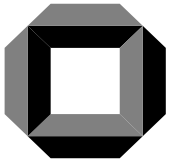

Universität Karlsruhe
Fakultät für Informatik
76128 Karlsruhe

Architektur vernetzter Systeme

Seminar Sommersemester 1997 & Wintersemester 1997/98

Herausgeber:
Arnd G. Grosse
Jörn Hartroth

Universität Karlsruhe
Institut für Telematik

Interner Bericht 02/98

ISSN 1432-7864

Zusammenfassung

Der vorliegende interne Bericht enthält die Beiträge von Studenten zum Seminar „Architektur vernetzter Systeme“, das im Sommersemester 1997 und im Wintersemester 1997/98 am Institut für Telematik der Universität Karlsruhe stattgefunden hat. Themen waren dabei die Performanzuntersuchung bei Vermittlungsinstanzen, alternative Kommunikationsparadigmen wie mobile Agenten und Push-Techniken als auch die Betrachtung verteilter Anwendungen im WWW oder die Optimierung des WWW durch den Harvest-Ansatz.

Abstract

This Technical Report includes student papers produced within assignments for the seminar “Architektur vernetzter Systeme”. The seminar was held at the Institute of Telematics of the University of Karlsruhe in summer 1997 and winter 1997/98. The topics for discussion included mechanisms like trader performance, alternative communication concepts like mobile agents and push technologies. Additionally, new distributed applications arise based on the WWW or being optimized by using distributed caches like Harvest.

Inhaltsverzeichnis

Vorwort

Neben der Fortentwicklung der Informationstechnik hat im speziellen die Entwicklung der Kommunikationstechnik einen rasanten Fortschritt erfahren. Hierdurch sind leistungsfähige Netze hoher Bandbreite und damit neue Anwendungsszenarien möglich geworden. Insbesondere der Bereich der verteilten Systemdienste und neuer, auf der leichten Verfügbarkeit von Kommunikationsdiensten aufsetzender, Programmierparadigmen, wie dieses z.B. die Entwicklung im Agentenbereich darstellt, bedürfen noch einer eingehenden Forschung, um eine nahtlose Integration auch von mobilen Teilnehmern auf diesem Gebiet zu ermöglichen.

Der hier vorliegende Seminarband stellt Konzepte und Mechanismen vor, die in dem genannten Spannungsfeld zwischen dem Entwurf von verteilten Anwendungen und darunterliegenden Kommunikationsdiensten angesiedelt sind. Er enthält eine Zusammenfassung zweier Seminare, die im Sommersemester 1997 und im Wintersemester 1997/98 am Institut für Telematik abgehalten wurden und damit eine Veranstaltungsreihe des Instituts fortgesetzt haben, die erstmals im Sommersemester 1992 unter dem Thema "Mechanismen für fehlertolerante verteilte Anwendungen" stattgefunden hat. Der damalige Schwerpunkt lag dabei auf dem speziellen Themengebiet der Fehlertoleranz. Dieses setzte sich über die Umbenennung der Reihe in "Daten in verteilten Systemen" fort, welche den Datenverteilungsaspekt in Systemen näher untersuchte. Durch die neuerliche Umbenennung des Seminars in "Architektur vernetzter Systeme" soll der tiefgreifenden Veränderung noch besser Rechnung getragen werden und damit den umfassenden Integrationsgedanken in vernetzen Systemen verstärken. Hierbei liegt der Schwerpunkt auf der neuartigen Gestaltung von vernetzten bzw. verteilten Systemen und zeigt mit den folgenden Beiträgen die Verlagerung hin zu speziellen Fragestellungen verteilter Systeme.

Die Zunahme an Diensten innerhalb vernetzter Systeme stellt den Benutzer der Systeme immer stärker vor die schwierige Frage, welche Dienste es innerhalb des Netzes gibt und wie er aus dieser Vielzahl den geeigneten auswählen kann. Eine Lösung dieser Problematik liegt in der Verwendung von Vermittlungsdiensten, sog. Tradern. Ein einzelner Trader ist dabei mit der Durchführung der Vermittlung oft überlastet, weshalb hier mehrere Trader gemeinsam den erwünschten Dienst erbringen können. Eine Diskussion, wie dabei der günstigste Trader ausgwählt werden kann, wird im ersten Beitrag beschrieben.

Mobile Agenten sind Programme, die von einem Rechner zum nächsten wandern können. Vorteile Mobiler Agentensysteme gegenüber herkömmlichen Client-Server Architekturen sind die geringere Kommunikation durch die Filterung relevanter Information am besuchten Rechner, die höhere Sicherheit durch die direkte Verbindung des Agenten zum besuchten Rechner und die bessere Bedienbarkeit für den Anwender, der sich nicht um komplexe Netzwerk-Syntax kümmern muß. Ein Überblick und einen Vergleich von unterschiedklichen Agentensystemen liefert der zweite Beitrag.

Der Zugriff in verteilten Systemen und insbesondere im Word Wide Web (WWW) ist hauptäschlich pull-orientiert, d.h. der Client fragt einen Server nach der gewünschten Information und der Server liefert dem Clienten daraufhin diese Information. Ein anderes Zugriffsparadigma besteht in der Verwendung von Push-Techniken, bei welchen der Server bei Informationsänderungen dem Client periodisch diese Informationen zusendet.

Einen Überlick über existierende Push-Techniken im WWW leistet der dritte Vortrag.

Der vierte Beitrag greift diese Problematik der auf und befaßt sich mit der Optimierung von Informationszugriffen im WWW unter Verwendung einer verteilten Caching-Struktur. Hierzu wird das verteilte System Harvest eingehender betrachtet und darin enthaltene Komponenten wie verteilte Caches, Informationsverwaltungs- und Sammelinstanzen näher vorgestellt.

Nachdem das World Wide Web die beherrschende Rolle für die Bereitstellung von Informationen in Weitverkehrsnetzwerken eingenommen hat, ist nachfolgend ein Trend zu beobachten, die dabei verwendeten technischen Mechanismen auch für komplexere verteilte Anwendungen zu verwenden. Der Web-Browser wird dabei als nahezu global verfügbare Plattforn für Informationsinhalte aller Art eingesetzt. Neu ist die Ersetzung des in der Originalform weitgehend statischen Datenbestands auf Seiten der Informationsanbieter durch Kopplung mit Middleware-Architekturen. In dem fünften Seminarbeitrag soll ein Überblick ueber den aktuellen Stand der Web-Middleware-Integration gegeben werden.

Bevor nun die einzelnen Ausarbeitungen der Seminarbeiträge präsentiert werden, möchten wir allen beteiligten Studenten für ihre engagierte Mitarbeit danken, ohne die weder der Erfolg der Seminare noch die Anfertigung des vorliegenden Berichts möglich gewesen wäre. Hierzu haben auch die nach den einzelnen Vorträgen stattfindenden Diskussionen maßgeblich beigetragen.

Karlsruhe, im Februar 1998

Arnd G. Grosse Jörn Hartroth

Trader Performanz

Holger Schönecker

Kurzfassung

Dieser Artikel betrachtet das Zwei- und das Drei-Parteien-Trading unter dem Aspekt der Leistungsfähigkeit in Bezug auf die Auswahl des „besten“ Servers. Mit Hilfe eines generischen Modells und Simulation wird die Antwortzeit verschiedener Auswahlstrategien unter wechselnden praktischen Bedingungen bestimmt. Insbesondere werden Strategien simuliert, die dynamische Statusinformation oder nur statische Daten verwenden, zufallsgesteuerte Strategien und Mischformen davon.

1 Trader und Trading

1.1 Die Konzepte des Trading

Trading ist in verteilten Systemen ein relativ neuartiges Konzept. Trading ist ein Dienst, der für eine Dienstanfrage einen geeigneten Dienstanbieter aussucht. Dieser Dienst kann als eine Weiterentwicklung des Name-Service betrachtet werden. Der Artikel [Kel93] von Ludwig Keller bietet hierzu einen Überblick.

Beim Trading betrachtet man selbständige, autonome Komponenten in einer Client-Server Umgebung. Der Prozeß des Tradings besteht nun darin, für Dienstanfragen geeignete Dienstleister zu finden.

Die Auswahl des Servers erfolgt in der Regel zunächst nach statischen Attributen der Server, die vom Trader verwaltet werden. Dies sind Parameter, die sich auf längere Sicht nur selten ändern. Bei einem Druckerserver wären das z. B. die verfügbaren Papierformate, Druckersprachen etc. Werden nach dieser Auswahl mehrere geeignete Server gefunden, kann der Trader zusätzlich dynamische Statusinformationen nutzen, um den „besten“ Dienstanbieter auszuwählen. Dynamische Statusinformationen sind z. B. aktuelle Auslastung, Anzahl der wartenden Aufträge etc.

Der Trader bietet gegenüber den Dienstnehmern *Transparenz*, das bedeutet, der Dienstnehmer kann von allen Parametern des Servers abstrahieren, die für diese Dienstleistung ohne Belang sind. Um all diese Parameter (z. B. Bezeichnung, Adresse, Status) kümmert sich der Trader.

Weiterhin bietet der Einsatz eines Traders *Flexibilität*. Änderung von Eigenschaften der Servern, Einführung von neuen Diensten werden vor dem Client verborgen. Der Trader stellt für ihn auch weiterhin die geeigneten Verbindungen über unveränderte Schnittstellen her.

Ein weiterer Aspekt beim Einsatz von Tradern ist die *Fehlertoleranz*. Aktuelle Statusinformation ermöglicht es dem Trader beim Ausfall von Servern zu reagieren und die Anfragen an andere geeignete Server weiterzuleiten.

Der wohl wichtigste Aspekt bei der Betrachtung der Performanz beim Einsatz von Tradern ist die *Optimierung*, die dieser bei der Auswahl der Server vornehmen kann. Eine geschickte Nutzung von dynamischen Informationen ermöglicht es dem Trader, den besten Dienstanbieter auszuwählen, und die den Clienten eine kürzere Antwortzeit bietet. Das Ausbalancieren der Last zwischen den Servern ermöglicht eine globale Optimierung und eine gesteigerte Leistung des gesamten Systems.

1.2 Grundsätzliche Probleme

Da ein Trader nur eine eingeschränkte Sicht auf die an sich gekapselten Funktionen der Server hat, ist es nicht immer möglich die nötigen statischen Informationen über die Leistung der angebotenen Dienste zu bekommen.

Sollen bei der Auswahl eines Servers auch dynamische Informationen sinnvoll berücksichtigt werden, so müssen diese eine gewisse Aktualität habe. Das bedeutet, daß diese Informationen mit möglichst kurzer Verzögerung dem Trader zugänglich gemacht werden müssen. In der Regel schicken die Server selbständig Daten über ihren momentanen Status an die Trader. Damit der Trader aktuelle Werte zur Verfügung hat, sollte dies in relativ kurzen Zeitabständen erfolgen. Dies hat jedoch eine erhöhte Belastung der Server, der Trader und vor allem des Netzes zur Folge. Es ist also nötig, einen Kompromiß zwischen der Aktualität der Information und der Belastung der beteiligten Komponenten zu finden. Das Problem der veralteten Statusinformation wird bei der Betrachtung der Performanz noch genauer untersucht.

Weitere Verzögerungen, welche die Genauigkeit der Statusinformation beeinträchtigen, können auch noch während der Vermittlung des Dienstes durch den Trader entstehen. Diese und andere Faktoren werden im folgenden vernachlässigt.

1.3 Auswahlstrategien

Stehen dem Trader nach der ersten Auswahl gemäß den statischen Kriterien mehrere geeignete Server zur Verfügung, kann er verschiedene Auswahlstrategien verwenden, um den besten Server zu finden (siehe auch [Kel93]. Die wichtigsten hiervon, die auch später auf ihre Leistungsfähigkeit untersucht werden, sollen im folgenden vorgestellt werden:

- *First Choice-Strategie:* Der erste passende Server wird vermittelt.
- *Random Choice-Strategie:* Aus allen passenden Servern wird einer zufällig ausgewählt. Hierbei können verschiedene Wahrscheinlichkeiten verwendet werden, welche die unterschiedliche Leistungsfähigkeit der Server berücksichtigen.
- *Cyclic Choice-Stategie:* Bei gleichwertigen Servern werden diese in eine Liste eingeordnet und zyklisch an die Anfragen zugewiesen.
- *Best Choice-Strategie:* Der Trader wählt denjenigen Server aus, der gemäß den dynamischen Eigenschaften (z. B. momentane Auslastung, Länge der Warteschlange etc.) die beste Wahl darstellt. Bei dieser Strategie müssen diese dynamischen Daten dem Trader zur Verfügung stehen.

– *Probing-Strategie:* Bei mehreren geeigneten Dienstangeboten wird zunächst eine zufällig bestimmte Untermenge betrachtet. Aus dieser Untermenge wird dann ein Server nach der Best Choice-Strategie ausgewählt.

Um die Selektion der Server nach diesen Kriterien durchführen zu können, muß ein Trader sowohl mit statischen als auch mit dynamischen Informationen über Server umgehen können. Der Trader stellt Funktionen bereit, Server sowie statische Informationen über deren Eigenschaften hinzuzufügen, zu verändern und wieder zu löschen. Zur Selektion nach dynamischen Gesichtspunkten muß der Trader darüber hinaus regelmäßig Informationen über den aktuellen Status von Servern entgegennehmen oder auch selbständig anfordern.

1.4 Design des Traders

Neben der bloßen Auswahl eines Servers (interface trading) übernimmt der Trader oft auch die Vermittlung der Dienstleistung zwischen Client und Server (service trading). Hierbei prüft der Trader, ob die entsprechenden Dienste verfügbar sind. Im weiteren wird hier ausschließlich die zweite, leistungsfähigere und kompliziertere Variante behandelt.

Trading muß nicht unbedingt als eigene Instanz realisiert sein. Trading, d.h. die Auswahl eines geeigneten Servers kann auch direkt beim Benutzer der Dienste stattfinden. Hierbei ist zu beachten, daß diese Benutzer bei der Auswahl der Server vollkommen autonom agieren, das heißt, sie haben keinerlei Informationen über weitere Benutzer, die eventuell die selben Dienste in Anspruch nehmen. Bei dieser Form des Trading Dienstes spricht man von *Zwei-Parteien-Trading.*

Eine weitere Form ist das *Three Party Trading.* Hier übernimmt der Trader die Vermittlung der Dienstleistung für mehrere Benutzer. Er kann also bei der Auswahl der Server Informationen über bereits von ihm getätigte Vermittlungen nutzen. Es ist jedoch zu beachten, daß auch mehrere Trader im Spiel sein können, die unabhängig voneinander arbeiten und keine Informationen untereinander austauschen.

Bei der Betrachtung des Designs eines Trader unter dem Aspekt der Performanz sollte man zwischen zwei Blickwinkeln unterscheiden. Aus der *Sicht des Benutzers* von Trading Diensten sind die folgenden beiden Kriterien relevant:

– Sollte der Prozeß des Trading an einen separaten Trader delegiert werden oder verwendet man besser Zwei-Parteien-Trading?

– Sind bestimmte Auswahlstrategien oder Formen des Tradings in der Performanz besser als andere?

Aus der Sicht des *Entwerfers von Trading-Diensten* sind die folgenden beiden Punkte wichtig:

– Welche Strategie schneidet unter den gegebenen Umständen (z. B. ungenaue oder veraltete Statusinformationen) am besten ab?

– Wie verhält sich ein System von konkurrierenden Tradern und Strategien?

Um diese Fragen zu untersuchen, wird im folgenden Kapitel zunächst ein Modell vorgestellt, anhand dessen die Performanz verschiedener Strategien und Konstellationen betrachtet wird.

