Manfred Hauswirth, Schahram Dustdar

Peer-to-Peer: Grundlagen und Architektur

1 Einleitung

Sind Peer-to-Peer-(P2P)-Systeme ein neuer Ansatz zum Design von verteilten Anwendungen? Eigentlich nicht. Zwar sind P2P-Systeme ein großer Erfolg, aber überspitzt formuliert, stellen sie „nur“ eine Rückbesinnung auf das ursprüngliche Internet-Prinzip dar: Jeder Internet-Node kann mit jedem anderen Internet-Node bidirektional kommunizieren, zum Beispiel, sowohl als FTP-Client als auch als FTP-Server fungieren. Dieses Basisprinzip, dass jeder Netzwerk-Knoten sowohl als Client als auch als Server (in P2P-Systemen zu *Servent* verschmolzen) fungieren kann, wurde im ursprünglich nicht aus dem akademischen Bereich stammenden P2P-Ansatz, „wiedererfunden”. Es lassen sich durchaus noch einige weitere P2P-Beispiele finden, beispielsweise könnte Usenet News als typisches hierarchisches P2P-System gesehen werden. Neu an P2P-Systemen sind die extrem hohe Anzahl an Teilnehmern und die Selbstorganisationsfähigkeiten einiger Systeme. Auch in P2P-Systemen sind Regeln vorgegeben, innerhalb derer die teilnehmenden Peers autonome Entscheidungen treffen können. Anders ausgedrückt ist das globale Verhalten eines P2P-Systems das Ergebnis der lokalen Entscheidungen der Teilnehmer und somit einem einfachen sozialen System ähnlicher als typischen verteilten Informationssystemen. Das interessante am P2P-Ansatz ist außerdem, dass er nicht nur auf technische Bereiche beschränkt ist. Viele soziale und ökonomische Systeme sind „P2P“ und selbstorganisierend. Typische Beispiele sind eBay, im biologischen Bereich Ameisenkolonien und in der Wirtschaft die Grundprinzipien von Märkten. P2P-Systeme eignen sich somit hervorragend als „Spielwiese” zur Erforschung mannigfaltiger Phänomene und Strukturen und bieten zum ersten Mal die Möglichkeit das Verhalten extrem großer Systeme in der Praxis zu erforschen. Andererseits erfordert die Erforschung von P2P-Systemen Ansätze und Wissen aus unterschiedlichsten Disziplinen – Statistik und Wahrscheinlichkeitstheorie, verteilte Systeme, Datenbanken, theoretische Biologie, Graphentheorie, Soziologie, um nur einige zu nennen.

Dies spannt einen sehr weiten Bogen, der in der Kürze dieses Beitrags natürlich nur überblicksweise abgehandelt werden kann. Wir beschränken uns daher auf eine datenbankorientierte Sichtweise und versuchen im Folgenden einen kompakten, wenn auch notwendigerweise unvollständigen Überblick über die Grundprinzipien von P2P-Systemen zu geben und die Funktionalitäten und Eigenschaften der wichtigsten Vertreter kurz zu präsentieren.

2 Grundprinzipien

Skalierbarkeit und Fehlertoleranz sind zentrale Probleme in verteilten Systemen, die eine grundlegende Motivation für eine P2P-Architektur darstellen, da sich damit „single-points-of-failure“ und Engpässe in Bezug auf Rechner- und Netzwerklast vermeiden lassen. Fällt ein Teil eines P2P-Systems aus (Rechnerausfall, Netzwerkunterbrechung), so hat dies nicht unbedingt den Ausfall des Gesamtsystems zur Folge. Mittels geeigneter Architekturen lässt sich sogar ein „Überleben“ innerhalb relativ weiter Grenzen gewährleisten. Ebenso kann durch die Verwendung aller im System vorhandenen Netzwerk- und Rechnerressourcen mit Hochlastszenarien recht gut umgegangen werden. Ein weiterer Vorteil ist die Ausnutzung lokalen Wissens, da in vielen P2P-Systemen Benutzer Informationen mit Metadaten annotieren können. Zum Beispiel könnten Bilddateien vom anbietenden Benutzer mit strukturierten Beschreibungen versehen werden und diese danach in Suchabfragen verwendet werden. Auf Grund der schieren Menge an Daten in einem P2P-System wäre dies zentral kaum durchführbar und stellt daher einen nicht zu unterschätzenden Vorteil dar.

Diese Vorteile werden aber mit bei weitem höherer Komplexität bezahlt. Um beispielsweise Ausfallssicherheit zu gewährleisten, müssen im Vergleich zu anderen Systemen weit komplexere Replikationsmechanismen eingesetzt werden (sowohl in Bezug auf Indexinformation, also auch für Daten und Metadaten). Ebenso lassen sich derzeit Transaktionen nicht einmal ansatzweise in P2P-Systemen implementieren und die Umsetzung nicht-trivialer Suchprädikate über Schlüssel (ebenso wie über Metadaten) steckt erst in den Kinderschuhen. Des Weiteren benötigt die verteilte

Metadatengenerierung natürlich viel höheren Aufwand in Bezug auf Integration. Dennoch sind die möglichen Vorteile so gewichtig, dass P2P-Systeme als architekturelle Alternative für bestimme Anwendungsgebiete höchst interessant sind, da auch traditionelle Ansätze keine letztgültige Alternative darstellen. Im Einzelfall wird daher abzuwiegen sein, ob es letztlich Sinn macht, einem P2P-Ansatz den Vorzug zu geben.

Peer-to-Peer-Systeme können durch folgende Eigenschaften charakterisiert werden:

- Rollensymmetrie: Jeder Peer ist sowohl Client als auch Server („Servent“).
- Dezentralisierung:
    - Es gibt keine zentrale Koordination, d.h., es gibt keinen zentralen Knoten, der die Interaktionen der Peers untereinander steuert oder koordiniert.
    - Es gibt keine zentrale Datenbasis. Dies bedeutet, dass jeder Peer einen Teil der im Gesamtsystem vorhandenen Daten speichert und zur Verfügung stellt, aber kein Peer den Gesamtdatenbestand verwaltet bzw. kennt.
    - Kein Peer hat eine globale Sicht des Systems. Jeder Peer kennt nur seine „Nachbarschaft“, d.h., jene Peers, mit denen er interagiert.
- Selbstorganisation: Das globale Verhalten (das Gesamtverhalten) des Systems entsteht aus den lokalen Interaktionen zwischen den Peers.
- Autonomie: Peers sind autonom in ihren Entscheidungen und ihrem Verhalten.
- Zuverlässigkeit: Peers und Netzwerkverbindungen sind nicht unzuverlässig, was durch entsprechende Mechanismen (Replikation, Reputationsmanagement, etc.) zu kompensieren ist.
- Verfügbarkeit: Die gesamten im System gespeicherten Daten müssen verfügbar sein. Diese Forderung bedeutet, dass trotz verteilter Speicherung, möglichen Verbindungsausfällen, unbekannter Glaubwürdigkeit der Peers, etc. die gesamten Daten jederzeit und von jedem Peer aus zugreifbar sein müssen.

