normalen Verbindungsobjekt geschehen, das auf ein entferntes Objekt verweist. Dieses entfernte Objekt muss in diesem Fall lediglich genauso behandelt werden wie ein lokales Objekt. Das Resultat einer solchen Vorgehensweise ist ein gemeinsamer Datenraum für alle Objekte. Auf die vorherige Architektur eines gemeinsamen LDAP-Datenraums bezogen trifft dies noch nicht zu, da bisher lediglich Benutzer Gruppen in dem Datenraum zur Verfügung stehen. Eine direkte Lösung besteht hier in einem Verweis, der eine Serverkennung beinhaltet, sodass das Ziel ohne weiteres identifiziert werden kann.

Eine grundlegendere Fragestellung ist jedoch, wie diese Verbindungen initial angelegt werden können. Dazu müssen Räume und Raumstrukturen auf entfernten Servern zu durchsuchen sein, um einen Zielpunkt wählen zu können. Entweder wird dabei der Zielpunkt aus einem Suchergebnis ausgewählt oder dieser kann in der Struktur von Räumen gefunden werden. In beiden Fällen muss eine Verfügbarkeit dieser Daten vorausgesetzt werden. In der im Abschnitt 6.2 dargestellten Architektur (Architektur 1) ist dies nicht der Fall, da der Datenraum ausschließlich aus Benutzern und Gruppen besteht. Daraus folgt, dass entweder ein zentraler Datenraum um Räume erweitert werden muss oder ein Kommunikationskanal zum Austausch dieser Informationen hinzugenommen wird.

6.3.1 Zentraler Datenraum mit Räumen

Ein Wissensraum unterscheidet sich aus technischer Sicht nicht von einem Container (vgl. Abschnitt 3.1.7). In beiden Fällen wird eine Liste von enthaltenen Objekten verwaltet. Konzeptionell ist der Raum dabei so ausgelegt, dass auch Benutzer und Verbindungen zu anderen Räumen enthalten sein können. Die Speicherung von Räumen und Containern unterscheidet sich daher jedoch nicht und so steht in beiden Fällen eine Liste der enthaltenen Objekte im Vordergrund (vgl. Abschnitt 6.1.1). Aufgrund der Klassenhierarchie mit dem Objekt als zentrale Klasse sind für eine vollständige Speicherung von Containern auch Attribute, Annotationen und Berechtigungen zu speichern. Bezogen auf das LDAP-Verzeichnis aus Architektur 1 kann dies mit einem entsprechenden Schema umgesetzt werden. Allerdings ist ein solches Verzeichnis kaum für die Speicherung von Objekten ausgelegt, da insbesondere eine Anpassung an zusätzliche Attribute nicht gegeben ist. Ein flexibles kollaboratives System wird jederzeit eine Registrierung von weiteren Attributen erlauben. Demgegenüber steht ein starres LDAP-Schema, das einmal definiert wird und allen Anforderungen genügen muss. Die Möglichkeit einer Speicherung von Objekten mit Verweisen auf andere Objekte innerhalb eines LDAP-Verzeichnisses ist von Ryan et al. (1999) für Corba dargestellt worden. Eine ähnliche Realisierung ist für Objekte aus dem Klassenmodell des Wissensraums mit den Einschränkungen eines starren Schemas möglich.

Abbildung 6.8 zeigt, wie mehrere Server einen Verbund bilden und dabei

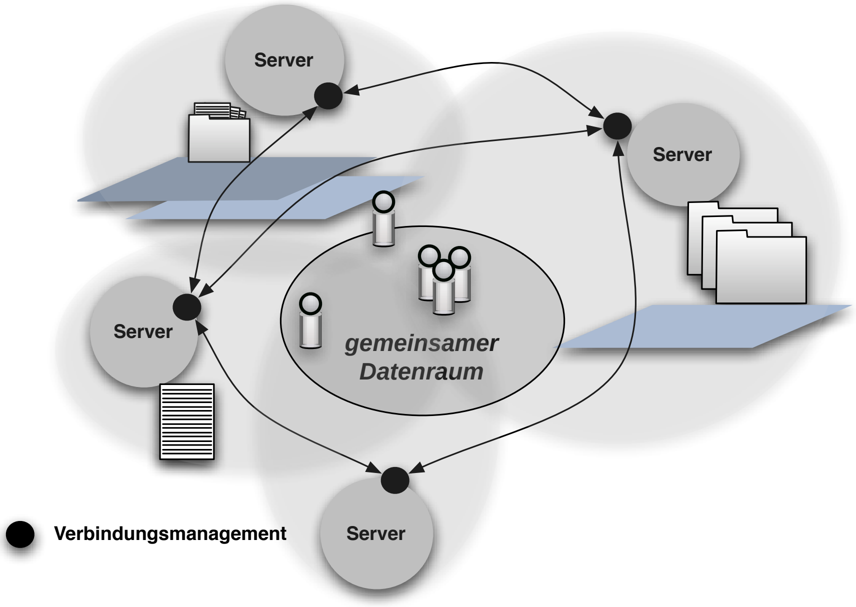


Abbildung 6.8: *Serververbund mit gemeinsamem Datenraum*

auf einen gemeinsamen Datenraum zurückgreifen. Wie im vorherigen Abschnitt beschrieben, können die Benutzer und Gruppen des Verbunds in einem LDAP-Verzeichnis gespeichert werden. Alle Server des Verbunds haben Zugriff auf dieses Verzeichnis und speichern dort ihre Benutzer- und Gruppendaten.

6.3.2 Kommunikation

Für Verbindungen zwischen Wissensräumen verschiedener Server muss von einem Startpunkt ausgehend ein Ziel gefunden werden. Dazu muss die Möglichkeit bestehen die Strukturen des Zielservers zu durchsuchen, um eine Auswahl zu treffen (vgl. Abschnitt 4.3). Dies beinhaltet eine direkte oder indirekte Erreichbarkeit aller Server. Zu diesem Zweck könnte jeder Server eine Liste mit anderen Servern des Verbunds führen. In dieser Liste sind alle notwendigen Informationen gespeichert, um Daten mit einem anderen Server des Verbunds auszutauschen. Die einfachste Umsetzung einer solchen Kommunikation ist eine vollständige Liste aller anderen Server. Auf dieser Basis kann eine direkte Kommunikation mit jedem Server erfolgen. Offensichtlich erfordert eine solche Kommunikation eine Authentifizierung zwischen den Servern bei einem Zugriff auf sensible Daten. Dazu existieren zwei Möglichkeiten: Entweder geschieht dies auf Ebene der Benutzer, indem der aktive Benutzer im Verbund authentifiziert wird, oder über ein Vertrauensverhältnis der Server untereinander.

Aufgrund der Autonomie von Servern im Verbund muss zwischen den Servern nicht unbedingt ein Vertrauensverhältnis bestehen, da sich die Zugriffsrechte genau wie im Betrieb mit einem einzelnen Server aus den Berechtigungen ergeben, die ein Objekt speichert (ACL).[10] Entscheidend ist demnach lediglich die eindeutige Wiedererkennung eines Servers. Analog zu einer Authentisierung von Benutzern müssen Server authentisiert werden, damit die Berechtigungen der Gruppen und Benutzer des Servers nicht von Dritten übernommen werden können. Ein Server könnte sich einfach unter einer gefälschten ID anmelden und damit würden die lokalen Gruppen und Benutzer ebenfalls unter dieser ID geführt.

Eine robuste Identifizierung von Servern untereinander gelingt z.B. durch einen Austausch von Zertifikaten zwischen den Parteien. Die Zertifikate werden dann direkt zur Identifikation eines Servers verwendet (vgl. Hess et al. 2002).

6.3.3 Suche

Die Suche nach Objekten lässt sich auf Basis einer Kommunikation zwischen den Servern realisieren. Dabei ist keine Kenntnis über den Verbund und die Verteilung von Objekten notwendig, sodass die Suche immer global im Verbund durchgeführt werden kann. Dazu muss lediglich ein entsprechender Suchdienst angesprochen werden, der die Resultate über die Verbindung zurückliefert. Eine einheitliche Schnittstelle vereinfacht eine Suche in einem heterogenen Verbund. Darüber hinaus sind aktuelle Standards im Bereich des Semantic Web[11] von Bedeutung, um eine Suche effektiv zu gestalten. Das Edutella-Projekt verwendet so den RDF-Standard, um Anfragen und Resultate geeignet zu beschreiben (Nejdl et al. 2002).

---

[10] Bei Verwendung von Rollen ergeben sich die Berechtigungen aus der Zuordnung zwischen Benutzern und Rollen.

[11] Publikationen und Spezifikationen im Bereich des Semantic Web sind verfügbar unter: http://www.w3.org/2001/sw/ [Stand: 20.12.2005].

6.3.4 Verbindungen zwischen Räumen

Eine Verbindung zwischen zwei Räumen kann technisch durch eine Speicherung von Objektreferenzen realisiert werden.[12] Im Gegensatz zu einem einzelnen Server müssen im Verbund Referenzen auf nicht lokale Räume bereitgestellt werden. Ausgehend von einer Suche nach Zielräumen muss diese prinzipiell lediglich auf mehrere Server ausgeweitet werden. Diese Überlegungen sind bereits im Szenario *Suchen innerhalb eines Serververbunds* beschrieben worden (vgl. Abschnitt 4.3).

Die einfachste Umsetzung einer Suche ist diese auf einem Server zu starten und an die anderen Server weiterzuleiten, die dann ihre lokalen Suchergebnisse zum Ausgangspunkt der Suche zurückgeben. Diese Vorgehensweise entspricht dem im GNUTELLA-Netzwerk verwendeten FLOOD-Algorithmus (vgl. Charwathe et al. 2003), der in dem Kontext eines Peer-to-Peer-Systems Probleme aufwirft und eine große Netzlast verursacht. In einem Serververbund mit wenigen Knoten ist diese Problematik allerdings zu vernachlässigen.

Die Suchergebnisse müssen nun in adäquater Weise dargestellt werden, um einen passenden Zielraum auswählen zu können und schließlich eine Verbindung zwischen den Räumen zu schaffen. Dazu sind wechselseitig Referenzen zu speichern, die den Speicherort eines Objekts mit einbeziehen. Im Wesentlichen muss eine eindeutige Identifizierung jedes Objekts im Verbund gewährleistet werden. Die Kodierung des Speicherorts erleichtert dabei ein Auffinden und Abrufen des Objekts (vgl. Abschnitt 5.3.1).

Die Existenz von Verbindungen zwischen Servern des Verbunds ermöglicht einen transparenten Wechsel des Raums im Verbund. An dieser Stelle kann ein Single Sign-On-Mechanismus Anwendung finden. Dabei wird für den Benutzer ein Ticket generiert, welches lediglich für den angestrebten Wechsel des Servers gültig ist. Dieses Ticket wird in dem gemeinsamen Datenraum gespeichert und ist damit für sämtliche Server des Verbunds abzufragen. Das Durchschreiten einer Inter-Serververbindung automatisiert den Authentifizierungsmechanismus, indem das generierte Ticket direkt an den Server übermittelt wird. Dieser entfernte Server ist nun in der Lage das Ticket über den gemeinsamen Datenraum zu validieren.

Der Benutzer wechselt damit ohne es zu bemerken von einem Server zu einem

---

[12] Es handelt sich normalerweise um bidirektionale Verbindungen. Grundsätzlich sollten sowohl bidirektionale als auch unidirektionale Verbindungen unterstützt werden (vgl. Objektmodell des Wissensraums in Abschnitt 3.1.7).

anderen. Innerhalb dieses entfernten Servers kann er sich nun wiederum frei bewegen und Objekte manipulieren. Es gibt allerdings Einschränkungen, da sich die beiden Server lediglich die Benutzerdaten im Datenraum teilen. Daher sind die privaten Arbeitsräume auf dem anderen Server nicht vorhanden und insbesondere ist es nicht möglich, weitere Objekte zwischen den Servern auszutauschen. Es existiert also keine Möglichkeit ein Dokument von einem Raum in einen anderen, entfernten Raum zu bewegen. Außerdem handelt es sich um einen passiven Datenraum ohne Austausch von Ereignissen. Dadurch ist eine gegenseitige Wahrnehmung der Benutzer nur eingeschränkt möglich. Dies gilt insbesondere für Aktivitäten anderer Benutzer, die vollständig verborgen bleiben, und damit ist zumindest ein synchrones gemeinsames Arbeiten nicht durchführbar. Damit sind die Anforderungen aus Szenario 4.4 und Szenario 4.5 nicht erfüllt.

Aufgrund dieser entscheidenden Einschränkung muss der Wechsel des Servers einem Benutzer verdeutlicht werden. Andernfalls kommt es zu Verwirrungen bei der Benutzung des Systems, da nicht alle Objekte uneingeschränkt zur Verfügung stehen. Dies gilt auch für den Rucksack des Benutzers, der aufgrund der Beschaffenheit des Datenraums auf jedem Server separat vorhanden ist und dadurch aus Benutzersicht zu fehlenden Objekten führen kann.

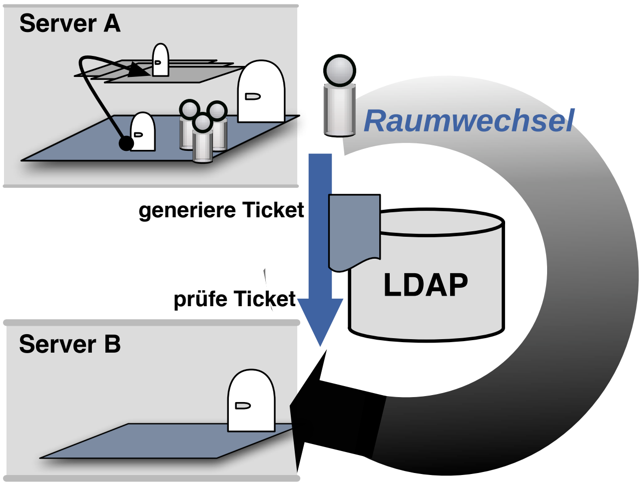


Abbildung 6.9: *Ticketaustausch im gemeinsamen Datenraum*

Abbildung 6.9 zeigt zwei Server mit gemeinsamem Datenraum. Ein Benutzer wechselt über eine Verbindung von einem Raum in einen anderen. Dabei wird der Benutzer auf einen anderen Server umgeleitet, auf dem sich der Zielraum befindet. Bei diesem Wechsel des Servers wird automatisch ein Ticket generiert, dass sowohl dem Zielserver übermittelt als auch im Datenraum gespeichert wird. Der Zielserver kann das Ticket daher durch einen Zugriff auf den Datenraum validieren und der Wechsel des Servers ist vollzogen.

6.3.5 Verteilte Benutzer, Gruppen und Berechtigungen

Im Gegensatz zur ersten Architektur eines gemeinsamen Datenraums ändert sich die Situation mit zusätzlichen Verbindungen zwischen entfernten Wissensräumen nicht. Im Unterschied zur vorigen Architektur erfolgt lediglich der Wechsel zwischen Servern automatisiert.

6.3.6 Bewertung der Architektur

Diese Architektur dient lediglich als weitere Ausbaustufe und stellt noch keine Umsetzung eines vollständig gemeinsamen Datenraums dar. Basierend auf Architektur 1 ist es notwendig Übergänge zwischen Wissensräumen im Verbund zu schaffen. Damit ist die Grundlage geschaffen, sich über Grenzen von Servern hinweg uneingeschränkt zu bewegen. Mit den Übergängen zu Wissensräumen anderer Server muss lediglich die Anmeldeprozedur automatisiert werden. Damit wird auf Basis der vorangegangenen Architektur lediglich ein Single Sign-On realisiert. Die Einschränkungen dieser Architektur für ein kooperatives Arbeiten von Benutzern verschiedener Server gelten allerdings genauso wie in der vorherigen Architektur.

Aufgrund der Übergänge zwischen Räumen bestehen hier Abhängigkeiten, die beim Ausscheiden eines Servers aus dem Verbund zu toten Verbindungen führen können. Offensichtlich kann diese Ablösung jedoch erkannt werden und so können diese Verbindungen entsprechend markiert oder gelöscht werden. In Bezug auf das Management von Benutzern sind keine Änderungen gegenüber der vorherigen Architektur zu verzeichnen, sodass sich der Verbund in dieser Hinsicht ohne weiteres auflösen lässt.

Die im Folgenden dargestellte Architektur führt die bisherigen Überlegungen einen Schritt weiter und bietet einen vollständig gemeinsamen Datenraum.

6.4 Architektur 3: Serververbund mit Handlungsraum

Diese Architektur setzt einen statischen Verbund von Servern voraus. Obwohl manuell neue Server hinzugenommen werden können, wird nicht von einem dynamischen Charakter des Verbunds ausgegangen. Es werden damit die Kernanforderungen des 1. Szenarios (vgl. Abschnitt 4.1) umgesetzt. Die Systeme sind zunächst unabhängig voneinander und schließen sich zu einem Verbund zusammen. Dieser Zusammenschluss geschieht dabei manuell und wird von den Administratoren der einzelnen Server gesteuert. Der Einsatzbereich umfasst daher mehrere Server innerhalb einer einzelnen Organisation oder Verbünde, die mehrere Organisationen beinhalten. Ziel ist es einen vollständig gemeinsamen Datenraum zu schaffen (vgl. Abschnitt 3.3).

Die meisten kollaborativen Systeme verfügen über eine fest eingebundene Persistenzschicht, die Objekte speichert und lädt. Generell ist es sinnvoll zumindest den Speicherort der Objekte austauschbar zu gestalten. So ist es möglich auf Datenbanken zurückzugreifen oder auch Daten im Dateisystem abzulegen. Dies ist insbesondere bei Knoten von Bedeutung, die über beschränkte Hardware verfügen und für die nicht unbedingt jedes Softwarepaket zur Verfügung steht. Auf technischer Ebene bietet sich hier eine Aufteilung der Persistenz in einen Persistenzmanager (vgl. Abschnitt 5.3) mit einer oder mehreren Persistenzebenen an.[13] Die Persistenzebenen übernehmen dabei direkt die Serialisierung und Speicherung von Objekten. Der Persistenzmanager sorgt für eine Zuordnung von Objekt und Speicherort und lehnt sich dabei an das „Data Access Object"-Pattern an (vgl. Alur et al. 2003, S. 462-495).

Dieses Konzept eines Persistenzmanagers ist so erweiterbar, dass auch entfernte Repositorys eingebunden werden können. Dabei kann es sich um andere Serversysteme handeln, ohne dass diese von anderen Persistenzebenen zu unterscheiden sind. Änderungen an Dateien erfolgen hier allerdings von mehr als einem Server. Dies ist bei der Implementierung zu beachten, sodass Zustandsänderungen von einem Server auf andere Server propagiert werden, die ein entferntes Objekt mit einbinden. Dort werden entsprechende Ereignisse ausgelöst, die über Publish-/Subscribe-Mechanismen abonniert werden können. Somit ist eine Synchronisation der Zustände gewährleistet.

---

[13] Dies zeigt sich auch im Aufbau des Corba Persistent Object Service (Kleindienst et al. 1996) mit Persistent Object Manager (POM) und Persistent Data Service.

6.4.1 Persistenzebenen

Die Architektur sieht mehrere parallele Persistenzebenen vor, damit neben lokalen Speicherorten auch verteilte Persistenzschichten eingebunden werden können. Dabei kommen Proxys zum Einsatz, die aber niemals direkt auf ein Objekt im Speicher verweisen, sondern über den Persistenzmanager zugeordnet werden. Diese Zwischenschicht bildet eine zusätzliche Abstraktionsebene, um verschiedene Persistenzschichten zu erlauben.

Abbildung 6.10: *Server-, Namensraum und Objekt-ID*

Die Unterscheidung der Ebenen kann durch verschiedene Namensräume erfolgen (vgl. Abschnitt 5.3.1). Neben der ID des Objekts bedeutet dies eine Speicherung einer Kennung für die Persistenzebene, in der das Objekt abgelegt worden ist. Dies kann dann sowohl ein lokales Speichermedium sein als auch eine Ressource im Internet. Diese Erweiterung ermöglicht den Datenraum eines einzelnen Servers und des gesamten Verbunds anzugleichen.

Selbst eine vollständige Überschneidung der Datenräume und damit, unter Einbeziehung von Ereignissen, ein gemeinsamer Handlungsraum ist denkbar. In diesem Fall kann auf jedes Objekt von überall zugegriffen werden und eine Übertragbarkeit von Objekten ist dadurch gegeben (vgl. Szenario 4.4). Folgende Speicherorte werden typischerweise zur persistenten Speicherung von Objekten genutzt:

- Datenbank: Eine lokale SQL-Datenbank oder auch eine Datenbank, die durch eine Netzwerkverbindung eingebunden wird.

- Dateisystem: Ein lokales Speichermedium, das eine persistente Speicherung im Dateisystem erlaubt. Im Gegensatz zu Datenbanken bieten Dateisysteme bereits ein Streaming für Dateien.

- Netzwerk: Das Objekt befindet sich irgendwo im Internet in einem Objekt-Repository. Dort sind die Daten wiederum in einer Datenbank oder dem Dateisystem gespeichert. Das Repository regelt alle Zugriffe auf Objekte und gewährleistet eine Konsistenz der Daten.

Eine weitere Notwendigkeit, die verschiedenen Server miteinander kommunizieren zu lassen, ist eine Ereignisverarbeitung zur Umsetzung eines gemeinsamen Handlungsraums (vgl. Abschnitt 3.3). Auf unterschiedlichen Servern können unterschiedliche Benutzer in dem gleichen Wissensraum aktiv sein und dadurch Ereignisse innerhalb dieses Raums auslösen. Zur synchronen Zusammenarbeit müssen die jeweiligen Ereignisse der anderen Benutzer für einen Benutzer angezeigt werden. Daher müssen sich die einzelnen Server über einen Kommunikationskanal miteinander austauschen und es müssen sich die Ereignisse im Verbund verbreiten, sodass jeder Abonnent eines Ereignisses über ein Auftreten informiert wird. Gleichzeitig lassen sich die Modifikationen von Objekten – etwa bei einer Verwendung von Replikationsmechanismen – durch die Ereignisverarbeitung synchronisieren: Ein Knoten im Verbund empfängt das Ereignis und führt die Modifikation eines Objekts auf einem lokal gespeicherten Replikat durch. Die Zustände der verschiedenen Replikate sind damit identisch (vgl. Abschnitt 5.6). Zu berücksichtigen ist allerdings das Auftreten und der Empfang von Ereignissen.

