Kapitel 2

Grundlagen

In diesem Kapitel sollen die Grundlagen der vorliegenden Arbeit vermittelt werden. Zunächst wird der Begriff der Awareness in der kollaborativen Gruppenarbeit präzisiert und mögliche Verfahren beleuchtet, die diese Awareness herstellen können.

Ohne die Vernetzung der Arbeitsplätze aller Gruppenmitglieder ist eine Unterstützung der Gruppenarbeit durch den Computer nicht möglich. Deshalb werden anschließend Kommunikationsstrategien in Computernetzwerken betrachtet, um die Rahmenbedingungen eines Awareness-Systems zu erkennen.

2.1 Awareness in der kooperativen Gruppenarbeit

Der Begriff der Gruppenarbeit ist von zentraler Bedeutung für diese Arbeit. In der Gruppenarbeit versuchen mehrere Personen zusammen ein gemeinsames Ziel zu erreichen, indem jeder zu einem gewissen Anteil zum Erreichen dieses Ziels beiträgt. Herausforderungen stellen dabei sowohl die Unterstützung der Kooperation der einzelnen Gruppenmitglieder untereinander als auch die Koordination der gemeinsamen Bestrebungen.

Die hohe Verfügbarkeit von Kommunikationsnetzwerken ermöglichte es, diese Gruppenarbeit von mitunter stark verteilten Gruppen durch den Computer zu unterstützen. Der Begriff der *Computer-Unterstützten Kooperativen Gruppenarbeit* (*Computer-Supported Cooperative Work*, *CSCW*) bezeichnet das interdisziplinäre Forschungsgebiet der Gruppenarbeit im Kontext von verteilten Systemen [SKB98]. Als *Groupware* hingegen werden Systeme bezeichnet, welche den Menschen beim Arbeiten in einer Gruppe unterstützen – weniger das Lösen von Problemen mit Hilfe des Computers [EGR91].

In diesem Abschnitt werden die verschiedenen Aspekte betrachtet, die bei der Unterstützung der Gruppenarbeit durch den Computer eine Rolle spielen. Dabei soll zunächst im Vordergrund stehen, wie die Interaktion der Mitglieder einer Gruppe erfolgen kann und welche Eigenschaften die Aufgaben besitzen, die von mehreren Personen bearbeitet werden.

2.1.1 Gruppeninteraktion

Um die Interaktionsmöglichkeiten mehrerer Gruppenmitglieder zu modellieren werden die Dimensionen Ort und Zeit verwendet. Diese Ort-Zeit-Taxonomie erlaubt es, die Synchronität (wann erreicht eine Aussage einen Interaktionspartner) und die räumliche Verteilung der Interaktion (wo befindet sich der Interaktionspartner) zu verdeutlichen. Ein umfassendes Groupware-System sollte alle möglichen Kombinationen der Ort- und Zeit-Dimensionen unterstützen [EGR91]. Diese Kombinationen sind die Folgenden:

- **Synchrone Interaktion:** Die Interaktion findet von Angesicht zu Angesicht statt. Alle Gruppenmitglieder befinden sich zum gleichen Zeitpunkt am gleichen Ort – sie könen direkt miteinander interagieren. Beispiele hierfür sind Seminare, Besprechungen und Konferenzen.

- **Verteilte synchrone Interaktion:** Die Gruppenmitglieder befinden sich zwar an unterschiedlichen Orten, können aber über ein Kommunikationsmedium unmittelbar miteinander interagieren. Dies kann ein Telefonat oder eine Instant-Messaging-Sitzung sein.

- **Asynchrone Interaktion:** Gruppenmitglieder besuchen den gleichen Ort zu unterschiedlichen Zeiten, sind aber dennoch in der Lage, über ein Kommunikationssystem miteinander zu interagieren. Ein derartiges Kommunikationssystem ist beispielsweise eine Magnettafel (Whiteboard). Man kann eine Nachricht daran heften, die andere Gruppenmitglieder zu einem späteren Zeitpunkt ansehen bzw. Kommentare dazu heften können.

- **Asynchrone verteilte Interaktion:** Gruppenmitglieder befinden sich weder am gleichen Ort, noch findet die Interaktion zum gleichen Zeitpunkt statt. Ein klassisches Beispiel hierfür ist die Post oder auch elektronische Post.

Abbildung 2.1 zeigt die aus diesen Kombinationen resultierende Ort-Zeit-Matrix mit beispielhaften Anwendungsfällen.

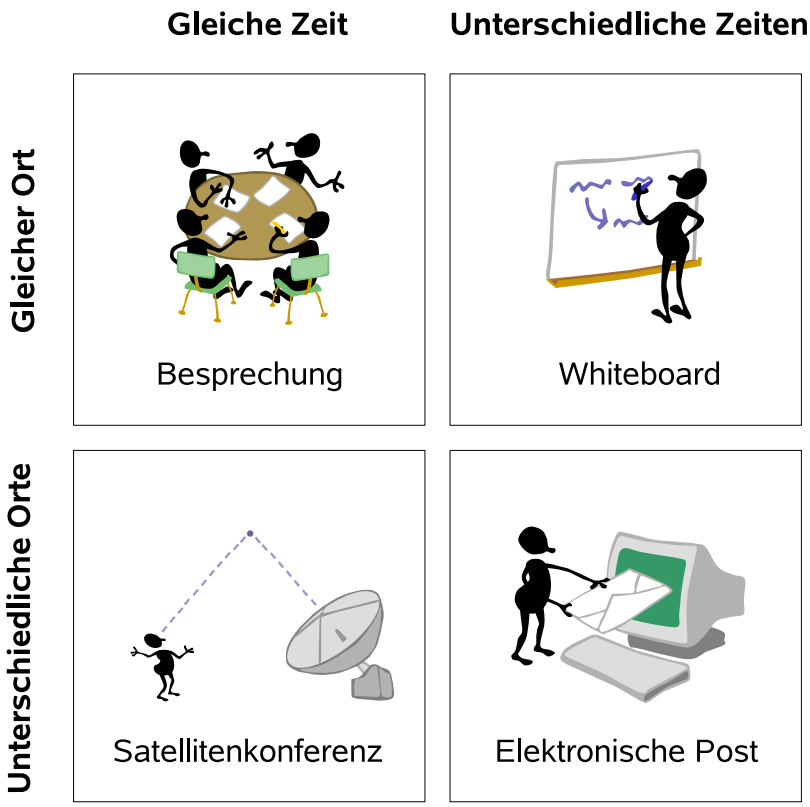


Abbildung 2.1: Die Ort-Zeit-Matrix mit Beispielanwendungen für die einzelnen Kombinationen

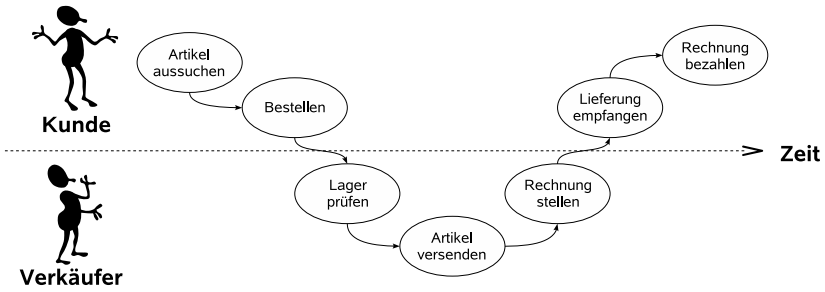
Da nun eine Grundlage für die Beschreibung von Interaktionen der Gruppenmitglieder gelegt ist, wird als nächstes betrachtet, welche Arten von Aufgaben es gibt, und wie sie in einer Gruppe bearbeitet werden können.

2.1.2 Aufgabenklassifikation

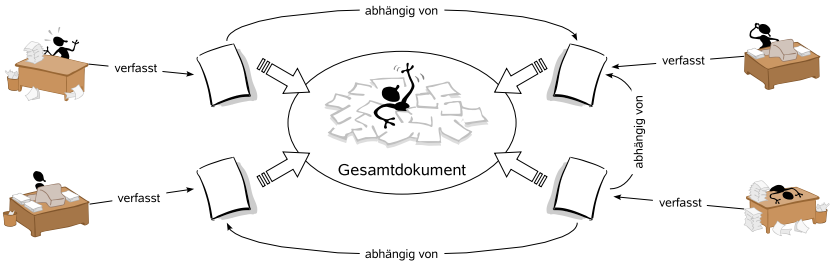
Nach Bürger kann man zwischen zwei Aufgabenarten unterscheiden – den *stark strukturierten* und den *schwach strukturierten* Aufgaben [Bür99].

Stark strukturierte Aufgaben *Stark strukturiert* sind diejenigen Aufgaben, die sich in einer fest definierten Folge von Schritten durchführen lassen. Der Computer kann durch sein Wissen über die Struktur der Aufgabe die Menschen in jedem Schritt unterstützen und die Koordination der Aufgabe vereinfachen. Diese Koordination wird als *explizite Koordination* bezeichnet [SKB98, Bür99]. Abbildung 2.2(a) zeigt einen einfachen Bestellprozess, der

als stark strukturierte Aufgabe angesehen werden kann.



(a) *Stark strukturierte Aufgabe:* Ein einfacher Bestellprozess



(b) *Schwach strukturierte Aufgabe:* Verteilte Dokumenterstellung

Abbildung 2.2: Aufgabenklassifikation (a) Ein einfacher Bestellprozess, in dem jeder Schritt im voraus bekannt ist. Der Computer kann durch sein Wissen über die Struktur in jedem Schritt unterstützend eingesetzt werden. (b) Änderungen an Dokumentteilen eines Autors haben Einfluss auf die Dokumentteile eines anderen Autors. Ohne einen geeigneten Synchronisierungsmechanismus wird der Aufwand für die Erstellung des Gesamtdokuments enorm wachsen.