Abhängig davon wie viele dieser Eigenschaften von einem System erfüllt werden, desto „mehr“ kann es als P2P-System bezeichnet werden. Nur wenige P2P-Systeme erfüllen allerdings alle diese Anforderungen, da sie nur in einigen Anwendungsszenarien auch tatsächlich gleichzeitig notwendig sind. Beispielsweise umgeht Napster einen Teil der inhärenten Komplexität durch einen zentralen Index der im System vorhandenen Daten, mit Referenzen auf die Peers, die die Daten zur Verfügung stellen. Es wird also eigentlich ein Client-Server-System für die Suche und Knotenverwaltung verwendet und nur die Datenübertragung selbst ist P2P. Damit werden natürlich die Skalierungsprobleme von Client-Server-Systemen nicht wirklich ausgemerzt. Gnutella, ein anderes bekanntes Peer-to-Peer-System, entspricht zwar der obigen Definition, tut dies allerdings mit Hilfe einer sehr viel Bandbreite kostenden Such- und Netzwerkverwaltungsstrategie.

Sehr häufig wird anstelle des Begriffs P2P-System übrigens auch der Begriff Overlay-Netzwerk verwendet. Dies soll verdeutlichen, dass ein P2P-System ein auf einer konkreten Netzwerk-Topologie aufgebautes, übergeordnetes Netzwerk mit „unabhängiger“ Topologie darstellt, das Basisnetzwerk also quasi „überlagert“. Der Aufbau und die Verwaltung dieser Overlay-Netzwerke können wohl als der eigentliche Beitrag von P2P-Systemen angesehen werden.

P2P-Architekturen lassen sich auf unterschiedlichsten Abstraktionsebenen einsetzen:

- **Netzwerkebene:** Routing von Anfragen in applikationsunabhängiger Weise über das physikalische Netzwerk.
- **Datenzugriffsebene:** Suche und Änderung von Ressourcen mit Hilfe applikationsspezifischer Zugriffsstrukturen.
- **Dienstebene:** Kombination und Erweiterung von Funktionalitäten der Datenzugriffsebene, um höherwertige Dienste zur Verfügung zu stellen. Die Bandbreite dieser Dienste kann von einfachem File-Sharing bis hin zu komplexen Geschäftsprozessen reichen.
- **Benutzerebene:** Gruppierung von Benutzern („Communities“) und Unterstützung von Benutzer-Interaktionen unter Verwendung der Dienstebene für Community-Management und Informationsaustausch.

Tabelle 1 zeigt typische Beispiele.

Ebene	Applikationsdomäne	Dienstklasse	Beispiel
Netzwerk	Internet	Routing	TCP/IP, DNS
Datenzugriff	Overlay-Netzwerke	Ressourcen-Lokation, „Data sharing“	Gnutella, Freenet
Dienst	P2P-Applikationen	Nachrichtendienste, verteilte Bearbeitung	SETI@Home, Napster, Groove, Skype
Benutzer	Benutzer-Communities	Kollaboration	eBay, Ciao

Tabelle 1: Beispiele für P2P auf unterschiedlichen Ebenen

In einem konkreten System kann das P2P-Paradigma unabhängig auf jeder dieser Ebenen auftreten. Die Kombination dieser Ebenen in einem konkreten System und die auf den einzelnen Ebenen verwendeten P2P-Konzepte charakterisieren ein konkretes P2P-System. Wie aus Tabelle 1 ersichtlich, variieren auch die Arten von Anwendungen beträchtlich. Das Spektrum reicht von „Data sharing“ über verteilte Verarbeitung, E-Commerce bis zu sozialen Netzwerken.

An „Overlay Networks“ lassen sich P2P-Eigenschaften wohl am klarsten beschreiben und dieser Bereich ist wohl auch am engsten mit den grundlegenden Fragestellungen im Datenbankbereich verbunden. Daher werden wir uns im Folgenden auf diesen Applikationsbereich konzentrieren, um ein Klassifikationsschema für P2P-Systeme zu definieren.

3 Klassifikation

Overlay-Networks lösen das Problem von verteilter Ressourcen-Lokation und im engeren Sinne sind alle P2P-Systeme wie Napster, Gnutella, Freenet, P-Grid, Chord, etc., als Ressourcen-Lokationssysteme zu bezeichnen. Ressourcen-Lokation in P2P-Systemen bedeutet, dass eine Gruppe G von Peers, jeder durch eine Adresse $p \in P$ identifiziert, Ressourcen $R(p) \subseteq R$ verwaltet, wobei jede Ressource $r \in R(p)$ durch einen Schlüssel $k \in K$ identifiziert wird. Jeder Peer ist mit $K(p) \subseteq K$ assoziiert. Die Lokation einer Ressource mit Schlüssel k (oder allgemeiner, eines Prädikats über k) bedeutet dann, dass – ausgehend von einem beliebigen Peer – der für diese Ressource zuständige Peer p gefunden werden soll, d.h. es gilt $k \in K(p)$. Anders ausgedrückt, handelt es sich um den Zugriff auf und die Verwaltung der binären Relation $I = \{(p, k) \mid k \in K(p)\} \subseteq K \times P$, d.h. der Indexinformation, wobei jeder Peer nur eine Teilmenge $I(p) \subseteq I$ kennt und verwaltet. Da ein Peer daher im Allgemeinen nicht in der Lage ist Anfragen direkt zu beantworten, wird eine solche Anfrage an andere Peers in der Nachbarschaft $N(p) \subseteq G$ weitergeleitet. Welche Peers aus $N(p)$ dazu selektiert werden, hängt vom jeweiligen P2P-System und dessen Suchstrategie ab.

Zur Interaktion mit anderen Peers sind in jedem P2P-System zwei grundlegende Protokolle notwendig:

Netzwerkverwaltungsprotokoll: Dieses Protokoll ermöglicht Peers in ein Overlay-Netzwerk einzusteigen bzw. es zu verlassen, wobei in fast allen Overlay-Netzwerken das Verlassen des Netzwerks implizit erfolgt, d.h. nicht angekündigt wird (Knotenausfall, Netzwerkunterbrechung, etc.) und daher durch geeignete Mechanismen abzufangen ist. Um an einem Overlay-Netzwerk teilnehmen zu können, muss ein Peer p zumindest ein Mitglied des Netzwerkes $p' \in G$ kennen und an dieses Mitglied eine *Join*-Nachricht schicken $p' \rightarrow join(p)$. Dies wird abhängig vom P2P-System weitere Nachrichten und eine Reorganisation der Nachbarschaftsverhältnisse und der von den Peers verwalteten Indexinformation auslösen. Ein Peer, der sich einem Netzwerk G anschließt, kann bereits Mitglied in einer anderen Gruppe G' sein, wodurch bei einigen Systemen die Möglichkeit besteht, die Netzwerke zu verbinden.

Datenverwaltungsprotokoll: Dieses Protokoll ermöglicht es Peers Daten zu suchen, einzufügen und zu löschen. Die entsprechenden Nachrichten sind $p' \rightarrow search(k)$, um Peers zu finden, die eine Ressource mit Schlüssel k besitzen, $p' \rightarrow insert(k, r)$, um eine Ressource r mit Schlüssel k in das Netzwerk einzufügen und $p' \rightarrow delete(k)$, um eine Ressource mit Schlüssel k zu löschen. Peers, die diese Nachrichten empfangen, können sie, falls die Anfrage nicht vom Peer selbst beantwortet werden kann, an andere Peers aus ihrer Nachbarschaft weiterleiten. Updates wie in Datenbanksystemen werden in den meisten P2P-Systemen nicht berücksichtigt, obwohl in einigen Systemen Protokolle dafür existieren, da P2P-System eher auf das Anbieten von Information und nicht auf Datenmanagement abzielen.