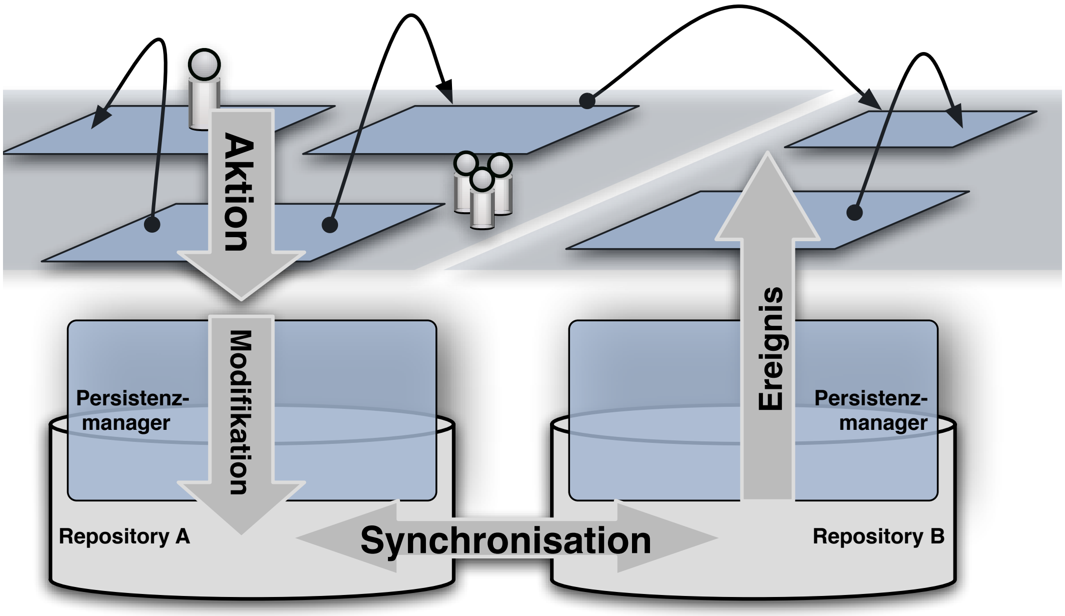


Abbildung 6.11: *Gemeinsamer Handlungsraum durch Persistenz-/Ereignisebene*

Abbildung 6.11 zeigt den unterliegenden Persistenzraum von Servern. Es handelt sich um ein verteiltes Objekt-Repository, das jedem Server eigene Daten zuordnet. Zwischen den einzelnen Persistenzebenen werden Objekte ausgetauscht und mit Hilfe von Ereignissen immer auf einem aktuellen Stand gehalten, sofern eine Verbindung zum Serververbund besteht.

Gleichzeitig sorgt die unterliegende Struktur für einen Austausch von Ereignissen zwischen Servern. Eine Aktion eines Benutzers auf Server A führt zu einer Modifikation eines Objekts. Diese Änderung wird in der Persistenzebene gespeichert und gleichzeitig an einen entfernten Server übertragen, der das Objekt verwendet und Daten zwischengespeichert hat. Der Status des Objekts ist auf Server B nun auf einem aktuellen Stand und das Ereignis wird ebenfalls auf Server B ausgelöst. Dadurch ist eine verteilte Verarbeitung von Ereignissen gewährleistet.

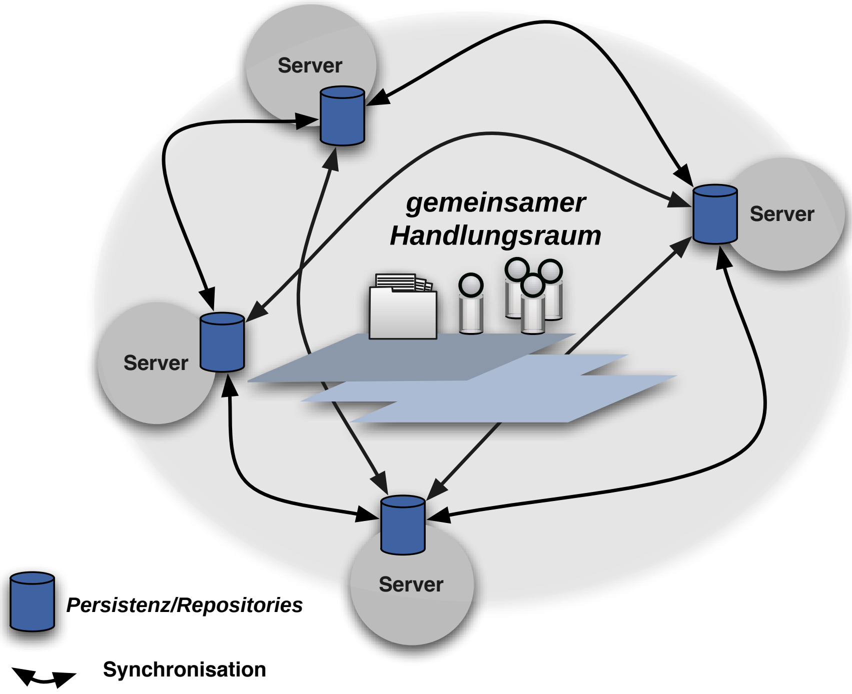


Abbildung 6.12: *Statischer Serververbund mit Handlungsraum*

Abbildung 6.12 zeigt den Verbund aus mehreren Servern. Diese synchronisieren ihre Persistenzschichten untereinander, sodass ein gemeinsamer Handlungsraum entsteht.

6.4.2 Ereignisverarbeitung

Die Realisierung eines Handlungsraums erfordert neben einem vollständig geteilten Datenraum einen Austausch von Ereignissen zwischen den beteiligten Knoten des Verbunds. Daher sind die verschiedenen Aspekte der Ereignisverarbeitung bei einer Verteilung zu berücksichtigen (vgl. Abschnitt 3.3). Während ein einfaches Publish/Subscribe-Modell gut in einem Verbund abzubilden ist, gestaltet sich die Situation wesentlich schwieriger für eine komplexere Verarbeitung.

6.4.3 Direkte Benachrichtigungen

Die direkten Benachrichtigungen können in einer lokalen Liste an einem Objekt gespeichert werden. Bei einem Ereignis wird diese Liste durchlaufen und alle dort eingetragenen Objekte werden per Callback aufgerufen und dadurch über das Eintreten des Ereignisses informiert (vgl. Abschnitt 6.1.8).

Diese Vorgehensweise kann im Verbund analog weitergeführt werden. Die Liste der zu benachrichtigenden Objekte enthält in diesem Fall neben den lokalen Objekten auch Objekte, die sich auf einem entfernten Server befinden (z.B. durch die Verwendung von Proxys). Auf diese Weise wird der Benachrichtigungsmechanismus dort genauso ausgeführt wie auf einem einzelnen Server. Es muss lediglich der Umweg über einen Kommunikationskanal zwischen den Servern in Kauf genommen werden.

Ein Unterschied ergibt sich allerdings durch die Verwendung von zwei Ereignisphasen. Während die zweite Benachrichtigungsphase wie bisher durchgeführt wird, kann die erste Phase von blockierbaren Ereignissen nicht ohne weiteres in den Verbund übertragen werden. Dies ist darauf zurückzuführen, dass wieder auf das Resultat des Callbacks gewartet werden müsste. Ein entferntes Objekt könnte die Ausführung blockieren. Bis das Resultat des Callbacks jedoch empfangen worden ist, kann die aktuelle Aktion nicht beendet werden.

Obwohl die Realisierung von Blockierungen der Ereignisse auch in einem Verbund durchführbar ist, erscheint der zusätzliche Aufwand nicht gerechtfertigt. Zudem kann eine Blockierung von Ereignissen durch entfernte Objekte unerwünscht sein.

Neben den direkten Benachrichtigungen können mit dem Raum als Kontext ebenfalls Ereignisse abonniert werden (vgl. Abschnitt 6.1.8).

6.4.4 Kontextuelle Benachrichtigungen

Bei kontextuellen Benachrichtigungen handelt es sich um Ereignisse, die nicht nur lokal verarbeitet werden. Auf ein raumorientiertes System bezogen kann die Umgebung als Kontext für ein enthaltenes Objekt genutzt werden. Es handelt sich also um ein zweites Ereignis, das parallel in der Umgebung ausgelöst wird. Der Mechanismus der Verarbeitung unterscheidet sich lediglich durch Hinzunahme weiterer Parameter von dem initialen Ereignis. Dabei muss angezeigt werden, dass es sich um ein Kontextereignis handelt und es muss das ursprüngliche Objekt im Ereignis abzufragen sein. Die Vorgehensweise ist analog zu einer direkten Benachrichtigung mit einer Auswertung einer Liste der zu benachrichtigenden Objekte. In jedem Objekt wird z.B. die entsprechende Methode ausgeführt und daher unterschiedet sich die Situation nicht von der direkten Ereignisverarbeitung in einem Objekt.

Je nach Art der Verteilung der Objekte über die Server kann es von Vorteil sein die Umgebung eines Objekts auf dem gleichen Server zu speichern (vgl. Abschnitt 3.3.1). In diesem Fall wird für die Ausweitung eines Ereignisses auf den umgebenden Raum kein zusätzlicher Kommunikationsaufwand benötigt.

6.4.5 Globale Benachrichtigungen

Eine weitere Stufe der Ereignisverarbeitung stellen globale Ereignisse dar. Dabei löst jede Aktion ein Ereignis an einer zentralen Stelle im Server aus (Dispatcher). Analog zu den direkten Ereignissen in einem Objekt kann es sinnvoll sein eine Blockierung von Ereignissen in der ersten Phase zu ermöglichen (vgl. Abschnitt 6.1.8).

Eine Umverteilung dieser Ereignisse in den Verbund ist jedoch offensichtlich zu aufwändig. Jede Aktion würde zu Kommunikation innerhalb eines Verbunds führen, auch wenn kein entferntes Objekt betroffen ist. Gleichzeitig spricht die Autonomie einzelner Server gegen solche globalen Ereignisse im Verbund. In jedem Server wären die Aktionen zu überwachen, die auf anderen Servern des Verbunds stattfinden.

Es besteht jedoch weiterhin die Möglichkeit diese globalen Ereignisse auf den jeweiligen Server zu beschränken. Damit beziehen sich alle Ereignisse auf die lokalen Objekte, die sich auf einem Server befinden. Die Ereignisse selbst werden technisch innerhalb des Aufrufs einer Methode verarbeitet. Damit können lokale Module immer noch die Ereignisse innerhalb eines Servers kontrollieren. Bezogen auf eine Verteilung von Objekten gemäß der Räume eines raumbasierten CSCW-Systems erfolgt hier eine klare Strukturierung. Mit Gruppenarbeitsräumen als Wurzelelementen bedeutet dies prinzipiell eine Aufteilung in Abhängigkeit der Gruppen. Auf einem Server sind jeweils alle Gruppen mit abhängigen Strukturen gespeichert (dies entspricht der Verteilungsstrategie „Gruppen“, die in Abschnitt 3.3.1 beschrieben worden ist).

Auf der anderen Seite stehen zufällige Verteilungen von Objekten. In diesem Fall erscheint die Auswahl lokaler Ereignisse eher willkürlich, da sich verschiedene Objekte innerhalb eines Raums auf unterschiedlichen Servern befinden können.

Insgesamt können die Ereignisse also im Rahmen von Publish-/Subscribe-Mechanismen über den Verbund verteilt werden. Dazu wird das Proxy-Pattern verwendet, das einen Platzhalter für entfernte Objekte darstellt. Die nächste Frage, die sich stellt, ist die Art der Verteilung von Objekten im Serververbund.

6.4.6 Verteilte Objekte

Der Aufenthaltsort der Objekte spielt bei einer Verteilung von Objekten innerhalb des Verbunds eine Rolle (vgl. Abschnitt 3.3.1). Dabei ist die Zugriffszeit von Bedeutung: Ein Objekt, welches sich lokal auf einem Knoten befindet, kann dort wesentlich schneller bereitgestellt werden, als ein Zugriff auf ein entferntes Objekt erfolgt. Aus diesem Grund ist zwischen drei grundsätzlichen Strategien der Verteilung zu unterscheiden:

- Keine Verschiebung von Objekten: Es werden Objekte zwischengespeichert, um lesende Zugriffe zu beschleunigen (Caching). Ansonsten werden die Daten lediglich lokal gehalten. Dies bedeutet, ein Objekt befindet sich auf dem Server, auf dem es initial angelegt worden ist. Zu unterscheiden sind hier demnach ebenfalls Strategien der Erzeugung von Objekten. So kann sich die Erzeugung von Objekten bei einem raumbasierten System auf die aktuelle Umgebung beziehen, also z.B. auf den Aufenthaltsort eines Benutzers. Falls sich der Benutzer in einen Raum bewegt, der sich auf einem entfernten Server befindet, und dort ein Objekt erzeugt, so wird dieses Objekt in dem entfernten Server erzeugt.

- Verschiebung von Objekten zwischen Servern: Ein Objekt wird dahin verschoben, wo die meisten Zugriffe auf das Objekt erfolgen. Diese Vorgehensweise bietet bei den meisten Zugriffen eine sehr gute Performanz. Analog zu der Verteilung von Communities bei NetEffect (Das et al.

1997) kann der Raum an dieser Stelle als Kontext dienen und eine Verschiebung von Objekten von einem Raum in einen anderen führt dann eine Verschiebung auf einen anderen Server mit sich (vgl. Abschnitt 3.3.1). Es handelt sich also um eine räumliche Partitionierung oder, wenn die Gruppenstruktur zugrunde gelegt wird, um eine funktionale Verteilung von Objekten (Cai et al. 2002). Dieses Vorgehen bietet sich an, falls die Zugriffe von lokalen Gruppen auf die Gruppenräume und darunter liegende Strukturen vorherrschen.

- Replikation von Objekten: Replikate von Objekten werden erzeugt, sobald auf einem Server ein entferntes Objekt referenziert wird. Dadurch ist auch ein Arbeiten ohne Kontakt zu dem Server möglich, der das Original enthält. Die verschiedenen Replikate müssen allerdings synchronisiert werden. Insbesondere eine strenge Konsistenz erfordert eine unmittelbare Synchronisation der Zustände (vgl. Abschnitt 5.3.6). Bei der Replikation ist zwischen Attributen und dem Inhalt von Dokumenten zu unterscheiden. Während sich Attribute einfach synchronisieren lassen, sind Inhalte wesentlich aufwändiger in Einklang zu bringen.

Diese drei Strategien beschreiben lediglich eine grundsätzliche Vorgehensweise. Falls Objekte nicht zwischen Servern verschoben werden sollen, kann es später zu einer stärker verwobenen Struktur mit erhöhtem Kommunikationsaufwand kommen. Dies ist auf eine Verschiebung von Objekten zwischen Räumen zurückzuführen. Der physische Ablageort des Objekts bleibt dann zu jedem Zeitpunkt erhalten, obwohl das Objekt in einen Raum auf einem anderen Server eingefügt werden könnte. Auf diesen Raum bezogen bedeutet dies, dass sich dort ein nicht lokales Objekt befindet. Wird also der Inhalt des Raums angefordert, so erfordert dies eine Kommunikation mit dem entfernten Server, um die Daten des Objekts dort abzufragen.

Die Vor- und Nachteile der Verfahren ergeben sich deshalb wie folgt:

- Lokalität von Objekten:
  - Vorteile: Die anfängliche Struktur ist sehr übersichtlich und es ist kein zusätzlicher Aufwand nötig, um die Objekte auf einen anderen Server zu übertragen.
  - Nachteile: Die Strukturen können bei der Verschiebung von Objekten sehr heterogen in einem hohen Grad verteilt werden, sodass

innerhalb eines Raums lokale Objekte und entfernte Objekte eines Servers zu finden sind. Daher kann der Kommunikationsaufwand bei der Abfrage des Inhalts eines Raums größer sein.

- Verschiebung von Objekten:
  - Vorteile: Zugriffe auf Objekte sind sehr effizient und Modifikationen werden robust ausgeführt.
  - Nachteile: Die Übertragung von Objekten kann aufwändig sein und durch die Verschiebung der Objekte sind bereits existierende Proxys auf anderen Servern, die auf das Objekt verwiesen haben, nicht mehr gültig. Daher müssen diese Proxys aktualisiert werden.
- Replikation von Objekten:
  - Vorteile: Verbesserung der Verfügbarkeit und Zugriffszeiten.
  - Nachteile: Die Synchronisation von Objektzuständen muss immer gewährleistet sein und ist aufwändig umzusetzen.

Bei einer Verschiebung von Objekten muss wieder zwischen zwei Strategien unterschieden werden:

- Räumlich/Zonen: Die Aufteilung der Objekte erfolgt räumlich entlang von Kontinenten oder Zonen.
- Gruppenverteilung: Gruppenstrukturen und zugehörige Objekte werden jeweils einem Server zugeordnet. Dadurch sind alle einer Gruppe zugeordneten Objekte lokal auf einem Server zu finden.

Wenn ein Objekt grundsätzlich von einem Server auf einen anderen verschoben werden soll, muss ein gewisses Vertrauensverhältnis in dem Verbund existieren. Darüber hinaus greifen in jedem Fall Berechtigungen von Räumen und Benutzern.

Im folgenden Abschnitt wird eine konkrete Strategie beschrieben, die lokale Strukturen bevorzugt.

6.4.7 Verteilungsstrategie: Räumliche Verschiebung von Objekten

Diese Strategie beinhaltet eine Verschiebung von Objekten zwischen den Servern, sodass alle Objekte innerhalb eines Wissensraums lokal sein sollen —

sich also auf dem gleichen Server wie der Raum befinden. Damit wird eine stark verwobene Struktur weitestgehend verhindert. Ein Bewegen eines Objekts kann so immer eine Verschiebung von einem Server auf einen anderen beinhalten.

Die *Ist-enthalten* Relation führt schließlich dazu, dass auch Räume innerhalb von Räumen lokal sind und damit sind die gesamten Baumstrukturen für eine Gruppe, ausgehend von dem Gruppenarbeitsraum, auf einen Server begrenzt. Wird ein Dokument von einem Raum in einen anderen Raum verschoben und liegt der Zielraum in einer anderen Baumstruktur als der Ausgangspunkt, so kann es zu einer Verschiebung des Objekts von einem Server auf einen anderen kommen.

Eine Partitionierung des Servers erfolgt dadurch ähnlich dem in Das et al. (1997) vorgestellten Ansatz der unterschiedlichen Gemeinschaften (Communities). Auf einem Server liegen eine oder mehrere virtuelle Gemeinschaften und sämtliche dazugehörenden Dokumente und Objekte. Eine einzelne Gemeinschaft ist also niemals zwischen verschiedenen Servern verteilt. Im Kontext einer Verteilung von Wissensräumen ist dies im Prinzip identisch zu einer geografischen Verteilung, da die Raumstrukturen direkt von Gruppen abhängig sind (vgl. Abschnitt 3.1.3). Die eigentliche Speicherung von Objekten wird durch die einzelnen Persistenzebenen vorgenommen (vgl. Abschnitt 7.6). Diese sind wiederum durch einen Persistenzmanager gekapselt, der die einem Objekt zugeordnete Persistenzebene ermittelt und die Verarbeitung dahin umleitet.

Durch die Verschiebung von Objekten von einem Server auf einen anderen wechseln diese ihre SID und sind damit nicht mehr auf dem entsprechenden Server lokalisierbar. Eine Lösung dieses Problems ist die Verwendung von Proxy-Ketten. Nach einer Verschiebung eines Objekts wird dazu auf dem Ausgangsserver ein Proxy mit Informationen über den Verbleib des Objekts hinterlassen. Ein anderer Proxy, der ursprünglich auf das Original verwiesen hat, findet nun einen weiteren Proxy und kann die Kette bis zum Objekt verfolgen.

Abbildung 6.13 zeigt eine Kette von Proxys, die sich auf unterschiedlichen Servern befinden. Zunächst besteht lediglich eine direkte Verknüpfung zu einem Objekt, das sich auf einem entfernten Server befindet. Nach der Verschiebung dieses Objekts auf einen weiteren Server wird ein weiterer Proxy dazu geschaltet und es entsteht eine Kette. Die ursprüngliche Verknüpfung von Server A besteht dadurch nicht mehr als direkte Verknüpfung. Bei einem Zugriff auf das Objekt kann die Kette jedoch wieder um das Zwischenelement reduziert werden und der Proxy weist wieder direkt auf das Objekt, welches sich nun

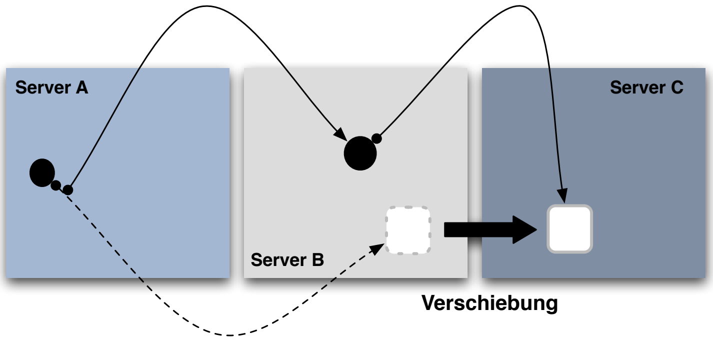


Abbildung 6.13: *Verkettung von Proxys bei einer Objektverschiebung*

auf einem anderen Server befindet.

Aus Gründen der Effizienz wird diese Proxy-Kette für weitere Zugriffe aufgelöst werden, sodass die Verweise aller Proxy-Objekte dieser Kette so aktualisiert werden, dass sie wieder auf das eigentliche Objekt verweisen. Der nächste Zugriff erfolgt dann wieder direkt über einen einzelnen Serverwechsel auf das Original.

6.4.8 Gateway-Server

Aus dem Bereich der Massive Multiplayer Online Games stammen Architekturen, die einen Gateway-Server vor ein Servercluster schalten, der zur Kommunikation mit den Clients dient. Diese Vorgehensweise kann für einen statischen Serververbund adaptiert werden, sodass die gesamte Kommunikation über diesen Server abgewickelt wird. Es handelt sich um eine zusätzliche Alternative für eine Architektur verteilter Wissensräume, die einen Cluster voraussetzt. Der Vorteil dieser Architektur liegt in einer einfacheren Umsetzung.

Im Vergleich zu Load-Balancing–Ansätzen von Webservern muss ein Gateway-Server im Bereich eines kollaborativen Systems zusätzliche Aufgaben bewältigen. Der Server nimmt Anfragen entgegen und ermittelt daraus, auf welche Strukturen zugegriffen wird. Die Objekte liegen verteilt auf den verschiedenen Serversystemen im Hintergrund vor. Diese Vorgehensweise wird von Cai et al. (2002) beschrieben und dort als Front-End- (zuständig für die Anfragen der Clients) und Back-End-Server beschrieben.