2 Grundmodell für die Performanzbewertung

2.1 Problemstellung

Als Modell zur Betrachtung der Trader Performanz wird das Problem der *„richtigen Warteschlange“* (*joining the right queue*) betrachtet. Jeder Server verwaltet für sich eine eigene Schlange mit wartenden Aufträgen. Der Trader, der neue Aufträge entgegennimmt, muß sich nun unmittelbar für eine Warteschlange von den geeigneten Server entscheiden und kann diese Entscheidung auch nicht wieder rückgängig machen. Das Problem besteht nun darin, diejenige Schlange auszuwählen, die den Benutzern die kürzeste Antwortzeit bieten kann. Das vorgestellte Modell wird auch in mehreren Schriften von A. Wolisz und V. Tschammer benutzt ([WT91], [TWW92], [WT93]).

2.2 Verwendete Größen

Als Maß für die Performanz wird die Dauer der *Antwortzeit*, nachfolgend mit D bezeichnet, betrachtet. Die Antwortzeit setzt sich Zusammen aus der Wartezeit in der Schlange und der Bearbeitungszeit des Auftrages. Alle sonstigen Verzögerungen (z. B. Berechnungszeiten des Traders, Netzwerktransporte, Anstellen an der Schlange etc.) sollen im folgenden vernachlässigt werden.

Die *Ankunftsrate* sei ein Poisson Strom und mit λ bezeichnet.

Die *Bearbeitungszeit* der Aufträge sei entweder konstant oder durch die Verteilungsfunktion $B(x)$ gegeben. In den späteren Betrachtungen der Performanz wird die Bearbeitungszeit entweder konstant oder exponential verteilt sein. Diese Verteilungen sind die beiden Grenzfälle. Die in der Praxis vorkommenden Bearbeitungszeiten haben in der Regel eine Varianz größer als null und kleiner als die der Exponentialverteilung.

Die *Anzahl der Server* wird mit N bezeichnet. Als praktisch interessante Werte werden $N = 5, 10, 20$ betrachtet.

Die *Anzahl der Trader* wird mit R bezeichnet.

Bei mehreren Tradern werden mit $\Pi_1, \Pi_2, ..., \Pi_R$, die *Wahrscheinlichkeiten der Nutzung* des entsprechenden Traders. Dabei ist $\Pi_1 + \Pi_2 + ... + \Pi_R = 1$.

Verwendet wird überdies die *Servernutzungsrate* ρ. Bei der Simulation der Performanz werden die in der Praxis relevante Werte $\rho = 0,5$ und $\rho = 0,7$ untersucht. Es ergibt sich für die Ankunftrate λ bei mehreren Servern $\lambda = N * \rho$.

Dynamische Information über die Auslastung der Server wird in regelmäßigen Zeitabständen Δ zu den Tradern geschickt. Diese *Delai-Zeit* Δ wird als Vielfaches der durchschnittlichen Bearbeitungszeit ausgedrückt.

Die durchschnittliche Bearbeitungszeit sei eins, alle anderen Größen wie z. B. λ und Δ werden als Vielfaches davon ausgedrückt.

2.3 Grenzen der Optimierung

Die wohl einfachste Regel, die ankommenden Aufträge gleichmäßig auf geeignete Server zu verteilen, ist eine probabilistische Zuordnung. Diese Methode ist einfach zu Realisieren und benötigt keine zusätzlichen, dynamischen Informationen über den Zustand der Server. Diese Methode ist all jenen vorzuziehen, die dynamische Statusinformation benötigen, sofern diese keine signifikannt bessere Leistung bringen. Sie bildet also die Obergrenze (hier schlechtester Wert) für die praktisch erreichbare Antwortwortzeit. Bei der probabilistischen Zuordnung werden die ankommenden Aufträge mit den Wahrscheinlichkeiten p_i, $i = 1, 2, ..., N$ auf die N Server verteilt. Bei gleichartigen Servern ist $p_i = 1/N$. Dieser Fall entspricht also N einzelnen Warteschlangen mit den jeweils identischen Ankunftsraten λ/N. Unglücklicherweise ist es bei dieser Konstellation möglich, daß ein Auftrag in einer Schlange wartet, während ein anderer Server frei ist. Wie in [LM82] berechnet, übersteigt die Wahrscheinlichkeit eines solchen Ereignisses 80% bei Nutzungsraten von 0,4 – 0,8 und N größer als 10 und strebt gegen eins, wenn man die Anzahl der Server vergrößert.

Dieses Ergebnis berechtigt die Suche nach besseren Methoden zur Auswahl von Servern, die auch dynamische Daten über die Auslastung der Server berücksichtigen.

Die beste Strategie für die Zuordnung der Aufträge besteht zweifelsohne darin, den Server auszuwählen, der im Augenblick gerade die geringste Auslastung hat. Dieser Idealfall, der durch keine andere der hier betrachteten Zuordnungsregeln zu übertreffen ist, entspricht der Multiserver Schlange. Hier warten alle ankommenden Aufträge an einer Warteschlange und werden von dort den freien Servern zugeteilt. In diesem Fall kann es nicht mehr vorkommen, daß Aufträge warten, während ein Server unbeschäftigt ist. Aus verschiedenen praktischen Gründe ist die Multiserver Schlange nicht immer möglich. Dieser Fall ist bei der Betrachtung der Performanz nützlich, denn er stellt die Untergrenze für die Antwortzeit dar, sozusagen den Grenzwert für die besten Methoden.

Oft sind die exakten Informationen über die aktuelle Auslastung der Server nicht verfügbar, und es muß mit weniger genauen gearbeitet werden. Dies hat, wie im nächsten Kapitel gezeigt wird, Auswirkungen auf die Leistungsfähigkeit des Systems. Ist die Information über die Belastung eines Servers nicht bekannt, so kann auch alternativ mit der Anzahl der wartenden Aufträgen als Größe gearbeitet werden. Wie in [WT91] erwähnt wäre bei einem homogenen System die beste Strategie, die Schlange mit der geringsten Anzahl wartender Aufträge auszuwählen. In heterogenen Systemen ist das Problem komplexer. Hier kann es unter globalen Gesichtspunkten und bei geringer Auslastung günstiger sein, mehr Aufträge einem schnelleren Server zuzuordnen, auch wenn andere unbeschäftigt sind. Bei hoher Auslastung sollten demgegenüber auch langsamere Server genutzt werden.

In den beschriebenen Fällen wurde bisher immer davon ausgegangen, daß die gewünschte Information (auch ungenaue) dem Trader *sofort* zur Verfügung steht. Dies ist jedoch nicht immer möglich. Wenn der Trader vor der Zuordnung eines Auftrages die aktuelle Statusinformation von allen relevanten Servern anfordert, ist mit einer Verzögerung zu rechnen, die nicht mehr zu vernachlässigen ist. Zusätzlich entsteht durch diese Anfragen eine erhebliche Belastung des Netzwerkes.

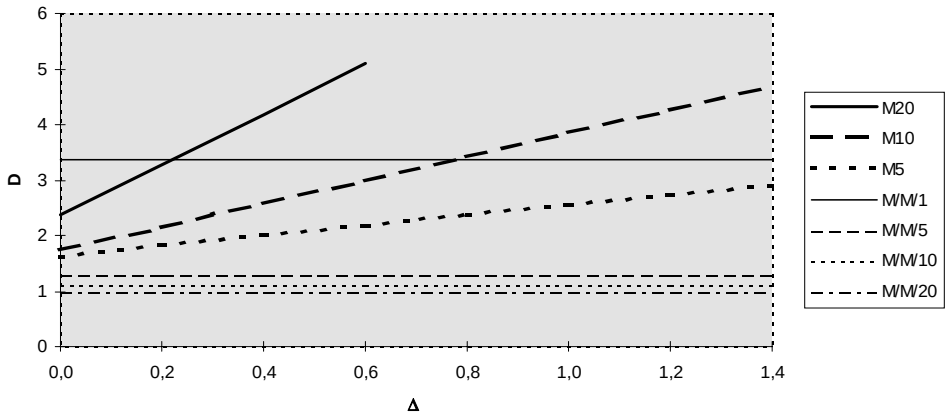

Abbildung1. Einfluß von veralteter Information beim Zwei-Parteien-Trading

Wenn in den weiteren Untersuchungen dynamische Staus Information genutzt wird, gehen wir davon aus, das diese in regelmäßigen Abständen Δ an den Trader geschickt wird. Der Trader arbeitet also schlimmstenfalls mit Daten, die um Δ Zeiteinheiten veraltet sind. Die dadurch bedingten Auswirkungen werden in den nächsten Kapiteln eingehender untersucht.

Wie bereits weiter oben erwähnt bilden konstante und exponentiell verteilte Bearbeitungszeiten von Aufträgen ebenfalls solche Grenzwerte.

Im nächsten Kapitel werden nun einige Ergebnisse von Simulationen vorgestellt, die von A. Wolisz und V. Tschammer in [WT93] und [WT91] vorgestellt wurden. Aus den vielen möglichen Kombinationen von Parametern werden hier nur Untersuchungen mit besonders interessanten Ergebnissen dargestellt.

3 Zwei-Parteien-Trading

3.1 Statische und Dynamische Serverauswahl

Wie bereits im vorhergehenden Kapitel erwähnt ist die optimale Strategie, einen Server auszuwählen, denjenigen mit der geringsten Arbeitslast zu nehmen. Nachfolgend soll nun betrachtet werden, ob diese Strategie auch dann sinnvoll ist, wenn der Trader mit veralteter Information arbeiten muß (wie in Abschnitt 2.2 definiert).

Ergebnisse einer Simulation des Einflusses von Δ mit $\rho = 0,7$ auf die Antwortzeit D sind in Abbildung 1 zu sehen. Der Buchstabe M steht für exponentiell verteilte Servicezeiten, die dahinter angegebene Zahl N für die Anzahl der betrachteten Server. Die Ergebnisse werden auch mit dem Idealfalle, der Multi Server Schlange (bezeichnet mit $M/M/N$) verglichen. Der mit $M/M/1$ bezeichnete Fall entspricht dabei gleichmäßig beanspruchten Servern, also wie in Abschnitt 2.3 bereits erwähnt, der einfach zu realisierenden probabilistischen Verteilung, die keine dynamische Information benötigt.

Wie in dem Diagramm zu erkennen ist, steigt die zu erwartende Antwortzeit mit wachsender Verzögerung Δ. Die Kurve steigt fast linear an. Für größere Verzögerungswerte Δ ist es bereits besser die probabilistische Strategie $M/M/1$ zu verwenden. Das Verhalten

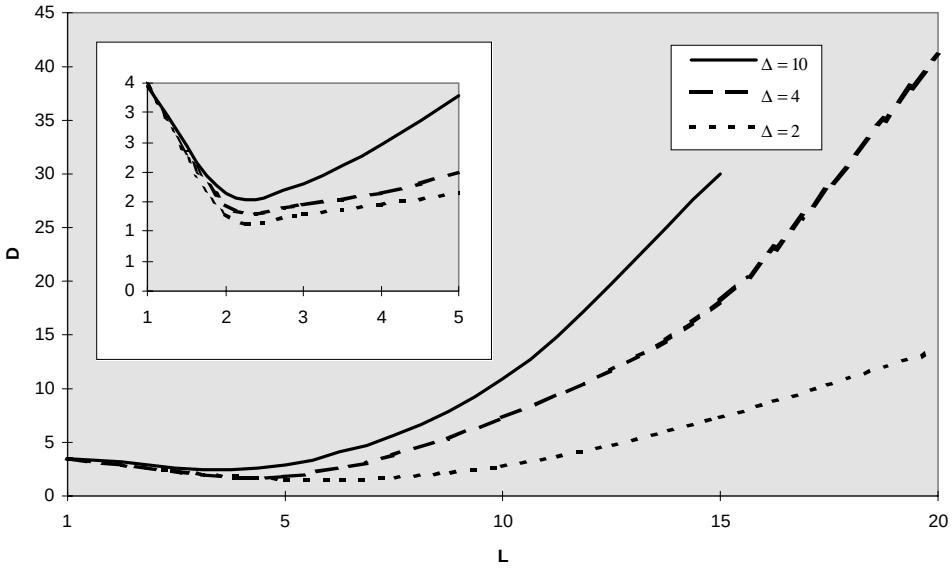

Abbildung2. Probing-Strategie: Einfluß der Größe der Untermenge L

der Strategie bei wachsendem Δ ist nicht überraschend. Dem Server, der zu einem bestimmten Zeitpunkt die geringste Arbeitslast hat, werden während der gesamten Zeit Δ alle Aufträge zugewiesen, was diesen leicht überlasten kann. Im nächsten Zyklus wird ein anderer Server mit geringster Auslastung gefunden und mit Aufträgen überhäuft. Der Zustand der Server pendelt bei großem Δ also zwischen Über- und Unterlast. Lediglich die Wahl eines sehr kleinen Wertes für Δ führt zu einer gedämpften Schwingung, große Werte Δ führen zu ungedämpften Schwingungen und somit zu der ungünstigen Über- und Unterauslastung. Für sehr große Δ entstehen Schwingungen mit doppelter Periode, später auch chaotische Schwingungen.

Genauso kann die Abhängigkeit von der Anzahl der Server N erklärt werden. Je mehr Server an dem Prozeß beteiligt sind desto höher ist die Ankunftrate λ beim Trader (siehe Abschnitt 2.2 $\lambda = N * \rho$) und desto mehr Aufträge werden während der Zeit Δ dem überlasteten Server zugewiesen.

Zusammenfassend kann man beobachten, daß diese Strategie bereits für kleine Werte von Δ (z .B. 0,8 bei 10 Servern oder sogar 0,4 bei 20 Servern) ineffizient wird und von der einfachen probabilistischen Strategie übertroffen wird.

3.2 Randomisierung

Bereits im vorigen Abschnitt 3.1 wurde gezeigt, daß die probabilistische Auswahl von Server unter bestimmten Umständen eine gute Wahl ist. Untersucht wurde bisher die rein zufällige Auswahl von Servern und die Auswahl des Servers mit der geringsten Arbeitslast unter Verwendung von etwas veralteter Statusinformation. In diesem Abschnitt soll nun untersucht werden, wie sich eine Kombination aus der Verwendung von veralteter Information und einer zufallsgesteuerten Strategie auf die Leistungsfähigkeit auswirkt.

Um einen fließenden Übergang zwischen diesen beiden Strategien zu schaffen, wird nach folgender Regel vorgegangen:

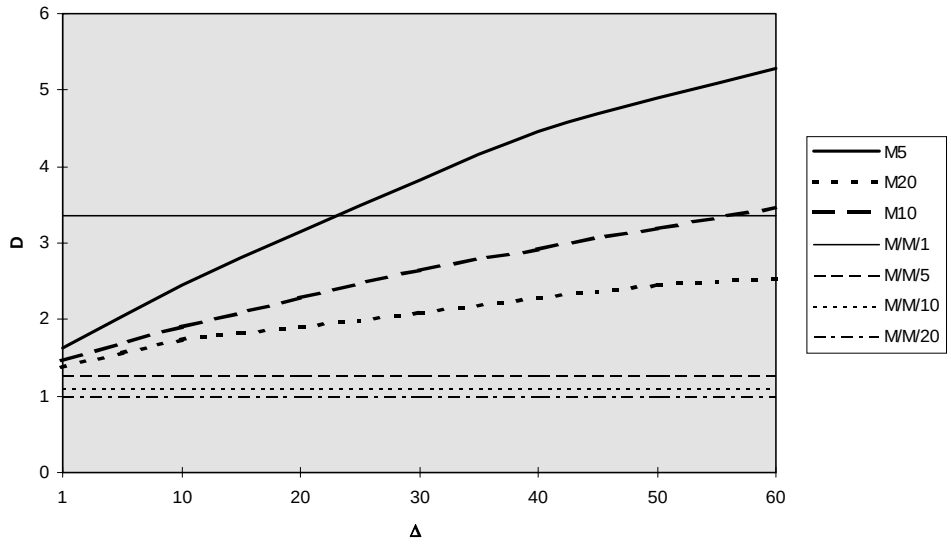

Abbildung3. Probing-Strategie: Einfluß von veralteter Information

- Zunächst werden aus den N verfügbaren Servern etwa $L < N$ zufällig ausgewählt.
- Aus diesen zufällig bestimmten L Servern wird unter der Zuhilfenahme von veralteter Statusinformation derjenige Server mit der geringsten Auslastung gewählt (entsprechend der Strategie in Abschnitt 3.1).