Diese Protokolle werden von allen P2P-Systemen auf systemspezifische Weise zur Verfügung gestellt, wobei eine sehr große Bandbreite in ihrer Umsetzung besteht. Die Verteilung der Indexinformation auf die teilnehmenden Peers, die Auswahl und Verwaltung der Nachbarschaftsbeziehungen, die von Peers über ihre Nachbarschaft verwaltete Information und die spezifischen Protokolle definieren die Eigenschaften eines konkreten P2P-Systems. Ein direkter Vergleich gestaltet sich daher oft als schwierig, da es sich um ein sehr generelles Problem handelt, bei dem viele funktionale Eigenschaften und Durchsatzcharakteristika zu berücksichtigen sind; die Umgebungen in denen P2P-Systeme eingesetzt werden durch ein komplexes Zusammenspiel von Parametern definiert werden; und die P2P-Ansätze selbst sehr unterschiedlich sind und für zum Teil sehr unterschiedliche Anwendungsbereiche ausgelegt wurden. Dennoch lassen sich die existierenden Ansätze anhand folgender Dimensionen klassifizieren:

Strukturierungsgrad: In unstrukturierten P2P-Systemen verwalten Peers keine Information über die Ressourcen anderer Peers, d.h. Indexinformation in Bezug auf eine Ressource $k \in K(p)$ wird nur von Peer p verwaltet und p verwaltet keine Informationen über von Peers in seiner Nachbarschaft $N(p)$ verwalteten Ressourcen, wodurch eine *gezielte* Weiterleitung von Anfragen nicht möglich ist. Ein typisches Beispiel dafür ist Gnutella, das Anfragen einfach so lange weiterleitet und Antworten liefert, bis die Lebensdauer einer Anfrage abläuft. Dieser Ansatz ermöglicht zwar einen extrem hohen Grad an Unabhängigkeit und Fehlertoleranz, geht aber auf Kosten von Effizienz und vorhersagbarem Systemverhalten. In strukturierten P2P-Systemen hingegen verwalten Peers lokal Wissen über die von anderen Peers verwalteten Ressourcen, wodurch eine zielorientierte Suche ermöglicht wird, jedoch zusätzlicher Aufwand für die Verwaltung dieser Information notwendig wird. Ein Beispiel für die hier gemeinte Information sind Routing-Tabellen, die auf dem aktuellen Stand gehalten werden müssen.

Hierarchiegrad: In flachen P2P-Systemen gibt es keine Unterscheidung in den Rollen, die ein Peer spielt. Dies ist der Ansatz von Systemen wie Gnutella. In hierarchischen Systemen hingegen existieren spezielle Rollen, d.h. bestimmte Peers stellen bestimmte Funktionalitäten, wie beispielsweise Suche, zur Verfügung. Ein Extrembeispiel dafür ist Napster, das nur einen (replizierten) Knoten besitzt, der Suchoperationen durchführen kann. Der Hauptvorteil von hierarchischen Systemen ist die leichtere Wartbarkeit und der höhere Durchsatz für Suchoperationen. Dies wird allerdings mit der Aufgabe einiger Vorteile der P2P-Architektur, wie höherer Fehlertoleranz und verteilter Verarbeitung, bezahlt.

Kopplungsgrad: In stark gekoppelten P2P-Systemen existiert zu jedem Zeitpunkt genau eine Gruppe, alle teilnehmenden Peers gehören dieser gobalen Gruppe an, und nur ein Peer (bis einige wenige Peers) kann sich dieser Gruppe jeweils anschließen bzw. sie verlassen. In diesem Systemtyp, wird den Peers beim Eintritt in die Gruppe eine statische, logische Identität zugewiesen, die die Rolle des Peers in Bezug auf die Gruppe genau festlegt, d.h., welche Daten er verwaltet oder die Art und Weise wie Nachrichten behandelt werden. In diesen Systemen sind die für Informationen möglichen Schlüsselintervalle mit jenen für die Peer-Identifikation zur Verfügung stehenden identisch. Ein typischer Vertreter für diesen Systemtyp ist Chord [Dabek et al. 2001]. Diese Systemstruktur beschränkt natürlich eindeutig die Möglichkeiten Peer-Populationen, die sich unabhängig von einander entwickeln, zu vereinigen bzw. zu trennen. Lose gekoppelte Systeme hingegen bieten diese Möglichkeiten. Gnutella ist ein typischer Vertreter für ein lose gekoppeltes P2P-System.

Tabelle 2 zeigt eine Einteilung einiger bekannter Systeme anhand dieser Kriterien:

	Strukturierungsgrad		**Hierarchiegrad**		**Kopplungsgrad**	
	Unstrukturiert	*Strukturiert*	*Flach*	*Hierarchisch*	*Lose*	*Eng*
Gnutella	✓		✓		✓	
FastTrack		✓		✓		✓
Freenet		✓	✓		✓	
Chord		✓	✓			✓
DKS		✓	✓			✓
P-Grid		✓	✓		✓	
Tapestry		✓	✓			✓
Pastry		✓	✓			✓
CAN		✓	✓			✓

Tabelle 2: Beispiele für P2P auf unterschiedlichen Ebenen

4 Selbstorganisation

Einen wesentlichen Anteil am Erfolg und an der von P2P-Systemen ausgehenden Faszination haben die in diesen Systemen auftretenden Selbstorganisationsphänomene. Diese Phänomene spielen eine Schlüsselrolle auf allen Systemebenen und lassen sich durch das teilweise Fehlen von zentraler Kontrolle erklären, die in Systemen dieser Größenordnung nur bedingt möglich ist. Damit ist das Verhalten von P2P-Systemen oft nur mit Hilfe statistischer Mittel (stochastische Prozesse) beschreibbar. Obwohl damit keine im informatischen Sinne „absoluten" Garantien mehr möglich sind, bringt die Fähigkeit zu Selbstorganisation wiederum andere sehr willkommene Eigenschaften, wie hohe Ausfallsicherheit oder hohe Flexibilität mit sich, da selbstorganisierende Systeme auf lokale Störungen wie beispielsweise den Ausfall oder die Überlastung einzelner lokaler Komponenten i.A. unempfindlich reagieren. Andere Komponenten können Ausfälle somit kompensieren. Einfach ausgedrückt handelt es sich bei Selbstorganisation um die Fähigkeit eines dynamischen Systems aus lokalen Interaktionen ein komplexes Systemverhalten zu entwickeln. Selbstorganisationsprozesse werden durch Zufallsprozesse ausgelöst wie zum Beispiel Kontakte zwischen Mitgliedern von Insektenkolonien oder Suchabfragen bzw. das Einfügen von Daten in einem P2P-System. Diese Fluktuationen führen zu permanenten „Störungen", die es einem System ermöglichen seinen möglichen Zustandsraum auszuloten bis ein dynamisches Gleichgewicht erreicht wird [Heylighen 1997].