Schwach strukturierte Aufgaben Aufgaben, die sich nicht in eine rigide Struktur bringen lassen werden als *schwach strukturierte* Aufgaben bezeichnet. Dies sind unter anderem Aufgaben die ein höheres Maß an Kreativität erfordern und daher keine vollends vorhersehbaren Arbeitsabläufe nach sich ziehen (z.B. das gemeinsame Verfassen eines Dokuments). Eine solche Aufgabe folgt dennoch einigen Regeln, deshalb wird hier der Begriff *schwach stukturiert* [Bür99] dem Begriff *unstrukturiert* [SKB98] vorgezogen.

Abbildung 2.2(b) zeigt als Beispiel für eine schwach strukturierte Aufgabe einen vereinfachten Prozess zur gemeinsamen Erstellung eines Doku-

ments. Besonderes Augenmerk liegt auf den Abhängigkeiten der einzelnen Teildokumente untereinander. Änderungungen an einem Teildokument können Änderungen an einem Teildokument eines anderen Autors erfordern. Ohne Koordinierung wird das Zusammensetzen der Teildokumente zu einer schwierigen Angelegenheit.

Beim Lösen einer gemeinsamen Aufgabe kommt es oft vor, dass die Aktionen eines einzelnen Gruppenmitglieds Einfluss auf die Arbeit der anderen Gruppenmitglieder haben kann. Zur Unterstützung der Zusammenarbeit aller Gruppenmitglieder ist der Austausch von Informationen über die Aktivitäten jedes einzelnen enorm wichtig. Das Bewusstsein um die Aktivitäten der anderen Gruppenmitglieder wird in der kooperativen Gruppenarbeit als *Awareness* bezeichnet.

2.1.3 Awareness

Nach Dourish und Belotti ist Awareness das Wissen über die Aktivitäten anderer Gruppenmitglieder, um einen Bezug für die eigene Aktivität herzustellen[1] [DB92].

Für das Beispiel aus Abbildung 2.2(a) bedeutet dies, dass der jeweilige Akteur immer weiß, wann eine Aktivität von ihm erforderlich ist. Nachdem der Verkäufer sich bewusst ist, dass er eine Bestellung zu bearbeiten hat, kann er damit beginnen, seinen Teil des Prozesses abzuarbeiten. Durch die eingehende Rechnung wird der Kunde sich bewusst, dass die Bestellung finanziell beglichen werden muss. Es ist auch denkbar, dass der Kunde über weitere Aktivitäten informiert ist, etwa wie weit der Bestellvorgang vom Verkäufer schon bearbeitet wurde. Es entsteht jedoch dadurch kein Mehrwert im Bezug auf den betrachteten Prozess – dies würde jedoch den Kunden während der Wartezeit zufriedener machen.

Das Beispiel aus Abbildung 2.2(b) kann ebenfalls aus Sicht der Awareness der Gruppenmitglieder betrachtet werden. Ist sich ein Autor A bewusst, dass ein anderer Autor B Änderungen an seinem Teildokument vorgenommen hat, auf die A in seinem Teildokument eingehen muss, dann kann er die Änderungen unmittelbar vornehmen oder zumindest im Hinterkopf behalten. Diese Koordination kann den Arbeitsablauf der Gruppe strukturieren und dadurch effizienter gestalten.

Als nächstes wird eine „systematische Einordnung des Informationsspektrums, das Awareness ausmacht“ vorstellt [Bür99]. Grundlage dieser Einordnung ist die Arbeit von Gutwin [Gut97], die aus vielen Veröffentlichungen

[1] „An understanding of the activities of others, which provide a context for your own activity.“

von Greenberg, Gutwin et al. zusammensetzt ist.

Arten von Awareness

Gutwin unterscheidet vier sich teilweise überlappende Arten von Awareness [Bür99].

Informale Awareness Die informale Awareness beschreibt das Wissen über anwesende Personen. In der realen Welt ist dies automatisch gegeben. Man bemerkt zum Beispiel, wenn eine Person den Raum betritt oder verlässt.

In einer virtuellen Umgebung benötigt man zunächst Informationsräume, über die dieser Bezug hergestellt werden kann. Die Frage lautet dann, mit welchen Personen man sich in einem Informationsraum befindet. Ein weit verbreiteter Anwendungsfall ist beispielsweise ein *Instant-Messaging*-System.

Gruppen-strukturelle Awareness Die gruppen-strukturelle Awareness betrifft das Wissen über die Rollen und Verantwortlichkeiten der verschiedenen Gruppenmitglieder. Mit Hilfe dieses Wissens kann man bei der gemeinsamen Arbeit Rückschlüsse über das Verhalten einzelner Gruppenmitglieder ziehen. Beim gemeinsamen Lösen einer stark strukturierten Aufgabe ist diese Art der Awareness nicht sehr ausschlaggebend, weil der Arbeitsfluss bereits vorher genau spezifiziert wurde. Bei schwach strukturierten Aufgaben wird das Rollenverständnis der Gruppenmitglieder in der Regel spontan ausgehandelt und deshalb ist eine Unterstützung durch den Computer nur schwer zu realisieren [Bür99].

Soziale Awareness Bei der sozialen Awareness handelt es sich um das Wissen über den sozialen und konversationsbezogenen Zustand der verschiedenen Gruppenmitglieder [GGR96]. Man ist sich zum Beispiel bewusst, ob der gegenüber gerade aufmerksam ist oder sich langweilt. Dies ist von einem computer-gestützten Awareness-System nur schwer zu unterstützen. In der Praxis bleibt in diesem Punkt oft nur die Videokonferenz, um einen derartigen Eindruck zu vermitteln.

Arbeitsbereich-Awareness In einem gemeinsamen Arbeitsbereich gibt es eine Menge von Informationsobjekten, um die sich die Arbeit dreht, eine Menge von Personen, die zusammen an einem gemeinsamen Ziel arbeiten, sowie eine Menge von Relationen zwischen Gruppenmitgliedern und Informationsobjekten. Bei der Awareness in einem gemeinsamen Arbeitsbereich handelt es sich um das Wissen über die Aktivitäten der einzelnen Personen

und den Zustand der einzelnen Informationsobjekte und die Kooperation so effektiv wie möglich zu gestalten. Eine Person ist sich im Idealfall über alle für die eigene Aufgabe relevanten Geschehnisse im Arbeitsbereich bewusst kann dementsprechend optimal handeln.

Über die Arbeitbereich-Awareness kann die informale und die gruppenstrukturelle Awareness teilweise unterstützt werden, da der Computer die Geschehnisse in einem gemeinsamen Arbeitsbereich gut beobachten kann. Die Betrachtung von Arbeitsbereich-Awareness bietet also einen guten Ansatzpunkt zur Unterstützung von Awareness durch den Computer [Bür99].

Als nächstes soll betrachtet werden, welche Möglichkeiten es gibt um die Arbeitsbereich-Awareness der Benutzer durch den Computer zu verbessern.

2.1.4 Unterstützung der Awareness im Arbeitsbereich durch den Computer

Die bisherigen Anforderungen, die an ein computergestütztes Awareness-System gestellt werden, lassen sich wie folgt zusammenfassen:

1. Das System kennt Gruppenmitglieder, Informationsobjekte und Relationen im Arbeitsbereich
2. Das System ist über Aktivitäten im Arbeitsbereich informiert.
3. Sofern es nötig ist, wird das System die Awareness eines Gruppenmitglieds verbessern, indem es ihm eine Nachricht mit relevanten Informationen zukommen lässt.
4. Das Gruppenmitglied soll nicht durch unnötige Nachrichten von seiner Arbeit abgelenkt werden.

Bei diesen Anforderungen handelt es sich um Annahmen, die nur ein ideales System erfüllen kann. Um die letzten beiden Punkte zu erfüllen müsste der Computer exakt wissen, was einen Benutzer interessiert und was ihm weiterhilft. Dies ist in einer praktischen Umsetzung nur näherungsweise möglich.

Das Awareness-System wird also zunächst noch als Blackbox aufgefasst (Abbildung 2.3). Dabei werden an dieser Stelle keine Annahmen über die Interaktionsformen der Gruppenarbeit (vgl. Abschnitt 2.1.1) und die angestrebte Aufgabenklasse (vgl. Abschnitt 2.1.2) getroffen.

Eine Reihe von Modellen wurden vorgeschlagen, um dieses Awareness-System zu realisieren. Darunter befinden sich die folgenden:

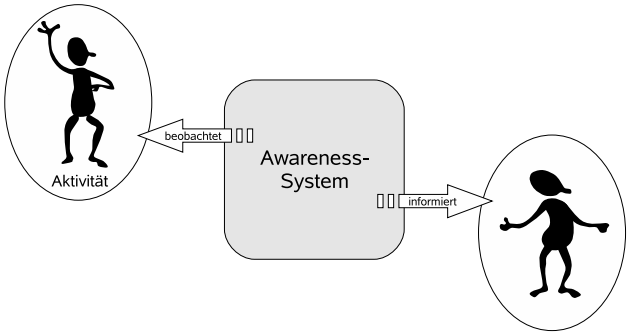


Abbildung 2.3: Das Awareness-System als Blackbox. Der eigentliche Mechanismus ist einfach zu beschreiben, die Herausforderung stellt jedoch das intelligente Benachrichtigen beim Auftreten von Aktivitäten dar. Verschiedene Modelle existieren, um ein derartiges Awareness-System zu realisieren.

Publish-Subscribe: In diesem Modell kann jedes Gruppenmitglied *Ereignisse* aussenden, die das Awareness-System über ihre Aktivitäten informiert (*Publish*). Gruppenmitglieder können über sogenannte *Interessen* dem Awareness-System mitteilen, über welche Ereignisse sie informiert werden möchten (*Subscribe*). Nun kann das System eindeutig bestimmen welche Gruppenmitglieder über welche Ereignisse informiert werden müssen.