Abbildung 2 zeigt nun die zu erwartende Antwortzeit D in Abhängigkeit von L für verschiedene Verzögerungszeiten Δ. Die Anzahl der Server N beträgt 20, die Auslastung $\rho = 20$. In diesem Diagramm entspricht nun $L = 1$ der rein zufallsgesteuerten Strategie, $L = N$ entspricht der ausschließlichen Verwendung von veralteter Statusinformation.

Die Abbildung zeigt eindeutig die Überlegenheit der Probing-Strategie gegenüber den beiden bisher betrachteten Verfahren. Die Wahl von $L = 2$ scheint für die meisten Bedingungen ideal. Lediglich für extrem kleine Werte von Δ kann L gößer sein, bzw. für große Werte von Δ ist die probabilistische Strategie ($L = 1$) besser. Wenn unverzögerte Information zur Verfügung steht, ist natürlich die Wahl des am geringsten belasteten Servers die geeignete Strategie ($L = N$). Abbildung 3 zeigt nun die Leistungsfähigkeit der Probing-Strategie über einen sehr weiten Bereich von Δ. Wie im direkten Vergleich mit Abbildung 1 zu sehen ist, reagiert diese Strategie weit weniger empfindlich auf höhere Verzögerungszeiten Δ als die ausschließliche Verwendung von Statusinformation ohne Zufallskomponente. Auch für hohe Werte von Δ ist diese Strategie noch leistungsfähiger als die rein probabilistische Methode (im Diagramm mit $M/M/1$ bezeichnet). Als grobe Annäherung kann man sagen, daß die Probing-Strategie für $\Delta \leq N$ wesentlich besser als die zufällige Serverauswahl ist. Im Gegensatz zur in Abschnitt 3.1 beschriebenen dynamischen Auswahl ist die Antwortzeit geringer, wenn mehr Server zum Einsatz kommen.

Eine andere Möglichkeit, den Zufall bei der Auswahl der Server mit ins Spiel zu bringen, wäre die Verwendung von ungenauer Statusinformation, durch das Hinzufügen von zufällig verteiltem Rauschen. Auch diese Methode führt zu einem besseren Verhalten des Systems, regelmäßigeren oder sogar gedämpften Schwingungen.

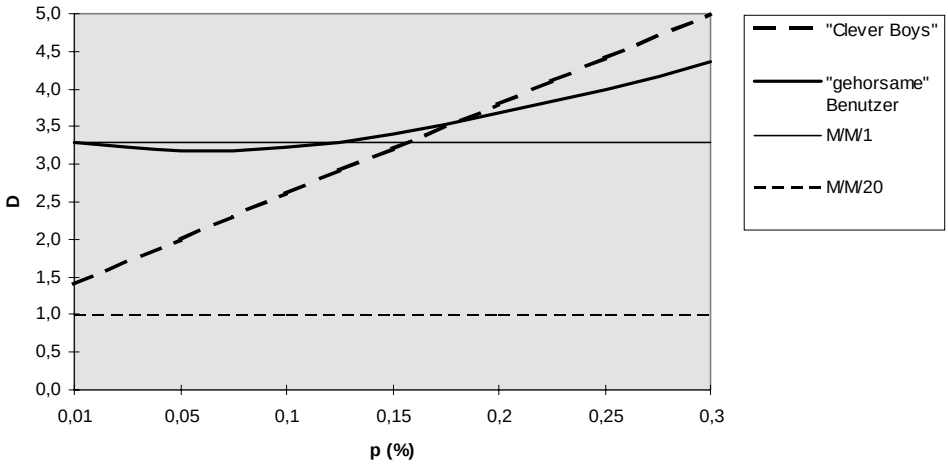

Abbildung4. Einfluß von Autonomen Benutzern

Zusammenfassend ist zu bemerken, daß im Fall von verzögerten Informationen das Arbeiten mit einer zufällig vorbestimmten Menge oder der Einsatz von ungenauen Informationen zu einer erheblichen Steigerung der Performanz führt.

3.3 Einfluß von autonomen Benutzern

Bisher wurden alle Strategien unter dem Gesichtspunkt der *sozialen Optimierung* betrachtet. Das bedeutet, der Trading Mechanismus versucht, die beste Antwortzeit für alle Anfragen zu erreichen. Aus der Sicht eines individuellen Benutzers ist dies nicht immer die optimale Lösung. Es ist in der Tat so, daß ein einzelner Benutzer eine für sich selbst bessere Regel zur Auswahl eines Servers finden könnte. Dies soll nun an einem einfachen Beispiel demonstriert werden.

Betrachten wir ein System, bei dem die Anfragen zufällig auf die zur Verfügung stehenden Server verteilt werden, um insgesamt eine optimale Leistung zu erreichen (soziale Optimierung). Trotzdem wird im Zeitabstand Δ die Statusinformation der Server abgefragt und steht den Benutzern zur Verfügung. Ein gewisser Anteil P der Benutzer, genannt „Clever Boys“, hält sich jedoch nicht an die Vorgabe und wählt seinen Server unter Zuhilfenahme der Statusinformation nach der Regel der geringsten Arbeitslast aus. Der Rest, die „gehorsamen Anwender“ wählen ihre Server weiterhin probabilistisch aus.

Abbildung 4 zeigt nun die Ergebnisse einer solchen Simulation. Wie man leicht erkennen kann, erhalten die „Clever Boys“ sehr viel bessere Antwortzeiten als die „gehorsamen“ Anwender, solange ihr Anteil P relativ gering ist. Man kann feststellen, daß sogar die „Gehorsamen“ von den „Cleveren“ etwas profitieren. Steigt die Rate P allerdings an, leiden beide Gruppen unter einer höheren Antwortzeit, als im Falle einer rein zufälligen Auswahl der Server hätten. Man kann nun eine Rate P^* so bestimmen, daß für alle $P < P^*$ die „cleveren“ Anwender im Schnitt eine bessere Antwortzeit erhalten als die „gehorsamen“. Auch andere Experimente, die hier nicht vorgestellt werden, zeigen, daß die „Clever Boys“ um so mehr von ihrer Strategie profitieren, je kleiner ihr Anteil P ist. Außerdem ist der Grenzwert P^* um so höher, je kleiner das Zeitintervall Δ gewählt wird. Unter praktischen Bedingungen ist der Wert P im Allgemeinen sehr klein und liegt

unter einem Prozent.

Aus den Ergebnissen dieser Simulation kann man zwei praktische Schlüsse ziehen. Zum einen kann schon ein relativ geringer Anteil von Komponenten, der sich nicht an die vorgegebenen Regel hält, die Leistungsfähigkeit des Systems beträchtlich verringern. Leider ist es in einem offenen System nicht möglich, die Anwender bezüglich ihrer Wahl der Server zu beeinflussen.

Eine Möglichkeit wäre jedoch, die Information, die den Benutzern zur Verfügung steht einzuschränken. In diesem Fall werden sie gezwungen, die Entscheidung für einen geeigneten Server an den Trading Dienst zu delegieren oder sie wären durch begrenzte Informationen in ihrer Auswahl eingeschränkt. Besonders hilfreich ist das Zurückhalten von Statusinformationen über die Auslastung der Server.

Dieses Phänomen kann aber in einer positiven Weise auch dazu genutzt werden, Anfragen mit zwei unterschiedlichen Prioritäten zu behandeln. Solange der Anteil der Anfragen mit hoher Priorität nicht den Wert von P^* übersteigt, können diese mit der Strategie der geringsten Serverlast zugeordnet werden. Bei den Anfragen mit geringer Priorität werden die Server zufällig ausgewählt.

Eine weitere Form, durch „ungehorsames“ Verhalten die Antwortzeit, sogar die des gesamten Systems zu verbessern besteht darin, die Statusinformation der Server zu analysieren und die Frequenz der Schwingungen vorherzusagen. Solange der Anteil dieser „Analysierer“ recht gering bleibt, profitiert das ganze System davon, dadurch daß die Amplitude der Schwingungen gedämpft werden.

4 Drei-Parteien-Trading

4.1 Statische und Dynamische Serverauswahl

Die folgenden Abschnitte behandeln nun das Drei-Parteien-Trading, wie es bereits in Kapitel 1.4 beschrieben wurde. Bei der Betrachtung der Performanz wird wie bisher von der Zeit, die für den Trading-Prozeß benötigt wird, sowie von der Übertragungszeit zum Server abgesehen. Der wichtigste Unterschied zum Zwei-Parteien- Trading besteht darin, daß der Trader Anforderungen für mehrere Benutzer behandelt und somit zusätzlich zur Server-Statusinformation noch über Daten der von ihm bereits zugeordneten Anfragen verfügt.

Die Bedeutung dieses Unterschieds zeigt sich selbst bei Strategien, die keine dynamische Statusinformationen verwenden. Betrachtet man beispielsweise die zufallsgesteuerte Auswahl der Server beim Zwei-Parteien-Trading (siehe Abschnitt 3.1) so ergibt sich keine Abhängigkeit der Antwortzeit von der Anzahl der Trader. Beim Drei-Parteien-Trading dagegen, bei dem Informationen über bisherige Zuordnungen zur Verfügung stehen, besteht in der Regel eine solche Abhängigkeit.

Der Trader darf Informationen über bisherige Zuordnungen nutzen und kann sich nun der in Abschnitt 1.3 beschriebenen *zyklischen Zuordnung* bedienen. Die folgende Simulation wurde für eine unterschiedliche Anzahl von Servern und für die zwei verschieden Auslastungen $\rho = 0,5$ und $\rho = 0,7$ durchgeführt.

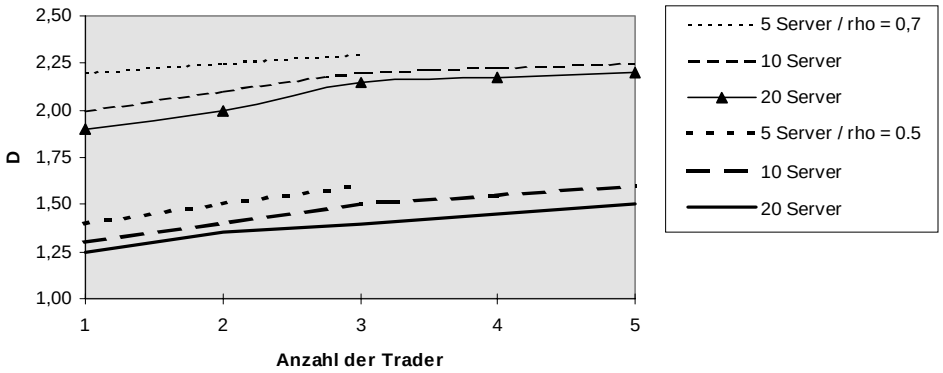

Abbildung5. Zyklische Zuordnung beim Drei-Parteien-Trading

Abbildung 5 zeigt nun die Ergebnisse dieser Simulation mit zyklischen Zuordnung der Server. Wie in der Abbildung zu sehen verschlechtert sich bei gegebener Anzahl von Servern die Antwortzeit bei steigender Anzahl von Tradern. Unter den gegebenen Rahmenbedingungen schneidet die zyklische Strategie im Vergleich zur probabilistischen jedoch noch wesentlich besser ab. Wie in Abbildung 1 zu sehen war, beträgt die mittlere Antwortzeit bei zufälliger Serverauswahl konstant $3\frac{1}{3}$, unabhängig von der Anzahl der Trader. Die zyklische Startegie liegt selbst bei vielen Tradern und Servern noch deutlich unter diesem Wert.

Nachfolgend soll nun diese Strategie mit solchen verglichen werden, die zusätzlich noch Statusinformation verwenden. Beim Drei-Parteien-Trading ist es nun möglich, neben dem letzten Status des Servers über seine Auslastung auch die vom Trader selbst den Servern zugeteilte Last mit einzubeziehen. Dieses Verfahren und die Berechnung des Schätzwertes für den aktuellen Server Status wird nun nachfolgend etwas genauer erläutert.

Es seien nun $t_k^0, k = 1, 2, ...$ die Zeitpunkte, zu denen die Auslastung des Servers i dem Trader zugänglich gemacht wird. Die Werte $Z_i(t_k^0)$ seien die jeweils erfaßten Wartezeiten in der Schlange des i-ten Servers $(i = 1, 2, ..., N)$ zu diesen Zeitpunkten. Im Zeitraum zwischen zwei aufeinanderfolgenden Meßpunkten $t_k^0 < t < t_{k+1}^0$ gibt es nun weitere Momente $t_k^p, k = 1, 2, ...; p = 1, 2, ...$ in denen Anfragen beim Trader eintreffen, die dieser jeweils einem Server zuordnet. Bevor der Trader nun einen neuen Auftrag zuordnet, berechnet er die Schätzwerte für die Auslastung $Y_i^-, i = 1, 2, ..., N$ der einzelnen Server wie folgt:

$$Y_i^-(t_k^p) = \left[Y_i(t_k^{p-1}) - (t_k^p - t_k^{p-1})\right]$$

mit

$$(Y_i^-(t_k^0)) = Z_i(t_k^0)$$

Diese Formeln berechnen im Prinzip rekursiv, welchen Teils ihrer Last die einzelnen Server seit der letzten Berechnung Y_i^- bereits abgearbeitet haben.

Die Werte nach erfolgter Zuordnung zum Server I berechnen sich nach folgender Formel:

$$(Y_i^-(t_k^p)) = \begin{cases} \left[Y_l^-(t_k^p)\right]^+ + b & : \quad i = l \\ Y_i^-(t_k^p) & : \quad i \neq l \end{cases}$$

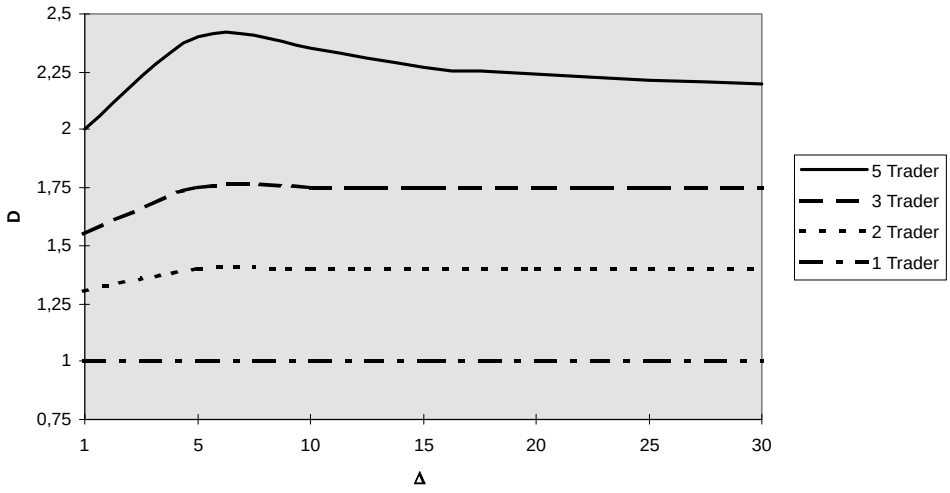

Abbildung6. Konkurrierenden Trader bei veralteter Information. $N = 20; \rho = 0,7$

Die Notation $[x]^+$ steht hier für x wenn $x > 0$ sonst 0 und b für die zusätzliche Last, die durch den Auftrag dem Server I zugewiesen wird. Neue Last wird dabei nur jeweils einem Server zugewiesen. Die Einheiten von Y_i^- und Z_i sind wieder die durchschnittliche Service Zeit.

Abbildung 6 zeigt nun das Ergebnis einer solchen Simulation. Mit steigender Anzahl der Trader verschlechtert sich die Antwortzeit, wird nur ein einziger Trader eingesetzt, entspricht dies dem Modell der Multiserverschlange aus Abschnitt 2.3. In diesem Idealfall ist die Schätzung der Auslastung der Server Y_i korrekt, da es keine konkurrierenden Trader gibt, die den Servern weitere Aufträge zuweisen. Wie bereits erwähnt, ist dieser Idealfall hinsichtlich der Leistungsfähigkeit der Zuordnung nicht zu übertreffen.