In der Informatik wurden solche Prozesse bisher vor allem im Bereich der künstlichen Intelligenz erforscht. Mit P2P-Systemen stehen jetzt aber zum ersten Mal sehr große Systeme zur Verfügung, um diese Prozesse und Phänomene genauer zu erforschen, da theoretische Ergebnisse in Experimenten praktisch und statistisch aussagekräftig verifiziert werden können. Andererseits ermöglichen die aus Experimenten erhaltenen Ergebnisse auch eine Weiterentwicklung der Theorie, wie am Beispiel von *Small World*-Graphen eindrucksvoll gezeigt wird. Vor allem mit Gnutella wurden einige interessante Experimente vorgenommen, die die Möglichkeit zur Selbstorganisation in P2P-Systemen verdeutlichen. Gnutella wurde für diese Experimente wegen seiner sehr einfachen Systemstruktur gewählt, deren Funktionieren, wie man inzwischen weiß, zu einem Grossteil auf Selbstorganisationsprozessen beruht. Studien des durch sehr einfache Interaktionen aufgebauten Gnutella-Netzwerkes haben zwei interessante Eigenschaften der resultierenden Netzwerk-Struktur gezeigt:

Power-Law-Verteilung der Knotenkonnektivität: Nur wenige Knoten im Gnutella-Netzwerk haben viele Links zu anderen Peers, die Mehrzahl der Peers haben nur eine sehr geringe Konnektivität. Dies kann als Resultat einer bevorzugten Anbindung an Knoten mit hoher Konnektivität beim Einstieg in das Netzwerk erklärt werden [Barabási & Réka 1999].

Geringer Netzwerkdurchmesser: Obwohl sich Gnutella „zufällig" entwickelt, hat der resultierende Graph einen relative geringen Durchmesser von ungefähr 7. Diese Eigenschaft ermöglicht es, dass Gnutella's Broadcast-Ansatz für die Suche mit einer relativ geringen Latenz bzw. *Time-To-Live* (TTL) funktioniert (die TTL ist hier als Maximalanzahl an Übertragungen einer Anfrage/Nachricht zwischen Peers zu verstehen). Die Erklärung dafür ist das so genannte *Small-World*-Phänomen, das bereits seit längerem für soziale Netzwerke bekannt ist [Kleinberg 2000]. Small-World-Graphen [Watts & Strogatz 1998] kombinieren eine typische Eigenschaft von zufälligen Graphen, nämlich geringen Netzwerkdurchmesser, mit einem hohen Grad an Clustering. Damit kann in dieser Art von Graphen Information schnell gefunden werden und „verwandte" Informationen sind gruppiert. Neben Gnutella zeigt auch der Netzwerkgraph von Freenet [Clarke et al. 2001] Small-World-Eigenschaften. Typischerweise sind Small-World-Graphen das Resultat eines Selbstorganisationsprozesses.

Außer in diesen Bereichen treten Selbstorganisationsprozesse auch auf allen anderen Ebenen auf. Als Beispiele dafür seien hier das Problem von Free-Riding und verteilte Verwaltung von Reputation und Vertrauensbeziehungen genannt, auf die hier aus Platzgründen nicht näher eingegangen werden kann. Als ein wichtiger Beitrag von P2P-Systemen ist daher die Einführung von Selbstorganisationsprinzipien in verteilte Anwendungsarchitekturen zu sehen. Bei adäquater Anwendung ermöglicht Selbstorganisation hohe Skalierbarkeit und Fehlertoleranz, zwei wesentlich am Erfolg des P2P-Paradigmas beteiligte Faktoren.

5 Lokationssysteme

Wie bereits ausgeführt können P2P-Prinzipien in unterschiedlichsten Ausprägungen und Anwendungen auftreten. Der derzeit als Kernbereich angesehene und für den Erfolg des P2P-Paradigmas verantwortliche Bereich ist jedoch sicherlich die Lokation von Ressourcen. Im Folgenden beschränken wir uns daher auf eine Gegenüberstellung von Systemen aus diesem Bereich.

5.1 Unstrukturierte P2P-Systeme

Alle derzeitigen unstrukturierten P2P-Systeme sind flach und lose gekoppelt. Der typische Repräsentant dieser Klasse ist Gnutella [Clip 2001]. Ursprünglich wurde Gnutella als „Quick-Hack" zum Austausch von Kochrezepten von Nullsoft-Entwicklern (Winamp) entwickelt. Die Protokolle wurden nie veröffentlicht sondern aus der nur wenige Stunden verfügbaren Software „re-engineered". Das Protokoll besteht aus 5 Nachrichtentypen für Netzwerkverwaltung (*Ping*, *Pong*) und Datenmanagement (*Query*, *QueryHit*, *Push*).

Das Routing von Nachrichten in Gnutella erfolgt durch einen einfachen „Constraint Broadcast"-Ansatz: Nach Erhalt einer Nachricht wird ihr *Time-To-Live* (TTL) Zähler verringert. Ist die TTL größer als 0 und wurde die Nachricht noch nicht empfangen (Vermeidung von Schleifen durch in die Pakete eingebettete Identifikatoren), wird die Nachricht an alle Peers in der Nachbarschaft, die mit dem Peer verbunden sind, weitergeleitet. Üblicherweise hat jeder Peer 4 aktive Verbindungen zu anderen Peers, d.h., jedes empfangene Paket, wird an 3 Peers weitergeleitet. Gleichzeitig überprüft jeder Empfänger, ob lokal Suchergebnisse vorhanden sind. Informationen über Suchergebnisse (*QueryHit*) werden entlang des Netzwerkpfades, auf dem eine Anfrage (*Query*) empfangen wurde, zurückgegeben.

Um an einem Gnutella-Netzwerk teilnehmen zu können, muss ein Peer einen Gnutella-Peer kennen. An diesen Peer wird eine *Ping*-Nachricht geschickt, die wie eine Suchanfrage weitergeleitet wird und von anderen Peers mit einer *Pong*-Nachricht beantwortet wird, die ihre Adresse und einige Systemparameter enthalten. Aus den empfangenen *Pong*-Nachrichten wählt der Peer maximal 4 aus und stellt zu diesen Verbindungen her (unter der Voraussetzung, dass diese noch weniger als 4 aktive Verbindungen haben). Der Transfer von Dateien (basierend auf den via *QueryHit* gelieferten Informationen) erfolgt über ein vereinfachtes HTTP-Protokoll direkt zwischen zwei Peers. *Push*-Nachrichten werden verwendet, wenn ein Peer sich hinter einer Firewall befindet und der Transfer daher als „Callback" initiiert werden muss.

Der Ansatz von Gnutella ist zwar einfach, aber effizient: Suchergebnisse werden schnell zurückgeliefert, und das Netzwerk ist sehr fehlertolerant. Sein Nachteil ist jedoch der hohe Bandbreitenverbrauch. Beispielsweise verursacht ein einziges *Ping* bei 4 Verbindungen (C) pro Peer und einer typischen TTL von 7, $2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26{,}240$ Nachrichten.

Neuere Ansätze verringern diesen hohen Bandbreitenverbrauch [Lv et al. 2002]. Dazu zählen *Expanding Ring Search* oder *Random Walker*, aber auch Ansätze aus der Perkulationstheorie [Sarshar et al. 2003], auf die allerdings aus Platzgründen nicht näher eingegangen werden kann.

5.2 Hierarchische P2P-Systeme

Hierarchische P2P-Systeme speichern Indexinformation auf einigen ausgezeichneten Peers, um die Suchgeschwindigkeit zu erhöhen. Das bekannteste System ist Napster, doch wurde der Ansatz in neueren Systemen, die üblicherweise als Super-Peer-Systeme bezeichnet werden, in Bezug auf Skalierbarkeit und Robustheit verbessert. Der bekannteste Vertreter dieser Systemfamilie ist FastTrack, die P2P-Suchmaschine hinter Kazaa. In FastTrack werden 3 Arten von Peers unterschieden:

- Ein Super-Super-Peer, der als zentraler Einstiegspunkt dient und Listen von Super-Peers zur Verfügung stellt.
- Super-Peers, die Indexinformation für an sie angebundene Peers verwalten und mit anderen Super-Peers interagieren (z.B. bei Suchvorgängen). Mehrere Super-Peers können mit einer Gruppe von Peers verbunden sein.
- Normale Peers, die wie in Napster ihre Ressourcen bei einem Super-Peer registrieren und an diesen Suchanfragen stellen. Die Übertragung gefundener Dateien erfolgt wie in Napster direkt zwischen den Peers ohne Zuhilfenahme eines Super-Peers.