Abbildung 6.14 zeigt einen Verbund von Servern mit einem zentralen Gateway, der alle Zugriffe auf diesen Verbund kontrolliert. Es handelt sich um einen statischen Verbund. Die Server kommunizieren im Gegensatz zu den üb-

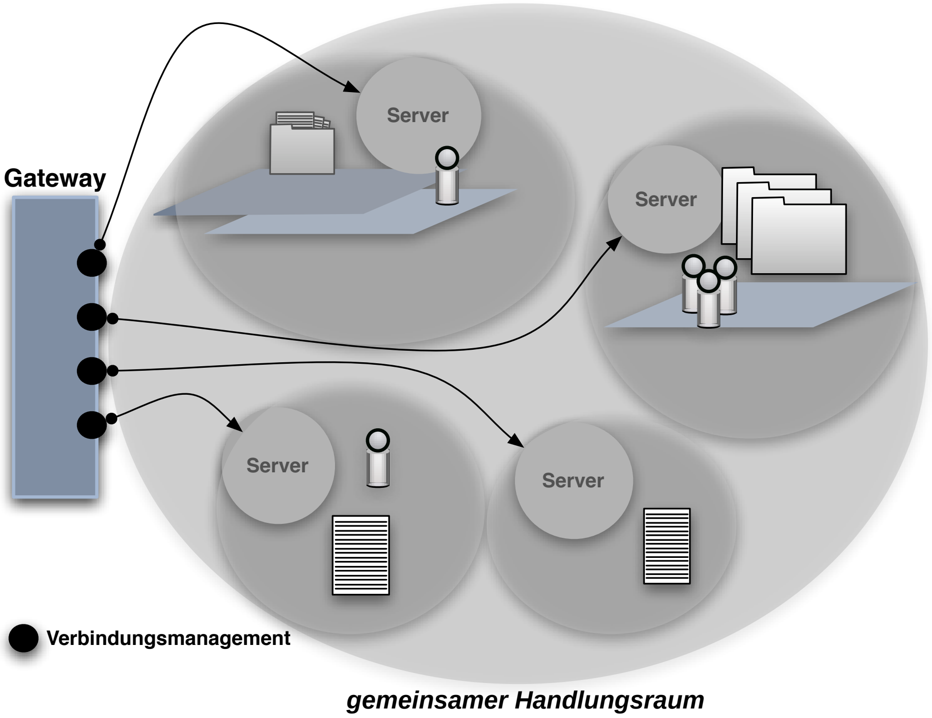


Abbildung 6.14: *Verbund mit zentralem Gateway*

rigen Architekturen nicht direkt untereinander. Stattdessen wird die gesamte Kommunikation über den Gateway geregelt.

6.4.9 Verteilte Benutzer, Gruppen und Berechtigungen

Mit direkten Kommunikationskanälen zwischen den Servern ist es nicht notwendig die Benutzer und Gruppen in einen zentralen Datenraum zu speichern. Stattdessen bietet der Serververbund mit dem gegenseitigen Zugriff auf Objekte einen gemeinsamen Datenraum, der alle Objekte umfasst. Benutzer und Gruppen können damit ebenso wie alle anderen Objekte behandelt werden und über Remote-Proxys auch auf entfernten Servern referenziert werden.

In diesem Kontext sind auch die Berechtigungen zu sehen, die jeweils an Objekte gebunden sind und den Benutzern und Gruppen zugeordnet werden. Hier werden für entfernte Benutzer/Gruppen ebenfalls Remote-Proxys eingetragen.

6.4.10 Bewertung der Architektur

Die beschriebene Architektur eines Serververbunds mit Handlungsraum setzt lediglich einen Teil der Anforderungen um und verfügt noch nicht über alternative Infrastrukturen wie Peer-to-Peer. Es wird ein statischer Verbund erzeugt, der einen gemeinsamen Handlungsraum für die Nutzer dieses Verbunds bereitstellt. Die verwendenten Verfahren sind nicht unbedingt auf eine Optimierung der Zugriffe ausgelegt, sondern orientieren sich an bestehenden Lösungen. Im Gegensatz zu den vorherigen Architekturen gelingt jedoch ein Austausch von Ereignissen und Objekten. Damit sind wesentliche Voraussetzungen kooperativen Arbeitens gegeben: Objekte werden auf Basis von Ereignissen synchronisiert und Zustandsänderungen können anderen Benutzern angezeigt werden. Gleichzeitig stehen Rucksack und Arbeitsraum eines Benutzers serverübergreifend zur Verfügung. Weiterhin ist eine Autonomie der einzelnen Server gegeben, da die lokalen Strukturen erhalten bleiben. Eine Autorisierung kann auf Basis eines Proxy als Verweis auf eine entfernte Gruppe oder einen Benutzer erfolgen.

Die Antwort auf die Frage einer möglichen Auflösung des Verbunds gestaltet sich für diese Architektur wesentlich komplizierter. Dies ist auf die starken Abhängigkeiten der Objekte unter der Verwendung von Remote-Proxys zurückzuführen. In Abhängigkeit von der Verteilungsstrategie von Objekten können diese nicht mehr aufgelöst werden. Eine Replikation der Zustände kann hier von Vorteil sein, sodass die Anfragen weiterhin beantwortet werden können. Allerdings beinhalten Räume weitere Objekte, die ebenfalls repliziert werden müssen. Dies hat zur Folge, dass ein replizierter Raum alle unterliegenden Strukturen mit replizieren muss. Gleichzeitig gefährdet dies die Konsistenz der Daten, falls ein Server wieder zu dem Verbund zurückkehrt. Bei einer endgültigen Ablösung kann auf replizierten Objekten weitergearbeitet und dadurch können eigene Zustände erzeugt werden.

6.5 Architektur 4: Netzwerkverbund

Eine vollständig transparente Umgebung verteilter Wissensräume als Verbund von statischen und temporären Knoten mit gemeinsamem Handlungsraum ist für ein kollaboratives System anzustreben. Im Gegensatz zu Architektur 3 handelt es sich um einen dynamischen Verbund, der automatisch durch die Teilnahme verschiedener Knoten an diesem Netzwerk entsteht (vgl. Abschnitt 3.4).

Ist dieser Zugang nicht beschränkt, so ist es notwendig Daten zu verschlüsseln und mit entsprechenden Berechtigungen zu versehen, da kein Vertrauensverhältnis vorausgesetzt werden kann. Auf Basis von Zugriffsrechten und Rollen kann ein kooperatives Arbeiten im Rahmen von Wissensräumen erfolgen.

Eine Peer-to-Peer-Infrastruktur bietet sich als Grundlage für einen solchen Verbund an. Im Gegensatz zu klassischen Client/Server-Systemen ist dabei jeder Knoten im Prinzip gleichberechtigt, wobei nicht unbedingt die gleichen Voraussetzungen und Kapazitäten gegeben sein müssen. Aus diesem Grund gibt es bereits eine Unterscheidung zwischen Peer und Superpeer (vgl. Mizrak et al. 2003), der über erweiterte Kapazitäten verfügen kann oder zusätzliche Aufgaben im Netzwerk hat. Dazu kann die Bereitstellung eines Webzugangs in das Peer-to-Peer-Netzwerk gehören.

Diese Sichtweise von Peers und Superpeers spiegelt eigentlich wiederum eine Art von Client- und Serversystemen wider. Der entscheidende Unterschied liegt jedoch darin, dass jeder Peer über die gleichen Grundfunktionen verfügt. Auf ein kollaboratives System mit Client und Server übertragen müsste der Client nun ebenfalls über die Funktionalität des Servers verfügen. Die einzelnen Knoten kommunizieren im Peer-to-Peer-Netz direkt miteinander und sind nicht unbedingt von einem Superpeer abhängig. Der Schritt von Client/Server zu Peer-to-Peer erlaubt es, die bisherigen Clients als Peers in das Netzwerk zu integrieren. Ein konkretes Beispiel dafür ist ein Whiteboard, welches bisher über den Server mit anderen Teilnehmern/Whiteboards kommuniziert hat. Im Peer-to-Peer-Netzwerk ist das Whiteboard lediglich ein weiterer Knoten im Netzwerk, der über die erforderlichen Grundeigenschaften verfügt und damit dem Prinzip von Peer-to-Peer gerecht wird. Aus diesem Grund kann überall das gleiche Basissystem verwendet werden, sodass eine Architektur basierend auf einem Microkernel sinnvoll ist (vgl. Abschnitt 5.4.2). Dieser bietet für alle Knoten eine einheitliche Grundlage, die mit verschiedenen Modulen erweitert werden kann.

Abbildung 6.15 zeigt einen Verbund mit verschiedenen Knoten, die zusammen einen gemeinsamen Handlungsraum bieten. Dies gelingt durch eine generelle Verfügbarkeit der Objekte. Der dynamische Charakter des Verbunds wird durch einen Peer, der zu dem Verbund hinzukommt (Peer V) und einem Peer, der den Verbund verlässt (Peer T) illustriert. Die Verbindungen zwischen den Knoten entspricht der Routingtabelle des Peer-to-Peer-Netzwerks und kann nicht direkt den Verbindungen im physikalischen Netzwerk zugeordnet werden. Neben drei normalen Peers sind zwei Superpeers am Netzwerk

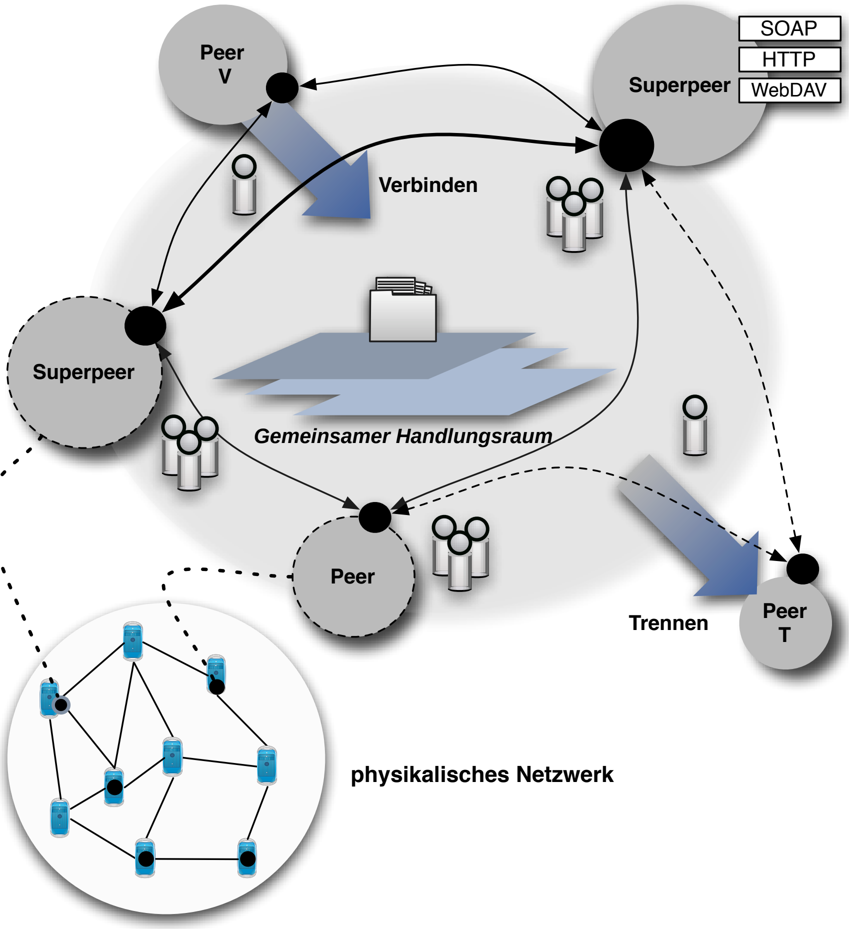


Abbildung 6.15: *Hybrider Serververbund mit Peer und Superpeer*

angeschlossen, die über umfangreichere Ressourcen verfügen und zusätzliche Aufgaben bewältigen müssen. So sind dem oben rechts abgebildeten Peer verschiedene Protokolle zugeordnet, die einen Zugriff mit herkömmlichen Clients in das Peer-to-Peer-Netzwerk ermöglichen. WebDAV erlaubt es neue Dokumente auf dem Server einzufügen, SOAP ist ein verbreitetes Protokoll zur Unterstützung von Diensten (Webservices, vgl. Abschnitt 5.7.2) und HTTP bietet einen Zugang mit dem Webbrowser.

Diese Architektur setzt die folgenden Anforderungen um (vgl. Abschnitt 3.4):

- **Gemeinsamer Handlungsraum eines Serververbunds**: Der Datenraum der Server überschneidet sich vollkommen, sodass sich Benutzer beliebig durch die virtuelle Umgebung bewegen und Daten ausgetauscht werden können.

- **Dynamischer Verbund**: Es können beliebig Server zu dem Verbund hinzukommen oder diesen wieder verlassen. Unterschieden wird zwischen temporären und statischen Knoten des Netzwerks.

Die Umsetzung einer solchen Architektur beinhaltet vielschichtige Problemstellungen. Die Verwendung einer existierenden Peer-to-Peer-Infrastruktur als Grundlage zur Verteilung von Objekten bietet sich an (vgl. Bopp et al. 2005a). Wie sich jedoch zeigt, sind die existierenden Systeme nicht als verteilte Object Repository ausgelegt, sondern fokussieren auf einen Austausch von Dateien, Ressourcen oder Diensten. Während die Objekte noch in serialisierter Form in einem solchen Dateisystem abzulegen sind, gestaltet sich der Austausch von Ereignissen schwierig. Nur über Umwege lässt sich hier eine synchrone Kommunikation realisieren. Aufgrund dessen ist es sinnvoller eine Peer-to-Peer-Infrastruktur, wie Pastry (Rowstron und Druschel 2001a) oder Chord (Stoica et al. 2001), zugrunde zu legen.[14] Dabei handelt es sich um Distributed Hash Tables (DHT), die dazu dienen Daten auszutauschen. Jede Nachricht wird von einem Knoten zu einem anderen Knoten übermittelt, dessen ID der ID der Nachricht am meisten entspricht. Da diese Systeme über ein eigenständiges Routing verfügen, werden sie auch als Overlay-Netzwerke bezeichnet.

Der Austausch von Objekten zwischen den Servern erfolgt durch den Einsatz verschiedener Persistenzebenen und IDs von Objekten, die ein Auffinden des Objekts im Verbund erlauben. Diese Ebene wird im Idealfall vollständig abstrahiert, sodass eine kollaborative Applikation auf den Objekten ohne Kenntnis über den Speicherort arbeiten kann. Als Basistechnologie einer solchen Persistenzebene unterstützen DHTs eine solche Zuordnung von ID und Objekt. Objekte werden dabei verteilt über das Netzwerk abgelegt, sodass lokale Manipulationen nicht ausgeschlossen werden können. Dazu verwenden Systeme wie OceanStore (Kubiatowicz et al. 2000) Public-Key-Verfahren, um die Daten vor Zugriffen zu schützen. Der Nachteil dieser Vorgehensweise liegt in einer Notwendigkeit zentraler Server, die die Zertifikate verwalten müssen (Rhea et al. 2003).

Des Weiteren sind Erweiterungen eines DHT vorzusehen, um einen Handlungsraum zu ermöglichen. Dazu sind Ereignisse über die einzelnen Knoten

---

[14] Auch JXTA bietet eine verteilte Infrastruktur, die für eine Umsetzung verteilter Wissensräume grundsätzlich in Frage kommt.

auszutauschen. Auch die generelle Verwaltung von Berechtigungen und die Suchfunktionen sind Bereiche, die nicht direkt in einem DHT bereitgestellt werden.

Abgesehen von diesen grundlegenden Betrachtungen ist die Rate der Knoten, die das Netz verlassen oder hinzukommen, zu beachten. Dieser so genannte Churn ist entscheidend für die Stabilität des Netzwerks (vgl. Rhea et al. 2004). Bestehende DHT-Implementierungen sind nicht unbedingt sehr resistent gegen eine hohe Rate, sodass in Abhängigkeit vom Aufbau des Netzes dieses in Betracht gezogen werden muss. Dazu muss das Verhältnis von festen Knoten zu temporären Knoten ermittelt und die Dauer der temporären Knoten im Netzwerk mit einbezogen werden.

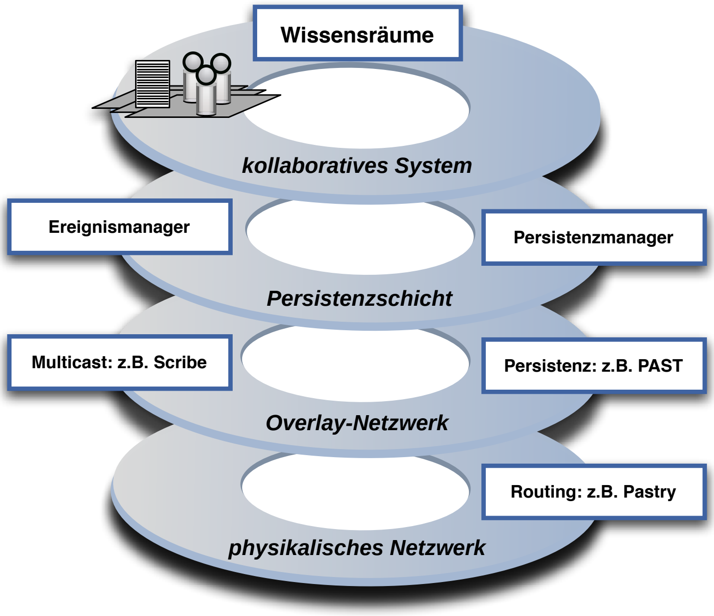


Abbildung 6.16: *Aufbau eines Overlay-Netzwerks*

Abbildung 6.16 zeigt eine Einteilung der Architektur in verschiedene Ebenen. Dabei bildet das Overlay-Netzwerk eine Schicht zwischen der Persistenz des kollaborativen Systems und dem physischem Netzwerk, die über einen Persistenzmanager (vgl. Abschnitt 5.3) eingebunden werden kann.

Pastry (Rowstron und Druschel 2001a) bildet in der skizzierten Architektur beispielsweise das Overlay-Netzwerk und übernimmt das Routing von Nachrichten zwischen den Knoten. Berücksichtigt wird hierbei auch der Ausfall von

Knoten. Als Erweiterung dieser Technologie sind Systeme wie Scribe zu sehen. Es handelt sich hierbei um ein ereignisbasiertes Nachrichtensystem, welches ein Multicasting im Netzwerk realisiert. Dieses ist für eine effiziente Verbreitung von Ereignissen im Verbund notwendig.

6.5.1 Identifikation von Knoten

Die Idee eines gemeinsamen Handlungsraums für Benutzer aller Server kann durch ein hierarchisches Peer-to-Peer-Netz realisiert werden, bei dem es verschiedene Knoten gibt. Diese zeichnen sich durch eine unterschiedliche Konfiguration aus und statische Knoten verfügen über mehr Module und Funktionen, die den anderen Teilnehmern bereitgestellt werden.

Jeder Knoten des Netzes verfügt über eine eigene Verwaltung von Objekten, Benutzern und Gruppen. Alle Objekte werden über den Verbund verteilt und stehen somit im gesamten Verbund zur Verfügung.

Die Notwendigkeit der eindeutigen Identifikation ergibt sich aus den Zugriffen auf Objekte durch entsprechende IDs. Ein Server, der sich hier als ein anderer Server ausgibt, kann damit beliebige Objekte fälschen und z.B. auch Gruppen oder Benutzer durch eigene austauschen.

Knoten werden in Pastry mit einer 128-Bit ID identifiziert, die mit einer Hashfunktion ermittelt wird. Objekte werden ebenfalls im gleichen Bereich mit 128-Bit adressiert, sodass jedes Objekt einem Knoten zugeordnet werden kann.

6.5.2 Speicherung von Objekten

Objekte werden im Sinne des DHT in Abhängigkeit von der OID verteilt gespeichert. Damit werden Objekte auf dem Knoten im Netzwerk gespeichert, dessen ID am nächsten zur OID liegt. Es handelt sich also um eine zufällige Verteilung, die durch eine Hashfunktion bestimmt wird (vgl. Abschnitt 3.3.1 auf Seite 49). Aufgrund der Gegebenheiten im Netzwerk ist eine Replikation des Objekts unbedingt notwendig, da der Speicherort nicht von überall gut erreichbar ist. Hier kommen Systeme wie PAST zum Einsatz, welche die Eigenschaften des darunter liegenden DHTs ausnutzen. Dabei werden die Daten auf die benachbarten Knoten repliziert, da sich im Pastry-Netzwerk benachbarte Knoten in der Regel nicht in physischer Nähe zueinander befinden (vgl. Rowstron und Druschel 2001b). Damit ist eine gute Verfügbarkeit der Daten gegeben. Gleichzeitig erfolgen nachfolgende Zugriffe auf unterschiedliche Replikate, sodass eine Konsistenz der Daten bei schreibenden Aktionen nicht unbedingt gegeben ist (vgl. ACID-Eigenschaften von Datenbanken im Abschnitt 5.3.5). Aus diesem Grund wird von Knutsson et al. (2004) ein Koordinationsknoten für die Verwaltung von Regionen in einem Massive Multiplayer Online Game vorgeschlagen. Diese Idee lässt sich direkt auf die Räume eines raumbasierten kollaborativen Systems adaptieren. Aus den Daten des Raums wird mit Hilfe einer kollisionsfreien Hashfunktion ein Wert ermittelt und der Knoten mit dieser ID im Netzwerk als Koordinationsknoten für diesen Raum und alle enthaltenen Objekte verwendet.

6.5.3 Zugriff auf Objekte

Der Zugriff auf Objekte erfolgt zunächst auf dem gleichen Weg, wie die Objekte auch gespeichert werden. Durch die Anwendung einer Hashfunktion kann die ID des Objekts ermittelt und darüber auch der Speicherort bestimmt werden. Dieses Verfahren hat den Nachteil, dass die Ausgangsdaten bekannt sein müssen, damit die Hashfunktion angewendet werden kann um das Objekt zu finden. Aus diesem Grund sind neben diesem direkten Zugriffsverfahren weitere Möglichkeiten des Zugriffs zu berücksichtigen:

- Auf Basis von Daten wird eine Hashfunktion angewendet, die zu der ID eines Objekts weist.
- Die Suche nach einem Schlüsselwort führt zu einem Objekt bzw. zu der ID des Objekts.
- Ein Objekt beinhaltet Verweise auf weitere Objekte, die in Form der ID gespeichert sind (vgl. Abschnitt 5.3).