Bei mehreren Trader ist ein interessanter Zusammenhang zwischen der Verzögerungszeit Δ und der Antwortzeit D zu bemerken. Die kürzeste Antwortzeit ist natürlich für sehr kleine Werte von Δ zu erwarten, man nähert sich nämlich der Multiserverschlange an, bei der die Information unmittelbar zur Verfügung steht.. Für mittlere Werte von Δ wird die Performanz deutlich schlechter um dann für längere Verzögerungszeiten wieder besser zu werden. Die Trader tendieren anscheinend dazu, sich zwischen den Zeitpunkten der Messung an die allgemeine Auslastung anzupassen, das heißt, sie verteilen die ankommenden Aufträge gleichmäßig auf die verfügbaren Server. Zusätzliche Informationen dagegen bringen die Trader in ihrer Strategie durcheinander.

Der Hauptunterschied, den man im Vergleich zum Zwei-Parteien-Trading beobachten kann, ist das asymptotische Verhalten der Antwortzeit für Δ gegen Unendlich. Die Antwortzeit strebt hier also nicht wie im Kapitel 3 zu sehen war gegen unendlich.

Wie im Vergleich von Abbildung 6 mit Abbildung 5 zu sehen ist, strebt die Antwortzeit für steigendes Δ in der Tat gegen die Werte, die bei der zyklischen Zuordnung erreicht werden. Im Falle von sehr vielen Trader ist der Einsatz der dynamischen Zuordnung mit ihrem hohen Aufwand nicht gerechtfertigt, denn die einfach zu realisierende Strategie der zyklischen Zuordnung liefert praktisch die gleiche Leistung.

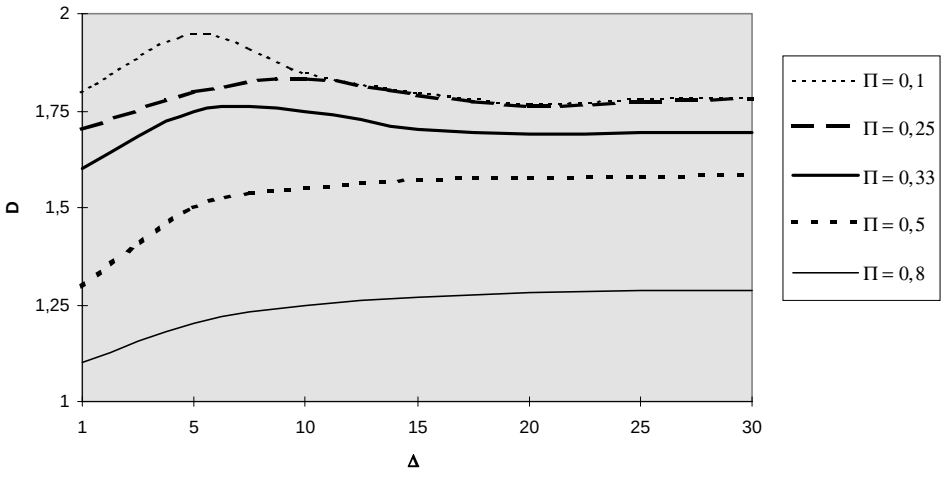

Abbildung7. Drei-Parteien-Trading bei unterschiedlicher Last. $N = 2; R = 3; \rho = 0,7$

4.2 Dynamische Serverauswahl bei ungleichmäßiger Lastverteilung

Im vorangegangenen Abschnitt wurde davon ausgegangen, daß alle Aufträge gleichmäßig auf die Trader verteilt wurden. Das bedeutet, daß die Ankukunftswahrscheinlichkeiten $\Pi_1, \Pi_2, ..., \Pi_R$ gleich waren. In Abbildung 7 werden nun für drei Trader ($R = 3$) folgende Fälle betrachtet:

a) $\Pi_1 = \Pi_2 = \Pi_3 = \frac{1}{3}$

b) $\Pi_1 = 0,5; \Pi_2 = \Pi_3 = 0,25$

c) $\Pi_1 = 0,8; \Pi_2 = \Pi_3 = 0,1$

Wie in dem Diagramm zu sehen ist, hat der Server, der die meisten Aufträge bearbeitet die beste Antwortzeit zu erwarten. Dieses Ergebnis ist auch logisch, denn da er die meisten Anfragen zuordnet, kann er auch den besten Schätzwert für die Auslastung der Server berechnen. Die beiden anderen Trader können nicht wissen, welchen Servern der stark beanspruchte erste Trader die Aufträge zuteilt, was zu ungenaueren Schätzwerten führt.

In Abschnitt 3.2 wurden die positiven Effekte beim Probing-Strategie bein Zwei-Parteien-Trading gezeigt. Wie Tschammer und Wolisz in [TWW92] zeigen, sind die Auswirkungen bei unterschiedlicher Lastverteilung nicht so positiv zu bewerten. Für beispielsweise $L = 2$ vorselektierte Server nähert man sich den Werten der probabilistischen Strategie an. Das bedeutet praktisch, daß höher belastete Server schlechter abschneiden, weniger belastete aber nicht so stark wie in Abbildung 7 zu sehen benachteiligt werden. Bei gleichmäßiger Verteilung der Last liefert die Probing-Strategie jedoch bessere Antwortzeiten als die beiden zuvor betrachteten Strategien, der zyklischen und der an der Auslastung der Server orientierten Regeln.

5 Zusammenfassung

In diesem Artikel wurde das Problem der dynamischen Serverauswahl mit Hilfe von dynamischer Statusinformation untersucht, und dabei angenommen, daß die Zeit, die benötigt wird, diese Information zu bekommen, im Vergleich zur Bearbeitungszeit des Auftrages nicht zu vernachlässigen ist. Diese Annahme scheint in vielen Fällen vernüfntig, beispielsweise bei Dienste, die häufig angesprochen werden, aber nur eine kurze Bearbeitungszeit benötigen wie zum Beispiel Nameserver, Systeme zur Verkehrsüberwachung oder Trader selbst. Dabei wurde der Fall des Zwei und des Drei- Parteien-Tradings betrachtet.

Beim *Zwei-Parteien-Trading* wurden die Grenzen der Verfahren aufgezeigt, die lediglich dynamische Statusinformation verwenden, die aber nicht aktuell ist. Es wurde gezeigt, daß die Kombination von Statusinformation mit einer zufallsgesteuerten Strategie eine wesentliche Verbesserung der Performanz bedeutet. Außerdem wurde anhand von Simulation der negative Einfluß von selbständigen Benutzern gezeigt. Systeme, bei denen die Benutzer Zugriff auf dynamische Statusinformation besitzen, sind in dieser Hinsicht sehr empfindlich.

Das *Drei-Parteien-Trading* bietet im Allgemeinen eine bessere Leistung, selbst wenn die Verzögerungszeit Δ gegen unendlich strebt, bleibt die Antwortzeit begrenzt. Dies folgt aus der Tatsache, daß der Trader Informationen über Informationen zu seinen bisherigen Zuordnungen verfügt. Alle Untersuchungen wurden unter der Annahme gemacht, daß der Prozeß des Tradings selbst sowie das Zuordnen zu einer Warteschlage keine Zeit benötigen und daß die Dienstzeit selbst exponential verteilt ist. Unter diesen Annahme währe ein einziger Trader ideal. Ein zentrale Trader kann aber andere Nachteile mit sich bringen. Er kann sich zum Beispiel aus Flaschenhals erweisen und in der Zuverlässigkeit geringer sein als mehrere gleichwertige Trader.

Zusammenfassend ist zu bemerken, daß es für die Wahl des Trading-Konzeptes und für die Wahl der Strategien kein Patentrezept gibt. Das Drei Parteien Trading sollte dort eingesetzt werden, wo dies möglich ist, da es bessere Antwortzeiten ermöglicht. Hier bietet selbst die recht einfache Strategie der zyklischen Zuordnung eine gute Performanz, da der Einfluß der Verzögerungszeit Δ eher gering ist. Wo diese Form des Tradings nicht möglich ist, sollte das Zwei Parteien Trading zum Einsatz kommen. Im Allgemeinen ist hier durch Server-Statusinformation und Zufall gesteuerte Probing-Strategie die beste Wahl. Sollte die Statusinformation mit sehr geringer Verzögerung $\Delta \ll 1$ zur Verfügung stehen, bietet eine Strategie, die diese Information nutzt bei beiden Formen des Tradings die beste Leistung.

Mobilität und Kommunikation in verteilten Agentensystemen

Björn Müller

Kurzfassung

Mobile Agenten sind Programme, die von einem Rechner zum nächsten wandern können. Sie interagieren mit den lokalen Ressourcen der Rechner, bis sie ihre Task beendet haben. Vorteile Mobiler Agentensysteme gegenüber herkömmlichen Client-Server Architekturen sind die geringere Kommunikation durch die Filterung relevanter Information am besuchten Rechner, die höhere Sicherheit durch die direkte Verbindung des Agenten zum besuchten Rechner und die bessere Bedienbarkeit für den Anwender, der sich nicht um komplexe Netzwerk-Syntax kümmern muß. Es existiereren bereits eine Reihe von Modellen, von denen hier Telescript, Ara, Messenger, TACOMA, Agent Tcl und Obliq vorgestellt und miteinander verglichen werden.

1 Einleitung

Motivation Neue Produkte und Anwendungen im Bereich der Telekommunikation und die immer stärkere Vernetzung der Rechner erfordern neue Lösungen, um dem Nutzer eine bessere Unterstützung für seine Probleme zu bieten. Traditionelle Kommunikation basiert auf dem Austausch von Nachrichten zwischen einem Server und einem Client. Beide Seiten müssen die Verbindung zum Gegenüber während des Nachrichtenaustauschs aufrechterhalten. Desweiteren wird eine große Menge an Daten über das Netz geschoben, die erst nach der Übertragung vom Client selektiert wird. Dadurch kann es vorkommen, daß ein Großteil der übertragenen Daten wieder vernichtet wird.

Mobile Agenten Um diese Probleme zu lösen, wurde das Konzept der Mobilen Agenten entwickelt. Mobile (wandernde) Agenten sind Programme, die von einem Ursprungsrechner abgeschickt werden und danach zwischen Netzwerkservern umherwandern. Sie interagieren mit den Ressourcen lokaler Netzwerkserver, bis die Task beendet ist. Als Erweiterung zu herkömmlichen Client/Server Modellen ist es Mobilen Agenten möglich, sich von Server zu Server zu bewegen und die für die Anwendung am besten geeigneten auszuwählen.

Vorteile Mobiler Agenten

– Leistungsverbesserung durch die Verringerung von Kommunikation. Es reicht aus, den Agenten zu übertragen, dieser filtert die relevante Information beim Server und speichert sie ab. So wird vermieden, daß für den Client unwichtige Information über das Netz geschickt wird.

– Das Modell für mobile Agenten entspricht der realen Welt. Der Agent repräsentiert seinen Client. Er handelt unabhängig von dem Client. Dadurch kann er die relevante Information filtern.

– Der Anwendungsprogrammierer muß sich nicht um komplexe Netzwerk-Syntax kümmern. Das einzige was notwendig ist, ist den Agenten zu bewegen, der kommunizieren will. Das Netzwerksystem unterstützt diese Bewegung, und so reicht eine Instruktion zur Wanderung aus.

– Die Fehlerbehandlung ist einfacher, da sich der Agent in direkter Verbindung mit dem Server befindet. Die Gefahr, daß es zu Inkonsistenzen kommt, ist geringer als im herkömmlichen Client/Server Modell, wo eine verteilte Fehlerbehandlung notwendig ist.

Gliederung In dieser Arbeit werden die bestehenden Konzepte miteinander verglichen. Hierzu erfolgt im nächsten Kapitel die Beschreibung des grundsätzlichen Aufbaus Mobiler Agentensysteme. Danach werden die bestehenden Konzepte vorgestellt und bewertet. Im letzten Kapitel erfolgt eine Zusammenfassung dieser Arbeit sowie ein Ausblick auf die Entwicklung in den nächsten Jahren.

2 Grundsätzlicher Aufbau

In diesem Kapitel werden zunächst der Ablauf einer Agenten-Verbindung und anschließend die Elemente des mobilen Agenten Netzwerks beschrieben. Der hier vorgestellte allgemeine Aufbau bezieht sich auf [CGH+95], wo das Konzept Mobiler Agenten unabhängig von einem bestimmten System vorgestellt wird. Die im nächsten Kapitel vorgestellten Modelle setzen verschiedene Schwerpunkte und besitzen deshalb nicht alle hier vorgestellten Komponenten. Dieses Kapitel soll jedoch später zur Bewertung der Modelle herangezogen werden.

2.1 Ablauf

Der Agent wird durch die Benutzer (Client)-Task initialisiert und mit Hilfe eines Nachrichtenkanals zum Server übertragen. Der Server kann direkt angegeben werden oder der Agent wird an einen Vermittlungsserver geschickt, der dann diejenigen Server vorschlagen kann, die zur Ausführung der Task am besten geeignet sind.
Nach der Ankunft des Agenten in einem Server wird er zu einem sogenannten Agent Meeting Point (AMP) gebracht. Der AMP ergänzt das Betriebssystem um die Agentenfunktionalität. Die ausführbaren Teile des Agenten werden gestartet. Wenn der Agent seine Task erfolgreich ausgeführt hat, werden die gewonnenen Informationen gespeichert und er besucht den nächsten Server. Desweiteren wird die Authentität und der Inhalt des Agenten untersucht. Wenn genügend Ressourcen vorhanden sind und für den Agenten genehmigt werden, wird dieser zu den Diensten weitergegeben.
Hieraus ergeben sich die folgenden Forderungen an den Agenten:

– Er muß die minimal notwendige Menge an Information speichern, um die Task beenden zu können.

– Er muß seine Befugnis den Servern beweisen können.

– Er muß auf seiner Route Wissen sammeln und aufgrund dieses Wissens Entscheidungen treffen können.

2.2 Agenten

Der Agent besteht aus drei Teilen: dem (Agenten-) *Paß*, dem *Table Of Contents* (TOC) und den *Komponenten*.

Paß Der Paß enthält die Information, die für den Agenten notwendig ist, um von AMP zu AMP zu wandern. Hierzu gehören die Authentifikation des Eigentümers, Informationen über Maßnahmen, die der AMP ergreifen soll, wenn ein Fehler auftritt, sowie Ziele und Statusinformationen des Agenten.

Table Of Contents und Komponenten Der TOC gibt zu jeder Komponente die Größe, den Typ und die Wichtigkeit an. Der Typ enthält Informationen, die zur Durchführung der Komponente notwendig sind. Die Wichtigkeit besagt, ob die Komponente zur Instanziierung des Agenten in dem AMP notwendig ist.

2.3 Sprache

Es gibt zwei Arten von Sprachen, die Sprachen zur Programmierung des Agenten und die Wissensrepräsentationssprachen.

Sprache zur Programmierung des Agenten In dieser Sprache ist der programmatische Inhalt des Agenten geschrieben. Ziel ist es, eine breite Auswahl an Sprachen zu unterstützen. In der Praxis haben sich einige jedoch als besonders geeignet erwiesen. Die Eigenschaften der Objektorientierung, der Mobilität, der Aufspaltung und Zusammenführung von Agenten, sowie verteiltes Programmieren, müssen durch die Sprache unterstützt werden.

Wissensrepräsentationssprachen In dieser Sprache werden Ziele, Aufgaben und Präferenzen des Agenten beschrieben. Es gibt viele verschiedene Arten von Wissensrepräsentationssprachen, z.B. den Knowledge Sharing Effort.