Super-Peers werden unter den teilnehmenden Peers dynamisch ausgewählt. Experimente [Yang & Garcia-Molina 2002] zeigen, dass redundante Super-Peers von Vorteil sind und dass Super-Peers

substantiell mehr abgehende Verbindungen haben sollten als in Gnutella (> 20), um die TTL zu minimieren und so den Durchsatz in Bezug auf Suchlatenz an Napster's Durchsatz bei gleichzeitig besserer Skalierbarkeit und Robustheit anzunähern.

5.3 Strukturierte P2P-Systeme

In strukturierten P2P-Systemen wird ein auf alle Teilnehmer aufgeteilter verteilter Index aufgebaut. Im Gegensatz zu hierarchischen P2P-Systemen verhindert diese gleichmäßige Verteilung der Indexinformation Engpässe und Asymmetrien. Dies wird jedoch mit weit komplexeren Wartungsaufgaben bezahlt, da nicht nur der Index, sondern auch die zur Verwaltung des Index aufgebaute verteilte Infrastruktur gewartet werden muss. Allen strukturierten P2P-Systemen ist gemein, dass sie sowohl einen Teil des Gesamtindex als auch Routing-Tabellen als Teil der globalen Zugriffsstruktur besitzen und Suchoperationen durch Weiterleitung an mittels der jeweiligen Suchstrategien aus der Routing-Tabelle ausgewählte Peers durchgeführt werden. Die Routing-Tabellen werden dabei so konstruiert, dass ihre Größe nur sehr langsam anwächst und Suche schnell durchgeführt werden kann – beide Größen wachsen üblicherweise logarithmisch zur Anzahl der Peers im System. Damit skalieren diese Systeme sehr gut.

Der Aufbau und die Verwaltung von Routing-Tabellen erfolgt üblicherweise nach einer der folgenden Strategien:

- *Ad-hoc Caching and Clustering* entlang der Suchpfade (Freenet [Clarke et al. 2001])
- *Distributed Hash Tables* (P-Grid [Aberer 2001], Chord [Dabek et al. 2001], Pastry [Rowstron & Druschel 2001], Tapestry [Zhao et al. 2001])
- Topologisches Routing (CAN [Ratnasamy et al. 2001])

Neben dieser Einteilung nach gewähltem Routing-Ansatz bestehen die Hauptunterschiede zwischen den Systemen in Bezug auf ihre funktionale Flexibilität bei Replikation, Lastverteilung, unterstützten Suchprädikaten, und Eigenschaften der Netzwerkverwaltungsprotokolle. Lose gekoppelte strukturierte P2P-Systeme wie Freenet und P-Grid haben mit unstrukturierten Systemen die flexible Netzwerk-Evolution gemeinsam. Eng gekoppelte Systeme wie CAN, Chord, Tapestry und Pastry hingegen benutzen rigidere Verwaltungsstrategien, um dadurch kontrollierbareren Durchsatz zu erhalten.

Freenet

Freenet [Clarke et al. 2001] ist ein dezentrales P2P-System für die Publikation, Speicherung und Replikation von Dateien, sowie eine Suchinfrastruktur zum Auffinden gespeicherter Inhalte. Freenet verwendet ein adaptives Routing-Schema, dass Suchanfragen in Richtung des wahrscheinlichen Speicherorts der gesuchten Information weiterleitet. Um dies effizient zu ermöglichen, werden die Routing-Tabellen von Freenet-Peers dynamisch im Zuge von Suchanfragen und Einfügeoperationen angepasst und optimiert. Wie sich zeigt, wird mit dieser Strategie, ähnlich wie bei Gnutella ein Small-World-Graph aufgebaut.

Nach dem Einklinken wird die zu Beginn leere Routing-Tabelle eines Peers sukzessive mit Informationen aus Suchabfragen und Antwortnachrichten (abhängig vom Füllgrad und der Antwortgröße) gefüllt. Auf diese Weise „lernt" der Peer die Struktur des Freenet-Netzwerkes und kann Nachrichten zielgerichteter weiterleiten. Die Routing-Tabelle eines Freenet-Peers speichert die Adressen benachbarter Peers sowie die Schlüssel und Daten, die diese speichern. Die zum Aufbau der Routing-Tabelle notwendige Information ist dem Peer einfach zugänglich, da Suchergebnisse entlang des Suchpfades retourniert werden.

Empfängt ein Peer ein Query, wird die Routing-Tabelle konsultiert und falls der entsprechende Schlüssel in der Routing-Tabelle enthalten ist, wird unmittelbar eine Antwort zurückgeschickt. Wenn nicht, wird die Anfrage an jenen Peer weitergeleitet, der den Schlüssel mit der geringsten lexikographischen Distanz zur Anfrage hat. Liefert dies kein Ergebnis, wird der nächste Eintrag versucht und so weiter. Bei Scheitern aller Einträge wird Backtracking durchgeführt. Algorithmisch gesehen implementiert Freenet somit eine *Depth-first*-Suchstrategie. Um Zyklen im Routing von Suchabfragen zu erkennen, enthalten die Anfragen wie in Gnutella eindeutige Identifikationsnummern. Des Weiteren verwendet Freenet ein einfaches, aber effizientes Schema zur kollisionsfreien Erzeugung von Schlüsseln.

Distributed Hash Tables

In *Distributed Hash Tables* (DHT) werden Peer-Identifikationen und Ressourcen-Schlüssel durch Hashing in denselben Schlüsselraum abgebildet. Durch diese Abbildung wird einem Peer die Verantwortung für einen bestimmten Teilraum übertragen, d.h. er ist für die Verwaltung für Schlüssel aus diesem Teilraum verantwortlich. Damit von jedem Peer aus jeder andere Peer erreicht werden kann, oder anders ausgedrückt beliebige Schlüssel gefunden werden können, baut jeder Peer eine Routing-Tabelle auf. Um die Suche effizient zu machen, enthält die Routing-Tabelle Verweise auf andere Peers mit exponentiell ansteigender Distanz zu seiner eigenen Position im Schlüsselraum. Dadurch wird im Prinzip ein *Small-World*-Graph aufgebaut, der Suche in *O(log(n))* Schritten (Nachrichten) bei *n* Peers ermöglicht.

Im Grunde basieren alle unter dem Oberbegriff DHT zusammengefassten P2P-Systeme (P-Grid, Chord, Pastry, etc.) auf Varianten dieses Ansatzes. Unterschiede bestehen im Wesentlichen in Bezug auf:

- Fixe Unterteilung des Schlüsselraums (P-Grid, Pastry) vs. variable Unterteilung des Schlüsselraums (Chord).
- Die Topologie des Schlüsselraums (Ring, Intervall, etc.).
- Verwaltung der Routing-Information (Mehrfacheinträge zur Erhöhung der Ausfallssicherheit, Berücksichtigung der Netzwerkdynamik).