Sitzungsmodell: Im Sitzungsmodell können sich die Gruppenmitglieder bei einer *Sitzung* anmelden, in der dem Awareness-System die Informationsobjekte und alle Gruppenmitglieder bekannt sind. Alle Teilnehmer einer Sitzung werden nun von den Aktivitäten im gemeinsamen Arbeitsbereich informiert. Hierbei spielt zeitliche und räumliche Verteilung keine Rolle. Das Sitzungsmodell ist im Grunde ein Spezialfall des Publish-Subscribe-Modells, in dem alle Teilnehmer beim Betreten der Sitzung die gleichen Interessen zugewiesen bekommen.

Nimbus-Fokus: Das von Benford und Fahlén [BF93] vorgestellte Nimbus-Fokus-Modell ist im Gegensatz zu den anderen hier betrachteten Systemen ein theoretisches Modell, für das eine konkrete Umsetzung offen gelassen wurde.

Als Interaktionsmetapher dienen virtuelle Welten, in denen sich die Gruppenmitglieder und Informationsobjekte befinden. Der *Nimbus* des Benutzers ist sein Aufenthaltsort in der virtuellen Welt. Ein Aufenthaltsort kann hierbei auch als Aktivität (bzw. Ereignis) aufgefasst werden. Sein *Fokus* beschreibt dabei die Orte, die der Benutzer betrachtet

oder die für ihn interessant sind. Man kann ihn auch als Interessensprofil ansehen.

Anhand des Nimbus und des Fokus der verschiedenen Gruppenmitglieder kann nun die Größe des gegenseitigen Bewusstseins ermittelt werden. Dieser errechnet sich aus dem Durchschnitt von Nimbus und Fokus. Je größer der Durchschnitt, desto größer ist das gegenseitige Interesse der Gruppenmitglieder.

GroupDesk: Das *GroupDesk*-Modell wurde in der Arbeit von Fuchs et al. entwickelt [FPBP95]. Alle Gruppenmitglieder und Informationsobjekte werden über Relationen in Beziehung gesetzt, was anders als in den beiden vorigen Modellen eine Strukturierung des Arbeitsbereichs voraussetzt. Die Awareness der Gruppenmitglieder kann bei den in Abschnitt 2.1.1 vorgestellten Interaktionsformen unterstützt werden.

Um die Benachrichtigung der Benutzer zu kontrollieren setzt GroupDesk so genannten Interessenskontexte ein. Dazu gehört eine *Ereignisliste*, die angibt, welche Operation auf einem Informationsobjekt ein bestimmtes Ereignis auslöst. Eine *Situationsbeschreibung* legt über Relationen zwischen den Informationobjekten fest, wie Ereignisse weitergeleitet werden. Schließlich kann ein Gruppenmitglied über eine Interessentenliste festlegen, wie wichtig eine Benachrichtigung durch ein bestimmtes Ereignis ist.

Die verschiedenen Modelle haben eines gemeinsam: Durch eine gezielte Ereignisbenachrichtigung wird der Benutzer über Aktivitäten im Arbeitsbereich informiert. Der Unterschied besteht lediglich in der Art, wie entschieden wird, welche Nachrichten für ein Gruppenmitglied von Interesse sind und welche nicht. Das Publish-Subscribe-Modell kann für jedes der vorgestellten Modelle als Grundbaustein der Realisierung dienen. Man muss hierbei nur die möglichen Interessen und Ereignisse passend modellieren.

Aus diesem Grund werden die Merkmale des Publish-Subscribe-Modells nun detaillierter beleuchtet.

2.1.5 Das Publish-Subscribe-Modell

Das Publish-Subscribe-Modell kann als Kommunikationsgrundlage für ein Awareness-System genutzt werden. Die Arbeit von Eugster et al. [EFGK03] bietet einen umfassenden Überblick über dieses Kommunikationsparadigma. Die Benutzer des Publish-Subscribe-Systems können *Ereignisse*, über die sie informiert werden möchten, durch *Interessen* spezifizieren.

Wie bereits betrachtet kann die Kommunikation zwischen den verschiedenen Benutzern entlang der Dimensionen Zeit und Ort aufgespalten werden. Eugster et al. schlagen zuätzlich vor, die Dimension Synchronisation in die Betrachtung mit einzubeziehen. Sie gibt auf der Ereignis Seite an, ob der Erzeuger blockiert wird, bis die Benachrichtigungen empfangen wurden. Auf der Empfänger-Seite gibt die Synchronisation an, ob der Interessent beim Empfangen einer Benachrichtugung nebenläufig informiert wird oder nicht.

Diese drei Dimensionen werden später bei der Analyse des entwickelten Awareness-Systems eine wichtige Rolle spielen, da sie darüber Auskunft geben können, welche Arten von Awareness das System unterstützen wird.

Als nächstes soll geklärt werden, welche Möglichkeiten der Benutzer hat, um seine Interessen zu formulieren. Um dem Awareness-System mitzuteilen, welche Benachrichtigungen man empfangen möchte, gibt es folgende Möglichkeiten [EFGK03]:

- Themenbasiertes Publish-Subscribe
- Inhaltsbasiertes Publish-Subscribe
- Typbasiertes Publish-Subscribe

Themenbasiertes Publish-Subscribe

Ein Thema ist ein zusammenfassender Begriff für eine Menge von Ereignissen die im System ausgelöst werden können. Der Benutzer meldet sein Interesse beim Awareness-System an, indem er dem System mitteilt, dass er über alle Ereignisse eines bestimmten Themas informiert werden möchte. Das Abonnement eines bestimmten Themas kann auch als Zugehörigkeit zu einer Gruppe angesehen werden (zum Beispiel bei IP Multicast oder eine Usenet Newsgroup). Ein Vorteil beim Einsatz von themenbasierten Publish-Subscribe-Systemen ist die Verfügbarkeit von effizienten Verfahren zum Verbreiten von Nachrichten [OAA+00].

Zusätzlich zu einer flachen Themenstruktur gibt es auch die Möglichkeit, die Themen durch eine Hierarchie flexibler gestalten zu können. So gibt es Systeme, bei denen der Beitritt zu einem bestimmten Thema den Beitritt zu Unterthemen impliziert. Über diese Erweiterungen hinaus bietet das Themenbasierte Publish-Subscribe nur begrenzte Flexibilität. Es stellt lediglich ein statisches Schema zur Verfügung.

Inhaltsbasiertes Publish-Subscribe

Im inhaltsbasierten Ansatz ist die Benachrichtigung eines Benutzers abhängig vom Inhalt eines Ereignisses, nicht von seinem Thema. Hierzu besitzen die

Ereignisse verschiedene Eigenschaften, die durch so genannte Attribute beschrieben werden. Der Interessent kann genau spezifizieren, welche Attributbelegungen eines Ereignisses gelten müssen, damit er von seinem Auftreten benachrichtigt werden soll.

Dieser Mechanismus erlaubt eine gezieltere Formulierung der Interessen im Vergleich zum themenbasierten Ansatz und ist damit wesentlich flexibler, die Umsetzung des Ereignisbenachrichtigung stellt allerdings eine größere Herausforderung dar als im themenbasierten Modell [OAA+00].

Attributschemata Damit der Benutzer des Systems Interessen formulieren kann, für die auch tatsächlich passende Ereignisse auftreten können, muss das System verfügbare Attribute und mögliche Werte vorher spezifizieren. Sowohl bei der Modellierung der Ereignisse als auch der Interessen ist daher ein gemeinsames Attributschema notwendig. Das Publish-Subscribe-System in Abbildung 2.4 verwendet die Attribute *name*, *aktion*, *objekt*, *wann* und *empänger*.

Ereignismodellierung Ein Ereignis kann verschiedene Eigenschaften besitzen. Es besitzt eine Reihe von *Attributen* mit zugewiesenen Werten, die die Ausprägung der Eigenschaft darstellen. Bei der Veröffentlichung eines Ereignisses weist der Erzeuger allen betroffenen Attributen den entsprechenden Wert zu und verbreitet es nach einem bestimmten Schema. In Abbildung 2.4 wird über das Systemprimitiv *subscribe*() ein Ereignis veröffentlicht. Ein Ereinis selbst wird hier als Tupel von Attributen und ihren zugewiesenen Werten dargestellt.

Interessenmodellierung Bevor ein Benutzer von einem Ereignis benachrichtigt wird, muss er alle seine Interessen (*Interessenprofil*) beim System anmelden (*Subscribe*). Erst wenn das System alle Interessenprofile aller Benutzer kennt kann es alle Interessenten beim Auftreten eines bestimmten Ereignisses informieren.

Um nun ein Interesse zu formulieren kann der Benutzer über eine vorgegebene Sprache einen Filter definieren. Für jedes relevante Attribut kann er eine prädikatenlogische Bedingung angeben, die erfüllt sein muss. Wenn alle diese Bedingungen von einem aufgetretenen Ereignis erfüllt werden, dann wird der Interessent benachrichtigt. Eine Erweiterung dieses Modells sieht die Kombination von elementaren Ereignissen vor, bei der ein Benutzer erst dann informiert wird, wenn mehrere Ereignisse zusammen auftreten [BMB+00].

Die Beispielinteressen in Abbildung 2.4 besitzen als Bedingung die Überprüfung auf Gleichheit. Hier wird eine Fragezeichen verwendet um anzuge-

ben, dass ein Attribut beim Vergleich nicht berücksichtigt werden soll. Man hätte diese Bedingung aber auch genauso gut weglassen können.

Abonnementbeschreibungen Um dem System mitzuteilen wie ein Interesse oder ein Ereignis definiert ist gibt es unterschiedliche Darstellungsformen [EFGK03]:

- *Stringbasiert:* Am häufigsten werden Filterkriterien durch stringbasierte Sprachen beschrieben. Diese Strings werden anschließend vom Awareness-System analysiert.
- *Vorlagenobjekte:* Aus dem Objektorientierten System JavaSpaces[2] stammt das vorlagenorientierte Abonnement. Der Benutzer übergibt beim Abonnieren ein Objekt, gegen das ein Ereignis verglichen wird. Der Benutzer wird benachrichtigt, wenn das aufgetretene Objekt von gleichen Typ ist und alle Attribute mit den Attributen des Vorlagenobjekts übereinstimmen. Attribute des Vorlagenobjekts, die den Wert *null* tragen, werden beim Vergleich ignoriert (*Wildcards*).
- *Ausführbarer Code:* Der Benutzer sendet als Abonnement ein Stück ausführbaren Code, der entscheiden kann, ob er über ein Ereignis benachrichtigt werden soll. Diese Darstellung wird am seltensten genutzt, weil die resultierenden Filter sehr schwierig zu optimieren sind.