2.4 Beweglichkeit

Wandernde Programme Hier werden zwei Möglichkeiten unterschieden:

– das Programme läuft bis es fertig ist

– das Programm entscheidet, während der Ausführung den Server zu wechseln

Im zweiten Fall stellt sich die Frage, wie die Variablen und der Zustand von einer Maschine auf eine andere übertragen werden können. Am einfachsten ist es, alle Informationen, die mit der Programmausführung zu tun haben, im Programm-Stack zu sammeln und diesen zu transportieren. Ein anderer Ansatz benutzt ein Variablenregister und überträgt dieses zusammen mit den Variablen.

Gesammelter Zustand Hier können zum einen neue Objekte, mit der neuen Information, zu den bestehenden hinzugefügt oder bestehende verändert werden. Die Entscheidung, welche Variante gewählt wird, hängt von der Komplexität der Task ab. Sucht der Agent nur nach dem günstigstem Preis für ein bestimmtes Produkt, so reicht es aus, eine Variable zu ändern.

2.5 Plätze

Der Agenten Treff-Punkt (AMP) setzt sich aus einer Menge von Objekt-orientierten Klassen zusammen. Der AMP besteht aus den folgenden Teilen:

Dienste zur Unterstützung der Mobilität Der AMP benötigt *Kommunikationstore*, die die Ankunft und den Weggang der Agenten managen, *Authentifikationsdienste*, die auf den Authentifikationsdiensten des verteilten Netzwerks aufsetzen. Ein weiterer Dienst ist die *Concierge* . Hier wird der Paß des Agenten untersucht, es wird untersucht, ob die für den Agenten notwendigen Einrichtungen vorhanden sind und dem Agent wird bei der Ankunft im AMP geholfen. Der *Shallow Request Handler* ist die Schnittstelle zwischen dem Agent und den Komponenten des AMP. Der *Event Delivery Service* überträgt Informationen über Dienste des AMP zu Agenten außerhalb des AMP, die Interesse daran haben.

Dienste zur Unterstützung der Task des Agenten Der *Ressourcen Manager* registriert alle aktiven Agenten mit den ihnen zugeordneten Ressourcen. Der *Agent Execution Environment* bietet Zugang zu den Basiseinrichtungen des AMP. Der *Agent Status Service* ermöglicht Agenten, wichtige Aktivitäten festzuhalten (z.B. Ankunft).

2.6 Ankunft und Weggang

Bei der Ankunft werden folgende Schritte durchgeführt: Der Agent kommt durch ein Kommunikationsportal an. Danach wird er zur Concierge weitergegeben. Hier geschieht die Authentifikation und Registrierung des Agenten. Dem Agent werden Ressourcen zugeordnet und die Komponenten des Agenten werden an die sie unterstützenden Dienstelemente verteilt. Danach erfolgt die Ausführung des Agenten
Beim Weggang des Agenten müssen seine Komponenten gestoppt und in eine transportierfähige Form gebracht werden. Desweiteren muß die Statusinformation des Agenten erneuert werden.

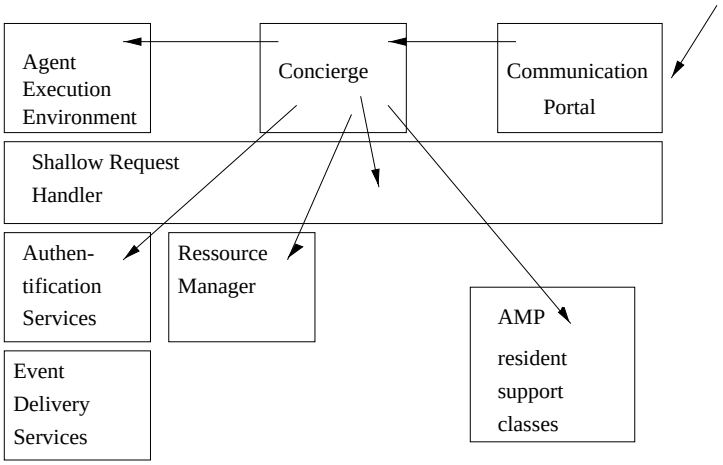

Abbildung8. Ankunft und Weggang

2.7 Sicherheit

Sicherheit ist einer der wichtigsten Punkte. Ein wichtiger Aspekt ist hier die digitale Unterschrift. Auch wenn es nicht möglich ist, alle Sicherheitsaspekte im Bereich der Mobilen Agenten abzudecken (Virus, Information hiding), gibt es doch einige Sicherheitsziele:

- Eindeutige Authentifikation des Ursprungs des Agenten.
- Datenintegrität; der ausführbare Code des Agenten muß vor Zugriffen geschützt sein. So darf z.B. ein AMP einem Agenten, der einen Höchstpreis für ein Flugticket von 1000 DM gespeichert hat, diesen Wert nicht heraufsetzen.
- Durch Zugangs- und Routenkontrolle ist eine Auswahl der AMPs durch den Client möglich.
- Privatheit und Integrität der gesammelten Informationen ist ein weiterer wichtiger Punkt, ohne den das Agentenkonzept keinen Sinn macht, weil der Client sicher sein muß, daß die gesammelte Information auch richtig ist.

3 Vorstellung bestehender Systeme

3.1 Bewertungskriterien

Um die nachfolgenden Konzepte miteinander vergleichen zu können, werden sie anhand folgender Kriterien bewertet:

Wanderung des Agenten Bei diesem Punkt geht es vor allem darum, ob der Agent selber während der Ausführung des Programms entscheiden kann, ob er den Server wechselt.

Schnittstellen Hier soll beurteilt werden, wie gut der Agent mit der Außenwelt (Anwender) interagieren kann.

Zuverlässigkeit Beim Wandern und bei der Ausführung des Agenten können Fehler auftreten. Die verschiedenen Modelle werden deshalb im Hinblick auf die Zuverlässigkeit untersucht. Hier geht es vor allem um die Sicherung des Agenten-Zustands, um ein Rücksetzen im Fehlerfall zu ermöglichen.

Sicherheit In Abschnitt 2.7 wurden einige wichtige Sicherheitsziele (Authentifikation, Datenintegrität, Zugangskontrolle und Privatheit) genannt. Anhand der Realisierung dieser Ziele werden die Modelle nun bewertet.

3.2 Telescript

Der Schwerpunkt von Telescript ist der elektronische Marktplatz. Anbieter und Konsumenten von Produkten und Dienstleistungen sollen sich hier finden und ihre Geschäfte elektronisch abwickeln.

Aufbau Die prinzipiellen Konzepte von Telescript sind Plätze, Agenten, Reisen, Meetings, Verbindungen und Freigaben. Teleskript ist objektorientiert. Der Anwender kann eigene Klassen definieren. Um dies zu erleichtern, existieren einige vordefinierte Klassen.

Plätze Ein Netzwerk von Computern wird als eine Sammlung von Plätzen verstanden. Ein Platz bietet einen Dienst einem ihn besuchenden mobilen Agenten an. Zur Unterstützung der Dienste kann der Besitzer des Platzes Klassen definieren. So ist der Platz um die gewünschten Funktionen erweiterbar. Plätze besitzen eine eindeutige Adresse.

Agenten Eine Kommunikationsanwendung wird als eine Sammlung von Agenten modelliert. Agenten sind unabhängig voneinander. Jedem Platz ist ein stationärer Agent zugeordnet. Ein Agent ist durch eine Klasse definiert, welche durch den Benutzer definiert werden kann. Somit kann er selber das Aussehen des Agenten bestimmen. Agenten besitzen, wie Plätze, eine eindeutige Adresse.

Reisen Agenten können von einem Platz zum anderen reisen. Dies ist, wie schon in der allgemeinen Einführung angeführt, eine typische Eigenschaft eines Agentensystems. Programme, die durch das Netz wandern, sind seit über 20 Jahren bekannt. Das besondere hier ist, daß die Programme während ihrer Ausführung bewegt werden, und daß über die Reiseroute dynamisch entschieden wird (siehe 2.3). Die Prozedur muß demnach in einer Sprache geschrieben sein, die eine Wanderung erlaubt. Die Teleskript Sprache erlaubt das Einpacken des Agenten (der Prozedur als auch des Zustands), um diesen zu einem anderen Computer zu transportieren. Der Agent selbst entscheidet, wann ein Transport notwendig wird.

Meetings Ein Meeting zwischen zwei Agenten findet in einem Platz statt. In diesem Meeting rufen die Agenten die Prozedur des anderen auf. Wanderungen finden demnach statt, um einen anderen Agenten zu treffen; entweder den stationären Agenten des Platzes oder einen beliebigen anderen.

Verbindungen Eine Verbindung erlaubt die Kommunikation von Agenten, die sich in unterschiedlichen Plätzen befinden. Verbindungen werden oft genutzt, um mit dem Nutzer zu interagieren.

Freigaben Agenten und Plätzen können bestimmte Freigaben erteilt werden. Freigaben gewähren Fähigkeiten. Diese können von den Agenten und Plätzen wahrgenommen, aber nicht erweitert werden. Eine Freigabe kann zum einen das Recht sein, eine bestimmte Instruktion ausführen zu dürfen, zum anderen das Recht, eine bestimmte Ressource nutzen zu dürfen.

Sprache Der stationäre Teil der Software ist in C geschrieben. Die Agenten und die Oberfläche der Plätze sind in Telescript geschrieben. Teleskript hat die folgenden Eigenschaften:

- Vollständigkeit: Jeder Algorithmus kann in Teleskript geschrieben werde.
- Objektorientierung: Der Programmierer definiert Klassen von Informationen (s.o.).
- Dynamik: Ein Informationsobjekt kann vom Agenten von einem Platz zum nächsten gebracht werden.
- Persistenz: Durch die Speicherung des Agenten sowie der Information ist ein Schutz vor Computerfehler gegeben.
- Tragbarkeit und Sicherheit: Ein Computer führt eine Agenten-Anweisung durch eine Telescript-Maschine aus. Der Agent kann dadurch nicht direkt auf den Prozessor und den Speicher zugreifen. So wird die Gefahr von Viren verringert.
- Kommunikationzentrierung: Bestimmte Instruktionen lassen einen Agenten komplexe Netzwerk-Tasks ausführen (z.B. go Instruktion).

Maschine Die Telescript-Maschine ist ein Softwareprogramm, das die Telescript-Sprache implementiert. Die Maschine verwendet drei Anwendungsprogrammschnittstellen (API): Ein Speicherungs-API um Plätze und Agenten vor Computerfehlern zu schützen. Ein Transport-API, um den Zugang zu Kommunikationsmedien zum Transport von Agenten zu besitzen. Sowie ein externes Anwendungs-API, um die Interaktion zwischen den Anwendungen in Teleskript und denen in C zu ermöglichen.

Bewertung Der Agent entscheidet, wann er wandern will. Die Wanderung wird durch eine einzelne Instruktion bewirkt (go). Die Task des Agenten wird am Abbruchpunkt fortgesetzt.
Die Sicherheit wird durch die Authentifikation der Agenten, sowie der indirekten Ausführung der Agenten-Anweisungen durch die Telescript-Maschine gewährleistet. Desweiteren ist durch die Objektorientierung eine Unterscheidung in einen öffentlichen und einen privaten Teil möglich.
Beim Transport wird der Agent, inklusive der gesammelten Daten, übertragen.
Schnittstellen zum Nutzer sind durch Instruktionen gegeben, durch die der Zustand des

Agenten abgefragt werden kann.
Fazit: Vordefinierte Dienste, die über die reine Transportfunktion hinausgehen, sind noch nicht gegeben. Sie können jedoch durch den Benutzer definiert werden. Teleskript ist somit im Vergleich zu Kapitel 2 relativ vollständig.

3.3 Ara

Aufbau Ara ist aus den im fogenden Bild gezeigten Komponenten aufgebaut.

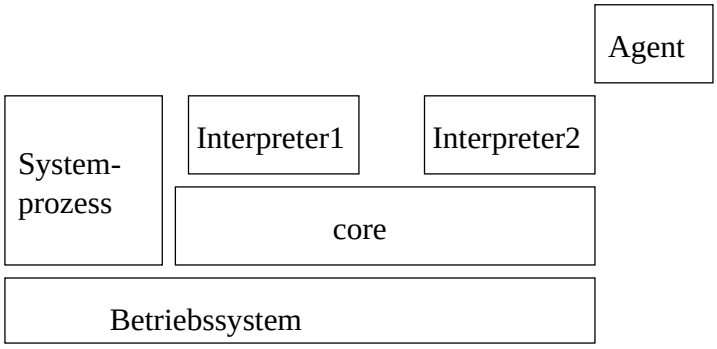

Abbildung9. Ara System

Core Der core ist die zentrale Komponente eines Ara Systems. Er leitet den Agentenprozeß und vermittelt die Interaktion. Basisfunktionen, wie die Wanderung, werden dem Agent hier zur Verfügung gestellt. Höhere Dienste werden jedoch durch Server-Agenten angeboten. Desweiteren regelt der core den Zugang eines Agenten zu einem Host-System oder zu einem anderen Agenten, um die Sicherheit und die Transportierbarkeit zu gewährleisten. Agenten werden unabhängig von ihrer Sprache behandelt (siehe Interpreter).

Interpreter Ein mobiler Agent wird durch einen Interpreter für seine Programmiersprache ausgeführt, wobei der core benutzt wird. Hier findet eine klare Trennung zwischen den sprachspezifischen Fragen, die sich im Interpreter befinden und den sprachunabhängigen Teilen des core statt. So können sich mehrere Interpreter für verschiedene Sprachen im Kern befinden.

Systemprozeß Verschiedene Tasks des Ara Systems werden durch Prozesse unterstützt, die spezielle Vorrechte besitzten (z.B. Verzicht auf die Vermittlung durch den core).

Agenten Ein mobiler Agent ist ein Programm mit der Möglichkeit, während der Ausführung zu wandern. Dabei wird die Identität und der Zustand nicht verändert. Auch Programme, die nicht wandern, werden als Agenten bezeichnet. Agenten besitzen einen global einzigartigen, unveränderlichen Namen. Sie gehören einer bestimmten Gruppe von Agenten an, durch die der Zugang zu den Ressourcen geregelt ist.

Das Erschaffen und die Zerstörung von Agenten sind Basisfunktionen, die vom core angeboten werden. Die erzeugten Agenten können ihr Programm parallel zu anderen ausführen, sie können sich vervielfältigen, ihre Operation stoppen, unterbrechen oder fortsetzen. Agenten kann eine bestimmtes Wirkungsgebiet durch den Erzeuger eingeräumt werden. Desweiteren ist es dem Platz möglich, die Ressourcen für den Agenten zu beschränken.
Um Fehler zu beheben, kann ein Agent Sicherungspunkte erzeugen. Hier ist es möglich, den Agenten auf einen vorher gesicherten Zustand zurückzusetzen.

Plätze Ein Platz ist ein virtueller Ort in einem Ara System. Ein Agent befindet sich entweder in einem Platz oder wandert zwischen ihnen. Ein Platz schafft ein Gebiet von logisch verbundenen Diensten unter einer gemeinsamen Sicherheitsordnung, die alle Agenten des Platzes kontrolliert. Dies geschieht durch Zugangs- und Ressourcenbeschränkungen. Plätze haben eindeutige Namen. Ara unterstützt im Moment nur einen Platz pro System, ohne Zugangskontrolle. Plätze sind im jetzigen Stadium keine bestimmten Objekte mit einer programmierbaren Schnittstelle.

Service Point Ein Service Point ist ein Treffpunkt für Agenten, der Interaktion unterstützt. Er besitzt einen Server-Agenten und mehrere Client-Agenten. Der Server ist für die Öffnung und das Schließen zuständig. Ein service point ist immer an einen Platz gebunden.

Mobilität Ara Agenten können an jedem Punkt ihrer Ausführung wandern. Die Wanderung wird durch eine spezielle Instruktion ausgelöst (Arago). Bei der Wanderung kann der externe Status des Agenten nicht mitgenommen werden. D.h. die Beziehung zu anderen Systemobjekten und Ressourcen geht verloren. Die Wanderung findet zwischen Plätzen statt.