Für Wissensräume ist insbesondere die letzte Variante des Zugriffs von Bedeutung, da in einem Raum mehrere Objekte enthalten sein können (vgl. Abschnitt 6.1.5). Nach einer erfolgreichen Anmeldeprozedur steht ein Benutzerobjekt als Ausgangsbasis zur Verfügung. Sämtliche Verweise dieses Objekts können zunächst ausgewertet werden. Zu nennen sind hier Mitgliedschaft in Gruppen und die Umgebung des Benutzers (vgl. Abschnitt 6.1.1). Mit dem Raum können wiederum sämtliche enthaltenen Objekte geladen werden.

Die Suche nach Objekten ist darüber hinaus eine Grundfunktion in einem kollaborativen System. Aufgrund der Gegebenheiten in einem DHT-Netzwerk ist diese Suche jedoch nicht direkt möglich. Als hybrides Peer-to-Peer-Netzwerk sind Superpeers (vgl. Abschnitt 3.4) für diesen Aufgabenbereich geeignet. Eine Zuordnung von Schlüsselwort zu der ID des Objekts wird hier gespeichert und anfragenden Peers zur Verfügung gestellt.

6.5.4 Synchronisierung

Die Verwendung von mehreren Replikaten erfordert eine Synchronisierung der Objektzustände (vgl. Abschnitt 5.3.6). Während PAST (Rowstron und Druschel 2001b) lediglich das Ablegen von Dateien ohne spätere Modifikationen vorsieht, ist es für verteilte Wissensräume unbedingt erforderlich Dokumente jederzeit auch zu ändern. Das SimMud (Knutsson et al. 2004) System sieht dafür einen Koordinationsknoten vor, dem sämtliche Replikate bekannt sind. Dieser Knoten regelt Zugriffe auf ein Objekt und sorgt für eine Einhaltung der ACID-Eigenschaften (vgl. ACID-Eigenschaften in Abschnitt 5.3.5). Bei einem Ausfall dieses Koordinationsknotens übernimmt der Knoten mit numerisch ähnlichster ID die Koordination.[15] Dazu repliziert der Koordinationsknoten vorher bereits die für eine Koordination notwendigen Daten an seine Nachbarn. Falls ein neuer Knoten hinzugefügt wird, dem durch die Nachrichtenzuordnung im DHT-Netzwerk die Rolle eines Koordinationsknotens zukommt und dem keine Informationen über den Raum vorliegen, fordert dieser neue Knoten die relevanten Daten von dem bisherigen Koordinationsknoten an und übernimmt dessen Aufgabe.

Die Synchronisierung der Objektzustände erfolgt mit Ereignissen. Darüber können Änderungen im Netzwerk verteilt und alle Replikate auf dem aktuellen Stand gehalten werden. Im Prinzip entsprechen die Ereignisse über Zustandsänderungen von Objekten einem Protokoll von Aktualisierungen der Objekte, die im Bayou-System in geordneter Form für eine Konsistenz der Daten sorgen (vgl. Petersen et al. 1997).

6.5.5 Ereignisse

Eine Komponente eines kollaborativen Systems stellt eine umfangreiche Ereignisverarbeitung dar (vgl. Abschnitt 5.6). Im Gegensatz zu einem klassischen Client/Server-System müssen in einer Peer-to-Peer-Umgebung die Ereignisse

---

[15] Aufgrund der Eigenschaften des DHT werden alle Nachrichten des ausgefallenen Knotens nun diesem Knoten zugestellt.

über mehrere Knoten ausgetauscht werden. Bezogen auf das Publish/Subscribe-Paradigma müssen Ereignisse überall hingelangen, wo Abonnements auf bestimmte Objekte und Ereignisse bestehen.

Mit Scribe (Rowstron et al. 2001) steht eine Multicast-Infrastruktur[16] zur Verfügung, die es innerhalb von Pastry erlaubt Nachrichten auf kurzen Wegen zu mehreren anderen Knoten im Netzwerk zu senden. Auf Basis dieser Infrastruktur ist es möglich auch Ereignisse effizient im Netzwerk zu verteilen. Dabei werden bestimmte Knoten des Netzwerks zu einem Multicast-Baum zusammengefasst, indem jeder Knoten des Baums eine Multicast-Liste speichert. Die Wurzel dieses Baums wird von einem Rendezvousknoten gebildet. Es handelt sich dabei um einen Knoten, der unter Verwendung einer Hashfunktion einem bestimmten Ereignis zugeordnet wird. Im raum- und objektorientierten kollaborativen System kann eine Abbildung der Ereignisse auf solche Rendezvousknoten so erfolgen, dass alle Ereignisse der Objekte innerhalb eines Raums auf denselben Rendezvousknoten abgebildet werden. Gleichzeitig verwenden alle Abonnenten dieselbe Zuordnung und gelangen so ebenfalls zu dem Rendezvousknoten. In Scribe wird die Ereignisverarbeitung über die Generierung von Themen vorgenommen, die innerhalb eines objektorientierten Systems über Objekte definiert werden.

Die Liste der Abonnentenknoten und die Knoten des Multicast-Baums sind unabhängig voneinander. Bei dem Abonnement eines Themas wird der Baum allerdings so aktualisiert, dass eine Route zu dem neuen Abonnenten hinzukommt. Dies gelingt durch eine schrittweise Anpassung der Multicast-Informationen in jedem Knoten (Nachfolgertabelle) während des Routingvorgangs von dem Rendezvous-Peer zu dem Abonnentenknoten durch die Übermittlung einer speziellen Nachricht. Das Abbestellen eines Themas gelingt auf ähnliche Weise durch die Übermittlung einer Abbestellnachricht durch den Multicast-Baum. Während des Routings dieser Nachricht werden alle Nachfolgertabellen modifiziert, sodass sie den Zielknoten nicht mehr enthalten.

6.5.6 Übersicht der Funktionsweise

Das Peer-to-Peer Overlay-Netzwerk arbeitet mit verschiedenen Erweiterungen des Pastry-DHTs. Für eine raumbasierte Infrastruktur sind eine verteilte Speicherung von Objekten mit Synchronisierung und eine Ereignisverarbeitung

---

[16] Eine Adressierung von mehreren gleichzeitigen Zielen wird als Multicast bezeichnet.

notwendig. Zur Optimierung bei Zugriffen wird Replikation mit dem PAST-System eingesetzt.

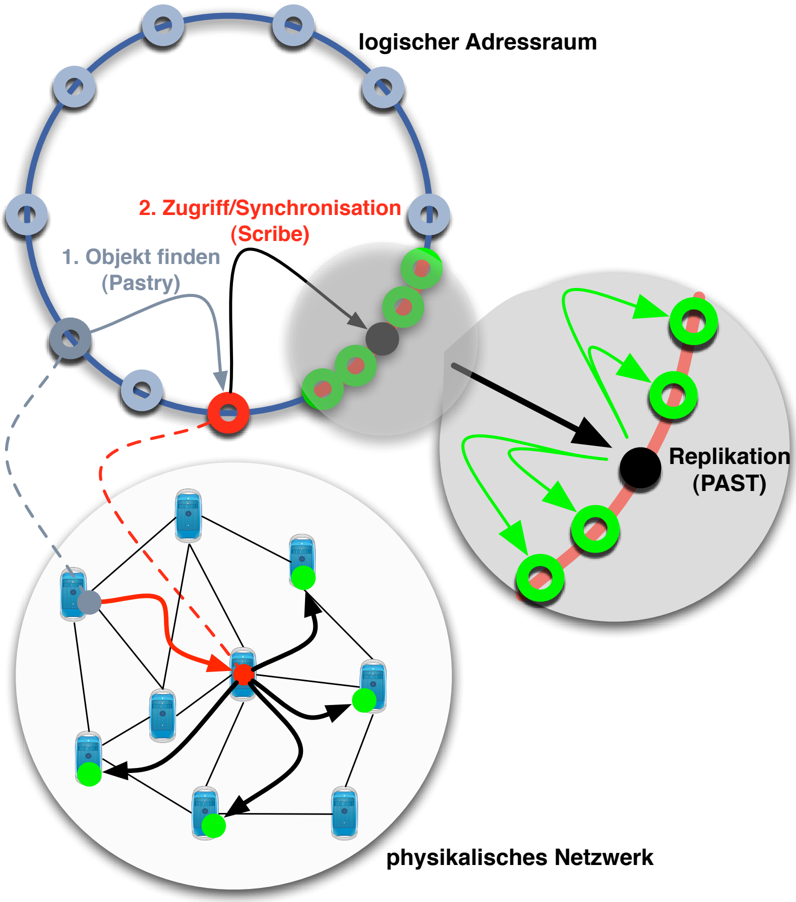


Abbildung 6.17: *Zugriff im DHT*

Abbildung 6.17 zeigt das Zusammenspiel der verschiedenen Systeme. Im ersten Schritt wird ein Objekt mit Hilfe von Pastry im Overlay-Netzwerk gefunden. Für dieses Objekt existieren mehrere Replikate. Der Zugriff wird über Scribe gesteuert, sodass alle Replikate synchronisiert werden. Diese befinden sich im Leaf Set des Knotens, der ein Objekt speichert.

6.5.7 Bewertung der Architektur

Die Architektur setzt einen gemeinsamen Handlungsraum um, der sämtliche Objekte umfasst und die Verfügbarkeit und Synchronisierung der Objekte gewährleistet. Der Verbund nutzt dabei die Eigenschaften des darunter liegenden DHTs, der Nachrichten über kurze Wege versendet und eine Selbstorganisation aufweist. Eine Synchronisierung gelingt mit Hilfe von Koordinationsknoten. Damit wird eigentlich eine Art Client/Server-Prinzip im Peer-to-Peer-Netz angewandt, um in diesem Bereich eine verbesserte Robustheit zu erreichen.

Eine Anforderung stellt die Sicherheit des Verbunds dar. Hier enthalten ist eine Verschlüsselung der Daten und die Authentisierung der Knoten im Verbund. An dieser Stelle greifen im Idealfall wiederum die Eigenschaften der verwendeten Peer-to-Peer-Infrastruktur. Aufgrund der zufälligen Verteilung der Objekte im Netzwerk muss der Zugriff geschützt werden. Dies gilt sowohl im Bereich eines Zugriffs über die Infrastruktur als auch über direkte lokale Zugriffe auf gespeicherte Daten. Das PAST-System (Rowstron und Druschel 2001b) verwendet dazu ein Public-Key-Verfahren und eine Verschlüsselung sämtlicher Inhalte. Zugriffe erfolgen dann über Zertifikate, die an den Speicherorten der Replikate geprüft werden müssen.

Den Nachteil dieser Architektur stellt der Aufwand der Implementierung dar, da z.B. die Speicherung von Objekten, Suchfunktionalität und Verteilung von Ereignissen von Grund auf neu entwickelt werden muss. Mit Systemen wie Pastry, Scribe und PAST stehen lediglich grundlegende Komponenten einer Umsetzung zur Verfügung, die geeignet verwendet werden müssen.

Gleichzeitig stellt sich die Frage, inwieweit die Stabilität eines kleinen Peer-to-Peer-Netzwerks gegeben ist, da der gleichzeitige Ausfall mehrerer Knoten die Verfügbarkeit von Objekten beeinträchtigen kann, falls die Anzahl der ausfallenden Knoten groß gegenüber dem Gesamtnetzwerk ist. Eine Möglichkeit dem entgegenzuwirken ist es, möglichst viele feste Knoten (Superpeers) in das Netz zu integrieren und so eine stabile Infrastruktur bereitzustellen.

6.6 Zusammenfassung

Die verschiedenen Architekturen decken die Anforderungen unterschiedlich ab. Ausgehend von Architektur 1 wird der geteilte Datenraum schrittweise erweitert bis hin zu einem Handlungsraum für die Benutzer in einer verteilten Umgebung. Gleichzeitig nimmt der Aufwand einer Umsetzung der Architekturen zu, sodass Architektur 4 eine von Grund auf neue Implementierung eines kollaborativen Systems erfordert. Dabei liegt eine neue Technologie zu Grunde, die sehr viel versprechende Ansätze bietet. Als kleine Einschränkung muss an dieser Stelle angeführt werden, dass bisher nur wenige Anwendungen existieren und damit die Erfahrungen mit DHTs in der Alltagspraxis eher gering sind.

Im Gegensatz dazu stehen die ersten drei Architekturen, die als Erweiterungen bestehender Client/Server-Architekturen gedacht sind und Übergänge zwischen verschiedenen Servern ermöglichen. Dabei liegt der Fokus auf serverübergreifende Benutzer, Gruppen und Objekte.

Im folgenden Kapitel werden verschiedene Entwurfsmuster vorgestellt, die innerhalb dieser Arbeit identifiziert worden sind. Obwohl diese bereits in den jeweiligen Architekturen berücksichtigt wurden, ist eine allgemeinere Form der Darstellung bei der Umsetzung einer Architektur verteilter Wissensräume von Bedeutung.

7 Entwurfsmuster verteilter Wissensräume

Dieses Kapitel beschäftigt sich mit den Entwurfsmustern von verteilten Wissensräumen. Dabei sind die meisten identifizierten Muster nicht ausschließlich für Wissensräume relevant, sondern beziehen sich ebenso auf allgemeine Multi-Server-Architekturen und verteilte Systeme.

Komplexe Multiserver-Umgebungen sind aufwändig zu analysieren, da die Verteilung von Dokumenten und Räumen für Benutzer transparent erfolgt. Weiterhin können bereits bei einfachen Abläufen mehrere Server miteinander kommunizieren, wodurch es schwierig wird die Abläufe genau nachzuvollziehen und herauszufinden, auf welchem Server ein Fehler aufgetreten ist. Verschiedene Techniken und Entwurfsmuster sind hilfreich, um Fehler aufzuspüren oder bereits im Vorfeld zu vermeiden. Besonders wirksam sind automatisierte Tests, die Abläufe simulieren und das Ein- und Ausgabeverhalten der Methoden kontrollieren. Bei Änderungen des Quelltextes kann so sichergestellt werden, dass das Verhalten der Methoden unverändert ist.

Die Anforderungen an eine Umsetzung verteilter Wissensräume sind in den vorherigen Kapiteln beschrieben und ergeben sich aus den verschiedenen Szenarien kooperativen Arbeitens. Ziel ist es, einen gemeinsamen Datenraum zwischen den Servern zu schaffen und Übergänge zwischen Wissensräumen zu ermöglichen. Ein kooperatives Arbeiten zwischen Benutzern verschiedener Server ist ohne Einschränkung möglich, falls ein gemeinsamer Handlungsraum zwischen allen Knoten eines Verbunds besteht (vgl. Abschnitt 3.3).

Der Benutzer bewegt sich transparent durch den Verbund von Servern und meldet sich an seinem Heimatserver an. Eine zentrale Benutzerverwaltung existiert nicht – jeder Server verfügt über einen eigenen Bestand an Benutzern und Gruppen. Die Identität eines Benutzers muss in jedem Fall sichergestellt sein und lokale ACLs oder Rollen entscheiden über Berechtigungen (vgl. Abschnitt 5.2.1).

Insgesamt sind aus den Anforderungen mehrere mögliche Architekturen erarbeitet worden, die jeweils einen Teil der Anforderungen umsetzen. Während die ersten drei Architekturen von einem Serververbund ausgehen, setzt die letzte Architektur einen dynamischen Verbund auf Basis eines Peer-to-Peer Overlay-Netzwerks um. Obwohl diese letzte Architektur den höchsten Grad der technischen Innovation beinhaltet, erfordert eine Umsetzung allerdings auch den größten Aufwand.

Auf technischer Ebene sind weitere Voraussetzungen identifizierbar. So bilden Microkernel-Architekturen (vgl. Abschnitt 5.4.2) eine ideale Basis, um von identischen Grundsystemen aus verschiedene Konfigurationen zu erzeugen. Die Erweiterung von neuen und bestehenden Systemen um eine austauschbare Persistenzschicht (vgl. Abschnitt 5.3) bietet sich weiterhin an, um verschiedene Datenquellen mit einzubeziehen und den Datenraum in dieser Hinsicht flexibel zu gestalten.

7.1 Proxy als Entwurfsmuster für verteilte Wissensräume

Ein Proxy ist ein Platzhalter für ein Objekt (Gamma et al. 1995, S. 207f.). Statt einer direkten Referenz wird der Proxy dazwischen geschaltet und kontrolliert dadurch die Zugriffe auf das zugehörige Objekt. Der Proxy verfügt über dieselbe Schnittstelle wie das dahinter liegende Objekt, sodass für eine Anwendung keine Unterschiede zwischen Proxy und Objekt existieren.

Das Proxy-Pattern ist ein zentrales Entwurfsmuster für eine Architektur verteilter Wissensräume. Ziel ist es Objekte zur Verfügung zu stellen, die nicht zwingend in der lokalen Persistenzebene gespeichert sein müssen. Weiterhin müssen die enthaltenen Objekte eines Raums nicht direkt im Speicher vorliegen, sodass ein Zugriff auf einen Wissensraum nicht automatisch alle abhängigen Objekte bereitstellen muss. Es kann zwischen den folgenden Anwendungen von Proxys unterschieden werden:

- **Remote-Proxy**: In dem Kontext von verschiedenen Adressräumen werden Proxys verwendet. Dabei verweist ein Remote-Proxy auf ein Objekt, das sich in einem anderen Adressraum als der Proxy befindet.

- **Virtual-Proxy**: Die Erzeugung des Objekts kann durch Verwendung eines Proxys so lange hinausgezögert werden, bis das Objekt wirklich benötigt wird. Dies hat Vorteile im Hinblick auf den Speicherverbrauch und die Ausführungszeit, die benötigt wird, um das Objekt anzulegen.

- **Protection-Proxy**: Der Zugriff auf Objekte kann mit einem Proxy kontrolliert werden, sodass z.B. die Zugriffsrechte zunächst überprüft werden, bevor eine Methode ausgeführt wird.
- **Smart-Reference**: Es ist möglich die Anzahl von Referenzen innerhalb eines Proxys nachzuhalten. Es besteht dadurch die Möglichkeit Objekte zu löschen, falls keine Referenz mehr existiert.

Abbildung 7.1 illustriert die Verwendung von Proxys in Wissensräumen. Die Objekte (Container C, Dokument D, Benutzer B und Verbindung V) sind indirekt über einen Proxy verbunden und in einem beliebigen Adressraum gespeichert.

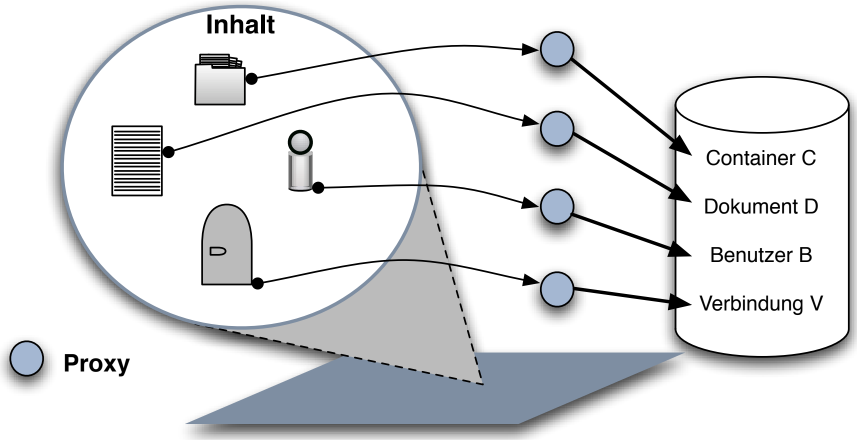


Abbildung 7.1: *Verwendung von Proxys im Wissensraum*

Für eine Architektur verteilter Wissensräume haben neben der Verwendung von Remote-Proxys auch die anderen Einsatzmöglichkeiten des Proxy-Entwurfmusters Bedeutung. Der Zugriff auf Objekte kann generell über Proxys abgesichert werden und die Verwendung von Virtual Proxys kann bei einer verteilten Architektur genauso Verwendung finden wie bei klassischen Client/Server-Anwendungen. In Bezug auf Wissensräume gilt allgemein aufgrund der Liste von enthaltenen Objekten (vgl. Abschnitt 3.1.2 und Abschnitt 3.1.7), dass Proxys von Bedeutung sind, um nicht alle enthaltenen Objekte im Speicher halten zu müssen.

7.2 Factorys in verteilten Systemen

Das Factory Pattern beschreibt ein Entwurfsmuster zur Erzeugung von Objekten. Dabei wird nicht der Konstruktor des Objekts genutzt, sondern die Erzeugung von einer Factory kontrolliert. In einem kollaborativen System ist dies von Bedeutung, um festzulegen, welche Benutzer bestimmte Objekte erzeugen dürfen. Darüber hinaus sind einige Objektklassen nicht isoliert zu sehen. So werden in einem raumbasierten kollaborativen System Arbeitsräume Benutzern oder Gruppen zugeordnet (vgl. Abschnitt 3.1.2).

Im Kontext einer Middleware dienen die Factorys weiterhin als Objekte, die für Clients Methoden zur Erstellung von Objekten bereitstellen, da es über entfernte Methodenaufrufe nicht direkt möglich ist Konstruktoren anzusprechen. Daher dienen Factorys als Objekte mit nur einer einzelnen Instanz (vgl. Singleton-Pattern: Gamma et al. 1995, S. 127f.) zur Bereitstellung von Methoden zur Erzeugung einer bestimmten Klasse von Objekten.

Bei mehreren untereinander verbundenen Systemen verfügt jedes dieser Systeme über eigene Factorys. Diese lokalen Factorys kontrollieren dabei die Erzeugung von Objekten, die auf diesem Knoten entstehen. Dabei ist die Speicherung nicht auf den lokalen Knoten beschränkt, sondern kann über einen Persistenzmanager (vgl. Abschnitt 5.3) auf anderen Knoten und in unterschiedlichen Persistenzebenen erfolgen.

Bei einem statischen Verbund bestehend aus verschiedenen Servern (vgl. Architektur 6.2–6.4) haben die lokalen Factorys jedes einzelnen Servers lediglich eine Bedeutung für die Erzeugung von lokalen Objekten. Damit bieten diese die Möglichkeit der Erzeugung von entfernten Objekten durch eine serverübergreifende Verwendung von Factorys.

7.3 Ereignisorientierte Kommunikation

Eine ereignisorientierte Kommunikation basiert auf der Abhängigkeit von Aktion und Modifikation (vgl. Abschnitt 5.6). Ein Großteil der Ereignisse beinhaltet eine Zustandsänderung von einem Objekt, sodass sich der Zustand eines Objekts mit einem Publish/Subscribe-Mechanismus überwachen lässt. Bei einer Benachrichtigung kann entsprechend reagiert werden. Im Kontext eines verteilten Systems kann dies genutzt werden, um Replikate zu synchronisieren. Da jede Änderung übertragen wird, kann dies als Basis für eine Konsistenz von Objekten dienen und gleichzeitig können serverübergreifende Ereignisse genutzt werden, um die Aktionen anderer Benutzer darzustellen (und damit Awarenessfunktionen zu realisieren).