Als exemplarisches Beispiel sei hier eine einfache auf Präfix-Routing [Plaxton et al. 1997] basierende Strategie skizziert. Ohne Beschränkung der Allgemeinheit nehmen wir dazu an, dass Ressourcen durch binäre Schlüssel identifiziert werden und in einem binären Baum organisiert werden. Der Einfachheit halber nehmen wir an, der Baum sei balanciert und habe die Höhe *d*. Nun assoziieren wir jeden Blattknoten mit einem Peer und übertragen ihm dadurch die Verantwortung für die Verwaltung aller Schlüssel, die den Pfad von der Wurzel des Baumes bis zum jeweiligen Blattknoten (Peer) als Präfix haben. Dies ist im linken Teil von Abb.1 für *d = 2* dargestellt. Da der Baum nicht zentral gespeichert werden kann, besteht die Strategie für die Verteilung des Baumes darin, dass jeder Peer für jede Ebene seines Pfades die Adresse mindestens eines Peers speichert, der erreicht werden kann, wenn man auf dieser Ebene der nicht dem eigenen Pfad entsprechenden Kante folgt. Die Gesamtheit dieser Informationen stellt die Routing-Tabelle des Peers dar. Der rechte Teil von Abb.1 zeigt diese Dekomposition des Baumes in Routing-Tabellen und zeigt als Beispiel zwei mögliche Routing-Einträge für Peer 3.

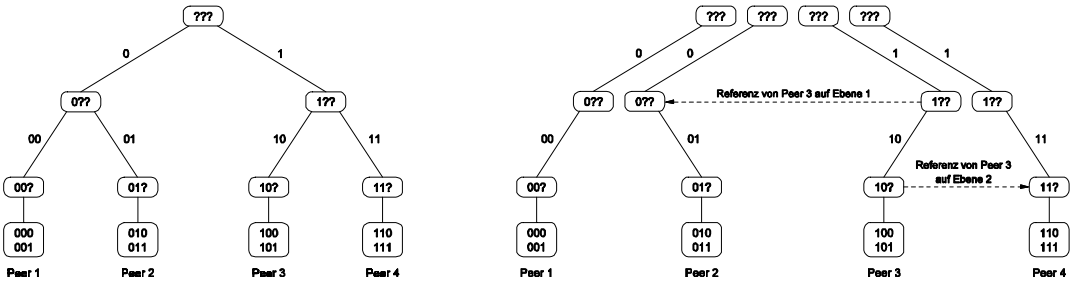

Abb.1: Aufbau einer exemplarischen DHT

Durch diese Strategie enthält die Routing-Tabelle eines Peers, wenn man den Pfad von seinem Blattknoten aus Richtung Wurzel entlang geht, auf jeder Ebene einen Verweis auf einen Peer mit exponentiell ansteigender Distanz. Des weiteren ist gewährleistet, dass von jedem Peer aus jeder andere Peer, d.h., Teil des Suchraums, in *O(log(n))* Schritten erreicht werden kann. Der für die Routing-Tabellen notwendige Speicherplatz wächst ebenfalls logarithmisch an, da die Höhe des Baumes logarithmisch zur Anzahl der Peers im System ist.

Topologisches Routing

Wie in DHTs wird bei topologischem Routing, wie beispielsweise in CAN [Ratnasamy et al. 2001] verwendet, die Zuordnung von Peers zu Ressourcen durch Hashing der Peer-Adressen und Ressourcen-Schlüssel in einen gemeinsamen Schlüsselraum durchgeführt. Bei topologischem Routing wird allerdings ein *d*-dimensionaler Schlüsselraum verwendet – genauer gesagt ein *d*-dimensionaler Torus, da die Schlüsselerzeugung modulo einer festen Basis durchgeführt wird – wodurch die erzeugten

Schlüssel *d*-dimensionale Vektoren einfacher Schlüssel sind. Abb. 2 zeigt ein Bespiel für CAN mit $d = 2$.

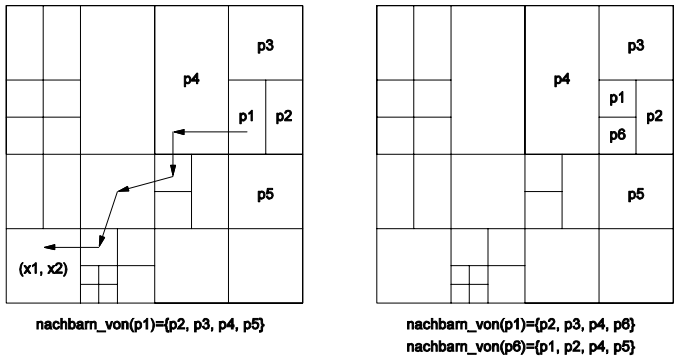

Abb. 2: Topologisches Routing in CAN

Schlüssel werden als Koordinaten im Gesamtschlüsselraum angesehen, der in den einzelnen Peers zugeteilte Teilräume unterteilt ist. Im Zuge des Aufbaus eines CAN-Netzwerkes wird der mehrdimensionale Raum immer weiter unterteilt und Peers merken sich in ihren Routing-Tabellen, welche Peers für angrenzende Teilräume zuständig sind. Dadurch wird ein einfacher Routing-Algorithmus ermöglicht, bei dem Peers Suchabfragen an Peers weiterleiten, die für Teilräume verantwortlich sind, deren Koordinaten näher am gesuchten Schlüssel (bzw. den gesuchten Koordinaten) liegen. In Abb. 2 leitet p1 beispielsweise die Suchanfrage an p4 weiter, da dieser in näher am gesuchten Intervall liegt (in Bezug auf eine Koordinatenachse) Die Suchkosten in CAN hängen somit nicht nur von der Größe des Netzwerkes, sondern auch von der gewählten Dimension ab und haben einen Erwartungswert von $O(dn^{(1/d)})$.

6 Referenzarchitekturen und Anwendungsbeispiele

Alle bisher beschriebenen Systeme zeigen ähnliche Eigenschaften und lösen ähnlich gelagerte Probleme in unterschiedlichem Ausmaß und auf verschiedenste Art und Weise. Die Prinzipien eines jeden Paradigmas lassen sich dennoch üblicherweise anhand von allgemeinen Referenzmodellen beschreiben. Auch im P2P-Bereich ist dies momentan im Gange. Dennoch gibt es derzeit noch keine, die gesamte Bandbreite an Anwendungsarchitekturen abdeckende Referenzarchitektur für P2P-Systeme. Es gibt einige Ansätze und Vorschläge wie beispielsweise JXTA [Gong 2001], die jedoch noch keinesfalls als generell anwendbar angesehen werden können.

Die Notwendigkeit einer Referenzarchitektur für P2P-Systeme ergibt sich gerade in diesem sehr dynamischen Bereich aus einer simplen Notwendigkeit: Interoperabilität. Derzeit sind so gut wie alle P2P-Anwendungen, selbst wenn man spezialisierte Bereiche, wie beispielsweise *Distributed Hash Tables* (DHT) betrachtet inkompatibel und verwenden uneinheitliche Abstraktionen. Für einen Erfolg auf breiter Basis, z.B. durch eine P2P-Middleware, die eine einheitliche Infrastruktur für Anwendungen zur Verfügung stellt und unterschiedliche P2P-Ansätze einbinden und abstrahieren kann, ist dies jedoch von zentraler Bedeutung. Eine Referenzarchitektur hat dabei die unterschiedlichen Ebenen auf denen das P2P-Paradigma auftreten kann – Netzwerk-, Datenzugriffs-, Dienst- und Benutzerebene – abzudecken, muss die Einschränkungen und Randbedingungen der unterschiedlichen System erfassen und beschreiben können, generelle APIs sowohl vertikal (Dienst) als auch horizontal (Protokollinteraktion und -kompatibilität) zur Verfügung stellen und die unterschiedlichen Ansätze unter gemeinsamen Abstraktionen vereinheitlichen. Neben JXTA [Gong 2001], das als erster Schritt in diese Richtung angesehen werden kann und im Folgenden kurz diskutiert wird, ist dies eines der Ziele des EU-Projekts Evergrow (http://www.evergrow.org/, Integrated Project Nr. 1935).