Beispiele für inhaltsbasierte Systeme sind *Gryphon* [BCM+, ASS+99], *Siena* [CRW00] und Jedi [CNF01].

Typbasiertes Publish-Subscribe

Eine weitere Erweiterung der Flexibilität von themenbasierten System stellt das typbasierte Publish-Subscribe dar. Die namensbasierte Themenbeschreibung wird dabei durch Typen ersetzt [EGD01]. Im Gegensatz zum vorlagenbasierten Ansatz in Abschnitt 2.1.5 wird hier der Typ nicht dynamisch ermittelt sondern bereits zur Compile-Zeit. Ansonsten sind beide Ansätze sehr ähnlich.

Um gezielte Interessen kompakt formulieren zu können bietet der inhaltsbasierte Ansatz größere Flexibilität als der themenbasierte. Im schlechtesten Fall haben N Benutzer paarweise disjunkte Abonnements und die resultierende Gruppenmächtigkeit für alle Themen würde $O(2^N)$ betragen [OAA+00].

[2] `http://java.sun.com/docs/books/jini/javaspaces/`

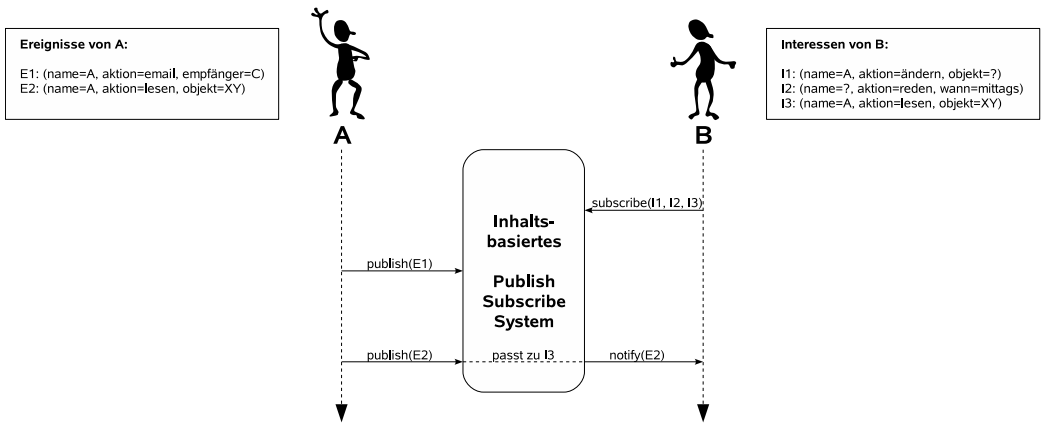


Abbildung 2.4: Beispielschema eines inhaltsbasierten Publish-Subscribe-Systems. B besitzt einige Interessen, die hier stringbasiert dargestellt sind. Ein Fragezeichen gibt an, dass ein Attribut beim Vergleich mit dem entsprechenden Attribut eines Ereignisses vernachlässigt werden kann. Nachdem B seine Interessen bekannt gegeben hat, erzeugt A zunächst ein Ereignis, dass keinem Interesse von B entspricht. Das Ereignis E2 stimmt schließlich mit dem Interesse I3 überein und B wird benachrichtigt.

Für den inhaltsbasierten Ansatz hingegen ist es kein Problem, die themenbasierten Abonnements zu integrieren. Ein Themen-Abonnement resultiert in genau einer Art Interesse. Der Unterschied zwischen dem typbasierten Ansatz und dem inhaltsbasierten fällt hingegen gering aus. Deswegen spielt der Typ-basierte Ansatz im folgenden keine Rolle mehr, da er mit leichten Einschränkungen auch vom inhaltsbasierten Ansatz erreicht werden kann.

Die Flexibilität bei der Spezifizierung von Interessen im inhaltsbasierten Publish-Subscribe hat jedoch ihren Preis. Der Benutzer hat erst dann einen Vorteil von der Verwendung eines Awareness-Systems, wenn der Aufwand zur Eingabe seiner Interessen wesentlich geringer ist, als der Nutzen, den er aus den Benachrichtigungen ziehen kann. Aufbauend auf dem Publish-Subscribe-Schema muss also ein Mechanismus existieren, der den Benutzer beim Erstellen von Interessen unterstützt, beispielsweise indem er aus dem Verhalten des Benutzers Rückschlüsse auf seine Interessen zieht. Dieser Aspekt wird jedoch in dieser Arbeit nicht weiter verfolgt.

Alle vorgestellten Verfahren haben eine Gemeinsamkeit: Sie sind davon abhängig, dass alle Teilnehmer mit dem Awareness-System kommunizieren können. Das Kommunikationsschema beeinflusst dabei auch, wie die Datenverarbeitung stattfindet und welche Rahmenbedingungen zu beachten sind. Wie Computersysteme über Netzwerke miteinander kommunizieren können

wird daher im nächsten Abschnitt detaillierter betrachtet.

2.2 Netzwerkarchitekturen

Mehrere Computer zu einem Netzwerk zusammenzuschließen hat erst dann Sinn, wenn man einen Nutzen daraus ziehen kann, den man ohne die Vernetzung nicht hat. Beispiele für Netzwerk-Anwendungen sind zum Beispiel der Zugriff auf Daten, die auf anderen Rechnern gespeichert sind oder die Kommunikation über elektronische Post. Die große Vielfalt an nutzbringenden Diensten ist wohl zweifelsfrei der Antrieb für die stetige Vergrößerung des Internets, aber auch im lokalen Netzwerk können die Fähigkeiten eines einzelnen Rechners durch den Zusammenschluss mit anderen erweitert werden.

Die Rollen in einem Netzwerk kann man verallgemeinern auf Anbieter von Diensten und Nutzer dieser Dienste. Wie Dienste im Netzwerk erreichbar sind und welche Kommunikation zu ihrer Nutzung nötig ist hat große Auswirkungen auf die Leistungsparameter dieser Dienst-Infrastruktur.

Zwei sehr weit verbreitete Architekturen für die Netzwerk-Kommunikation sind das Client-Server-Modell und das Peer-to-Peer-Modell. Im folgenden werden diese beiden Architekturen vorgestellt und ihre Eigenschaften untersucht.

2.2.1 Das Client-Server-Modell

Viele Arbeiten, die sich mit dem Client-Server-Modell auseinandersetzen fangen mit der Feststellung an, dass es keine einheitliche Definition für den Client-Server-Begriff in der Literatur gibt [DJT96]. Ein weit verbreitetes Einverständnis besteht jedoch darin, dass das Client-Server-Modell auf der logischen Trennung zwischen den Dienstanbietern (Server) und deren Nutzern (Clients) basiert [CR97]. Abbildung 2.5 zeigt eine schematische Darstellung – mehrere Clients greifen über das Netzwerk auf den Server zu. Wie dieser Zugriff erfolgt spielt bei der Charakterisierung des Modells keine Rolle.

Diese Trennung hat den Vorteil, dass ein bestimmter Dienst in einem Netzwerk über eine wohldefinierte Schnittstelle vielen Clients zur Verfügung steht. Auch das Aktualisieren eines Dienstes ist komfortabel, solange sich die Schnittstelle nicht verändert. Man muss lediglich die Server-Software austauschen und alle Clients greifen automatisch auf den erneuerten Dienst zu.

Fällt ein Server aus, so sind alle Clients unmittelbar davon betroffen, weil der gewünschte Dienst nicht mehr verfügbar ist. In diesem Modell besteht daher eine große Abhängigkeit von der Zuverlässigkeit und Leistung des

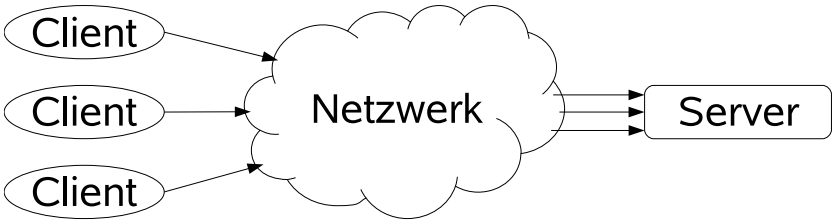


Abbildung 2.5: Kommunikationsschema des Client-Server-Modells

Servers. In der Literatur wird diese zentrale Rolle als *single point of failure* kritisiert [ADN+95, ATS04, LCC+02], die Verfügbarkeit des Servers ist das Nadelöhr des Systems.

Ein Ausfall kann zum Beispiel durch einen Defekt oder ein Netzwerk-Problem entstehen, aber auch bei Überlastung des Servers durch enorm viele Client-Anfragen zu einem Zeitpunkt. Die Ausfall-Sicherheit und Skalierbarkeit im großen Maßstab ist zwar möglich, jedoch mit erheblichen Kosten verbunden. Ein gutes Beispiel hierfür ist der beliebte Internet-Suchdienst Google[3], dessen System sich mit tausenden von Anfragen pro Sekunde konfrontiert sieht und diese Last im Jahre 2003 unter anderem mit dem Zusammenschluss von 15.000 handelsüblichen Computern bewältigt hat [BDH03]. Bis heute hat Google mehr als acht Milliarden Webseiten indexiert[4] – mit steigender Tendenz.

In den späten achtziger und frühen neunziger Jahren vollzog sich der Wechsel von monolithischen Mainframe-Systemen hin zu flexibleren Client-Server-Systemen [CR97]. Das *World Wide Web* und andere Client-Server-Dienste verursachten in den neunziger Jahren den größten Anteil der Internet-Bandbreitennutzung. In den letzten Jahren scheint diese Dominanz jedoch durchbrochen zu sein, denn die Bandbreitennutzung wird klar von einer neuen Anwendung bestimmt: dem Peer-to-Peer-Informationaustausch [Ste05]. Der folgende Abschnitt geht auf die verschiedenen Eigenschaften des Peer-to-Peer-Modells ein.