Eine Kommunikation zwischen einzelnen Komponenten eines Systems kann ebenfalls mit Hilfe von Ereignissen geregelt werden. Dies entspricht einer losen Kopplung, die eine hohe Flexibilität verspricht und im Gegensatz zu einer festen Verknüpfung zwischen Komponenten/Modulen steht. Damit steht einer Austauschbarkeit einzelner Systemkomponenten nichts im Wege und eine generelle Erweiterbarkeit des Systems wird unterstützt, da sich neue Komponenten ohne weiteres in die Ereignisverarbeitung einbringen können. Gleichzeitig ist ein solches ereignisorientiertes System auf eine Ereignisverarbeitung zwischen verschiedenen Ebenen vorbereitet (vgl. SEDA in Abschnitt 5.5.2).

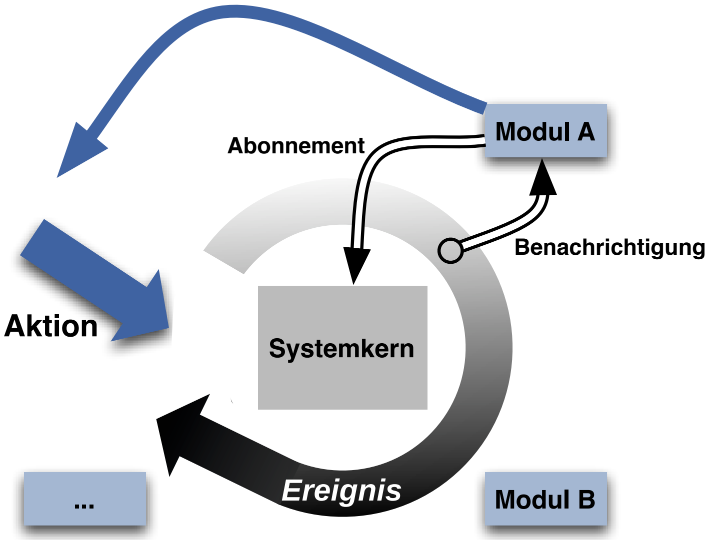


Abbildung 7.2: *Ereignisorientierte Kommunikation über den Systemkern*

Abbildung 7.2 zeigt eine Kommunikation zwischen Komponenten eines Systems. Dabei erfolgt ein Abonnement eines Ereignisses über den Systemkern, der die Verwaltung übernimmt. Eine Aktion löst ein entsprechendes Ereignis aus und eine Benachrichtigung an die Abonnenten erfolgt. Wie in der Abbildung dargestellt ist, kann das Modul A dann wieder eine Aktion ausführen, die wiederum mit einem Ereignis verbunden ist, welches von einem weiteren Modul empfangen und ausgewertet werden kann.

Für verteilte Wissensräume stellt dies eine gute Basis dar, wenn der Raum als Kontext für Ereignisse verwendet wird. Dadurch kann der Raum als geschlossene Einheit betrachtet werden und das Abonnement von Ereignissen wird durch den Raum gekapselt.

7.4 Interest Management

Aus dem Area of Interest Management der CVE-Systeme (vgl. Abschnitt 2.2) lässt sich eine allgemeinere Form eines Interest Managements für verteilte Systeme ableiten. Diese ergibt sich aus einem beliebigen Kontext für den Benutzer/Client, sodass Ereignisse gefiltert werden können oder generell Nachrichten in Abhängigkeit des Kontextes versendet werden. Für ein raumbasiertes System kann diese Beziehung mit Hilfe von Räumen hergestellt werden, wie dies bereits für eine Vergabe der Berechtigungen und Ereignisse beschrieben worden ist (vgl. Abschnitt 3.1.4). Für einen Benutzer bildet der Raum in einer virtuellen Umgebung einen Kontext, der sich beispielsweise für Awareness-Funktionen nutzen lässt.

Dies lässt sich analog auf verteilte Wissensräume übertragen, die ebenso einen Kontext für enthaltene Objekte bieten können. Dies trifft insbesondere zu, wenn eine räumliche Verteilungsstrategie von Objekten verwenden wird und Räume wiederum in Abhängigkeit von Gruppen zugeordnet werden (vgl. Abschnitt 3.3.1). Damit kommt einem Interest Management eine größere Bedeutung zu als bei einem einzelnen Server, da eine Übertragung von Ereignissen über Servergrenzen hinweg einen erhöhten Kommunikationsaufwand beinhaltet. Zudem kann die Verarbeitung von Ereignissen direkt über den Raum erfolgen, so dass die Area of Interest automatisch durch die Strukturen eines raumbasierten Systems gegeben ist. Im Gegensatz dazu muss in grafischen CVE-Systemen die Sichtweite und Orientierung eines Avatars ständig berücksichtigt werden.

7.5 Microkernel als Basis kollaborativer Systeme

Ein modulares System stellt eine günstige Architektur für spätere Erweiterungen und Anpassungen dar. Aufgrund der Eigenschaften eines Microkernels eignet sich dieser besonders gut als grundlegender Rahmen eines kollaborativen Systems (vgl. Bopp und Hampel 2005c). Insbesondere in Bezug auf einen dynamischen Verbund von Knoten erlaubt der Microkernel einen Einsatz auf Knoten mit unterschiedlichen Eigenschaften. In Abhängigkeiten von den verfügbaren Ressourcen kann die Konfiguration des Microkernels variiert werden. Der Kernel kann sowohl auf großen Serversystemen eingesetzt werden als auch auf mobilen Geräten, die lediglich über geringe Ressourcen verfügen. Dabei werden folgende grundlegende Funktionen für Module bereitgestellt (vgl. Abschnitt 5.4.2):

- **Auswertung der Abhängigkeiten zwischen Modulen**: Es bestehen Beziehungen zwischen den einzelnen Modulen.
- **Laden von Modulen**: Zu Beginn werden die Module in der Reihenfolge ihrer Abhängigkeiten vom Kernel geladen.
- **Ausführen von Modulen**: In jedem Modul wird die Methode *run*() beim Start des Kernels aufgerufen. Dabei läuft jedes Modul in einem eigenen Thread, sodass die Ausführung nicht das System blockieren kann.
- **Bereitstellen von Modulen**: Es besteht die Möglichkeit die Module des Kernels abzufragen. Dabei kann entweder die gesamte Modulkonfiguration abgefragt werden oder einzelne Module durch ihren Namen oder einen Alias. Des Weiteren besteht die Möglichkeit alle Module eines bestimmten Typs zu erfragen.

Die Funktionalität des Systems wird durch die verwendeten Module erzielt, wobei zwischen verschiedenen Arten von Modulen unterschieden werden muss.

7.5.1 Module

Ein Modul stellt dem System zusätzliche Funktionalität bereit. Dabei gibt es Module, die unbedingt erforderlich sind, und Module, die dem System lediglich weitere Eigenschaften hinzufügen. Die folgende Liste stellt typische Klassen von Modulen dar, die in einem kollaborativen System Verwendung finden.

- **Sicherheit**: Diese Module fragen die Berechtigungen von Benutzern im Kontext von Aktionen ab. Jede Zustandsänderung des Systems muss von einem Sicherheitsmodul blockierbar sein.
- **Persistenz**: Es können mehrere Persistenzschichten existieren, die die Objekte speichern. Dabei kann die Speicherung z.B. im Dateisystem erfolgen oder auch in einer Datenbank.

- **GUI**: Es handelt sich um eine Benutzerschnittstelle, die eine Sicht auf die Objekte des Systems bietet (vgl. auch verschiedene Sichten auf Wissensräume im Abschnitt 3.1.2).
- **Netzwerk**: Ein Netzwerkmodul verarbeitet Kommunikation zwischen Client/Server oder Peer-to-Peer auf Basis eines bestimmten Protokolls.

Der Microkernel lädt alle Module beim Start des Systems und stellt die Funktionalität damit bereit. Die Persistenz von Attributen der Module kann aufgrund der Existenz von Persistenzmodulen nur mit Umwegen realisiert werden. Dann ist es durchaus möglich Module wie Objekte zu behandeln und mit der gleichen Schnittstelle zu versehen.[1]

Der Vorgang des Ladens von Modulen erfordert die Spezifikation der Abhängigkeiten, um eine Reihenfolge zu ermitteln. Zusätzlich werden Daten wie der Name des Moduls und die Version in einer Konfigurationsdatei festgelegt und von dem Microkernel zu Beginn ausgewertet. Der Name des Moduls wird dabei verwendet, um eine Identifikation innerhalb des Systems zu gewährleisten. Darüber hinaus muss jedes Modul über ein Attribut verfügen, welches den Typ des Moduls beschreibt, sodass es möglich ist alle Module einer bestimmten Art abzufragen.

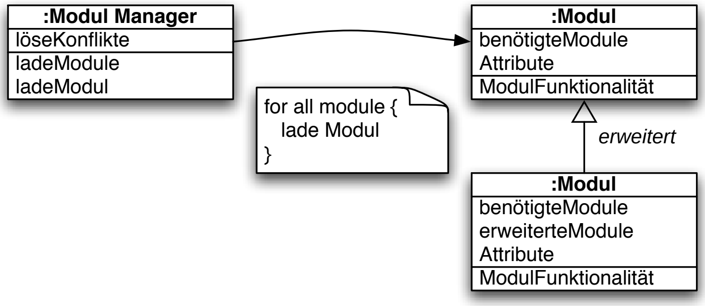


Abbildung 7.3: *Struktur der Module*

Abbildung 7.3 illustriert die Struktur von Modulen im Zusammenspiel mit dem Modulmanager. Dieser wertet Abhängigkeiten aus und löst Konflikte zwischen Modulen. Falls keine Lösung für eine bestimmte Situation gefunden werden kann, so wird der Ladevorgang abgebrochen und eine entsprechende Meldung ausgegeben (z.B. falls kreisförmige Abhängigkeiten gefunden werden).

---

[1] Das sTeam-System löst dies mit einer vorgegebenen Persistenzschicht (eine MySQL-Datenbank), die fest im System verankert ist.

Die Konfiguration von Modulen muss verschiedene Aspekte berücksichtigen:

- **Name des Moduls**: Der Name des Moduls muss eindeutig sein und dient zur Identifikation innerhalb des Kernels.
- **Version der Implementierung**: Von einem Modul können verschiedene Versionen existieren, die über unterschiedliche Funktionalität verfügen. Die Abhängigkeiten zwischen Modulen müssen dies berücksichtigen.
- **Benötigte Module**: Die Funktion einiger Module setzt andere Module voraus, die ebenfalls installiert sein müssen.
- **Aufgabenbereich des Moduls**: Der Bereich eines Moduls muss definiert sein, sodass indirekt auf das Modul zugegriffen werden kann. Ein Modul kann auch die Aufgaben eines anderen Moduls übernehmen und dieses damit ersetzen. Ein Beispiel sind Sicherheitsfunktionen, sodass Anfragen in Bezug auf Berechtigungen abgewickelt werden können, ohne explizit auf ein bestimmtes Modul zuzugreifen.

Abbildung 7.4 zeigt die Architektur eines Microkernels mit verschiedenen Modulen. Das zentrale Element wird von dem Modulmanager eingenommen, der für eine Erweiterbarkeit des Kernels sorgt. Gleichzeitig werden die Netzwerkeigenschaften, Ereignisse und Persistenzmerkmale durch entsprechende Komponenten bereitgestellt.

Die Ereignisverarbeitung, die direkt im Kern implementiert und nicht austauschbar ist, nimmt eine zentrale Stellung ein. Sie regelt die Kommunikation im System, sodass Objekte und Module Ereignisse auf verschiedenen Ebenen abonnieren und dadurch die Abläufe im Server überwachen können (vgl. Abschnitt 5.6).

7.5.2 Ereignisverarbeitung im Microkernel

Die Ereignisverarbeitung nimmt eine zentrale Bedeutung bei einer Kollaboration von Benutzern ein (vgl. Caballé et al. 2004). Nur durch einen Austausch von Ereignissen ist es möglich über Änderungen informiert zu werden und den Zustand der Applikation zu aktualisieren. Alle Teilnehmer befinden sich dadurch auf einem aktuellen Zustand und nehmen die Modifikationen anderer Benutzer wahr.

Die Verarbeitung von Ereignissen findet nach einem Publish-/Subscribe-Modell statt (vgl. Abschnitt 5.6). Mit dem Wissensraum als grundlegende

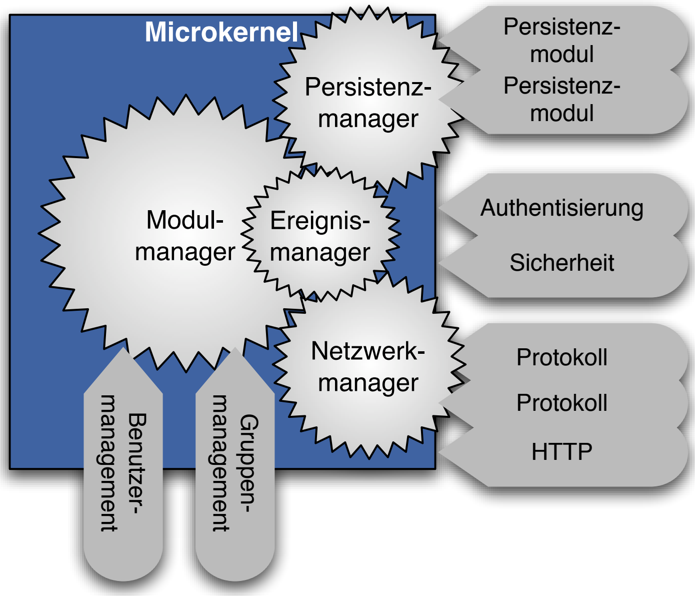


Abbildung 7.4: *Microkernel*

Komponente der Architektur ist die Verarbeitung von Ereignissen auf die Umgebung als Kontext eines Objekts auszudehnen (vgl. Abschnitt 3.1.4). Wie bereits beschrieben, besitzt jedes Objekt eine eindeutige Umgebung, die auch als Kontext für die Berechtigungen genutzt werden kann. Auf ähnliche Weise werden Ereignisse nicht nur innerhalb des Objekts, sondern auch im umgebenden Raum verarbeitet. Ein Ereignis wird dabei mit einem zusätzlichen Flag (*Observed*) versehen und kann so von einem lokalen Ereignis unterschieden werden (vgl. Interest Management in Abschnitt 7.4).

Bei der Einhaltung der Reihenfolge von Ereignissen wird zwischen optimistischen und pessimistischen Verfahren unterschieden (Greenberk und Marwoord 1994). Bei einem optimistischen Verfahren spielt die Reihenfolge der Zugriffe keine Rolle – Inkonsistenzen müssen bemerkt und repariert werden. Bei einem pessimistischen Verfahren muss eine bestimmte Reihenfolge eingehalten werden, sodass es zu keinen Inkonsistenzen kommen kann.

Im Rahmen eines kollaborativen Systems ist eine Reihenfolge von Ereignissen weniger entscheidend. Die Zustände von einzelnen Attributen sind weitgehend unabhängig und im synchronen Betrieb ist es nicht von besonderer Bedeutung, welche Aktionen von anderen Benutzern zuerst erfolgen.

Trotzdem stellt die Erweiterung der Ereignisse um einen Wert für die Zeit der Aktion hier eine Lösung dar. Aufgrund der zweiphasigen Verarbeitung kann in der Phase der möglichen Blockierungen von Ereignissen bereits die Zeit ermittelt werden, da diese Phase direkt vor dem Ereignis durchlaufen wird.

7.6 Persistenzmanager

Als wesentliches Kriterium kollaborativer Systeme ist die Persistenz der Daten sicherzustellen. Dies erfolgt in der Regel durch die Verwendung einer einzelnen Persistenzschicht. Eine Austauschbarkeit einer solchen Persistenzschicht bietet eine optimale Flexibilität und erlaubt es sowohl Datenbanken als auch Dateisysteme zu nutzen. Dadurch ist es möglich sich den Ressourcen der jeweiligen Umgebung anzupassen.

In Anlehnung an das „Data Access Object“-Pattern (vgl. Alur et al. 2003, S. 462-495) dient der Persistenzmanager dazu Objekt und Daten zu entkoppeln. Damit wird neben dem Austausch einer einzelnen Persistenzebene auch ein Betrieb mit mehreren Persistenzschichten ermöglicht. Einer Einbindung von zusätzlichen Repositorys steht aus diesem Grund nichts im Wege.

Die Persistenzschicht speichert Objekte oder deren Replikate. Dazu werden die Attribute bei einer Modifikation eines Objekts gespeichert. Die Schnittstelle zu der Persistenzschicht besteht aus einer Reihe von Funktionen:

- *neuesObjekt(Objekt)*: Legt ein neues Objekt in einer Persistenzschicht an.
- *speicherObjekt(Objekt)*: Speichert ein Objekt in der dem Objekt zugeordneten Persistenzschicht.
- *ladeObjekt(int serverID, int namespaceID, int objektID)*: Lädt ein Objekt aus einer Persistenzschicht und gibt den Verweis auf das geladene Objekt zurück. Aus dem Namespace wird das zugehörige Persistenzmodul ausgewählt und das Objekt von dort geladen. Ein Persistenzmodul mit Zugriff auf entfernte Objekte ist möglich.
- *findeObjekte(string query)*: Realisiert Suchfunktionalität in den Persistenzschichten. Der Suchbegriff „query“ kann dabei sowohl der Name eines Objekts als auch ein Suchmuster (z.B. ein regulärer Ausdruck) sein.

Abbildung 7.5 zeigt das UML-Diagramm eines Ladevorgangs für ein Objekt. Dabei werden dem Persistenzmanager neben der ID eines Objekts, die

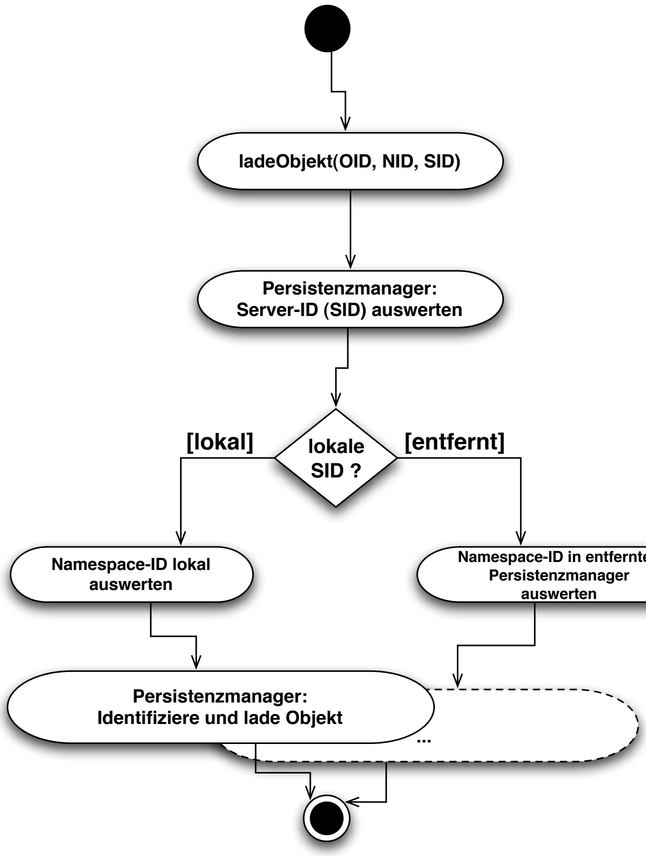


Abbildung 7.5: *Aktivitätsdiagramm: Auflösung von Namespace-ID (NID) und Server-ID (SID)*

IDs für den Datenraum (Namespace) und den Server übergeben. Mit diesen Daten wertet der Persistenzmanager zunächst die SID aus und leitet die Anfrage für entfernte Objekte an den entsprechenden Persistenzmanager eines anderen Servers weiter. Eine lokale ID führt zu einer direkten Auswertung der Namespace-ID und einer Delegation des Ladevorgangs an das entsprechende Persistenzmodul.

Abbildung 7.6 zeigt die Struktur der Komponenten. Der Persistenzmanager steht im Zentrum mit Funktionalität zum Laden und Speichern von Objekten. Einzelne Persistenzschichten registrieren sich und werden bei einer Erzeugung von neuen Objekten angesprochen. Dann können diese eine entsprechende Rückmeldung geben, um ein Objekt in die eigene Schicht zu übernehmen. Eine

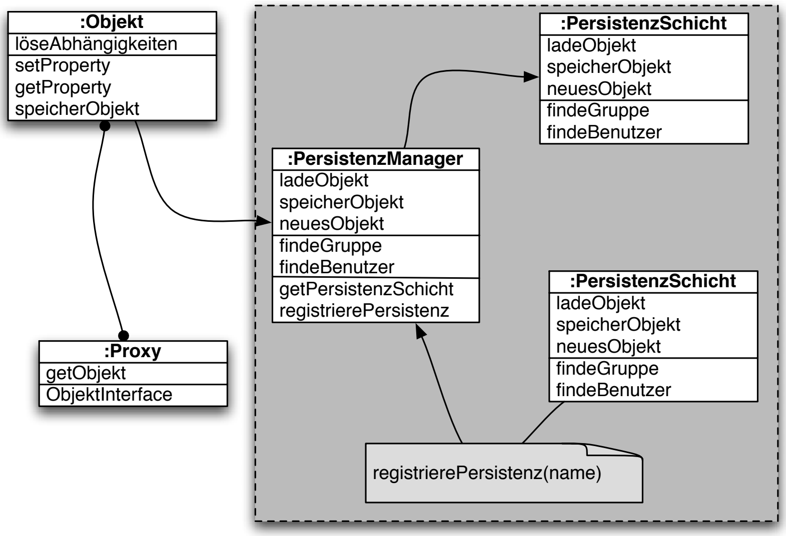


Abbildung 7.6: *Zusammenhang von Persistenzmanager und Persistenzschichten*

besondere Stellung nehmen die Funktionen findeGruppe() und findeBenutzer() ein, die eine Gruppe oder ein Benutzerobjekt über einen eindeutigen Bezeichner finden. Dies ist auf den Vorgang der Authentisierung und die Vergabe von Rechten zurückzuführen (vgl. Abschnitt 5.2.1).

Im direkten Zusammenhang zu einem Persistenzmanager steht das Proxy-Pattern, welches Verweise auf die Daten von Objekten realisiert, die in verschiedenen Persistenzschichten gespeichert sein können.