6.1 JXTA

JXTA [Gong 2001] definiert eine Drei-Schichten-Architektur, XML-basierte Protokolle, und einige grundlegende Abstraktionen wie Peer Groups, Pipes und Advertisements, um eine einheitliche Programmierplattform für P2P-Applikationen zu schaffen und Interoperabilität und Unabhängigkeit von Software- und Hardwareplattformen zu ermöglichen. Abb.3 zeigt die Architektur von JXTA.

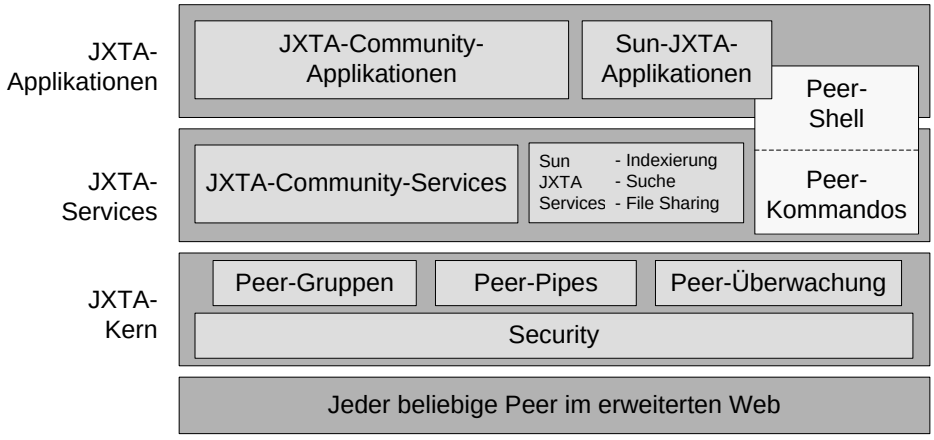

Abb.3: Die JXTA-Architektur

Der JXTA-Kern stellt Kommunikationsprimitive (Adressierung, Peer-Gruppen), Routing sowie eine Monitoring-Schnittstelle zur Verfügung. Darauf bauen dann die bekannten P2P-Dienste wie Indizierung, Suche und File-Sharing auf, welche wiederum als Grundlage für P2P-Applikationen dienen. Während andere P2P-Systeme sich nicht um generell anwendbare architekturelle Gesichtspunkte kümmern, ist dies genau das Ziel von JXTA. Dies hat allerdings auch zur Folge, dass die in JXTA implementierten und damit dem Entwickler zur Verfügung stehenden Funktionalitäten zwar über ein sehr sauberes API mit sauberer *Separation of Concerns* zugreifbar sind, die Anwendbarkeit aber hinter dem aktuellen Stand der Entwicklung zurückliegt. Stark vereinfachend kann gesagt werden, dass JXTA vor allem auf hierarchische P2P-Ansätze ausgerichtet ist, unstrukturierte Systeme mit einigem Overhead abgebildet werden können, strukturierte Ansätze wie DHTs aber nicht richtig in die Philosophie von JXTA passen. Im Moment stellt JXTA eine Implementierungsarchitektur dar, aber noch keine für die praktische Anwendung geeignete Programmierplattform. Nichtsdestotrotz sind die in JXTA vorgeschlagenen Ansätze für jeden interessant, der P2P-Applikationen entwirft. Sicherlich werden auch etliche in JXTA vorgeschlagene Ideen in andere Architekturen Eingang finden.

6.2 Groove

Um auch P2P-Systeme jenseits von File-Sharing-Applikationen abzudecken, präsentieren wir hier noch exemplarisch ein Anwendungsbeispiel aus der Groupware-Domäne, um weitere zu berücksichtigende architekturelle Parameter zu verdeutlichen. Groove ist eine Peer-to-Peer-Plattform für die unstrukturierte Kollaboration und Kommunikation von Teams und wurde vor kurzem von Microsoft aufgekauft. Mit anderen Worten: Eine Ablaufsteuerung (Workflow) wird nicht unterstützt. Groove ist im derzeitigen Entwicklungsstand keine „echte“ P2P-Lösung, da es nach wie vor auf einen zentralen, stets erreichbaren Groove-Relay-Server (siehe Abb. 4) aufsetzt und damit eine ähnliche Architektur wie Napster besitzt. Die Kommunikation zwischen Groove peers basiert auf einem propriäterem Nachrichtenformat (SSTP). Jeder Groove peer registriert sich (seinen Workspace) beim Groove Relay Server (über das http Protokoll). Der Nachrichtenaustausch zwischen peers mit einer Firewall dazwischcen geschieht über den Relay-Server.

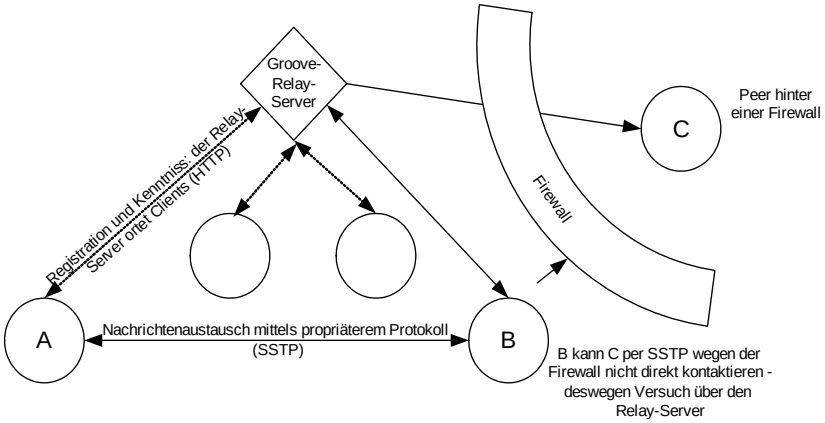

Abb. 4: Groove-Anwendungsszenario

Groove-Teilnehmer arbeiten in gemeinsamen Workspaces. Diese Workspaces sind jedoch asynchron, d.h., es ist nicht notwendig, dass alle Teilnehmer auch tatsächlich zur selben Zeit online und miteinander verbunden sind. In diesen Workspaces werden sämtliche Änderungen der Teilnehmer an einer gemeinsamen Aktivität auch allen anderen Teilnehmern angezeigt. Der dafür eingesetzte zentralisierte Replikationsmechanismus basiert auf Nachrichten, die an die anderen Mitglieder des Workspaces gesendet werden. Dieser Replikationsmechanismus basiert auf einem „store-and-forward"-Prinzip und wird bei Groove als Relay-Service bezeichnet. Werden Änderungen (Deltas) von einem Teilnehmer, der offline ist, vorgenommen, so werden diese Deltas von dem Relay-Server zwischengespeichert und danach simultan an die anderen Workspace-Teilnehmer propagiert. Die Granularität der Replikation ist immer ein Workspace. Dieser Mechanismus ermöglicht das Wechseln des Mobilitätszustands und Geräts. Kurz gesagt, die Teilnehmer werden wie bei einem Notifikationssystem aufgrund von Ereignissen (Events) notifiziert. Der Relay-Server dient auch als (proprietärer) Directory-Service, um Teilnehmer von Workspaces zu lokalisieren. Aufbauend auf der Basisfunktionalität gibt es auch die Möglichkeit, weitere Werkzeuge in den Client zu integrieren bzw. Groove-Funktionalität mittels Web-Services in eigene Software zu integrieren. Im Sicherheitsbereich unterstützt Groove Ende-zu-Ende-Public-Key-Verschlüsselung, Authentifizierung, Unleugbarkeit des Ursprungs (*Non-Repudiation*) und Datenintegrität.