Bewertung Agenten können in jedem Punkt ihrer Ausführung wandern. Dies geschieht durch eine spezielle Instruktion (Arago). Am Zielpunkt der Wanderung läuft der Agent direkt am Abbruchpunkt weiter.
Sicherheit wird durch Zugangskontrolle und Ressourcenzuteilung gewährleistet. Wobei die Zugangskontrolle im Moment noch nicht realisiert ist.
Durch Sicherungspunkte kann ein Agent im Fehlerfall auf seinen alten Zustand zurückgesetzt werden.
Ara unterstützt mehrere Sprachen (siehe Interpreter). Dadurch ist jedoch das Prinzip der Objektorientierung nur teilweise gegeben.
Fazit: Der Hauptunterschied zu Telescript ist die Unterstützung mehrerer Sprachen. Plätze sind im Moment keine einzelnen Objekte, sondern an einen Server gebunden. Die Programmierung der Agenten hängt vön der gewählten Sprache ab.

3.4 Messenger

Messenger basiert auf dem Austausch von protokollunspezifischen Programmen.

Aufbau

Messenger Wenn zwei hosts miteinander kommunizieren wollen, tauschen sie Programme aus. Diese werden Messenger genannt. Ein Messenger enthält sowohl Daten als auch die Regeln, zum Aufbau eines Protokolls.

Die Messenger Plattform Alle hosts, die an einer Messenger-Kommunikation beteiligt sind, teilen sich eine gemeinsame Repräsentation der Messenger und unterstützen eine lokale Ausführungsumgebung. Diese Ausführungsumgebung wird Messenger Plattform genannt. Ein Messenger wird sequentiell bearbeitet, es können sich aber mehrere Messenger in der Plattform befinden, die gleichzeitig bearbeitet werden. Eine Messenger Plattform besteht aus den folgenden Komponenten:

- Eine Menge von *Kanälen* ermöglicht es einer Messenger Plattform, Messenger mit anderen Plattformen auszutauschen.
- Messenger werden nach ihrer Ankunft in unabhängige *Messenger-Prozesse* umgewandelt.
- Die ausführenden Prozesse können sich gemeinsame Daten durch ein *Wörterbuch* teilen.
- *Prozeß-Warteschlangen* erlauben eine Sortierung der Messenger nach ihrer Wichtigkeit.

Sprache Ein Messenger ist eine Anzahl von Instruktionen, die in einer Messenger- Programmiersprache ausgedrückt werden. Diese Sprache ist allen Plattformen gemeinsam. Die Sprache enthält Anweisungen, um den Messenger an andere Plattformen zu senden und um neue Messenger zu bilden.

Bewertung Messenger erlaubt eine Kommunkation zwischen zwei Rechnern, ohne daß der Empfänger das Protokoll des Senders kennen muß. Eine Wanderung der Messenger ist möglich.
Eine dirkete Unterstützung der Agentenfunktionalität (Authentität, Routenwahl usw.) ist hier nicht gegeben.
Fazit: Messenger kann nur als unterste Schicht eines Agentensystems dienen.

3.5 TACOMA

TACOMA wurde entwickelt, um Agenten im Internet zu unterstützen.

Aufbau

Agenten Bei TACOMA wird keine Unterscheidung zwischen Client, Server und Agent gemacht; alle werden als Agenten betrachtet. Der Agent selbst entscheidet, wann er wandern will, das System muß ihn hierbei lediglich unterstützen.

Hefte, Taschen und Schränke Will ein Agent seinen Ort wechseln, so muß er seinen Zustand mitnehmen. Zu diesem Zweck wurden Hefte eingerichtet, die Einheiten von Daten darstellen, die zu einem Agent gehören. Hefte, die mit einem Agenten zusammenhängen, werden zu Taschen zusammengefaßt. Ein Agent wird also beim Wandern seine Tasche mitnehmen. Zum anderen gibt es Hefte, die Daten zum stationären Gebrauch enthalten. Diese werden zu Schränken zusammengefaßt.

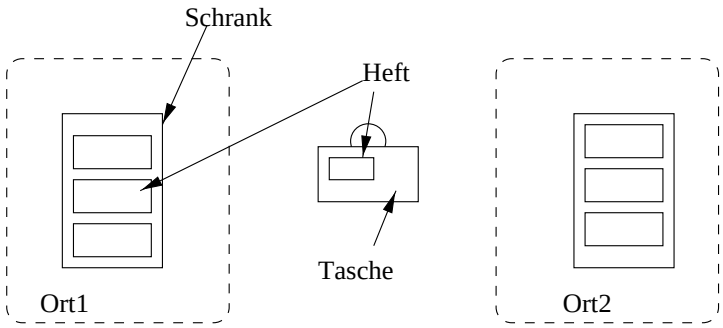

Abbildung10. TACOMA

Treffen Agenten können sich treffen. Dies geschieht durch den Austausch einer Tasche an einem Ort. Das Treffen muß einen Ankunftspunkt haben. Dieser wird bridge head genannt.

Dieser Ankunftspunkt für Gast-Agenten dient zur Authentifikation, Zugangskontrolle, zum Zählen und zur Fehlervermeidung. Zu diesem Zweck dient ein spezieller Agent.

Bewertung Die Wanderung wird durch eine einzelne Instruktion veranlaßt (meet). Tacoma unterstützt nicht die Unterbrechung eines ausführenden Agenten. Agenten treffen sich an einem Ort und tauschen dort Daten aus. Sicherheitsmaßnahmen werden durch einen stationären Agenten gewährleistet. Desweiteren enthält TACOMA Agenten, die einen verlorenen Agent neu starten können.
Fazit: TACOMA bietet die Möglichkeit, Anwendungen durch spezielle benutzerdefinierte Agenten zu erweitern. Ein großer Nachteil ist, daß eine Unterbrechung laufender Agenten nicht möglich ist. Spezielle Schnittstellen zum Nutzer sind nicht gegeben. Es wird nur eine Sprache unterstützt.

3.6 Agent-Tcl

Zweck von Agent Tcl ist es, die Schwächen bisheriger Systeme zu vermeiden, wobei man sich hier an Telescript, TACOMA und Messenger orientiert hat.

Aufbau Die Architektur von Agent Tcl besteht aus 4 Schichten:

1. Einem API für jeden Transportmechanismus.
2. Einnem Server für jedes Netzwerkgebiet, mit den folgenden Aufgaben:

- Ankommende Agenten annehmen, die Authentifikation des Owners übernehmen und die Agenten zu dem richtigen Interpreter weiterleiten.
- Das Senden von Nachrichten von einem Agent zu einem anderen ermöglichen.
- Es einem Agent ermöglichen, sich selbst oder eine Kopie an einen anderen Platz zu senden.
- Die Speicherung des internen Status der Agenten ermöglichen.
- Wiederherstellen von Agenten im Fehlerfall.

3. Interpreter für jede verfügbare Agentensprache. Ein Interpreter besteht aus den folgenden Komponenten:
   - Dem Interpreter selbst.
   - Einem Sicherheitsmodul, das fehlerhafte Aktionen der Agenten verhindert.
   - Einem Statusmodul, das den internen Staus eines ausführenden Agenten speichert und wieder herstellt.
   - Einem API, das mit dem Server interagiert, um Wanderung, Kommunikation und das Setzen von Sicherungspunkten zu ermöglichen.
4. Den Agenten selber.

Wanderung Zur Wanderung genügt eine spezielle Instruktion (Agentjump). Die Ausführung des Agenten wird an der neuen Maschine direkt am Abbruchpunkt fortgesetzt.

Bewertung Die Wanderung ist durch eine spezielle Instruktion möglich (Agentjump). Es werden mehrere Sprachen unterstützt. Sicherheit sowie Zuverlässigkeit wird durch den Server gewährleistet.
Fazit: Agent-Tcl ist ähnlich vollständig wie Teleskript. Vorteile sind die Unterstützung mehrerer Agenten-Sprachen und implizite Sicherheits- und Zuverlässigkeitsmaßnahmen. Im Moment(1995) unterstützt Agent-Tcl jedoch nur eine Sprache (Tcl), die nicht objektorientiert ist.

3.7 Obliq

Ziel von Obliq ist es, die Benutzerschnittstelle eines Agenten bei der Wanderung mitzunehmen.

Aufbau Ein Agent kann sich von einem Ort des Netzwerks zu einem anderen bewegen. Ein *Koffer* ist ein Teil von Daten, die ein Agent mitnimmt, wenn er wandert. Er enthält den Langzeitspeicher des Agenten. Beispielsweise können hierin die Orte gespeichert sein, die der Agent besuchen will, die Tasks, die an jedem Ort ausgeführt werden sollen und die Resultate der Ausführungen.

Ein *briefing* sind Daten, die der Agent an jedem Ort erhält. Hierin enthalten können Ratschläge für den Agenten (z.B. zu lange Wartezeit) und ortsspezifische Daten sein.
Ein *Agenten-Server* für einen bestimmten Ort ist ein Programm, das den Code des Netzwerks akzeptiert, diesen ausführt und mit dem lokalen briefing unterstützt.
Eine *hop-Instruktion* wird von Agenten genutzt, um sich von einem Ort zum nächsten zu bewegen. Hierbei wird die Prozedur zur Ausführung zum neuen Server transportiert. Anschließend wird der Koffer kopiert. Der Agent wird nun mit dem lokalen Koffer und dem lokalen briefing aufgerufen.
Ein Agent ist eine benutzerdefinierte Prozedur, die durch einen Koffer und ein briefing parametrisiert ist. Wenn ein Agent eine Benutzeroberfläche besitzt, so wird diese beim Ortswechsel gespeichert und am neuen Ort wieder aufgebaut.

Bewertung Obliq ist eine eigene Sprache. Die Wanderung wird durch eine Instruktion veranlaßt (hop). Die Ausführung beginnt am Abbruchpunkt. Jede Obliq-Anwendung kann in eine mobile umgewandelt werden. Das besondere ist die Mitnahme der Benutzerschnittstelle. Durch die Kopie des Koffers ist im Fehlerfall ein Rücksetzen möglich. Sonstige Agentenfunktionalität wird hier nicht unterstützt.

4 Zusammenfassung und Ausblick

Durch Mobile Agenten ist es möglich, Nachteile des herkömmlichen Client/Server Konzepts zu beseitigen. Hierzu gehören die geringere Kommunikation und die leichtere Bedienbarkeit durch den Nutzer. Die bereits realisierten Modelle sind im Vergleich zum Grundmodell im 2. Kapitel unterschiedlich vollständig.
Telescript bietet, durch die Möglichkeit Agenten und Server um benutzerdefinierte Klassen zu erweitern, ein relativ vollständiges Konzept an. Die Wanderung des Agenten ist durch eine Instruktion gelöst, und eine Unterbrechung und Fortsetzung der Task am Zielserver ist möglich. Über die reine Transportfunktion hinausgehende Dienste sind noch nicht realisiert.
Ara legt einen Schwerpunkt auf die Unterstützung mehrerer Sprachen. Die Wanderung wird wie in Telescript unterstützt. Desweiteren kann ein Agent im Fehlerfall auf einen vorherigen Zustand zurückgesetzt werden. Im Moment ist ein Platz an einen Server gebunden.
Messenger unterstützt nicht direkt die Agentenfunktionalität. Vergleichbar ist hier nur die Wanderung, bei der Daten zusammen mit Instruktionen übertragen werden. Ziel ist hier der Austausch protokollunspezifischer Programme.
Im TACOMA Modell wird sowohl der Client als auch der Server als Agent verstanden. Ein sich in Ausführung befindlicher Agent kann nicht unterbrochen werden. Das Wandern wird auch hier durch eine Instruktion unterstützt.
Agent-Tcl wird als eine Weiterentwicklung bestehender Systeme verstanden. Es soll eine Unabhängigkeit von der Agentensprache erreicht werden. Im Moment (1995) wird jedoch nur Tcl unterstützt. Die Wanderung wird auch hier durch eine Instruktion realisiert. Sicherheitsmaßnahmen und Zuverlässigkeit wurden realisiert.
Obliq unterstützt die Wanderung durch eine Instruktion. Die Benutzerschnittstelle wird

bei der Wanderung mitgenommen. Eine Kopie des Koffers erlaubt ein Rücksetzen im Fehlerfall. Sonstige Agentenfunktionalität wird hier nicht unterstützt.

Die weitere Arbeit auf diesem Gebiet wird vor allem darin bestehen, dem Nutzer ein vollständiges System zu bieten. Dazu gehört die Unterstützung mehrerer Programmiersprachen für den Agent, das Anbieten umfangreicher vordefinierter Dienste, sowie automatische Sicherheits- und Zuverlässigkeitsmaßnahmen. Die leichte Interaktion zwischen Agent und Nutzer ist ein weiterer wichtiger Aspekt.

Überblick Push-Technologie

Thomas Kresken

Kurzfassung

Seit einigen Monaten werden sogenannte Push-Verfahren als geradezu revolutionäre Methoden zur Verteilung von Daten im Internet und auch in Intranets angepriesen. Da ist vom nahen Ende der Browser und des Internets in seiner jetzigen Form ebenso die Rede wie von einer neue Ära des Informationszeitalters.

Diese Arbeit gibt eine Einführung in Push-Verfahren, ihre Funktionsweise, Fähigkeiten, Einsatzmöglichkeiten und die Probleme, mit denen sie heute noch behaftet sind. Dazu werden auch bestimmte Aspekte der Netzwerktechnik angesprochen. Um einen Überblick über die Fähigkeiten verfügbarer Systeme zu bekommen, werden drei Systeme beispielhaft vorgestellt.

1 Einleitung

Wer sich heute Informationen aus dem World Wide Web (WWW) beschaffen möchte, benutzt dazu in der Regel einen Browser, mit dessen Hilfe er die entsprechende WWW-Seiten sucht und sich schließlich die gewünschten Informationen auf seinen Rechner holt. Ein Problem stellt bei diesem *Pull* genannten Verfahren die Suche selbst dar. Es ist oft äußerst schwierig, die gewünschten Informationen zielgerichtet und zeiteffizient ausfindig zu machen.

Durch das weltweit sprunghaft gestiegene Interesse am Internet in den letzten drei Jahren hat sich auch die Anzahl der im WWW angebotenen Seiten drastisch erhöht. Neben den Bereichen Forschung, Industrie und Politik tummelt sich eine noch größere Anzahl Seiten von Privatleuten und Vereinen. Hier genau die gesuchten Informationen möglichst schnell zu finden, kann sich zu einer sehr zeit- und kostenaufwendigen Aufgabe entwickeln.

Zwar gibt es eine Anzahl verfügbarer Suchmaschinen mit beträchtlichen Datenbeständen, doch ist die Suche auch hier sehr ungenau, und die Aktualität der Daten ist nicht immer hoch. So kann es vorkommen, daß die Ziele nicht mehr verfügbar sind oder sich die darauf angebotenen Informationen bereits geändert haben. Selbst wenn das Ziel und mit ihm die gewünschten Informationen vorhanden sind, kann aufgrund von Server- oder Netzwerk-Problemen ein Zugriff zumindest momentan unmöglich sein. Oder es kommt in besonders verkehrsstarken Zeiten zu hohen Übertragungszeiten beim Zugriff auf einen Server („World Wide Waiting“).

Möchte ein Nutzer zudem stets auf dem aktuellen Stand über ihn interessierende Themen sein, so bleibt ihm im allgemeinen nichts anderes übrig, als in regelmäßigen Abständen die entsprechenden WWW-Seiten zu besuchen und etwaige Änderungen zu suchen.

Umgekehrt finden sich viele, besonders kleinere Firmen in der Situation, daß sie zwar interessante Produkte anzubieten haben, der Zustrom zu ihren WWW-Seiten dennoch sehr gering ist. Da die Investitionen in ansprechende WWW-Seiten oft hoch sind, besteht

ein nicht geringes Interesse, daß dieses Angebot auch genutzt wird. Es gibt jedoch nur wenige Möglichkeiten für diese Unternehmen, auf sich aufmerksam zu machen. Außer den Einträgen bei diversen Suchmaschinen und eventuell vorhandenen Verweisen von WWW-Seiten anderer bleibt nur, auf Interessenten zu warten.