7.6.1 Proxy

Ein Proxy dient als Platzhalter für das Objekt mit den enthaltenen Daten (vgl. Abschnitt 7.1). Technisch wird der Proxy daher als eine spezielle Klasse umgesetzt, die einen Zeiger auf ein Datenobjekt darstellt. Dies ist insbesondere bei der Serialisierung von Objekten zu berücksichtigen: Jeder Verweis auf ein anderes Objekt wird als Proxy mit Server-ID, Namensraum-ID und Objekt-ID gespeichert werden (vgl. Abbildung 6.10). Verweise zwischen Objekten sind in dieser Form serialisierbar und können ein Objekt beim Laden wieder in den korrekten Zustand überführen. Dabei wird zunächst lediglich der Proxy als Platzhalter in den Speicher geladen, der auf das eigentliche Objekt mit

Attributen und weiteren Daten in der Persistenzschicht verweist. Für einen Persistenzmanager sind dem Proxy direkt die verschiedenen IDs zugeordnet, die es ermöglichen die korrekte Persistenzschicht zuzuordnen und das Objekt zu laden.

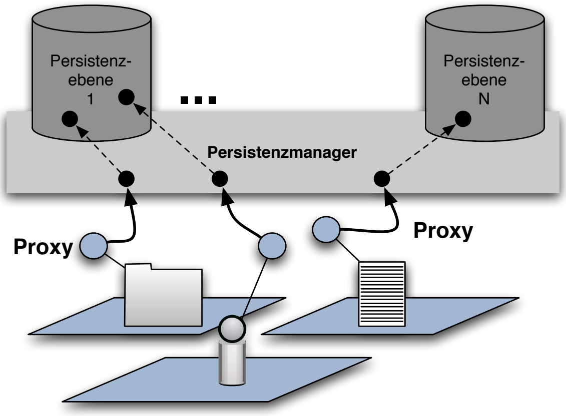


Abbildung 7.7: *Dynamischer Proxy-Zugriff*

Abbildung 7.7 zeigt einen indirekten Zugriff durch ein Proxy-Objekt. Der Persistenzmanager verwaltet den eigentlichen Verweis auf die Daten, die sich auf unterschiedlichen Persistenzebenen befinden können.

Abgesehen von diesen Beobachtungen kann ein Objekt nicht isoliert betrachtet werden. Jedes Objekt speichert Referenzen auf andere Objekte — im einfachsten Fall ist die Ist-enthalten-Relation (vgl. Abschnitt 3.1.3) zu nennen. Bei einer starken Partitionierung einfacher Strukturen (z.B. innerhalb eines Baums) kann der Kommunikationsaufwand sehr groß werden.

Zur Fehlererkennung wird jedem Proxy ein Status zugeordnet. Dabei kann ein Proxy die folgenden Zustände annehmen (es gibt dabei gültige und ungültige Zustände — die ungültigen sind mit *fehlgeschlagen* gekennzeichnet):

- *In der Datenbank*: Das Objekt befindet sich in der Datenbank und muss bei einem Zugriff zunächst in den Speicher geladen werden.

- *Speicherung wird vorgenommen*: Das Objekt befindet sich im Speicher und muss noch in die Datenbank gespeichert werden.

- *Speicherung erfolgreich*: Das Objekt befindet sich im Speicher und der aktuelle Zustand ist in der Datenbank gespeichert.

- *Löschung fehlgeschlagen*: Ein Löschen des Objekts ist fehlgeschlagen.

- *Bereitstellung fehlgeschlagen*: Das Objekt kann nicht von dem Persistenzmodul bereitgestellt werden. Dies ist insbesondere bei entfernten Objekten zu beachten, wenn das Repository temporär oder endgültig nicht erreichbar ist.

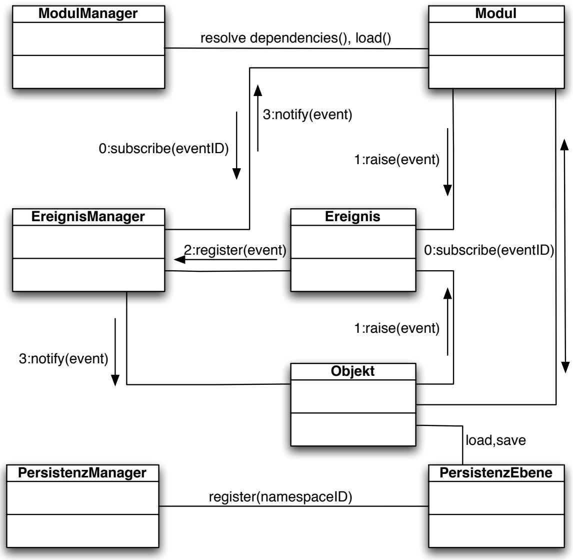


Abbildung 7.8: *Persistenz und Ereignisse im Kernel*

Abbildung 7.8 zeigt das Zusammenspiel der einzelnen Komponenten des Kernels. Die Kommunikation findet über den Ereignismanager statt, der Registrierungen (0:subscribe) zu bestimmten Ereignissen entgegennimmt und verwaltet. Beim Auftreten eines Ereignisses (1:raise) wird dieses dem Manager mitgeteilt (2:register) und schließlich eine Benachrichtigung (3:notify) über das Ereignis an ein Objekt übermittelt. Die Persistenz der Objekte wird durch den Persistenzmanager geregelt, der im Falle einer Änderung eine Speicherung in einer der Persistenzschichten vornimmt.

7.7 Webservices zur Kopplung von Systemen

Mit Hilfe von Webservices kann ein System Dienste über festgelegte Schnittstellen bereitstellen.[2] Damit kann eine Interoperabilität von verschiedenen Systemen erreicht werden, da die Spezifikation eines Webservices automatisch durch eine WSDL-Datei beschrieben ist (vgl. Abschnitt 5.7.2) und sich in unterschiedliche Umgebungen einbetten lässt.

Aufgrund dieser Eigenschaften bietet ein Webservice eine Grundlage, um verschiedene Systeme miteinander zu verbinden. Darüber hinaus besteht auch die Möglichkeit, verschiedene Dienste zu koppeln und ein neues Gesamtsystem daraus zu schaffen. So beschreiben Prpitsch et al. (2005) einen Ansatz der Kopplung von digitaler Bibliothek, Planungssystem und CSCW-System. Ebenso gelingt eine Einbindung von Webservices in ein CSCW-System unter Einbeziehung von Amazon- und Google-Diensten (Hampel et al. 2004). Damit stellt das CSCW-System einerseits selbst standardisierte Dienste bereit und ist darüber hinaus in der Lage Dienste in Anspruch zu nehmen. Auch eine Kombination von eigenen und fremden Diensten in neue komplexe Webservices ist in diesem Kontext möglich.

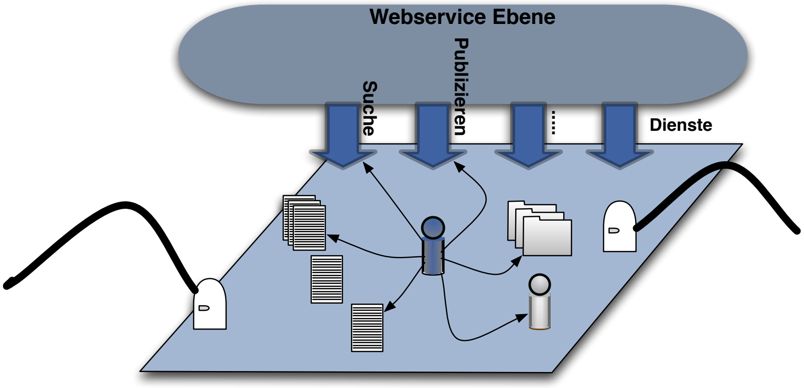


Abbildung 7.9: *Verschiedene Dienste werden in einem Raum bereitgestellt*

Im Kontext von Wissensräumen, die einen Treffpunkt für Benutzer darstellen, bietet es sich sowohl an lokale als auch externe Dienste innerhalb dieser Räume bereitzustellen. Abbildung 7.9 zeigt, wie einem Benutzer in einem Raum verschiedene Aktionen zur Verfügung stehen. Dazu zählen verschiedene

---

[2] Die Webservice-Aktivitäten und Spezifikationen des W3C können unter http://www.w3.org/2002/ws/ [Stand: 10.12.2005] abgerufen werden.

Dienste, die entweder lokal zur Verfügung stehen oder extern über Webservices angebunden sind. Hier ist beispielsweise Suche und Publizieren dargestellt, ohne eine direkte Verknüpfung zu der dahinter liegenden Funktionalität aufzuzeigen. So kann es sich sowohl um lokale Dienste handeln als auch um eine externe Suche in einem beliebigen Datenbestand und das Publizieren von lokalen Dokumenten in entfernte Repositorys (wie z.B. eine Bibliothek).

7.8 Peer-to-Peer-Netzwerktopologie als Basis verteilter Wissensräume

Für verteilte Wissensräume sind flexible Netzwerktopologien ohne zentrale Serverinstanzen von Vorteil. Es besteht keine vorgegebene Struktur, sondern die Kooperation in Räumen wird lediglich durch die Vergabe von Zugriffsrechten gesteuert. Jeder Teilnehmer einer bestimmten Gruppe, die zur Mitarbeit in einen Raum eingeladen worden ist, kann diesen Raum betreten und dort aktiv werden. Damit handelt es sich um ein sehr dynamisches Netzwerk von Wissensräumen, in dem alle Teilnehmer unabhängig von Serverstrukturen miteinander kooperieren können. Dabei kann das Peer-to-Peer-Netzwerk durchaus einen hybriden Charakter aufweisen und über unterschiedliche Knoten verfügen, sodass einige feste Peers als Einstiegspunkte in das Overlay-Netzwerk dienen und gleichzeitig weitere Aufgaben wahrnehmen können. Weiterhin ist es sinnvoll die Ressourcen jedes Knotens zu berücksichtigen, da dedizierte Superpeers über gute Netzwerkeigenschaften und große Speicherkapazität verfügen. Diese Knoten stabilisieren insbesondere kleine Peer-to-Peer-Netzwerke, da sie im Prinzip das Backbone darstellen.

Aufgrund der Eigenschaften von Peer-to-Peer-Netzwerken ist zu berücksichtigen, dass bei einem Ausfall einer großen Menge von Knoten (in Bezug zur Gesamtzahl der Knoten) die Möglichkeit eines Zusammenbruchs besteht, wenn z.B. kein Knoten aus der Routingtabelle mehr erreichbar ist. Ebenso kann es passieren, dass Objekte nicht mehr auffindbar sind, wenn alle Knoten mit Replikaten gleichzeitig ausscheiden. Bei kleinen Netzen kann dem entgegengewirkt werden, indem eine große Anzahl permanenter Knoten (Superpeers) bereitgestellt wird.

Eine Verwendung von Peer-to-Peer-Netzwerktopologien erfolgt im Idealfall vollständig gekapselt mit existierenden Infrastrukturen, deren Funktionsweise abstrahiert zur Verfügung steht.[3] Dadurch kann eine solche Netzwerkebene separat entwickelt werden. Dies ist von besonderer Bedeutung aufgrund der Komplexität einer Peer-to-Peer-Anwendung im Gegensatz zu klassischen Client/Server-Systemen.

Für Wissensräume sind eine verteilte Verarbeitung von Ereignissen und die Bereitstellung von Objekten ausschlaggebend, die die Grundlage eines gemeinsamen Handlungsraums darstellen (vgl. Abschnitt 3.3). Dies wird im Idealfall vollständig gekapselt von einer Peer-to-Peer-Infrastruktur bereitgestellt. Dabei ist allerdings die Charakteristik des Peer-to-Peer-Netzwerks zu berücksichtigen, da Wissensräume ganz bestimmte Strukturen aufweisen, die in die Art der Verteilung einfließen können.

7.9 Gateway zu verteilten Wissensräumen

Im Gegensatz zu einem Peer-to-Peer-Netzwerk steht ein Cluster, das viele Ressourcen an einer zentralen Stellen bündelt. Bezogen auf Wissensräume stehen zunächst unabhängige Strukturen von Wissensräumen nebeneinander und wachsen im Cluster zusammen. An dieser Stelle nimmt das Proxy-Pattern eine wichtige Stellung ein, um Verweise auf andere Wissensräume zu realisieren (vgl. Abschnitt 7.1).

Eine einheitliche Sicht auf die Wissensraumstrukturen innerhalb eines Clusters kann durch einen Gateway realisiert werden. Dieser bietet einen Zugang und verwaltet die Strukturen selbstständig. Im Prinzip werden in dem Gateway Wissensräume direkt zusammengeführt. Sowohl Gruppen und Benutzer als auch Ereignisse und Übergänge zwischen den Räumen werden von dem vorgeschalteten Gateway verwaltet. Damit kommt dem Gateway die Rolle eines Vermittlers zu, da dieser gleichzeitig auf mehrere Server zugreift, um eine einheitliche Sicht für den Benutzer zu schaffen. Dabei müssen die Abläufe in den einzelnen Servern überwacht werden.

Abbildung 7.10 zeigt, wie in einem Gateway eine Aktion eines Benutzers verarbeitet wird. Zur Durchführung müssen in dem Gateway verschiedene Objekte vorgehalten werden, wie der Benutzer B, der die Anfrage durchführt, die Gruppe G und der Raum R. Der Raum berechtigt dabei die Gruppe, der auch der Benutzer angehört. Die Aktion selbst kann nun auf dem Server C

---

[3] Ein Beispiel für eine solche Infrastruktur ist der Distributed Hash Table Open-DHT (Rhea et al. 2005).

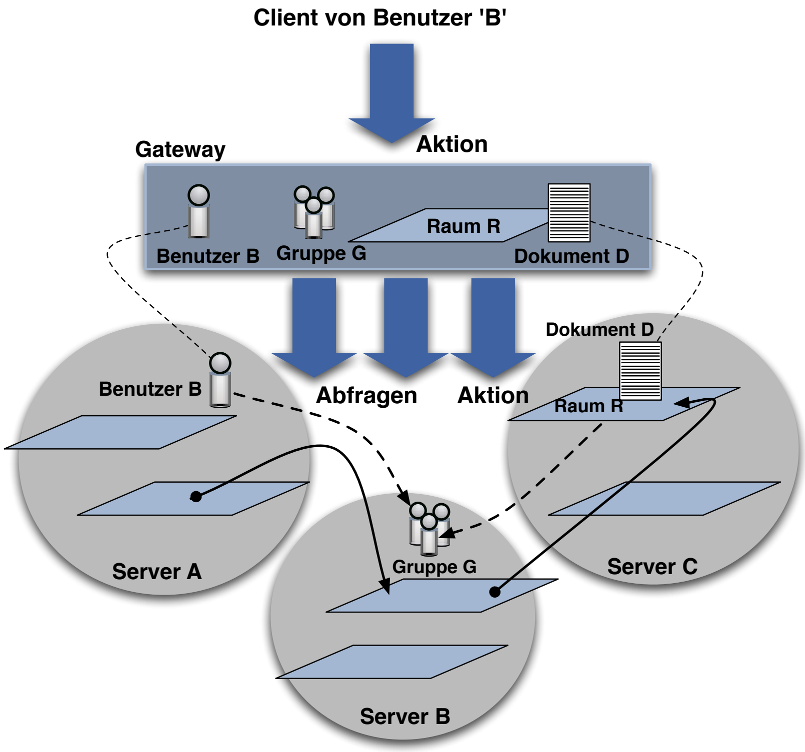


Abbildung 7.10: *Aktionen in Wissensräumen über einen Gateway*

ausgeführt werden (Aktion von Benutzer B auf Dokument D). Dazu ist eine einheitliche Benutzerverwaltung im Verbund notwendig, damit Benutzer B auf Server C aktiv werden kann.

Die Umsetzung eines Gateways erfordert kaum Eingriffe in die Architektur bestehender raumbasierter Systeme. Ein Großteil der Funktionalität muss in dem Gateway realisiert werden, der im Prinzip als Client angesehen werden kann. Das Resultat ist eine Zwischenschicht zwischen der eigentlichen Client-Applikation und dem Server, die selbst auf mehrere Server zugreift und dadurch Wissensräume zusammenführt.

Der Gateway kann nicht für den Zugriff auf ein Cluster eingesetzt werden, sondern auch auf beliebige verteilte Server genutzt werden. In diesem Fall sind allerdings Nachteile im Sinne einer schlechten Erreichbarkeit von Servern in Kauf zu nehmen.

7.10 Zusammenfassung

Im Zusammenhang mit den Architekturen aus Kapitel 6 sind verschiedene Entwurfsmuster zu berücksichtigen. An zentraler Stelle steht dabei der Proxy als Verweis auf entfernte Objekte. Damit können lokale Objekte in den Speicher geladen werden, ohne dass direkt auf entfernte Objekte zugegriffen werden muss. Ein weiteres sehr grundlegendes Entwurfsmuster stellt das Factory-Pattern dar. Im Kontext von verteilten Wissensräumen kann dies als zentrale Instanz zur Erzeugung von Objekten auf einem Server dienen.

Für die Umsetzung eines Handlungsraums zwischen Servern sind Ereignisse von Bedeutung (vgl. Abschnitt 3.3). Aus diesem Grund sind ereignisorientierte Architekturen für eine Verteilung von Wissensräumen von Vorteil. Dies spiegelt sich in der Konsistenz von Zuständen der Objekte (z.B. bei einer Replikation) in der Abfolge von Aktion, Ereignis und Synchronisation wider. In diesem Kontext ist auch ein Interest Management zu sehen, welches insbesondere in verteilten Umgebungen von Vorteil ist, da die Verteilung von Ereignissen dabei bereits auf Seite des Servers gefiltert wird. Dadurch sinkt das Kommunikationsaufkommen zwischen einzelnen Servern und zu den Clients. Ein Wissensraum stellt hier eine Area of Interest für Benutzer dar, die sich in diesem Raum befinden.

Als komplexeres Muster sind Microkernel und Persistenzmanager identifiziert worden. Für einen gemeinsamen Datenraum, der nicht alle Objekte enthält, können über den Persistenzmanager verschiedene Speicherorte in Abhängigkeit von der Objektklasse eingebunden werden. Damit gelingt z.B. die Einbindung eines zentralen Dienstes für Benutzer und Gruppen. Weiterhin dient der Microkernel in einem verteilten kollaborativen System zum Einsatz auf Knoten mit beliebigen Ressourcen. Dabei ist die Konfigurierbarkeit ein wichtiges Merkmal für einen Einsatz als Peer und Superpeer in einem Peer-to-Peer-Netzwerk.

Eine Kopplung und Vernetzung von Servern ist auf verschiedene Weise möglich. Zunächst können Webservices als standardisierte Schnittstellen dienen, um auch unterschiedliche Systeme zu verbinden. Als dynamisches Netzwerk bieten sich Peer-to-Peer-Netzwerktopologien an, die eine Infrastruktur für verteilte Wissensräume darstellen können. Diese bezieht ehemalige Server und Clients im Prinzip gleichberechtigt in das Overlay-Netzwerk ein, sodass alle Peers über ein einheitliches Grundsystem verfügen und Wissensräume überall verfügbar sind. Die einzige Unterscheidung zwischen verschiedenen Knoten erfolgt anhand der Ressourcen als Peer und Superpeer.

Im Gegensatz dazu steht eine Gateway-Architektur, die stärker auf zentralisierte Serverstrukturen (z.B. Cluster) aufbaut und einen einheitlichen Zugang zu den dahinter liegenden Strukturen für die Clients schafft. Der Gateway nimmt dabei die Rolle eines Vermittlers zwischen den Servern ein.

8 Zusammenfassung

Ziel dieser Arbeit ist es Übergänge zwischen Wissensräumen verschiedener, bisher unverbundener, Bereiche zu schaffen. Dabei liegt der Fokus auf einer Erweiterung von bestehenden Architekturen um einen gemeinsamen Datenraum. Dies kann entweder mit einer Verbindung mehrerer Server der gleichen Systemklasse erreicht werden, aber auch durch eine Kopplung unterschiedlicher Systeme, die über einheitliche Schnittstellen verfügen. Neben einem passiven Datenraum ist das Konzept des Handlungsraums als aktiver Raum mit Ereignissen eingeführt worden, der durch eine gegenseitige Wahrnehmbarkeit von Aktivitäten gekennzeichnet ist. Ziel des Handlungsraum ist damit menschliche Zusammenarbeit serverübergreifend zu unterstützen. Auf technischer Ebene zeichnet sich ein Handlungsraum durch einen Austausch von Ereignissen aus, die als Grundlage für eine Synchronisation von Objekten dienen können. Dagegen bietet ein Datenraum zunächst eine Möglichkeit für Server gegenseitig auf Objekte zuzugreifen und einen Datenaustausch damit zu ermöglichen. Ein minimaler Datenraum wird hier durch die Benutzer gebildet, die sich dadurch systemübergreifend anmelden können.

Durch eine Analyse existierender Systeme und Technologien ist deutlich geworden, dass sich bestehende CSCW-Systeme nur aufwändig erweitern lassen, ohne wirklich allen Anforderungen gerecht zu werden. Dabei sind sowohl klassische Architekturen als auch neue Trends wie Peer-to-Peer und Grid-Computing einbezogen worden. Für eine vollständige Neuentwicklung sind aktuelle Forschungen im Bereich der Distributed Hash Tables zu nennen, die immer mehr an Bedeutung gewinnen und deren Einsatz sich nicht nur auf verteilte Dateisysteme und Tauschbörsen beschränkt. Sämtliche der genannten Technologien beinhalten wichtige Erkenntnisse für eine Verteilung von Objekten und den Austausch von Daten.

Zentrale Grundlage und Ausgangspunkt dieser Arbeit bildet der Wissensraum. Dieser stellt eine Umgebung für den Benutzer dar und kann verschiedene Objekte beinhalten. Aufgrund des Aufbaus eines Wissensraums ist dieser jedoch nicht isoliert zu betrachten, sondern es sind sowohl bestehende Verbindungen zu Räumen als auch eine Struktur von Räumen zu berücksichtigen. Daraus ergeben sich verschiedene Verteilungsstrategien über einen Verbund, die zum Beispiel von Architekturen im Bereich von virtuellen Umgebungen abgeleitet sind. Hintergrund einer günstigen Verteilung ist eine effiziente Kommunikation und ein schnelles Abrufen von Räumen mit enthaltenen Objekten. So finden sich Verteilungsstrategien bei den kollaborativen virtuellen Umgebungen (CVE) und massiven Online-Spielen (MMOG). In der Regel wird hier aufgrund der Charakteristik der virtuellen Welt in mehrere Zonen aufgeteilt. Eine andere Strategie ist eine Aufteilung von Objekten in Abhängigkeit einer Zugehörigkeit zu bestimmten Gruppen. In Bezug auf Wissensräume kann hier eine Parallele zu Gruppenarbeitsräumen gezogen werden, da sich hier ebenfalls alle Materialien einer Gruppe zuordnen lassen. Aus diesem Grund ist diese Aufteilung der Objekte im Rahmen dieser Arbeit für die Architektur eines gemeinsamen Handlungsraums favorisiert worden.