2.2.2 Das Peer-to-Peer-Modell

In diesem Ansatz ist im Gegensatz zum Client-Server-Modell jeder teilnehmende Knoten (Peer) des Peer-to-Peer-Netzwerks sowohl Anbieter als auch Nutzer von Diensten. Damit ein Knoten auf die Dienste der anderen Knoten

[3] `http://www.google.com`
[4] `http://www.google.de/intl/de/corporate/`

im Netzwerk zugreifen kann, muss er zunächst um ihre Existenz wissen. Da er in der Regel nicht Kontaktinformationen zu allen Knoten kennt bzw. speichern kann, muss er sich auf eine Untermenge von Nachbarn beschränken. Die Knoten bilden einen Zusammenschluss oberhalb der Netzwerk-Schicht, d.h. sie kommunizieren über ein bestehendes physikalisches Netzwerk (normalerweise IP-basiert). Deshalb bezeichnet man diesen logischen Netzwerk-Graphen auch als *Overlay-Netzwerk*.

In der Evolution der Peer-to-Peer-Systeme sind verschiedene Arten von Overlay-Topologien entstanden, die eine einheitliche Definition erschweren. Die Arbeit von Androutsellis und Theotokis [ATS04] gibt einen umfassenden Überblick über aktuelle Peer-to-Peer-Systeme. Sie legen als Kategorisierungsgrundlage den Grad an Zentralisierung und die Struktur des Overlay-Netzwerks zugrunde.

Zentralisierung

Im engeren Sinne des Peer-to-Peer-Begriffs (Peer, engl. für gleichrangig bzw. gleichgestellt) sind alle teilnehmenden Knoten äquivalent was ihre Funktionalität und zu erfüllende Aufgaben betrifft – sie arbeiten völlig *dezentralisiert* (siehe Abbildung 2.6(a)). Server können höchstens verwendet werden um nebenrangige Aufgaben zu erfüllen (z.B. Bootstrapping[5], oder die Verwaltung von Reputationsdaten), die Abwesenheit dieser Server darf allerdings nicht die Funktionsfähigkeit des Netzwerks beeinträchtigen. Das Gnutella-Netzwerk[6] verfolgt diesen Ansatz.

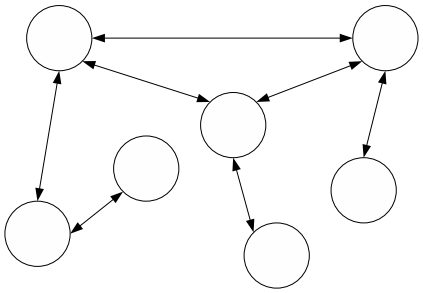
Da nicht davon auszugehen ist, dass jeder Knoten über die gleiche Leistungsfähigkeit verfügt, setzen einige Systeme so genannte Super-Knoten ein, die mehr Aufgaben im Vergleich zu normalen Knoten übernehmen (siehe Abbildung 2.6(b)). Diese Peer-to-Peer-Netze werden daher als *teilweise zentralisierte* Umgebungen bezeichnet. Das KaZaA-Netzwerk[7] ist ein prominenter Vertreter dieser Vorgehensweise.

Im *hybrid dezentralisierten* Ansatz dienen zentrale Server zur Verwaltung von Knoten-Informationen. Wenn ein Knoten nun zum Beispiel einen bestimmten Inhalt sucht, so wendet er sich zunächst an den Server, um abzufragen, welcher andere Knoten diesen bereitstellt und nimmt anschließend

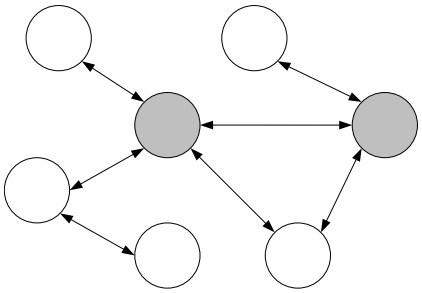
[5]Bootstrapping bedeutet sinngemäß übersetzt, sich an den eigenen Stiefeln aus dem Sumpf herauszuziehen. Dies ist die englische Variante der Münchhausen-Geschichte, in der sich Münchhausen an den eigenen Haaren aus dem Sumpf gezogen hat. In der Informatik bedeutet das Bootstrapping, dass man ein einfaches System benutzt, um ein komplizierteres in Gang zu setzen. (Quelle: `http://www.wikipedia.de`)

[6]`http://www.gnutella.com`

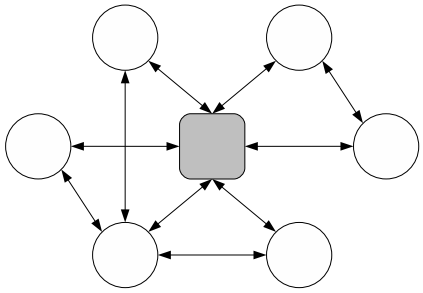
[7]`http://www.kazaa.com`



(a) Dezentralisiertes System



(b) Teilweise zentralisiertes System



(c) Hybrid dezentralisiertes System

Abbildung 2.6: Die verschiedenen Zentralisierungsgrade. (a) Im dezentralisierten Ansatz sind alle Knoten gleichberechtigt. (b) In teilweise zentralisierten Umgebungen werden leistungsfähigere Knoten (grau hervorgehoben) mit zusätzlichen Aufgaben betraut. (c) In hybrid dezentralisierten Systemen werden Server (grau hervorgehoben) zur Bereitstellung von globalen Diensten (z.B. Datei-Indizierung) verwendet. Die Knoten stellen Anfragen an den Server und kommunizieren anschließend direkt miteinander.

direkt Kontakt mit diesem auf (siehe Abbildung 2.6(c)). Der Nachteil ist jedoch, wie beim Client-Server-Ansatz auch, dass ein Ausfall dieses Servers die Funktionsfähigkeit des Gesamtnetzwerks erheblich beeinträchtigt oder unmöglich macht. Die erste Version des Napster-Netzwerks[8] setzte Server ein, um die Musik-Datenbanken der angemeldeten Clients zu indizieren.

Nachdem nun geklärt ist, welche Abhängigkeiten bei der Kommunikation der Knoten untereinander bestehen, muss noch betrachtet werden, wie die Struktur des gebildeten Overlay-Netzwerk-Graphen bestimmt ist.

[8] `http://www.napster.com`

Struktur

Die einzelnen Knoten bilden durch ihr Wissen über andere Knoten einen Overlay-Netzwerk-Graphen. Nach Androutsellis und Theotokis gibt die Struktur an, welche Regeln der Bildung dieses Graphen zugrunde liegen.

Im einfachsten Fall ist das Peer-to-Peer-Netz *unstrukturiert*. Das bedeutet, dass der Zusammenschluss der Knoten keinerlei System folgt. Um dem Netzwerk beizutreten, muss ein Knoten lediglich einen weiteren Knoten kennen und ihn über seine Existenz informieren.

Sucht ein Knoten in diesem Netz nach einem bestimmten Inhalt so bleibt ihm nur die Möglichkeit, so viele andere Knoten wie möglich zu befragen – in der Hoffnung, dass ein anderer diesen Inhalt bereitstellt. Da jeder Knoten lokal entscheidet, ob er die Suchanfrage erfüllen kann, sind in unstrukturierten Peer-to-Peer-Netzwerken auch komplexere Suchkriterien wie Schlüsselwort-Suchen (beispielsweise alle Bilder mit einer bestimmten Mindestauflösung) oder Volltextsuche auf einem Dokumentenbestand (alle Texte, die ein bestimmtes Wort enthalten) möglich.

Der naive Ansatz zur Verbreitung von Suchanfragen ist der Broadcast. Dies ist offensichtlich nicht ressourcenschonend, da das Netz mit Suchanfragen geflutet wird. Außerdem wird die Last auf dem Gesamtsystem umso größer, je mehr Knoten dem Netzwerk beitreten – die Skalierung bei großer Knotenanzahl kann nicht gewährleistet werden. Um die Netzlast im Gnutella-Netzwerk zu reduzieren wird die Nachricht nach einer bestimmten Anzahl von Hops nicht mehr weitergeleitet [ATS04]. Dies hat zur Folge, dass eine Suche nicht immer erfolgreich endet, auch wenn der Inhalt im Netz verfügbar ist. In [LCC+02] kommen die Autoren zu dem Schluss, dass Suchanfragen auf der Basis von *Random Walks* eine effizientere und ressourcenschonendere Methode darstellen – aber auch sie können nicht verhindern, dass vorhandene Inhalte nicht gefunden werden.

Androutsellis und Theotokis nennen folgende Eigenschaften, die erfüllt sein sollten, wenn man ein Informationssystem auf der Basis von unstrukturierten Peer-to-Peer-Umgebungen einsetzt:

Eine Anwendung lässt sich nach Androutsellis und Theotokis gut in unstrukturierten Peer-to-Peer-Umgebungen einsetzen, wenn die Anwendung folgenden Anforderungen genügt:

1. Schlüsselwort-Suche ist eine häufig verwendete Operation.
2. Inhalte werden von einer ausreichenden Anzahl von Knoten repliziert.
3. Die Knotenpopulation ist stark fluktuierend.
4. Benutzer akzeptieren einen *Best-Effort*-Dienst.

5. Die Größe des Netzwerks ist nicht zu groß, um Skalierungs-Probleme herbeizuführen (diese resultieren vor Allem aus der Verbreitung von Suchanfragen).

Um die Skalierungsprobleme in unstrukturierten Netzen zu lösen sind in den letzten Jahren so genannte *strukturierte* Peer-to-Peer-Netzwerke in das Interesse der Forschung gelangt. In diesen Netzen unterliegt die Bildung der Overlay-Topologie strikten Regeln was die Nachbarschaftsbeziehungen der Knoten und die Plazierung von Inhalten im Netz angeht.