1.1 Poll Pull und Push

Um diesen Problemen entgegenzuwirken und die Informationsbeschaffung so einfach wie möglich zu gestalten, hat man mit den *Push*-Verfahren die Situation umgekehrt: Nicht mehr der Nutzer sucht sich seine Informationen, sondern die Informationen kommen zu ihm.

Der Server verschickt dabei Daten an den Rechner des Nutzers, ohne von ihm dazu direkt aufgefordert worden zu sein. Im Idealfall würde der Server dem Nutzer ohne jegliches Zutun die Daten an seinen Rechner schicken, für die sich der Nutzer beim Server als Interessent eingetragen hat.

Um den Datentransfer zu initiieren, „abonniert“ der Nutzer zuvor einen Kanal[1], über den ihm dann die Daten geschickt werden. Dabei ist es möglich, die Daten nach bestimmten Kriterien zu selektieren, also eine möglichst individuelle Auswahl zu treffen.

Dieses oft mit dem Fernsehen oder Radio verglichene Broadcasting ist zwar heute bereits technisch möglich, wird aber aufgrund der damit verbundenen Investitionen in die benötigte Hardware noch nicht realisiert (s. Abschnitt 4). Daher installiert der Nutzer auf seinem Rechner einen Client, der dafür verantwortlich ist, in bestimmten Zeitabständen beim Server nachzufragen, ob neue Daten vorliegen. Diese Vorform des reinen Push-Verfahrens wird *Poll Pull* oder manchmal auch *Smart Pull* oder *Pull Push* genannt. Finden kann man derartige Bezeichnungen beispielsweise bei BackWeb [Bac97]. Marimba dagegen bezeichnet dieses Verfahren weiterhin als Pull [Mar97].

Ein wichtiger Aspekt bei Push-Verfahren sind Rückmeldungen vom Nutzer an den Anbieter. Dabei soll zum einen das Nutzerverhalten zur individuellen Anpassung der Informationen analysiert werden. Zum anderen können die so gewonnenen Nutzerprofile für Werbung benutzt werden.

Beim Push wird der Nutzer selbst nicht mehr zur Informationsbeschaffung aktiv, lediglich zum Einrichten oder Abbrechen einer Verbindung („Abonnieren“ bzw. „Abbestellen“ eines Kanals) oder zum Ändern der Selektionsparamter.

2 Der Kanal

Zentraler Bestandteil einer auf Push basierenden Datenverteilung ist der sogenannte Kanal. Alle Anbieter von Push-Systemen bauen die Architektur ihrer Systeme auf diesem Modell auf. Dennoch gibt es zwischen den einzelnen Anbietern erhebliche Unterschiede in der Definition dessen, was ein Kanal in ihrem System repräsentiert.

[1] Der Begriff des Kanals ist hier nicht im technischen Sinne zu verstehen, auch gibt es zwischen den einzelnen Anbietern der Push-Verfahren teilweise beträchtliche Unterschiede in der Definition des Begriffs (s. Abschnitt 2)

2.1 Definition

Die allgemeinste Definition vertritt die Firma Intermind: „Push technologies create automated, intelligent communications relationships between information publishers and subscribers – relationships which have come to be called *channels*.“ [RJ97] Hier wird der Begriff des Kanals noch an den technischen Begriff angelehnt, ohne jedoch die Art der Verbindung näher zu spezifizieren.

Allgemein wird diese Sicht eines Kanals von den anderen Anbietern nicht getragen. Sie stellen die zu übertragenden Daten in den Vordergrund und verstehen unter einem Kanal schlicht die Gesamtheit dessen, was dem Nutzer an Daten zur Verfügung gestellt wird. Die Verwendung desselben Begriffs für zwei so unterschiedliche Bedeutungen läßt sich damit erklären, daß der englische Begriff *channel* sowohl die Bedeutung „Übertragungsstrecke“ als auch „Programm“ im Sinne eines Radio- oder Fernsehprogramms haben kann.

Bei PointCast [Poi97], dem Vorreiter auf dem Gebiet der Push-Verfahren, umfaßt ein Kanal alle Informationen, die einem bestimmten Thema zuzuordnen sind. Da sich PointCast auf das Verteilen von Nachrichten spezialisiert hat, werden Spartenkanäle angeboten: Wetterkanal, Sportkanal, aktuelle Nachrichten uvm.

Die Firma BackWeb wiederum versteht unter einem Kanal die Gesamtheit aller zu übertragenden Daten, die von beliebiger Art sein können [Bac97]. Marimba hat sich auf die Verteilung von Software spezialisiert und sieht daher einen Kanal als ein mehrteiliges ausführbares Programm an [Mar97]. Bei Microsoft schließlich stellt der Kanal die Meta-Daten dar, mit Hilfe derer die Verteilung der Daten näher beschrieben wird [Ell97].

Der Bereich der Push-Verfahren ist noch weit von einer Standardisierung entfernt, daher gibt es keine Definition des Begriffs, der allen Interpretationen gerecht wird. Obwohl die Bedeutung des Wortes Kanal im Deutschen mehr ins Technische geht, wird hier eine allgemeinere Bedeutung verwendet, um dem üblichen Gebrauch des Begriffs in der Literatur zu folgen.

2.2 Kanalmodelle

Wie schon bei der Bedeutung des Begriffs Kanal so gibt es auch bei den verwendeten Kanalmodellen erhebliche Unterschiede. Stellvertretend sollen hier zwei Extreme betrachtet werden.

Im einfachsten Fall steht hinter dem Kanalmodell lediglich der URL[2] des betreffenden Servers. Diese Sichtweise findet man z.B. bei Netscape (www.netscape.com).

Im WWW gibt es zwei Gruppen von Knoten – Browser und Server: Ein Browser kann von einem beliebigen Server eine WWW-Seite anfordern oder explizit Daten an ihn übertragen. WWW-Seiten werden nur auf Server-Seite gehalten, nicht aber beim Client. Bild 11 verdeutlicht dieses Modell.

[2] Uniform Resource Locator, die eindeutige Kennzeichnung eines WWW-Servers und sich darauf befindender WWW-Seiten.

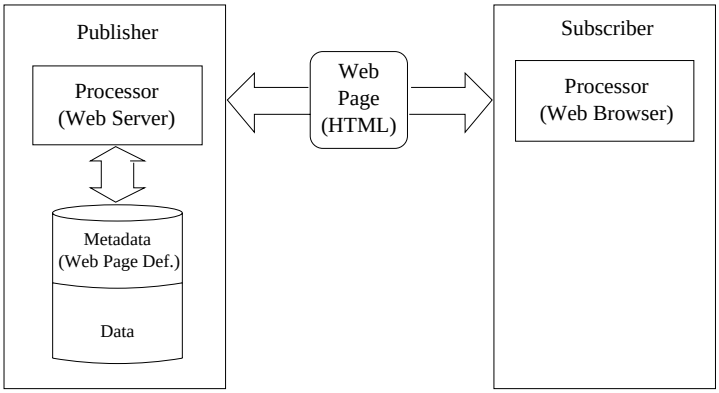

Abbildung11. Grundlegende WWW-Architektur.

Am anderen Ende der Komplexitätsskala steht beispielsweise das in Bild 12 gezeigte Kanalmodell der Firma Intermind, das das oben beschriebene Modell der WWW-Architektur erweitert [RJ97].

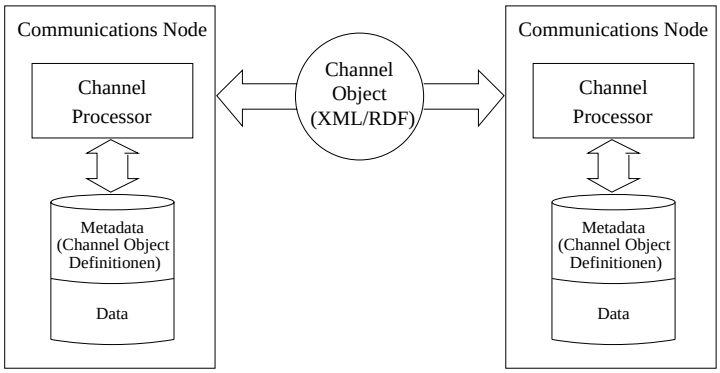

Abbildung12. Grundlegende Kanal-Architektur.

Dieses Kanalmodell weist drei wesentliche Veränderungen gegenüber dem Modell des WWW auf:

- Es gibt auf der architektonischen Ebene keine Unterscheidung mehr zwischen den Knoten, jeder Knoten ist ein *Kanalprozessor* und enthält einen Speicher für die übertragenen Daten. Zwischen Anbieter und Nutzer gibt es eine dauerhafte Verbindung, daher wird eine Speicherung der den Kanal definierenden Meta-Daten und der über den Kanal übertragenen Daten benötigt.
- Kanalobjekte sind im Grunde nichts anderes als HTML-Seiten, die um Meta-Daten zur Kontrolle der Kommunikation zwischen den Kanalprozessoren erweitert wurden. Diese Meta-Daten sollen die Auslieferung und Verarbeitung der Daten automatisieren. Die Gesamtheit dieser Meta-Daten, die einen Anbieter oder Nutzer beschreiben, nennt man auch *Profile*.
- Um diese Meta-Daten verarbeiten zu können, müssen die Kanalprozessoren entsprechende Methoden enthalten. Diese Methoden übertragen, empfangen und verarbeiten die Nutzdaten.

Bild 12 zeigt nur zwei Knoten, doch können zwischen dem eigentlichen Empfänger und Sender beliebig viele Knoten liegen. Die Art der Beziehung zwischen den Knoten ist bidirektional und kann eine 1:1-, 1:N- oder M:N-Beziehung sein, wobei N und M theoretisch beliebig groß werden können.

Es muß betont werden, daß Bild 12 nur *ein* Beispiel für eine derartige Architektur ist. Jede Firma hat ihr eigenes, speziell an ihre Bedürfnisse angepaßtes Modell, das ihrem Push-Verfahren zugrunde liegt.

3 Aufgaben von Server und Client

Die Kanäle bilden das gedankliche Modell, das den Verbindungen zwischen Anbietern von Daten und ihren Nutzern zugrunde liegt. Auf Anbieterseite werden Daten auf einem Server zur Verfügung gestellt, und der Nutzer erhält die gewünschten Daten durch den Einsatz eines Clients.

3.1 Server

Der Server hält die Daten bereit, die vom Client angefordert werden können, aber auch Meta-Daten, die die Verwaltung dieser Daten und der Nutzer betreffen (diese *Profiles* werden weiter unten ausführlich behandelt).

Die zu einem Kanal gehörenden Daten werden oft in einem Verzeichnis(baum) zusammengefaßt. Bei einer der periodischen Anfragen des Clients werden dann die geänderten oder wahlweise alle Daten übertragen.

3.2 Client

Es gibt prinzipiell zwei Möglichkeiten, auf welche Weise aktuelle Daten auf den Rechner des Nutzers kommen: Entweder der Client fordert den Server periodisch zur Übertragung auf, oder der Client wird bei einer Aktualisierung der Daten vom Server informiert, um dann eine Übertragung anzufordern. Am meisten verbreitet ist erstere Vorgehensweise (z.B. bei BackWeb [Bac97]).

Um den Nutzer bzw. dessen Rechner jedoch nicht bei seinen eigentlichen Aufgaben zu behindern, warten Push-Clients i.a., bis der Netzanschluß des Nutzers ruht. Dadurch laufen Anforderung und Übertragung im Idealfall vollständig im Hintergrund ab, unbemerkt vom Nutzer. Wird der Netzanschluß während der Übertragung wieder aktiv, unterbricht der Client die Übertragung für die Dauer der Nutzeraktivität und setzt sie dann an der Unterbrechungsstelle fort [Bac97, Mar97].

Um dem Nutzer stets eine konsistente Sicht auf seine Daten zu ermöglichen, macht der Client dem Nutzer die aktualisierten Daten erst nach ihrer vollständigen Übertragung zugänglich.

Bei Nutzern, die nicht über eine dauerhafte Verbindung zum Internet verfügen, kann der Client den jeweiligen Provider des Anschlusses automatisch anwählen und dann Kontakt

zum Push-Server aufnehmen. Je nach Konfiguration des Client können dabei nicht unerhebliche Zusatzkosten entstehen (Telefon, Internet-Nutzung usw.), die möglicherweise nicht den tatsächlich Bedarf des Nutzers an den erhaltenen Daten widerspiegeln.

Daten, die ein Client erhält, werden lokal nicht wie bei einem Browser in einem Cache gehalten, sondern gespiegelt. Dadurch liegen einmal übertragene Daten bis zur nächsten Aktualisierung auf Client-Seite vor und müssen für beliebig häufige Zugriffe nicht wiederholt übertragen werden. Das bedeutet aber auch, daß diese Daten auf jeden Fall übertragen werden, ganz gleich, ob und wie oft sie der Nutzer überhaupt verwendet. Von einer bedarfsabhängigen Kommunikation zwischen Client und Server sind aktuelle Push-Verfahren also weit entfernt.

Standardmäßig sind Clients so konfiguriert, daß nicht alle Daten, die beim Server vorliegen übertragen werden, sondern nur die, die sich seit der letzten Übertragung geändert haben. Alternativ wäre vom Nutzer stets die Übertragung aller vorliegenden Daten einstellbar.

3.3 Profiling

Um überhaupt ein Profile erstellen zu können, muß der Client die entsprechenden Daten zum Server übertragen. Auch das ist ein Unterschied zum herkömmlichen Pull-Verfahren, bei dem der Datenfluß nur in Richtung Client geht[3]. Im einfachsten Fall enthält das Profile lediglich technische Angaben: welche Kanäle abonniert sind, in welchen Abständen sich der Client beim Server meldet, die maximale Datenmenge pro Übertragung usw. Fortgeschrittene Systeme ermöglichen dem Nutzer eine nähere Spezifizierung der gewünschten Daten.

Deutlich darüber hinaus gehen Systeme wie das von BackWeb (s. Abschnitt 9, [Bac97]). Hier ist es prinzipiell für den Server möglich, Informationen vom Client über das Nutzerverhalten zu bekommen: wie lange der Nutzer über das Vorliegen neuer Daten informiert wurde, wie lange der Nutzer sich überhaupt mit den Daten beschäftigt hat, ob der Nutzer zu einer beworbenen WWW-Seite gegangen ist uvm. Inwieweit diese Art der Datenerfassung für den *Nutzer* vorteilhaft ist, bleibt wohl fraglich.

3.4 Filterung

Zur Filterung der vom Server zu übertragenden Datenmenge könnte ein Profile beispielsweise bestimmte Stichwörter enthalten, die das Interesse des Nutzers an bestimmten Themen eines Informationskanals widerspiegeln. Auf diese Weise soll eine möglichst individuelle Sicht auf einen Kanal ermöglicht werden. Diese „Individualisierung“ des Kanals steckt allerdings noch in den Kinderschuhen (s. Abschnitt 10, [KSK97]).

Das Profile wird nach der Erstellung bzw. bei einer Aktualisierung vom Client zun Server übertragen. Bei den zukünftig zu erwartenden mächtigeren Filterungsmöglichkeiten hätte das den Vorteil, daß nur die Daten zum Nutzer übertragen werden, die dieser mit hoher Wahrscheinlichkeit auch wünscht. Ändert sich das Profile jedoch oft, wie zum

[3] Cookies bilden beim Pull-Verfahren eine, wenn auch umstrittene Ausnahme.

Beispiel durch automatische Anpassung an das Leseverhalten des Nutzers bei einem Informationskanal [KSK97], hätte diese Vorgehensweise eine häufige Übertragung der Profiles unzähliger Nutzer zu ihrem jeweiligen Server zur Folge.

Filterung wäre daher auch beim Client möglich, doch setzte das voraus, daß stets alle neuen Daten zunächst zum Client übertragen werden, ganz gleich, ob diese überhaupt gewünscht werden. Was wiederum eine immense Netzlast zur Folge hätte.