Schließlich können Objekte auch ohne einen semantischen Zusammenhang mit Hilfe einer bestimmten funktionalen Verknüpfung verteilt werden, die sich z.B. positiv auf eine Verfügbarkeit und Erreichbarkeit von Objekten auswirkt (wie z.B. bei Distributed Hash Tables).

Mit dem Ziel einer verteilten Struktur von Wissensräumen sind zunächst verschiedene Szenarien aus technischer Perspektive erarbeitet worden: Wissensräume schließen sich unabhängig vom physischen Speicherort zusammen und es entstehen Verbindungen zwischen beliebigen Räumen. Der Verbund von Wissensräumen kann sowohl erweitert, als auch im Zuge einer Abtrennung von Wissensräumen verkleinert werden. Unterschieden werden muss grundsätzlich zwischen statischen und dynamischen Verbünden. Ein Vertreter der dynamischen Verbünde sind Peer-to-Peer Filesharing-Systeme. Hier wird die Quote von wechselnden Knoten als so genannter Churn erfasst, der die Änderung des Overlay-Netzwerks widerspiegelt. Im Gegensatz dazu ändern sich statische Verbünde kaum und werden lediglich durch manuelle Eingriffe von Administratoren angepasst. Zu berücksichtigen ist hier auch ein temporäres Ausscheiden von Teilnehmern aufgrund von Netzwerkausfällen oder Systemabstürzen.

Neben diesen Szenarien der Entstehung und Veränderung von Verbünden sind die Verknüpfungen zwischen Wissensräumen und die Aktivitäten von Benutzern von Bedeutung. Wenn sich Benutzer zwischen verteilten Strukturen bewegen, werden auch Dokumente mitgenommen und zwischen verteilten Raumstrukturen übertragen. Innerhalb von Wissensräumen werden Aktionen ausgeführt, die unabhängig von einer physischen Verteilung der Wissensräume sind. Schließlich sind Aktionen mit Ereignissen verbunden, die überall zur Verfügung stehen müssen.

Aus diesen verschiedenen Szenarien sind eine Vielzahl von Anforderungen abgeleitet worden. Hierzu zählt ein gemeinsamer Datenraum zum Austausch von Objekten zwischen verschiedenen Servern. Dabei muss eine Autonomie der einzelnen Systeme berücksichtigt werden, die einen eigenen Bestand an Objekten, Benutzern und Gruppen verwalten. Die Grundlage von Kooperationen wird durch Übergänge zwischen entfernten Wissensräumen geschaffen. Damit ist ein im Idealfall transparenter Wechsel von einem Wissensraum zu einem entfernten Wissensraum möglich. In diesem entfernten Wissensraum ist eine Basis einer Kooperation der Benutzer durch die gegenseitige Wahrnehmbarkeit der Aktivitäten gegeben. Zudem muss eine Vergabe von Berechtigungen an beliebige Gruppen und Benutzer möglich sein, um diese in lokale Wissensräume einzuladen.

Das Ergebnis dieser Arbeit sind verschiedene Musterarchitekturen und Entwurfsmuster, die unterschiedliche Ausbaustufen von verteilten Wissensräumen darstellen. Besonders hervorzuheben ist die Architektur eines dynamischen Verbunds, die auf einem Overlay-Netzwerk von Repositorys basiert. Durch den gemeinsamen Datenraum aller beteiligter Knoten und einer Verteilung von Ereignissen an alle Teilnehmer entsteht ein gemeinsamer Handlungsraum. Dieser bildet die technische Grundlage für kooperative Medienfunktionen in verteilten Umgebungen.

Während die bisher existierenden Systeme im Bereich von Peer-to-Peer zunächst von einem Austausch von Dateien und später auf verteilte Dateisysteme abzielten, sind die in dieser Arbeit beschriebenen Architekturen auf eine Verteilung von Objekten ausgelegt.

Die Grundlage dieser Architektur bildet der Wissensraum als Treffpunkt für Benutzer. Im Gegensatz zu den klassischen Peer-to-Peer-Systemen handelt es sich um eine Architektur, die ein kollaboratives Arbeiten von Benutzern zwischen den beteiligten Knoten des Netzes ermöglicht. Um diesem Anliegen gerecht zu werden ist eine Umsetzung auf Basis eines Microkernels von Bedeutung. Die Verwendung einer plattformunabhängigen Sprache, wie zum Beispiel Java, ist ebenfalls von Vorteil, da das gleiche System unabhängig von der jeweiligen Plattform dadurch auf beliebigen Knoten einsetzbar ist. Die Grenzen zwischen Server und Client lösen sich in Anlehnung an die Konzepte der Peer-to-Peer-Vernetzung auf. Trotzdem bleiben die bewährten Eigenschaften zentralisierter Architekturen durch verlässliche Serversysteme mit umfangreichen

Ressourcen erhalten.

Die Auswahl der Architektur richtet sich nach den jeweiligen Gegebenheiten und unter Berücksichtigung aller Faktoren. Bereits die Erweiterung von Servern um einen gemeinsamen Datenraum kann auf unterschiedliche Weise erfolgen: Es kann sich entweder um ein zentrales Verzeichnis handeln, welches dem Verbund untergeordnet ist und von jedem Server aus modifiziert werden kann (Dazu ist ein Vertrauensverhältnis zwischen allen Server notwendig). Auf der anderen Seite stehen Verzeichnisse, die lediglich eingebunden werden, aber nicht beliebig administriert werden können. So kann entweder ein einzelner Server eines Verbunds die Administration und Synchronisation im Verzeichnis vornehmen oder diese Kontrolle liegt vollständig ausserhalb des Serververbunds. Gleichzeitig ist allen anderen Servern lediglich ein lesender Zugriff erlaubt. Aus diesem Grund ist eine Architektur, die jeder Situation gerecht wird, nur schwierig möglich und die Auswahl einer Architektur muss sich den gegebenen Verhältnissen anpassen.

Für eine alltagstaugliche Umsetzung von verteilten Wissensräumen steht eine Gateway-Architektur, die bestehende Wissensraum-Infrastrukturen berücksichtigt. Dabei wird der Gateway generell vor alle Anfragen geschaltet und sorgt als Vermittler für eine Weiterleitung und für eine einheitliche Wahrnehmung der virtuellen Umgebung verteilter Wissensräume. Dies hat den Vorteil, dass bestehende Server nur im geringen Maße angepasst werden müssen.

Bei einer Umsetzung ist eine sorgsame Auswahl von Entwurfsmustern von großer Bedeutung. Hier nimmt der Remote-Proxy eine zentrale Stellung ein, um Verweise auf entfernte Objekte zu realisieren. Dabei erfolgt eine Erweiterung von lokal eindeutigen IDs um einen Namensraum, sodass auch eine eindeutige Identifizierung im Verbund ermöglicht wird. Darüber hinaus sind Ereignisse für eine Synchronisation von Objektzuständen und eine gegenseitige Wahrnehmung von Aktivitäten von Bedeutung. Gleichzeitig ist ein Interest Management notwendig, damit die Kommunikation zwischen den Knoten in einem Verbund optimiert werden kann. Im Bezug zu Wissensräumen kann hier die Umgebung als ein Kontext für die enthaltenen Objekte dienen.

Ein Microkernel mit entsprechenden Modulen kann in verschiedenen Umgebungen eingesetzt werden. Durch die Konfiguration der Module kann dieser sehr flexibel an unterschiedliche Anforderungen angepasst werden. Auf dieser Basis ist in dieser Arbeit eine Architektur verteilter Wissensräume beschrieben worden. Eine austauschbare Persistenzschicht ist dabei ein wichtiges Kriterium, um eine flexible Speicherung von Daten zu ermöglichen. Im Zuge des Modul-konzepts können einzelne Module hier Persistenzschichten darstellen, die durch den Persistenzmanager gekapselt werden.

Eine weitere Grundlage wird durch verteilte Repositorys gebildet, die einen logischen nächsten Schritt im Bereich von Peer-to-Peer-Systemen darstellen. Als eine Basis solcher Systeme wurden Distributed Hash Tables untersucht, die bereits eine weite Verbreitung im Bereich von Filesharing und verteilten Dateisystemen haben. Für Wissensräume können diese mit einer verteilten Ereignisverarbeitung eine grundlegende Infrastruktur darstellen. Dazu ist eine Kapselung in verschiedene Architekturebenen von Vorteil.

Mit den Entwurfsmustern und den verschiedenen Architekturen, die unterschiedliche Stufen einer Umsetzung des Konzepts von verteilten Wissensräumen darstellen, sind alle notwendigen Grundlagen einer Umsetzung verteilter Wissensräume gegeben. Diese Grundlagen sind für unterschiedliche Situationen und Anforderungen ausgelegt und ermöglichen verschiedene Architekturen verteilter Wissensräume, denen jeweils bestimmte Muster zugrunde liegen.

Literaturverzeichnis

Adar und Huberman 2000 ADAR, E. ; HUBERMAN, B.: Free Riding on Gnutella. In: *First Monday* 5 (2000), Oktober, Nr. 10

Alur et al. 2003 ALUR, D. ; CRUPI, J. ; MAIKS, D.: *Data Access Object*. 2. Auflage. Upper Saddle River, N.J., USA : Prentice Hall / Sun Microsystems Press, 2003

Amir et al. 1994 AMIR, Y. ; DOLEV, D. ; MELLIAR-SMITH, P. ; MOSER, L.: Robust and Efficient Replication using Group Communication / Institute of Computer Science, The Hebrew University of Jerusalem. 1994 (CS94-20). – Forschungsbericht

Androutsellis-Theotokis und Spinellis 2004 ANDROUTSELLIS-THEOTOKIS, S. ; SPINELLIS, D.: A Survey of Peer-to-Peer Content Distribution Technologies. In: *ACM Computer Survey* 36 (2004), Nr. 4, S. 335–371

Atkinson et al. 1989 ATKINSON, M. ; BANCILHON, F. ; DEWITT, D. ; DITTRICH, K. ; MAIER, D. ; ZDONIK, S.: The Object-Oriented Database System Manifesto. In: *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*. Kyoto, Japan, 1989, S. 223–240

Atkinson und Morrison 1995 ATKINSON, M. ; MORRISON, R.: Orthogonally Persistent Object Systems. In: *The VLDB Journal* 4 (1995), Nr. 3, S. 319–402

Barkley 1997 BARKLEY, J.: Comparing simple Role based Access Control Models and Access Control Lists. In: *Proceedings of the 2nd ACM Workshop on Role-based Access Control*, ACM Press, 1997, S. 127–132

Barrus et al. 1996 BARRUS, J. ; WATERS, R. ; ANDERSON, D.: Locales: Supporting Large Multiuser Virtual Environments. In: *IEEE Computer Graphics and Applications* 16 (1996), Nr. 6, S. 50–57

Benford et al. 1995 Benford, S. ; Bowers, J. ; Fahlén, L. E. ; Greenhalgh, C. ; Snowdon, D.: User Embodiment in Collaborative Virtual Environments. In: *Proceedings of the ACM Conference Human Factors in Computing Systems (CHI)* Bd. 1, 1995, S. 242–249

Benford et al. 2001 Benford, S. ; Greenhalgh, C. ; Rodden, T. ; Pycock, J.: Collaborative Virtual Environments. In: *Communications of the ACM* 44 (2001), Nr. 7, S. 79–85

Benford et al. 1993 Benford, S. ; Mariani, J. ; Navarro, L. ; Prinz, W. ; Rodden, T.: MOCCA: An Environment for CSCW Applications. In: *Proceedings of the Conference on Organizational Computing Systems (COCS).* New York, NY, USA : ACM Press, 1993, S. 172–177

Bentley und Appelt 1997 Bentley, R. ; Appelt, W.: Designing a System for Co-operative Work on the World-Wide-Web: Experiences with the BSCW System. In: J.F. Nunamaker, R.H. S. (Hrsg.): *Proceedings of the 30th Annual Hawaii International Conference on System Sciences (HICCS)* Bd. 4, IEEE Computer Society Press, 1997, S. 297–306

Bernstein 1998 Bernstein, P.: Repositories and Object Oriented Databases. In: *ACM SIGMOD Record* 27 (1998), Nr. 1, S. 88–96

Bernstein et al. 1987 Bernstein, P. ; Goodman, N. ; Hadzilacos, V.: *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987

Berry 2004 Berry, R.: Performability and Grid in an On Demand World. In: *Workshop on Grid Performability, Modelling and Performability*, 2004

Birrel und Nelson 1984 Birrel, A. ; Nelson, B.: Implementing Remote Procedure Calls. In: *ACM Transactions on Computer Systems* 2 (1984), Nr. 1

Bollmeyer 1997 Bollmeyer, J.: HyperMUD. In: *SIGGROUP Bull.* 18 (1997), Nr. 1, S. 35–36

Bonifacio et al. 2002 Bonifacio, M. ; Bouquet, P. ; Mameli, G. ; Nori, M.: KEx: A Peer-to-Peer Solution for Distributed Knowledge Management. In: *Lecture Notes in Artificial Intelligence* 2569 (2002), Dezember, S. 490–500

Bopp 2000 Bopp, T.: *Konzeption und prototypenhafte Entwicklung eines Servers zur Kooperationsunterstützung in computergestützten Lernumgebungen*, Universität Paderborn, Lehrstuhl für Informatik, Fachgruppe Informatik und Gesellschaft, Diplomarbeit, Mai 2000

Bopp und Hampel 2004 Bopp, T. ; Hampel, T.: Szenarien kooperativen Lernens und Arbeitens über Servergrenzen hinweg. In: *Tagungsband der 2. E-Learning Fachtagung Informatik, GI-Edition, Lecture Notes in Informatics*, 2004, S. 361–362

Bopp und Hampel 2005a Bopp, T. ; Hampel, T.: *Integration of New Technologies into a Room-based CSCW System.* Oktober 2005

Bopp und Hampel 2005b Bopp, T. ; Hampel, T.: A Microkernel Architecture for Collaborative Systems / Heinz Nixdorf Institut. 2005 (tr-ri-05-270). – Forschungsbericht

Bopp und Hampel 2005c Bopp, T. ; Hampel, T.: A Microkernel Architecture for Distributed Mobile Environments. In: *Proceedings of the 7th International Conference on Enterprise Information Systems (ICEIS)*, 2005, S. 151–156

Bopp et al. 2004 Bopp, T. ; Hampel, T. ; Essmann, B.: Connecting Virtual Spaces. In: *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS)*, 2004, S. 475–479

Bopp et al. 2005a Bopp, T. ; Hampel, T. ; Karadavut, T.Y. ; Konietzny, C.: A DHT Architecture for CSCW-Systems / Heinz Nixdorf Institut. 2005 (tr-ri-05-272). – Forschungsbericht

Bopp et al. 2005b Bopp, T. ; Hampel, T. ; Vitt, S.: Lüüd liehrn tohoop schrieven: Integration einer Wiki-Sicht in ein raumbasiertes CSCW/L-Systemm – Anforderungen und Umsetzungen. In: *Tagungsband der 3. E-Learning Fachtagung Informatik, GI-Edition, Lecture Notes in Informatics*, 2005, S. 23–34

Bote-Lorenzo et al. 2003 Bote-Lorenzo, M. ; Dimitriadis, Y. ; Gómez-Sánchez, E.: Grid Characteristics and Uses: a Grid Definition. In: *Proceedings of the 1st European Across Grids Conference, ACG'03*, Springer Verlag, 2003, S. 291–298

Bruckman 1994 Bruckman, A.: Programming for Fun: MUDs as a Context for Collaborative Learning. In: *Proceedings of the National Educational Computing Conference*, 1994

Buschmann et al. 1996 Buschmann, F. ; Meunier, R. ; Rohnert, H. ; Sommerlad, P. ; Stal, M.: *Pattern-Oriented Software Architecture - A System of Patterns*. John-Wiley and Sons, 1996

Caballé et al. 2004 Caballé, S. ; Xhafa, F. ; Daradoumis, T. ; Marques, J. M.: Towards a Generic Platform for Developing CSCL Applications using Grid Infrastructure. In: *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2004, S. 200–207

Cai et al. 2002 Cai, W. ; Xavier, P. ; Turner, S. J. ; Lee, B.: A Scalable Architecture for Supporting Interactive Games on the Internet. In: *Proceedings of the 16th Workshop on Parallel Distributed Simulation*, 2002, S. 60–67

Cardellini et al. 1999 Cardellini, V. ; Colajanni, M. ; Yu, P. S.: Dynamic Load Balancing on Web-Server Systems. In: *IEEE Internet Computing* 3 (1999), Nr. 3, S. 28–39

Carzaniga et al. 2000 Carzaniga, A. ; Rosenblum, D. S. ; Wolf, A. L.: Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service. In: *Proceedings of the Symposium on Principles of Distributed Computing*, 2000, S. 219–227

Casanova 2002 Casanova, H.: Distributed Computing Research Issues in Grid Computing. In: *ACM SIGACT News — Distributed Computing Column* 33 (2002), Nr. 3, S. 50–70

Charwathe et al. 2003 Charwathe, Y. ; Ratnasamy, S. ; Breslau, L. ; Lanham, N. ; Shenker, S.: Making Gnutella-like P2P Systems Scalable. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, ACM Press, 2003, S. 407–418

Childers et al. 2000 Childers, L. ; Disz, T. ; Olson, R. ; Papka, M. ; Stevens, R. ; Udeshi, T.: Access Grid: Immersive Group-to-Group Collaborative Visualization. In: *Proceedings of the 4th International Immersive Projection Technology Workshop*, 2000

Churchill und Bly 1999 Churchill, E. F. ; Bly, S.: It's all in the Words: Supporting Work Activites with Lightweight Tools. In: *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, ACM Press, 1999, S. 40–49

Clarke et al. 2001 Clarke, I. ; Sandberg, O. ; Wiley, B. ; Hong, T. W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: *Lecture Notes in Computer Science* 2009 (2001), S. 46–66

Cooper et al. 2000 Cooper, B. ; Crespo, A. ; Garcia-Molina, H.: Implementing a Reliable Digital Object Archive. In: *Lecture Notes in Computer Science* 1923 (2000), S. 128–143

Coulouris et al. 2002 Coulouris, G. ; Dollimore, J. ; Kindberg, T.: *Verteilte Systeme: Konzepte und Design.* 3., überarbeitete Auflage. Addison-Wesley, 2002

Crawley und Oudshoorn 1994 Crawley, S. ; Oudshoorn, A.: Orthogonal persistence and Ada. In: *Proceedings of the Conference on TRI-Ada.* New York, NY, USA : ACM Press, 1994, S. 298–308

Cugola et al. 1998 Cugola, G. ; Nitto, E. D. ; Fuggetta, A.: Exploiting an Event-based Infrastructure to develop complex Distributed Systems. In: *Proceedings of the 20th International Conference on Software Engineering (ICSE).* Washington, DC, USA : IEEE Computer Society, 1998, S. 261–270

Cugola und Picco 2002 Cugola, G. ; Picco, G.P.: Peer-to-Peer for Collaborative Applications. In: *Proceedings of the International Workshop on Mobile Teamwork Support*, 2002, S. 359–364

Curtis und Nichols 1994 Curtis, P. ; Nichols, D. A.: MUDs Grow Up: Social Virtual Reality in the Real World. In: *Proceedings of the IEEE Computer Conference (COMPCON)*, 1994, S. 193–200

Dabek et al. 2001 Dabek, F. ; Kaashoek, M. F. ; Karger, D. ; Morris, R. ; Stoica, I.: Wide-area Cooperative Storage with CFS. In: *Proceedings* *of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oktober 2001, S. 202–215

Das et al. 1997 Das, T. K. ; Singh, G. ; Mitchell, A. ; Kumar, P. S. ; McGee, K.: NetEffect: A Network Architecture for Large-scale Multi-user Virtual Worlds. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM Press, 1997, S. 157–163

Dewan und Shen 1998 Dewan, P. ; Shen, H.: Flexible meta Access-control for Collaborative Applications. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, ACM Press, 1998, S. 247–256

Dingledine et al. 2001 Dingledine, R. ; Freedman, A. J. ; Molnar, D.: The Free Haven Project: Distributed Anonymous Storage Service. In: *Lecture Notes in Computer Science* 2009 (2001), S. 67–95

Effert und Wunderlich 2004 Effert, S. ; Wunderlich, C.: *Entwicklung einer verteilter Microkernel-Architektur für kooperative Wissensräume*, Universität Paderborn, Fakultät EIM, Diplomarbeit, September 2004

Engelbart und English 1968 Engelbart, D. C. ; English, W. K.: A Research Center for Augmenting Human Intellect. In: *Proceedings of the 1968 Fall Joint Computer Conference*, Dezember 1968, S. 395–410

Eßmann et al. 2004 Essmann, B. ; Hampel, T. ; Bopp, T.: A Network Component Architecture for Collaboration in Mobile Settings. In: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, 2004, S. 337–343

Eugster und Baehni 2002 Eugster, P. T. ; Baehni, S.: Abstracting Remote Object Interaction in a Peer-2-Peer Environment. In: *Proceedings of the joint ACM-ISCOPE Conference on Java Grande*, ACM Press, 2002, S. 46–55

Ferraiolo und Kuhn 1992 Ferraiolo, D. ; Kuhn, R.: Role Based Access Control. In: *Proceedings of the 15th National Computer Security Conference*, 1992, S. 554–563

Foster 2005 FOSTER, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: *Proceedings of the International Conference on Network and Parallel Computing*, Springer Verlag, 2005, S. 2–13

Foster et al. 1998 FOSTER, I. ; KESSELMAN, C. ; TSUDIK, G. ; TUECKE, S.: A Security Architecture for Computational Grids. In: *Proceedings of the 5th ACM Conference on Computers and Communications Security*, November 1998, S. 83–92

Foster und Kesselmann 1999 FOSTER, I. ; KESSELMANN, C.: *The Grid: Blueprint for a new computing Infrastructure.* Morgan Kaufmann Publishers Inc., 1999

Fuchs et al. 1995 FUCHS, L. ; PANKOKE-BABATZ, U. ; PRINZ, W.: Supporting Cooperative Awareness with local Event Mechanisms: The GroupDesk System. In: *Proceedings of the European Computer Supported Cooperative Work Conference (ECSCW '95)*, Kluwer Academic Publishers, 1995, S. 247–262