Eine mögliche Abstraktion, die in allen aktuellen strukturierten Systemen realisiert werden kann, ist das *schlüsselbasierte Routing* (Key-Based Routing, KBR) [DZD+03]. Anhand eines Schlüssels kann das Routing-Verfahren Knoten im Netz eindeutig ermitteln. Hierdurch kann im Vergleich zu unstrukturierten Netzen sichergestellt werden, dass ein verfügbarer Inhalt auch tatsächlich gefunden wird. Mit Hilfe des KBR können weitere Dienste wie *Dynamische Hash-Tabellen* (DHT), *Anycast* und *Multicast* (CAST) sowie *Verteilte Objekt-Auffindung und Routing* (Distributed Object Location and Routing, DOLR) realisiert werden. Die DHTs sind eine verteilte Variante der Hash-Tabellen, die zu einem festen Schlüssel einen Wert effizient zurückliefern. Durch CAST kann eine einzelne Nachricht an irgendein bzw. alle Mitglieder einer Gruppe gesendet werden und mit DOLR kann man den Ort eines bestimmten Objekts im Netzwerk ermitteln.

Ein großer Vorteil in strukturierten Netzen liegt darin, dass diese mächtigen Dienst-Primitive effizient realisiert werden können, sowohl was die Nachrichtenanzahl angeht, als auch der Speicherbedarf auf jedem Knoten, um die Nachrichten durch das Netz zu leiten. Beides kann logarithmisch in der Gesamtanzahl der Knoten bewältigt werden [ATS04]. Ein großer Anstieg in der Knotenzahl hat kaum Auswirkungen auf die Leistungsfähigkeit des Gesamtsystems.

Eine schwierige Aufgabe für strukturierte Umgebungen besteht jedoch in der Erhaltung der Struktur, die für das effiziente Zustellen von Nachrichten benötigt wird. Gerade wenn die Knotenpopulation häufigen Schwankungen unterliegt ist der Wartungsaufwand groß [LCC+02].

Um in einem strukturierten Netz nach einem bestimmten Inhalt zu suchen, muss man zunächst seinen Schlüssel kennen. Erst dann ist eine effiziente Suche möglich. Eine schwierigere Abfrage wie zum Beispiel eine Schlüsselwort-Suche, in der man bestimmte Eigenschaften eines Inhalts abfragen will, werden dadurch schwierig. Einige Ansätze zur Lösung dieser Problematik sind in [HHH+02, RV03, BHPW04] zu finden.

In den letzten Jahren ist eine Vielzahl von verschiedenen Systemen für strukturierte Peer-to-Peer-Systeme veröffentlicht worden. Beispiele hier-

für sind Chord [SMLN+03], CAN [RFH+01], Pastry [RD01], Tapestry [ZKJ01], Kademlia [MM02] und Koorde [KK03]. Da strukturierte Peer-to-Peer-Umgebungen für diese Arbeit eine zentrale Rolle spielen, werden im folgenden Abschnitt einige ausgewählte Verfahren näher betrachtet.

2.3 Strukturierte Peer-to-Peer-Systeme

Strukturierte Peer-to-Peer-Systeme bieten gute Leistung und Skalierbarkeit, wenn es um die Suche nach einem bestimmten Schlüssel im Netzwerk geht. Anwendungen, die auf diese Dienste zurückgreifen, können sich die guten Eigenschaften der strukturierten Overlay-Netzwerke zu Nutze machen. In diesem Abschnitt werden einige ausgewählte Systeme, die Gegenstand der aktuellen Forschung sind, näher vorgestellt.

Gegenstand der Untersuchung sind folgende Eigenschaften:

Nachrichtenaufwand: Wie viele Nachrichten müssen ausgetauscht werden, um herauszufinden, welchem Knoten ein gegebener Schlüssel zugewiesen ist? Hierbei sei an dieser Stelle erwähnt, dass in einem Netzwerk mit N Knoten und einem konstanten maximalen Grad k die optimale Hop-Distanz $log_k N - 1$ beträgt [KK03].

Speicheraufwand: Welche Speicherkomplexität ergibt sich auf jedem Knoten, um das effiziente Routen zu gewährleisten?

Verwaltungsaufwand: Wie schnell kann das System seine Struktur reparieren, wenn ein Knoten das Netz betritt bzw. verlässt?

Transportaufwand: Kann das System die darunterliegende Netzwerk-Topologie ausnutzen, um die Anzahl der Hops gering zu halten?

Alle vorgestellten Verfahren weisen sowohl den Knoten als auch den Schlüsseln Bezeichner zu, die durch eine Hash-Funktion berechnet werden. Das Chord-System [SMLN+03] ordnet die Bezeichner der Knoten und Schlüssel in einer Ring-Struktur an. CAN [RFH+01] benutzt einen d-dimensionalen Torus als Bezeichner-Raum und Kademlia [MM02] wiederum verwendet eine XOR-Metrik, um die Bezeichner in einer Baum-Struktur anzuordnen.

2.3.1 Chord

Das von Stoica et al. entwickelte Chord-System [SMLN+03] unterstützt nur eine einzige Operation:

Für einen gegebenen Schlüssel kann ein zugehöriger Knoten im Overlay-Netzwerk berechnet werden.

Um dies zu erreichen weist Chord zunächst allen Knoten und Schlüsseln einen m-Bit langen Hashwert als Bezeichner zu (im Folgenden wird angenommen, dass dieser Bezeichner ganzzahlig interpretiert wird). Dabei muss m derart gewählt sein, dass mit hoher Wahrscheinlichkeit keine zwei gleichen Bezeichner berechnet werden. Diese Bezeichner werden in einem Ring *modulo* 2^m angeordnet (siehe Abbildung 2.7(a)). Ein Schlüssel wird nun dem ersten Knoten zugeordnet, dessen Bezeichner gleich oder größer dem Bezeichner des Schlüssels ist. Durch den Einsatz des so genannten *konsistenten Hashings* [KLL+97] wird erreicht, dass jeder Knoten ungefähr die gleiche Anzahl von Schlüsseln zugewiesen bekommt. Das konsistente Hashing verwendet dazu SHA-1 [sha95] als Basisfunktion.

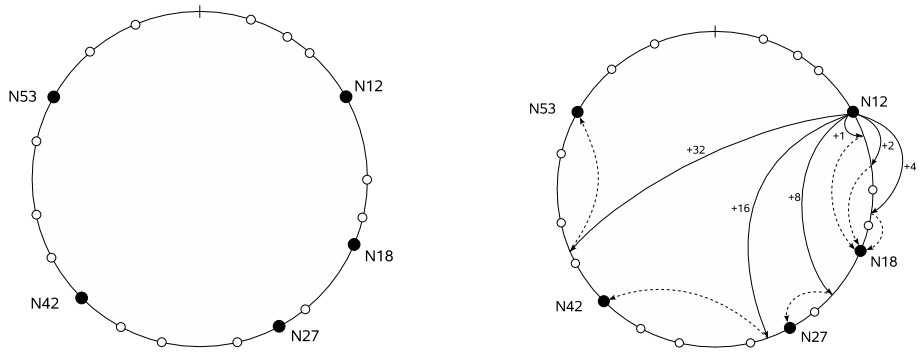
Routing im Chord-Ring

Jeder Knoten speichert Routing-Informationen in einer so genannten Finger-Tabelle. Der i-te Eintrag dieser Tabelle speichert die Adresse des ersten Knotensbezeichners, der mindestens 2^{i-1} vom jeweiligen Tabellen-Eigentümer im Ring entfernt ist. Abbildung 2.7(b) zeigt, wie die Finger-Tabelle (siehe Tabelle 2.1) eines Beispiel-Knotens zusammengesetzt wird.

Das Auffinden eines Schlüssels im Ring erfolgt über einen Algorithmus, der sich die Eigenschaften der Finger-Tabellen zu Nutze macht. Zuerst überprüft der suchende Knoten, ob der Schlüsselbezeichner zwischen dem eigenen und dem Bezeichner des direkten Nachbarn liegt. Wenn ja, dann ist der verantwortliche Knoten bereits gefunden. Ansonsten gibt der Knoten die Suchanfrage an den Knoten in der Finger-Tabelle weiter, der am nähesten am Schlüsselbezeichner liegt. Dieser Knoten führt dann seinerseits eine Suche aus. Bis schließlich ein Knoten gefunden wurde, dessen direkter Nachbar verantwortlich ist. Dieses Verfahren kann sowohl iterativ als auch rekursiv umgesetzt werden.

Nachrichten- und Speicheraufwand

Da eine Suchanfrage in jedem Schritt mindestens die Hälfte der verbleibenden Distanz zum Zielknoten zurücklegt wird die Anfrage mit hoher Wahrscheinlichkeit nach $O(\log N)$-Schritten abgearbeitet sein, wobei N die Anzahl der Knoten im Netzwerk ist [SMLN+03]. Die Finger-Tabellen werden mit Einträgen exponentiell steigender Distanzen erzeugt und daher beträgt



(a) Ein beispielhafter Chord-Ring

(b) Finger-Einträge eines Chord-Knotens

Abbildung 2.7: Ein beispielhafter Chord-Ring mit einigen Knoten (durch N gekennzeichnet) und Schlüsseln für $m = 6$. (a) Die Anordnung der Knoten und Schlüssel im Ring (b) Die Routing-Information, die der Knoten N12 speichert (nach [SMLN+03])

die Speicherkomplexität ebenfalls $O(\log N)$.

Verwaltungsaufwand

Bisher wurde davon ausgegangen, dass der Chord-Ring aus einer konstanten Anzahl von Knoten besteht. Nun soll betrachtet werden, wie das System beim Betreten neuer Knoten und dem Verlassen alter reagiert – dies hat erheblichen Einfluss auf die Korrektheit der Finger-Tabellen und damit die Routing-Leistung. An dieser Stelle sei erwähnt, dass das Routing im Ring selbst dann noch funktioniert, wenn jeder Knoten nur seinen Nachbarn kennt. Dann ist jedoch die Laufzeit linear in der Anzahl der Knoten im System. Um die Finger-Tabellen zu reparieren führen alle Knoten in regelmäßigen Zeitabständen das Reparatur-Protokoll *stabilize* durch.