7 Abschließende Bemerkungen

Nach Diskussion der dem P2P-Ansatz zu Grunde liegenden Prinzipien sowie repräsentativer Systeme und Architekturen stellt sich nun natürlich die Frage nach weiteren Anwendungsgebieten und nach Entscheidungskriterien, wann und ob eine verteilte Applikation als P2P-Applikation ausgelegt werden sollte. Die Beantwortung dieser Frage ist keineswegs trivial. Zum einen existieren sehr gute Client-Server-Infrastrukturen, die von einer Vielzahl von Bibliotheken und Sprachen hervorragend unterstützt werden und von Programmieren gut verstanden werden. Zum anderen kann sich gerade in einem so neuen und daher sehr dynamischen Bereich wie dem der P2P-Systeme der aktuelle Stand jederzeit ändern. Des Weiteren befinden sich viele derzeit vorgeschlagene P2P-Systeme noch in einem sehr experimentellen Stadium, wobei einige derzeit nur in Form von Modellen und Simulationen existieren. Zum Teil existieren keine oder nur unvollständige Implementierungen, die für einen Praxiseinsatz noch nicht geeignet sind. Außerdem sind derzeit die konkreten Anwendungsfälle für P2P-Architekturen auf Grund fehlender Erfahrungen noch schwer abzuschätzen. Dennoch scheint sich abzuzeichnen, dass der Trend weg von Client-Server-basierten Systemen hin zu symmetrischeren Architekturen führt. Die Gründe hierfür liegen wohl zum Großteil in der Flexibilität dieser Architekturen, der besseren Nutzung derzeit brachliegender Ressourcen und in der Möglichkeit, Systeme besser zu entkoppeln und dadurch leichter wartbar zu machen.

Auf Grund von Platzbeschränkungen, konnte in diesem Artikel nur ein allgemeiner Überblick über existierende P2P-Systeme gegeben werden. Für detailliertere Vergleiche sei auf [Milojicic et al. 2002] und [Risson & Moors 2004] verwiesen.

Bibliographie

[Aberer 2001] Aberer, K.: P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In: Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS).

[Barabási & Réka 1999] Barabási, A.; Réka, A.: Emergence of scaling in random networks. Science, 286: 509–512, 1999.

[Clarke et al. 2001] Clarke, I.; Sandberg, O.; Wiley, B.; Hong, T.: Freenet: A Distributed Anonymous Infor-mation Storage and Retrieval System. In: Proceedings of Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unob-servability, LLNCS 2009, Springer Verlag 2001

[Clip2 2001] The Gnutella Protocol Specification v0.4 (Document Revision 1.2), 15. Juni 2001, http://www.clip2.com/GnutellaProtocol04.pdf.

[Dabek et al. 2001] Dabek, F.; Brunskill, E.; Kaashoek, M. F.; Karger, D.; Morris, R.; Stoica, I.; Balakrishnan, H.: Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In: Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII).

[Gong 2001] Gong, L.: JXTA: A Network Programming Environment. IEEE Internet Computing, 5(3):88–95.

[Heylighen 1997] Heylighen, F.: Self-organization. Principia Cybernetica Web, Jan. 1997. http://pespmc1.vub.ac.be/SELFORG.html

[Kleinberg 2000] Kleinberg, J.: The Small-World Phenomenon: An algorithmic Perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing.

[Lv et al. 2002] Lv, Q.; Cao, P.; Cohen, E.; Li, K.; Shenker, S.: Search and Replication in Unstructured Peer-to-Peer Networks. In: Proceedings of 16th ACM International Conference on Supercomputing (ICS'02).

[Milojicic et al. 2002] Milojicic, D. S.; Kalogeraki, V.; Lukose, R.; Nagaraja, K.; Pruyne, J. ; Richard, B.; Rollins, S.; Xu, Z.: Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, Mar. 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf.

[Plaxton et al. 1997] Plaxton, C. G.; Rajaraman, R.; Richa, A. W.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures, S. 311–320, 1997.

[Ratnasamy et al 2001] Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R.; Shenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the ACM SIGCOMM, 2001.

[Risson & Moors 2004] Risson J.; Moors, T.: A survey of research towards robust P2P networks: search methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sidney, Australia. http://uluru.ee.unsw.edu.au/~john/tr-unsw-ee-p2p-1-1.pdf.

[Rowstron & Druschel 2001] Rowstron, A.; Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Lecture Notes in Computer Science 2218, S. 329–350, 2001.

[Sarshar et al. 2003] Sarshar, N.; Roychowdury, V.; Boykin, P. O.: Percolation-based Search on unstructured Peer-To-Peer Networks, 2003. http://www.ee.ucla.edu/~nima/Publications/search ITPTS.pdf.

[Watts & Strogatz 1998] Watts, D.; Strogatz, S.: Collective dynamics of small-world networks. In: Nature, 393, 1998.

[Yang & Garcia-Molina 2002] Yang, B.; Garcia-Molina, H.: Improving Search in Peer-to-Peer Networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), S. 5–14, 2002.

[Zhao et al. 2001] Zhao, B. Y.; Kubiatowicz, J.; Joseph, A. D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Computer Science Division, 2001.
http://oceanstore.cs.berkeley.edu/publications/papers/pdf/tapestry_sigcomm_tr.pdf.

Autoren

Dr. Manfred Hauswirth arbeitet seit Januar 2002 als Postdoc am Institut für Verteilte Informationssysteme, der École Polytechnique Fédérale de Lausanne (EPFL), Schweiz. Er studierte Informatik an der TU Wien (Dipl.-Ing. 1994, Dr. techn. 1999). Seine Forschungsinteressen umfassen Peer-to-Peer-Systeme, Verteiltes Informationsmanagement, Selbstorganisierende Systeme, Semantic Web und Sicherheit in Verteilten Systemen. Manfred Hauswirth ist Autor zahlreicher Beiträge für internationale Konferenzen, Journale und Bücher und ist Mitautor des Buches „Software-Architekturen für Verteilte Systeme“, Springer, 2003. Weitere Informationen sind unter http://lsirpeople.epfl.ch/hauswirth/ zu finden.

Prof. Dr. Schahram Dustdar arbeitet derzeit in der Distributed Systems Group, Technischen Universität Wien. Er habilitierte sich 2002 in Angewandter Informatik an der TU Wien. Seit 1999 arbeitet er auch als Mitgründer und Chief-Scientist bei der Caramba Labs Software AG (CarambaLabs.com), einem Venture Capital co-finanzierten Softwarehaus im Bereich Workflow und Groupware. Er ist Mitautor des Buches „Software-Architekturen für Verteilte Systeme“, Springer, 2003. Weitere Informationen sind unter http://www.infosys.tuwien.ac.at/Staff/sd/ zu finden.