Gamma et al. 1995 GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns: Elements of Reuseable Object-Oriented Software.* Addison-Wesley, 1995

Gibson 1984 GIBSON, W.: *Neuromancer.* Heyne Verlag, 1984

Gokhale und Schmidt 1996 GOKHALE, A. ; SCHMIDT, D.: Measuring the Performance of Communication Middleware on High-speed Networks. In: *Applications, technologies, architectures, and protocols for computer supported communications*, ACM PRESS, 1996, S. 306–317

Goland et al. 1999 GOLAND, Y. ; WHITEHEAD, E. ; FAIZI, A. ; CARTER, S. ; JENSEN, D.: *RFC 2518 — HTTP Extensions for Distributed Authoring — WEBDAV.* Februar 1999

Gong 2001 GONG, L.: Project JXTA: A technology overview / Sun Microsystems. URL `http://www.jxta.org/project/www/docs/TechOverview.pdf [Stand: April 2005]`, April 2001. – Forschungsbericht

Govindaraju et al. 2000 GOVINDARAJU, M. ; SLOMINSKI, A. ; CHOPPELA, V. ; BRAMLEY, R. ; GANNON, D.: Requirements for and Evaluation of RMI Protocols for Scientific Computing. In: *Proceedings of the Supercomputing Conference*, 2000, S. 76–102

Gray und Reuter 1993 Gray, J. ; Reuter, A.: *Transaction Processing: Concepts and Techniques.* Kaufman Publishers, 1993

Greenalgh et al. 2000 Greenalgh, C. ; Purbrick, J. ; Snowdown, D.: Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. In: *Proceedings of the 3rd International Conference on Collaborative Virtual Environments*, 2000, S. 119–127

Greenberk und Marwoord 1994 Greenberk, S. ; Marwoord, D.: Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 1994, S. 207–217

Greif 1988 Greif, I.: *Computer Supported Cooperative Work: A Book of Readings.* Morgan Kaufmann Publishers, 1988

Gutknecht et al. 1996 Gutknecht, O. ; Ferber, J. ; Michel, F.: Integrating Tools and Infrastructures for Generic Multi-Agent Systems. In: *Proceedings of the 5th International Conference on Autonomous Agents*, 1996, S. 441–448

Haake et al. 2004a Haake, J. M. ; Haake, A. ; Schümmer, T. ; Bourimi, M. ; Landgraf, B.: End-user Controlled Group Formation and Access Rights Management in a Shared Workspace System. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, ACM Press, 2004, S. 554–563

Haake et al. 2004b Haake, J. M. ; Schümmer, T. ; Haake, A. ; Bourimi, M. ; Landgraf, B.: Supporting flexible collaborative Distance Learning in the CURE Platform. In: *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004

Hampel 1999 Hampel, T.: sTeam — Cooperation and Structuring Information in a Team. In: *Proceedings of the World Conference on the WWW and Internet*, 1999, S. 469–474

Hampel 2002 Hampel, T.: *Virtuelle Wissensräume. Ein Ansatz für die kooperative Wissensorganisation*, Universität Paderborn, Fachbereich 17 - Informatik, Dissertation, 2002

Hampel und Bopp 2001 Hampel, T. ; Bopp, T.: Magellan, the Paderborn Approach to Distributed Knowledge Organization. In: *Proceedings of the ED-MEDIA*, 2001, S. 649–655

Hampel und Bopp 2004 Hampel, T. ; Bopp, T.: Verteilte Wissensräume und ihre Musterarchitekturen / Heinz Nixdorf Institut. 2004 (tr-ri-05-269). – Forschungsbericht

Hampel et al. 2004 Hampel, T. ; Halbsgut, J. ; Bopp, T.: Heterogenous Integration of Services into an open and standardized Web Service. In: *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS)*, 2004, S. 182–189

Hampel und Keil-Slawik 2001 Hampel, T. ; Keil-Slawik, R.: sTeam - Designing an integrative infrastructure for Web-based Computer-supported cooperative Learning. In: *Proceedings of the 10th International Conference on World Wide Web*, 2001, S. 76–85

Hampel und Keil-Slawik 2002 Hampel, T. ; Keil-Slawik, R.: sTeam: Structuring Information in a Team – Distributed Knowledge Management in Cooperative Learning Environments. In: *Journal of Educational Resources in Computing* 1 (2002), Nr. 2, S. 1–27

Harrison und Dourish 1996 Harrison, S. ; Dourish, P.: Re-Place-ing Space: The Roles of Place and Space in Collaborative Systems. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 1996, S. 67–76

Hauswirth et al. 2005 Hauswirth, M. ; Podnar, I. ; Decker, S.: On P2P Collaboration Infrastructures. In: *Proceedings of the 14th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*, 2005, S. 66–71

Henderson und Card 1986 Henderson, D. A. ; Card, S.: Rooms: The use of multiple Virtual Workspaces to reduce Space Contention in a Window-based Graphical User Interface. In: *ACM Transactions on Graphics* 5 (1986), Nr. 3, S. 211–243

Hess et al. 2002 Hess, A. ; Jacobson, J. ; Mills, H. ; Wamsley, R. ; Seamons, K. ; Smith, B.: Advanced Client/Server Authentication in TLS. In: *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2002

Hosseini et al. 2002 Hosseini, M. ; Pettifer, S. ; Georganas, N. D.: Visibility-based Interest Management in Collaborative Virtual Environments. In: *Proceedings of the 4th International Conference on Collaborative Virtual Environments (CVE)*, ACM Press, 2002, S. 143–144

Hubbold et al. 1999 Hubbold, R. ; Cook, J. ; Keates, M. ; Gibson, S. ; Howard, T. ; Murta, A. ; West, A. ; Pettifer, S.: GNU/MAVERIK: A Micro-Kernel for Lage-Scale Virtual Environments. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 1999, S. 66–73

Huizinga und Mann 1996 Huizinga, D. M. ; Mann, P.: Disconnected Operation for Heterogeneous Servers. In: *Proceedings of the ACM Symposium on Applied Computing*, ACM Press, 1996, S. 312–321

IBM Gamegrid 2005 Hachman, M.: *IBM tests Grid with Games: www.eweek.com/article2/0,1759,1497382,00.asp.* November 2005

IRC 1993 Oikarinen, J. ; Reed, D.: *RFC 1495 – Internet Relay Chat Protocol.* 1993

Jabber 2004 Saint-Andre, P. (Hrsg.): *RFC 3920 – Extensible Messaging and Presence Protocoll (XMPP).* Oktober 2004

Jakobsson und Taylor 2003 Jakobsson, M. ; Taylor, T. L.: The Sopranos Meets EverQuest — Social Networking in Massively Multiplayer Online Games. In: *FineArt Forum* 17 (2003), Nr. 8

Jin et al. 2002 Jin, C. ; Zheng, W. ; Zhou, F. ; Wu, Y.: A Distributed Persistent Object Store for Scalable Service. In: *ACM SIGOPS, Operating Systems Review* 36 (2002), Nr. 4, S. 36–49

Kappe 1995 Kappe, F.: A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. In: *J.UCS: Journal of Universal Computer Science* 1 (1995), Nr. 2, S. 84–104

Karger et al. 1997 Karger, D. ; Lehman, E. ; Leighton, T. ; Levine, M. ; Lewin, D. ; Panigrahy, R.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In: *Proceedings of the ACM Symposium on Theory of Computing*, Mai 1997, S. 654–663

Keil-Slawik und Selke 1998 Keil-Slawik, R. ; Selke, H.: Mythen und Alltagspraxis von Technik und Lernen. In: *Informatik-Forum* 12 (1998), S. 9–17

Kleindienst et al. 1996 Kleindienst, J. ; Plášil, F. ; Tůma, P.: Lessons learned from implementing the CORBA Persistent Object Service. In: *Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, ACM Press, 1996, S. 150–167

Knutsson et al. 2004 Knutsson, B. ; Lu, H. ; Xu, W. ; Hopkins, B.: Peer-to-Peer Support for Massively Multiplayer Games. In: *Proceedings of the Conference on Computer Communications (INFOCOM)*, 2004

Kubiatowicz et al. 2000 Kubiatowicz, J. ; Bindel, D. ; Chen, Y. ; Eaton, P. ; Geels, D. ; Gummadi, R. ; Rhea, S. ; Weatherspoon, H. ; Weimer, W. ; Wells, C. ; Zhao, Ben: OceanStore: An Architecture for Global-scale Persistent Storage. In: *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* ACM (Veranst.), November 2000, S. 190–201

Lamport 1978 Lamport, L.: Time, Clocks, and the ordering of Events in a Distributed System. In: *Communications of the ACM* 21 (1978), Nr. 7, S. 558–565

Lampson 1974 Lampson, B.: Protection. In: *ACM Operating Systems* 8 (1974), Nr. 1, S. 18–24

Lampson et al. 1992 Lampson, B. ; Abadi, M. ; Burrows, A. ; Wobber, E.: Authentication in Distributed Systems: Theory and Practice. In: *ACM Transactions on Computer Systems* 10 (1992), Nr. 4, S. 265–310

Lavender und Schmidt 1997 LAVENDER, G. ; SCHMIDT, D.: Active Object — An Object behavioral Pattern for Concurrent Programming. In: *Proceedings of the 2nd Annual Conference on the Pattern Languages of Programs*, 1997, S. 1–7

Lee und Anderson 1990 LEE, P. ; ANDERSON, T.: *Dependable Computing and Fault-tolerant Systems, Volume 3, 2nd Edition.* Kap. Fault Tolerance — Principles and Practice, Springer Verlag, 1990

Li und Moon 2001 LI, Q. ; MOON, B.: Distributed Cooperative Apache. In: *Proceedings of the 10th International Conference on World Wide Web*, 2001, S. 555–564

Liskov et al. 1996 LISKOV, B. ; ADYA, A. ; CASTRO, M. ; GHEMAWAT, S. ; GRUBER, R. ; MAHESHWARI, U. ; MYERS, A. C. ; DAY, M. ; SHRIRA, L.: Safe and Efficient Sharing of Persistent Objects in Thor. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM Press, 1996, S. 318–329

Lunney und McCaughey 2003 LUNNEY, T. ; MCCAUGHEY, A.: Object Persistence in Java. In: *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java (PPPJ '03).* New York, NY, USA : Computer Science Press, Inc., 2003, S. 115–120

Mariani und Rodden 1991 MARIANI, J. ; RODDEN, T.: The impact of CSCW on database technology. In: *Proceedings of the IFIP workshop on CSCW*, 1991, S. 146–161

Maymounkov und Mazieres 2002 MAYMOUNKOV, P. ; MAZIERES, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, S. 53–65

McQuillan und Walden 1977 MCQUILLAN, J. ; WALDEN, D.: *Computer Networks Vol. 1, No. 5.* Kap. The ARPA Network Design Decisions, S. 243–289, August 1977

Meyer 1995 MEYER, A.: Pen Computing: A Technology Overview and a Vision. In: *SIGCHI Bulletin* 27 (1995), Nr. 3, S. 46–90

Milojicic et al. 2002 Milojicic, D. ; Kalogeraki, V. ; Lukosi, R. ; Nagaraja, K. ; Pruyne, J. ; Richard, B. ; Rollins, S. ; Xu, Z.: Peer-to-Peer Computing / HPL-2002-57, HP Lab. 2002. – Forschungsbericht

Minar und Hedlung 2001 Minar, N. ; Hedlung, M.: *Peer-to-Peer.* Kap. A Network of Peers. Peer-to-Peer Models Through the History of the Internet, S. 3–20, O'Reilly, 2001

Mizrak et al. 2003 Mizrak, A. ; Cheng, Y. ; Kumar, V. ; Savage, S.: Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In: *Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP).* San Jose, CA, USA, Juni 2003, S. 104–111

Moss und Hosking 1996 Moss, J. ; Hosking, T.: Approaches to Adding Persistence to Java. In: *Proceedings of the 1st International Workshop on Persistence and Java*, September 1996

Muthitacharoen et al. 2002 Muthitacharoen, A. ; Morris, R. ; Gil, T. ; Chen, B.: Ivy: A Read/Write Peer-to-Peer File System. In: *SIGOPS Operating Systems Review* 36 (2002), Nr. SI, S. 31–44

Nardi et al. 2000 Nardi, B. A. ; Whittaker, S. ; Bradner, E.: Interaction and Outeraction: Instant Messaging in Action. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW '00)*, ACM Press, 2000, S. 79–88

Nejdl et al. 2002 Nejdl, W. ; Wolf, B. ; Qu, C. ; Decker, S. ; Naeve, A. ; Nilsson, M. ; Palmer, M. ; Risch, T.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. In: *Proceedings of the 11th International Conference on World Wide Web.* New York, NY, USA : ACM Press, 2002, S. 604–615

Nester et al. 1999 Nester, C. ; Phillippsen, M. ; Haumacher, B.: A More Efficient RMI in Java. In: *Proceedings of the ACM Conference on Java Grande*, 1999, S. 152–159

Newcomer 2002 Newcomer, Eric: *Understanding Web Services.* Addison-Wesley, 2002

Ng et al. 2003 Ng, W. S. ; Ooi, B. C. ; Tan, K. L. ; Zhou, A. Y.: PeerDB: A P2P-based System for Distributed Data Sharing. In: *Proceedings of the International Conference on Data Engineering (ICDE)*, 2003, S. 633–644

Oliveira et al. 1999 Oliveira, M. ; Crowcroft, J. ; Brutzman, D. ; Slater, M.: Components for Distributed Virtual Environments. In: *ACM Symposium on Virtual Reality Software and Technology*, 1999, S. 176–177

Park und Hwang 2003 Park, J. S. ; Hwang, J.: Role-based Access Control for Collaborative Enterprise in Peer-to-Peer Computing Environments. In: *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, ACM Press, 2003, S. 93–99

Petersen et al. 1997 Petersen, K. ; Spreitzer, M. ; Terry, D. ; Theimer, M. ; Demers, A.: Flexible Update Propagation for Weakly Consistent Replication. In: *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, 1997, S. 288–301

Pfister et al. 1998 Pfister, H.-R. ; Schluckmann, C. ; Beck-Wilson, J. ; Wessner, M.: *Cooperative Buildings.* Kap. The Metaphor of Virtual Rooms in the Cooperative Learning Environment Clear, Springer, 1998

Phan et al. 2002 Phan, T. ; Huang, L. ; Dulan, C.: Challenge: Integrating Mobile Wireless Devices into the Computational Grid. In: *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, ACM Press, 2002, S. 271–278

Prpitsch et al. 2005 Prpitsch, Christian ; Lützenkirchen, F. ; Richter, H. ; Bopp, T. ; Hampel, T.: Systemkonvergenz von digitalen Bibliotheken, Planungssystemen und CSCW-Systemen. In: *DeLFI*, 2005, S. 533–537

Pugh et al. 1987 Pugh, J. ; LaLonde, W. ; Thomas, D.: Introducing Object-oriented Programming into the Computer Science Curriculum. In: *Proceedings of the 18th SIGCSE Technical Symposium on Computer Science Education*, 1987, S. 98–102

Quarterman und Hoskins 1986 Quarterman, J. S. ; Hoskins, J. C.: Notable Computer Networks. In: *Communications of the ACM* 29 (1986), Oktober, Nr. 10, S. 932–971

Rashid et al. 1989 Rashid, R. ; Baron, R. ; Forin, A. ; Golub, D. ; Jones, M. ; Julin, D. ; Orr, D. ; Sanzi, R.: Mach: A Foundation for Open Systems. In: *Proceedings of the 2nd IEEE Workshop on Workstation Operating Systems (WWOS)*, September 1989, S. 109–113

Rhea et al. 2003 Rhea, S. ; Eaton, P. ; Geels, D. ; Weatherspoon, H. ; Zhao, B. ; Kubiatowicz, J.: Pond: The Oceanstore Prototype. In: *Proceedings of the Conference on File and Storage Technologies* USENIX (Veranst.), 2003, S. 1–14

Rhea et al. 2004 Rhea, S. ; Geels, D. ; Roscoe, T.: Handling Churn in a DHT. In: *USENIX Annual Technical Conference*, 2004, S. 127–140

Rhea et al. 2005 Rhea, S. ; Godfrey, B. ; Karp, B. ; Kubiatowicz, J. ; Ratnasamy, S. ; Shenker, S. ; Stoica, I. ; Yu, H.: OpenDHT: a public DHT service and its uses. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM).* New York, NY, USA : ACM Press, 2005, S. 73–84

Rheingold 1993 Rheingold, H.: *The virtual Community: Homesteading on the Electronic Frontier.* Addison-Wesley, 1993

Ritter 1995 Ritter, N.: An Infrastructure for Cooperative Applications based on conventional Database Transactions. In: *SIGOIS Bulletin* 15 (1995), Nr. 3, S. 40–48

Roseman und Greenberg 1996 Roseman, M. ; Greenberg, S.: TeamRooms: Network Places for Collaboration. In: *Proceedings of the ACM Conference on Computer supported Cooperative Work*, ACM Press, 1996, S. 325–333

Rowstron und Druschel 2001a Rowstron, A. ; Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: *Lecture Notes in Computer Science* 2218 (2001), S. 329–350

Rowstron et al. 2001 Rowstron, A. ; Kermarrec, A.-M. ; Castro, M. ; Druschel, P.: SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In: *Proceedings of the Conference on Networked Group Communication*, 2001, S. 30–43

Rowstron und Druschel 2001b Rowstron, A. I. T. ; Druschel, P.: Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In: *Proceedings of the Symposium on Operating Systems Principles*, 2001, S. 188–201

Ryan et al. 1999 Ryan, V. ; Lee, R. ; Seligman, S.: *RFC 2714 – Schema for Representing CORBA Object References in an LDAP Directory.* 1999

Samar 1996 Samar, V.: Unified Login with Pluggable Authentication Modules (PAM). In: *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, 1996, S. 1–10

Saroiu et al. 2002 Saroiu, S. ; Gummadi, P. ; Gribble, S.: A Measurement Study of Peer-to-Peer File Sharing Systems. In: *Proceedings of Multimedia Computing and Networking (MMCN)*, 2002, S. 156–170

Schmidt et al. 2004 Schmidt, C. ; Hampel, T. ; Bopp, T.: We've got mail! — Eine neue Qualität der Integration von Nachrichtendiensten in die kooperative Wissensorganisation. In: *Tagungsband der 2. E-Learning Fachtagung Informatik, GI-Edition, Lecture Notes in Informatics*, 2004, S. 211–222

Schmidt 1994 Schmidt, D. C.: The ADAPTIVE communication environment: Object-Oriented Network Programming for Developing Client/Server Applications. (1994), S. 214–225

Seigneur et al. 2003 Seigneur, J. ; Biegel, G. ; Damsgaard, C.: P2P with JXTA-Java pipes. In: *Proceedings of the 2nd International Conference on the Principles and Practice of Programming in Java*, 2003, S. 207–212

Shirky 2000 Shirky, C.: *What is P2P... and what Isn't.* 2000. – URL `www.openp2p.com/lpt/a/p2p/2000/11/24/shirky1-whatisp2p.html`

Shirky 2001 Shirky, C.: *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* Kap. Listening to Napster, S. 21–37, O'Reilly, 2001

Smith et al. 2003 Smith, D. A. ; Kay, A. C. ; Reed, D. P.: Croquet – A Collaboration System Architecture. In: *Proceedings of the 1st IEEE Conference on Creating, Connecting and Collaborating through Computing*, 2003, S. 2–9

van Steen et al. 1997 Steen, M. van ; Homburg, P. ; Tanenbaum, A. S.: The Architectural Design of Globe: A Wide-Area Distributed System. Netherlands, 1997 (IR-422). – Forschungsbericht

Steinman et al. 1998 Steinman, J. ; Wallace, J. ; Davani, D. ; Elizandro, D.: Scalable Distributed Military Simulations using the SPEEDES Object-oriented Simulation Framework. In: *Proceedings of the Conference on Object-Oriented Simulation (OOS)*, 1998, S. 3–23

Stoica et al. 2001 Stoica, I. ; Morris, R. ; Karger, D. ; Kaashoek, F. ; Balakrishnan, H.: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001, S. 149–160

Sugano et al. 1997 Sugano, H. ; Otani, K. ; Ueda, H. ; Hiraiwa, S. ; Endo, S. ; Kohda, Y.: SpaceFusion: A Multi-server Architecture for Shared Virtual Environments. In: *Proceedings of the 2nd Symposium on Virtual Reality Modeling Language*, ACM Press, 1997, S. 51–58

Tanenbaum und van Steen 2003 Tanenbaum, A. S. ; Steen, M. van: *Verteilte Systeme*. Pearson Studium, 2003

Wang 1999 Wang, W.: Team-and-role-based organizational context and access control for cooperative hypermedia environments. In: *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia : Returning to our diverse Roots*, ACM Press, 1999, S. 37–46

Waters et al. 1997 Waters, R. ; Anderson, D. ; Barrus, J. ; Brogan, D. ; Casey, M. ; McKeown, S. ; Nitta, T. ; Sterns, I. ; Yerazunis, W.: Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtine Modifiability. In: *Presence: Teleoperators and Virtual Environments* 6 (1997), Nr. 4, S. 461–480

Watsen und Zyda 1998 Watsen, K. ; Zyda, M.: Bamboo – A Portable System for Dynamically Extensible, Real-Time, Networked Virtual Environments. In: *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, 1998, S. 252–259

Welsh et al. 2001 Welsh, M. ; Culler, D. ; Brewer, E.: SEDA: An Architecture for well-conditioned, scalable Internet Services. In: *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, ACM Press, 2001, S. 230–243

Wessner et al. 2003 Wessner, M. ; Dawabi, P. ; Fernandez, A.: Supporting Face-to-Face Learning With Handheld Devices. In: Wasson, B. (Hrsg.) ; Ludvigsen, S. (Hrsg.) ; Hoppe, U. (Hrsg.): *Proceedings of the International Conference on Computer Supported Learning Environments: Designing for Change in Network Learning Environments*, Kluwer, 2003, S. 487–491

Wiebe 1986 Wiebe, D.: A Distributed Repository for Immutable Persistent Objects. In: *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, 1986, S. 453–465

Wolfson et al. 1997 Wolfson, O. ; Jajodia, S. ; Huang, Y.: An adaptive Data Replication Algorithm. In: *ACM Transactions on Database Systems* 22 (1997), Nr. 2, S. 255–314

Yeong et al. 1995 Yeong, W. ; Howes, T. ; Kille, S.: *RFC 1777 – Lightweight Directory Access Protocol.* März 1995

Zyda et al. 1992 Zyda, M. ; Pratt, D. ; Monahan, J. ; Wilson, K.: NPSNET: Constructing a 3D Virtual World. In: *Symposium on Interactive 3D Graphics*, 1992, S. 147–156