Tabelle 2.1: Die Finger-Tabelle des Knotens N12 aus Abbildung 2.7(b)

i	Entfernung	Knoten
0	N12+1	N18
1	N12+2	N18
2	N12+4	N18
3	N12+8	N27
4	N12+16	N42
5	N12+32	N53

Möchte ein neuer Knoten k ein bestehendes Chord-System betreten, so muss er mindestens einen Knoten l im System kennen. Nachdem k eine *join*-Anfrage an l gestellt hat, sucht l den direkten Nachfolger von k im Ring. Das *stabilize*-Protokoll repariert dann im nächsten Durchgang den entstandenen Bruch in der Ring-Topologie.

Das Chord-System muss keine Unterscheidung zwischen verlassenden und ausfallenden Knoten machen. Die Leistung des Systems kann jedoch verbessert werden, wenn ein verlassender Knoten seinen Vorgänger und Nachfolger benachrichtigt und sogleich die Informationen zur Reparatur des Rings übermittelt – d.h. dem Vorgänger melden, dass er einen neuen Nachfolger hat und dem eigenen Nachfolger die neu zu verwaltenden Schlüssel übermitteln.

Während Knoten das System betreten und verlassen bleibt der Anfrage-Aufwand mit hoher Wahrscheinlichkeit bei $O(\log N)$, obwohl das *stabilize*-Protokoll den Ring ständig reparieren muss. Für einen Beweis sei auf [SMLN+03] verwiesen.

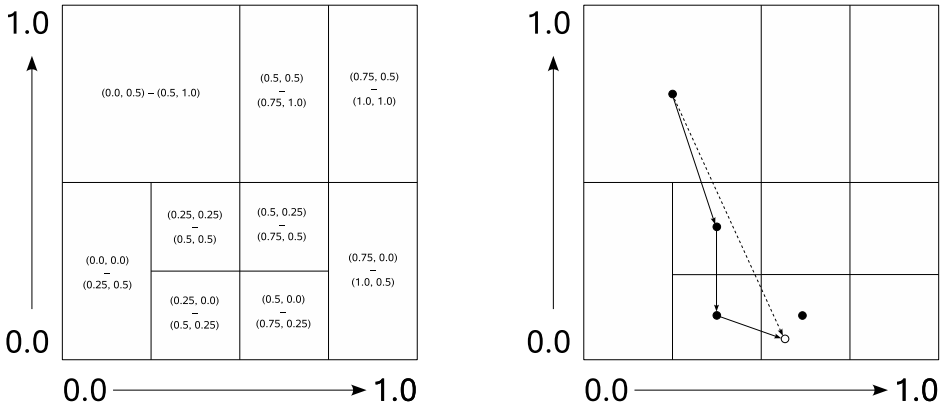
Transportaufwand

Das Chord-Protokoll sieht keine Möglichkeit vor, die zugrundeliegende Netzwerk-Architektur auszunutzen, um Transport-Hops zu minimieren.

2.3.2 CAN

Ein *Content Addressable Network* (*CAN*) ist nach Ratnasamy et al. [RFH+01] ein verteiltes System, das für ein globales Netzwerk wie das Internet effizient eine Hashtabelle zur Verfügung stellt. Die Autoren gehen davon aus, dass eine derartige Hashtabelle ein grundlegender Baustein für skalierbare verteilte Peer-to-Peer-Systeme darstellt.

Das entwickelte CAN-System unterstützt das Einfügen, Auslesen und Entfernen von (Schlüssel,Wert)-Paaren. Dazu wird ein virtueller d-dimensionaler Koordinatenraum (ein d-Torus) verwendet, der in verschiedene Bereiche (Zonen) aufgeteilt wird. Dabei ist d ein wählbarer Konfigurationsparameter, der allerdings zur Laufzeit des Systems nicht mehr geändert werden kann (denn das hätte eine Änderung der Hashfunktion zur Folge). Alle Zonen werden auf die Knoten im System aufgeteilt – jede Zone (und damit jeder Knoten) ist dabei für die Speicherung eines bestimmten Teils der verteilten Hashtabelle verantwortlich. Abbildung 2.8(a) zeigt die beispielhafte Einteilung eines zweidimensionalen Koordinatenraums.



(a) Einteilung eines zweidimensionalen CAN-Koordinatenraums

(b) Ein Routing-Pfad

Abbildung 2.8: Ein zweidimensionaler Koordinatenraum, aus Darstellungsgründen wird der Torus als Ebene gezeigt. (a) Eine beispielhafte Einteilung in Zonen (b) Ein Routing-Pfad zur Zielzone eines Schlüssels.

Routing im CAN-Raum

Jeder Knoten ist für eine Zone des CAN-Koordinatenraums zuständig. In seiner Routing-Tabelle speichern die Knoten Kontaktinformationen der Nachbar-Zonen ab, damit Nachrichten weitergeleitet werden können.

Um ein (Schlüssel,Wert)-Paar in System abzulegen wird der Schlüssel zunächst durch eine uniforme Hashfunktion auf einen Punkt im Koordinatenraum abgebildet. Dieser Punkt liegt innerhalb einer bestimmten Zone. Wenn der Knoten, der den Schlüssel zugreifen will, nicht selbst für die durch diesen Punkt getroffene Zone zuständig ist, so muss er die Anfrage an eine andere Zone weiterleiten. Dies geschieht auf der Geraden im kartesischen Koordinatenraum. Die Anfrage wird also durch alle Zonen, welche die Gerade schneidet, weitergeleitet, bis sie in der Zielzone angelangt ist.

Realitäten

Um eine höhere Verfügbarkeit der verwalteten Schlüssel zu erreichen, kann man in einem System mehrere Koordinatensysteme (so genannte Realitäten) gleichzeitig einsetzen. Jeder Knoten verwaltet dann eine Zone aus jeder vorhandenen Realität – die Koordinatensysteme sollten jedoch so gewählt werden, dass keine deckungsgleichen Zonen auf einem Knoten abgelegt werden. Wird nun ein Schlüssel im System abgelegt, so wird er in jeder Realität gespeichert und damit auch auf unterschiedlichen Knoten. Ist ein Schlüssel

in einer Realität nicht mehr zugreifbar, weil zum Beispiel der verantwortliche Knoten ausgefallen ist, so ist er erst dann verloren, wenn alle Knoten, die ihn gespeichert haben, ausfallen.

Nachrichten- und Speicheraufwand

In einem System mit N Knoten und Dimension d beträgt die durchschnittliche Weglänge einer Anfrage $O(d \cdot N^{1/d})$. Dazu muss jeder Knoten nur $2d$ viele Nachbarn kennen. Um hierbei logarithmische Skalierung zu erreichen muss man vorher wissen, wie groß das entstehende Netz sein wird und d entsprechend wählen. Die Entkopplung der Netzgröße von der Größe der Routing-Tabelle ist hier bewusst vorgenommen, um bei sehr stark wachsender Netzgröße die Kontrolle über den Speicherbedarf zu behalten.

Verwaltungsaufwand

Damit ein neuer Knoten das CAN-Netzwerk betreten kann, muss er einen Knoten kennen, der bereits an das Netzwerk angebunden ist. Er erzeugt einen zufälligen Punkt P und sendet eine *join*-Anfrage mit dem Argument P an den bekannten Knoten. Nun wird die Anfrage an den Punkt weitergeleitet der Besitzer der Zone von P ist. Nun wird der Knoten seine Zone halbieren und eine Hälfte an den neuen Knoten übergeben (mitsamt allen (Schlüssel,Wert)-Paaren). Die Routing-Tabelle des neuen Knoten kann daraufhin erzeugt werden aus der übermittelten Tabelle des alten Knotens. Die Routing-Tabellen der anderen Nachbarn werden über periodische *update*-Nachrichten automatisch aktualisiert werden. Insgesamt sind lediglich $O(d)$ andere Knoten von dem Beitreten eines neuen Knotens betroffen.

Wenn ein Knoten das Netzwerk verlassen will, dann kann er durch Benachrichtigung seiner Nachbarn und Übermittlung der verwalteten Hashtabellen-Einträge mit wenig Aufwand austreten.

Jeder Knoten sendet in regelmäßigen Abständen eine *update*-Nachricht an seine Nachbarn. Bleibt für eine bestimmte Zeit diese Nachricht aus, so gehen die Knoten von einem Ausfall aus. In diesem Fall sendet ein Knoten eine *takeover*-Nachricht an alle betroffenen Nachrichten. Anhand eines bestimmten Kriteriums (z.B. der Knoten mit dem geringsten verwalteten Datenvolumen) einigen sich alle Nachbarn darauf, wer die Zone übernehmen wird. Interessant ist an dieser Stelle, dass man auch Kriterien wie Lastverteilung einbeziehen kann. Die verlorenen Hashtabellen-Einträge sind dann erst wieder Verfügbar, wenn der eigentliche Eigentümer des Eintrags die Daten erneut ins Netz gibt.

Transportaufwand

Die ursprüngliche Version von CAN nutzt eine kartesische Metrik um den Routing-Pfad zu bestimmen. Dies bestimmt kürzeste Wege auf Anwendungsebene, hat aber auf Netzwerk-Ebene größeren Transport-Aufwand zur Folge. Die Entwickler von CAN schlagen daher vor, an dieser Stelle eine Metrik einzusetzen, die stattdessen auf Netzwerk-Distanzen (*IP-Hops*) bzw. der Paketumlaufzeit (RTT) basiert. Auf diesem Wege kann auch geographischer Nähe im physikalischen Netzwerk Rechnung getragen werden.

2.3.3 Kademlia

Das von Maymounkov und Mazières entwickelte Kademlia-System [MM02] vergibt für alle Knoten und alle Schlüssel einen b-Bit Bezeichner (Bezeichner werden genau wie bei Chord durch *konsistentes Hashing* erzeugt, die Autoren schlagen einen Wert von 160 für b vor). Schlüssel werden auf denjenigen Knoten gespeichert, deren Knotenbezeichner einen geringen Abstand zum Schlüsselbezeichner besitzen. Der Abstand $d(x, y)$ zwischen einem Knotenbezeichner x und einem Schlüsselbezeichner y wird über die sogenannte XOR-Metrik bestimmt. Tabelle 2.2 zeigt die Eigenschaften dieser Metrik.

Tabelle 2.2: Eigenschaften der XOR-Metrik

Eigenschaft	Beschreibung
$d(x, y) = x \otimes y$	Abstandsmaß d
$d(x, x) = 0$	Identische Bezeichner haben Abstand 0
$x \neq y \Rightarrow d(x, y) > 0$	Nichtidentische Bezeichner haben positiven Abstand
$\forall x, y \Rightarrow d(x, y) = d(y, x)$	Symmetrie
$d(x, y) + d(y, z) \geq d(x, z)$	Dreiecksungleichung
$\forall x, \Delta > 0\ \exists_1 y : d(x, y) = \Delta$	Unidirektionalität

Routing im Kademlia-Baum

Jeder Knoten verwaltet eine Liste von b so genannten *k-Buckets*. Der i-te Eintrag dieser Liste (ein k-Bucket) enthält bis zu k Knoten, die sich in einen Abstand von 2^i bis 2^{i+1} zum Knoten selbst befinden. k ist dabei eine systemweite Replikationskonstante, die so gewählt sein sollte, dass ein gleichzeiter

Ausfall aller k Knoten innerhalb eines bestimmten Zeitraums sehr unwahrscheinlich ist (die Autoren schlagen z.B. $k = 20$ vor). Abbildung 2.9 zeigt einen XOR-Baum für $b = 3$.

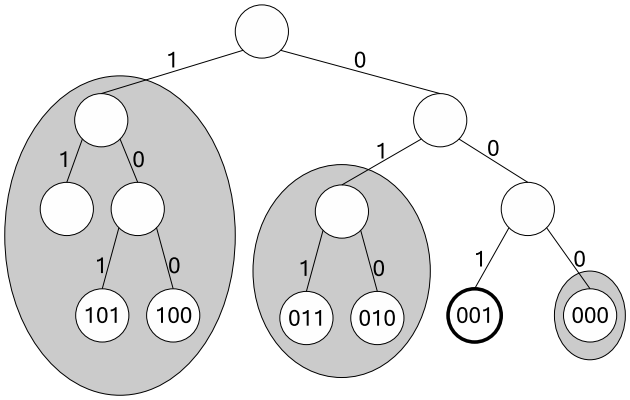


Abbildung 2.9: Ein XOR-Baum mit einer Bezeicherlänge von 3 Bits. Die Blätter in den grau hinterlegten Bereichen stellen die verschiedenen k-Buckets (mit $k = 2$) dar, die auf Knoten 001 gespeichert werden. (Angelehnt an [Zie05])

Jede Nachricht, die im Kademlia-System weitergeleitet wird, enthält auch den Bezeichner des Senders. Auf diese Weise können weiterleitende Knoten jeder Nachricht Routing-Informationen entnehmen. Der Sender einer Nachricht, der in den i-ten k-Bucket passt, wird am Ende der Liste eingefügt, sofern er nicht schon enthalten ist und noch keine k Knoten dort gespeichert wurden. Ist der Bucket voll, wird der *älteste* Knoten kontaktiert. Ist dieser Knoten noch erreichbar wird das Routing-Information des Pakets ignoriert. Dahinter steckt die Annahme, dass Knoten, die schon lange im System verweilen, auch weiterhin erreichbar sein werden (die Arbeit von Saroiu et al. untersucht hierzu das Gnutella-Netzwerk [SGG02]). Deshalb sind diese Knoten verlässlicher für das Routing.

Um nun eine Anfrage zum Zielknoten weiterzuleiten benutzt ein Knoten den *RPC*[9]-Mechanismus `FIND_NODE`. Diese Prozedur liefert die k Knoten, die nach der Routing-Tabelle des angefragten Knoten am nähesten am Zielbezeichner liegen. Aus den zurückgelieferten Knoten wählt der Anfragende α viele aus, und sendet asynchrone Anfragen rekursiv an diese weiter. Über den Parameter α kann man Bandbreite gegen bessere Suchleistung abwägen.

Da die XOR-Metrik unidirektional ist, konvergieren alle Anfragen nach dem gleichen Schlüssel entlang des selben Pfads. Hierdurch ist es möglich

[9]Entfernter Prozeduraufruf, engl. Remote Procedure Call

bereits auf dem Weg Ergebnisse zu cachen um die Systemleistung zu verbessern.

Nachrichten- und Speicheraufwand

Jeder Knoten verwaltet b Listen mit jeweils k Einträgen. Durch die Auswahl eines festen Werts für b ist die Größe der Routing-Tabelle zwar unabhängig von der Netzgröße, der Parameter b kann allerdings maximal mit $\log n$ gewählt werden – daher beträgt die Speicherkomplexität auf jedem Knoten $O(k \cdot \log n)$. In jedem Routing-Schritt nähert man sich dem Zielnoten um mindestens ein Bit, daher kann eine Anfrage in $O(\log n)$ Schritten abgearbeitet werden.

Verwaltungsaufwand

Ein Knoten im Kademlia-Netzwerk kann aus jeder Nachricht, die er weiterleitet, Routing-Informationen entnehmen. Dies hat offensichtlich den Vorteil, dass kein zusätzliches Protokoll zur Reparatur der Routing-Tabellen erforderlich ist. Die Routing-Tabellen aktualisieren sich vielmehr bei jeder weitergeleiteten Nachricht.

Transportaufwand

Durch den α-Parameter kann beeinflusst werden, wie viele Knoten gleichzeitig kontaktiert werden. Hierdurch hat jeder Knoten die Möglichkeit, die Nachrichten an diejenigen Knoten zu senden, bei denen die geringste Latenz zu erwarten ist. An der Stelle der Auswahl kann also ein Kriterium angelegt werden, um möglichst kurze *reale* Umlaufzeiten zu begünstigen.

2.3.4 Leistungsvergleich

TODO Dieser Abschnitt ist alles andere als fertig. Hier muss noch der Leistungsvergleich ausgearbeitet werden (Tabelle vervollständigen und Argumente vorbringen)

Die im vorherigen Abschnitt vorgestellten System wurden ausgewählt, um ein Gefühl zu vermitteln, wie strukturierte Peer-to-Peer-Systeme funktionieren. Nun werden die Leistungsmerkmale der vorgestellten und weiterer Systeme gegenübergestellt.

Verglichen wird CAN, Chord, Pastry, Tapestry, Kademlia und Koorde.

[LSG+04] meint, dass die verschiedenen Verfahren durch individuell angepasste Parameter ungefähr die gleiche Leistung erbringen. Nur die Auswahl der Parameter ist der ausschlaggebende Punkt.

Tabelle 2.3: Vergleich der verschiedenen strukturierten Systeme.

System	CAN[a]	Chord	Pastry	Kademlia[b]
Veröffentlichungsjahr	2001	2001	2001	2002
Veröffentlichung	[RFH+01]	[SMLN+03]	[RD01]	[MM02]
Nachrichtenaufwand	$O(\frac{d}{4}N^{\frac{1}{d}})$	$O(\log N)$	$O(\log N)$	$O(\log N)$
Speicheraufwand	2d	$O(\log N)$	$O(\log N)$	$O(k \cdot \log N)$
Verwaltungsaufwand				
Transportaufwand				

N ist die Gesamtknotenanzahl des betrachteten Netzwerks.
[a]Die Dimension des gewählten Bezeichnerraums ist mit d gekennzeichnet.
[b]k ist eine systemweite Replikationskonstante.

Kelips [GBL+03]
OneHop [GLR04]

Übersicht

Im ersten Teil dieses Kapitels wurden die verschiedenen Aspekte betrachtet, die bei der computergestützen Gruppenarbeit eine Rolle spielen. Es wurde untersucht, welche Interaktionsformen es bei der Kooperation von Gruppenmitgliedern gibt und welcher Art ihre Aufgaben sein können. Anschließend wurde beschrieben, wie der Benutzer eines Awareness-Systems durch das Bewusstsein über die Aktivitäten der anderen Gruppenmitglieder einen Vorteil für die eigene Arbeit erlangen kann und wie man dieses Bewusstsein durch den Computer vermitteln kann.

Im zweiten Teil ging es um Möglichkeiten, wie Computer miteinander kommunizieren können, denn die Kommunikation der Computer bietet schließlich die Grundlage eines Awareness-Systems. Im Client-Server-Ansatz werden die Berechnungen auf einer zentralen Instanz durchgeführt, was die Gestaltung und Umsetzung von Algorithmen überschaubar macht. Der Nachteil lag allerdings in den Kosten, die durch die Skalierung bei stark wachsenden Benutzerzahlen entstehen und den zentralen Ausfallpunkt, der das gesamte System in Mitleidenschaft ziehen kann. An dieser Stelle verspricht das Peer-to-Peer-Modell hohe Skalierbarkeit, gute Fehlertoleranz und Selbstorganisation. Die Berechnung wird auf Benutzerseite des Netzes verlagert, auf der viele ungenutzte Ressourcen zur Verfügung stehen. Der Nachteil besteht darin, dass die Entwicklung von Algorithmen mehr Aufwand nach sich

zieht, denn man muss die Aufgaben effizient auf mehrere Knoten verteilen. Schließlich wurden einige Systeme vorgestellt, die das Peer-to-Peer-Modell umsetzen und gute Leistung bei großer Zuverlässigkeit erlauben.

An dieser Stelle soll nun angesetzt werden, um die Anforderungen an das zu entwickelnde Awareness-System zu sammeln.