4 Datenübertragung

Zur Übertragung der Daten zwischen Client und Server benutzen Push-Systeme als Kommunikationsprotokoll in der Regel TCP, seltener UDP. Auf TCP bzw. UDP setzt dann normalerweise ein vom jeweiligen Anbieter selbst entwickeltes Protokoll auf, das den genauen Ablauf steuert (Warten auf Ruhen des Netzanschlusses, Wiederaufsetzpunkte usw.).

4.1 Unicast

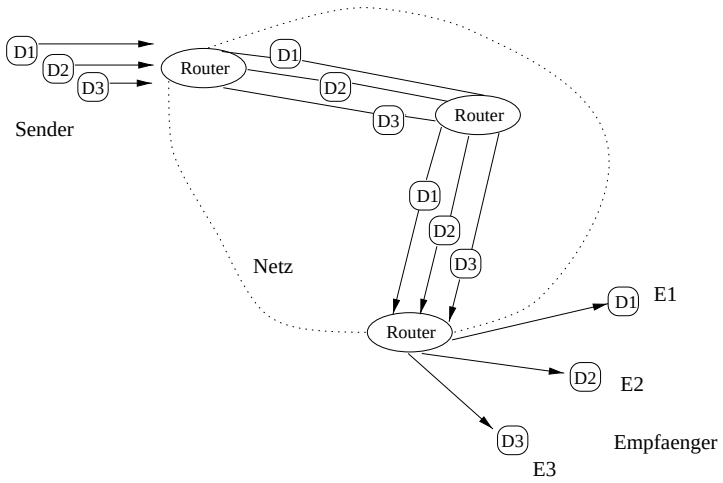

Abbildung13. Datenübertragung bei Unicast.

Bei der *Unicast* genannten Art der Dantenübertragung wird, wie in Bild 13 dargestellt, ein separater Datenstrom zu jedem einzelnen Nutzer geschickt, auch wenn jeder dieselbe Nachricht bekommen soll. Diese Vorgehensweise führt bereits heute bei noch nicht allzu großer Verbreitung von Push-Systemen zu einer erheblichen Belastung der Netze.

Nach einer Analyse des Internet-Verkehrs durch Optimal Networks verursachte PointCast, der führende Anbieter von Push-Systemen und Nachrichtenkanälen, bereits 1996 eine Belastung von 18%, obwohl nur 12% der Internet-Nutzer diesen Dienst überhaupt in Anspruch nahmen [GC97]. Auch wenn es sich dabei nicht um 18% des gesamten Internet-Verkehrs, sondern „nur“ um 18% des durch das WWW verursachten Verkehrs im Internet handelt, läßt diese Studie erkennen, wie groß die Netzlast sein könnte, wenn Push-Systeme erst einmal eine weite Verbreitung erfahren. Denn schon heute hat das

WWW einen Anteil von ca. 40% am Internet-Verkehr, womit also PointCast allein auf einen Anteil von ca. 7,2% am gesamten Internet-Verkehr kommt.

Bei einer 1:1-Verbindung zwischen Server und Client, wie es beim Pull der Fall ist, stellt die Datenübertragung mit einem separaten Datenstrom den einzig effektiven Weg dar, die Daten zu übertragen. Diese Art der Datenübertragung wird aber dann ineffektiv, wenn 1:N-Verbindungen vorliegen. Die dann durch Unicast erzeugte (überflüssige) Netzlast kann durch das sogenannte *Multicast* verhindert werden.

4.2 Multicast

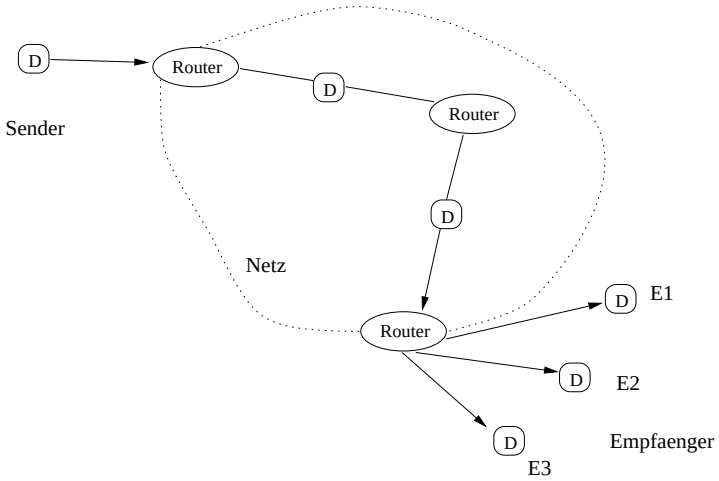

Abbildung14. Datenübertragung bei Multicast.

Multicast [JJ97a, JJ97b] unterstützt 1:N-Verbindungen, indem es die Übertragung einer einzigen Nachricht an eine Nutzergruppe ermöglicht, statt jedem Mitglied dieser Gruppe eine einzelne Nachricht zu senden.

Bild 14 zeigt den Unterschied zum herkömmlichen Unicast: Die Nachricht wird erst dann vervielfältigt, wenn es nötig wird, sie auf verschiedenen Wegen zu übertragen. So würde beispielsweise ein vierter Empfänger, der direkt hinter dem ersten Router sitzt, das Multicast-Protokoll veranlassen, eine weitere Kopie des Datenpakets zu diesem Empfänger zu schicken. Davon unbeeinflußt bliebe die im Bild zu sehende Übertragung des Datenpakets für die Empfänger E1 bis E3.

Der Vorteil dieser Erweiterung des Internet-Protokolls liegt auf der Hand – die Belastung der Netze könnte sich in vertretbaren Grenzen halten. Die für die Einführung von Multicast-Routern nötige Technik ist schon seit einiger Zeit einsetzbar. Allerdings sind solche Router noch nicht weit verbreitet. Die Begründung klingt nach einem Teufelskreis: Die Hardware-Investition wird ohnehin gescheut, zudem gibt es keine verbreiteten Anwendungen, die Multicast benötigen. Da aber die für eine entsprechende Anwendung benötigte Infrastruktur nicht vorhanden ist, werden auch keine Programme entwickelt, die auf Multicast aufsetzen.

Um die Netzlast in einem vertretbaren Rahmen zu halten und eine dauerhafte Akzeptanz

seitens der Netzbetreiber zu erreichen, werden Push-Verfahren auf längere Sicht nicht auf Multicast verzichten können.

4.3 Proxies

Eine andere Möglichkeit, unnötiges Datenaufkommen zu verringern, bieten Proxies. Dabei handelt es sich um Rechner, die eine Verbindung zwischen dem Internet und einem anderen, i.a. von einer Firma intern betriebenen Netz (*Intranet*) herstellen. Proxies dienen normalerweise als Schutz vor unbefugtem Zugriff von außen auf das Intranet (*Firewalls*), können URLs filtern und den Zugriff von innen auf das Internet protokollieren, haben aber auch Cache-Funktion für WWW-Seiten, die interne Nutzer aus dem Internet abfragen.

Gerade diese Cache-Funktion machen sich manche Anbieter von Push-Systemen zunutze. So soll nach ihrer Auffassung ein Push-Proxy installiert werden, der dann den Datenverkehr zwischen dem Intranet und dem Internet deutlich reduzieren soll. Auch für das Internet bringen Proxies eine erhebliche Entlastung. Hat eine große Zahl von Nutzern am Intranet dieselben Kanäle abonniert, müssen die fraglichen Daten nicht bei jeder Anforderung neu vom Server des Kanal-Anbieters übertragen werden, sondern liegen im Kanal-Cache des Proxies bereit. Gegenüber herkömmlichen Proxies können Push-Proxies außerdem Verbindungen zu den Servern der Kanal-Anbieter aufrechterhalten und die Anzahl der Verbindungen optimieren.

Alle namhaften Anbieter von Push-Systemen (z.B. Marimba, BackWeb) haben für ihre Systeme Proxies angekündigt. Filterung mit Proxies ist allerdings bisher für die Anbieter noch kein Thema. Als problematisch erweist sich in der Praxis aber noch die Einbindung in vorhandene Firewall-Systeme. Außerdem sieht es so aus, als ob diese Proxies Multicast nicht unterstützen.

5 Anwendungsmöglichkeiten

Das erste Push-System auf dem Markt war Anfang 1996 PointCast, das auch heute noch das mit Abstand am weitesten verbreitete System ist [GC97]. Im Laufe der Jahre 1996 und 1997 folgten dann weitere Anbieter, von denen Marimba (Castanet), BackWeb und Intermind die bekanntesten sein dürften.

5.1 Kommerzielle Verwendung

Informationsdienste Die meisten der heute angebotenen Push-Systeme zielen auf die Verteilung von Informationen, die normalerweise in beliebiger Form (HTML, Graphiken, Audio, Video usw.) dargestellt werden können. Nutzer abonnieren Nachrichtenkanäle, durch die sie dann mit aktuellen Informationen versorgt werden.

Hier findet sich der Schwerpunkt des Angebots. Einige Anbieter von Push-Systemen sind gleichzeitig auch Anbieter von Nachrichten- oder sonstigen Informationskanälen. Diese Kanäle sind größtenteils kostenlos nutzbar, allerdings um den Preis der Werbung, die man sich mit der Kanalnutzung auf den Monitor holt.

Software-Verteilung Eine weitere Einsatzmöglichkeit von Push-Systemen bietet sich für die Software-Verteilung und -Aktualisierung. Dieser Bereich ist im Vergleich zur Informationsverteilung bis jetzt eher schwach entwickelt, aber beispielsweise nutzt die Firma McAfee (Virenschutz) einen BackWeb-Kanal zur Aktualisierung ihrer Virenliste beim Nutzer.

Durch diese Art der Software-Aktualisierung wollen die Hersteller einerseits Kosten und Verwaltungsaufwand, andererseits aber auch den Installations- und Konfigurationsaufwand der Anwender senken.

Weiterhin sollen die Anwender auf diese Weise stets die aktuellste Version eines Programms nutzen können, also auch von einer Behebung etwaiger (schwerwiegender) Fehler, beispielsweise Sicherheitslücken, sofort profitieren können.

Hier wird auch die Netzbelastung in Grenzen gehalten, denn in der Regel müssen zur Aktualisierung eines Programmes nicht alle Dateien neu übertragen werden. Aber wesentlicher ist, daß im Vergleich zu Informationsdiensten die Aktualisierung in deutlich größeren zeitlichen Abständen auftritt.

5.2 Firmeninterne Verwendung

Innerhalb von Firmennetzen (Intranets) können Push-Systeme ebenfalls zur Verteilung aktueller Informationen und zur Software-Aktualisierung zum Einsatz kommen.

Der Vorteil eines internen Informationskanals wäre, daß neue Informationen sofort diejenigen Mitarbeiter erreichen könnten, für die sie von Nutzen sind. Allerdings muß der entsprechende Kanal auch verwaltet werden, verursacht also wiederum Aufwand. Außerdem existieren seit Jahren Groupware-Systeme und E-Mail-Listen, die diese Aufgabe ebenfalls erfüllen. Groupware wie Lotus' Domino kann beispielsweise die Ergebnisse einer Datenbankabfrage in HTML-Seiten konvertieren und an den Nutzer senden (s. Abschnitt 8).

Bei der Aktualisierung verbreiteter Programme könnten die EDV-Abteilungen einer größeren Firma durch Push-Systeme beträchtlich Zeit und Aufwand sparen. Die Anwendung von Push-Techniken könnte dabei allerdings zu einer Änderung der Unternehmenskultur führen, da auch andere Bereiche wie z.B. das Accounting, die interne Abrechnung von Dienstleistungen der Abteilungen untereinander, stark beeinflußt würde.

Ein weiterer möglicher Einsatz könnte das Change Management sein, die Verwaltung und Planung von Änderungen jeglichen Umfangs in einem Unternehmen. Push-Server könnten hier helfen, bestimmte Änderungen schneller zu realisieren.

Die Firma Connect (www.connect.com) integriert Push-Techniken in ihre Fernwartungs-Software. Mit diesen Programmen können verteilt installierte Server zentral verwaltet werden. Durch den Einsatz von Push-Technologien in diesem Bereich soll die Fernwartung einer großen Anzahl Server erleichtert werden, die die gleichen Daten anbieten. Dabei wird vor allem an die Aktualisierung von Servern gedacht, die kommerzielle Angebote bereithalten.

6 Ausgewählte Anbieter von Push-Verfahren

Zur Zeit drängen mehr als ein Dutzend Firmen auf den heiß umkämpften Markt rund um Push-Systeme und Informationskanäle.

So unterschiedlich die Möglichkeiten dieser Systeme auch sind, gemeinsam ist allen, daß sie im höchsten Maße proprietäre Lösungen darstellen: Ein Server kann nur mit einem Client aus gleichem Hause kommunizieren. Von einem Standard, der diesen Mißstand behebt, ist man noch weit entfernt.

Der Markt teilt sich grob in zwei Gruppen: Die Anbieter von Push-Systemen und die Anbieter von Push-Systemen und dazugehörenden Kanälen. Ebenso, wie man keine Server und Clients unterschiedlicher Anbieter miteinander kombinieren kann, so können auch die Push-Systeme eines Anbieters nicht die Kanäle der Konkurrenz nutzen.

Im folgenden werden drei Systeme kurz vorgestellt: Marimba ist reiner System-Anbieter (Castanet) [Mar97], wohingegen BackWeb auch Informationskanäle bereitstellt [Bac97]. WebCanal ist ein neues, auf Multicast aufsetzendes System [Lia97]. Für eine vertiefende Betrachtung dieser und anderer System wird auf die Literatur verwiesen.

In den letzten Wochen mehreren sich die Berichte über Push-Ergänzungen zu Browsern. Dabei handelt es sich lediglich um Verfahren, die in vom Nutzer festgelegten Abständen kontrollieren, ob sich angegebene WWW-Seiten geändert haben. Deswegen soll hier nicht näher auf Browser-Erweiterungen eingegangen werden.

6.1 Marimbas Castanet

Bei Marimbas Castanet (www.marimba.com) [Mar97] wird vom Server (Transmitter) ein Kanal zur Verfügung gestellt, der von einem Client (Tuner) empfangen werden kann. Zur Zeit können mit Castanet nur Java-Programme übertragen werden, für deren kontrollierte Ausführung dann der Client zuständig ist. Für einen späteren Zeitpunkt ist nicht nur die Aktualisierung von Java-Programmen, sondern von jeglicher Software angestrebt.

Ein Kanal ist für Marimba ein mehrteiliges ausführbares Java-Programm. Marimba bietet auch ein *Java Channel API* an, mit dem der Client in der Lage sein soll, Kanäle (Programme) zu kontrollieren wie ein Browser Java-Applets. Castanet setzt dabei auf dem Java-Sicherheitsmodell auf, erlaubt jedoch dem ausgeführten Programm das Beschreiben bestimmter, vorher festgelegter Verzeichnisse.

Zum Castanet-System gehören neben Transmitter, Channel und Tuner auch Repeater und Proxies: Repeater als „selbstaktualisierende Transmitter“ sollen zur Reduzierung der zu übertragenden Datenmenge eingesetzt werden. Sie abonnieren wie ein Tuner nur bestimmte Kanäle und geben an die hinter ihnen liegenden Repeater, Proxies oder Tuner nur noch die gewünschten Daten weiter.

Castanet-Proxies sollen die bereits beschriebenen Aufgaben eines Push-Proxy (s. Abschnitt 4.3) mit denen einer Firewall kombinieren.

Bei der Datenübertragung ist Castanet ein verbindungsorientiertes Unicast-System und läuft im Hintergrund: Übertragen werden nur geänderte Daten und nur dann, wenn der

Netzanschluß des Rechners nicht aktiv ist. Bei einer Unterbrechung wird die Übertragung an der Unterbrechungsstelle fortgesetzt.

6.2 BackWeb

Das BackWeb-System (www.backweb.com) [Bac97] besteht aus der Channel Software (Server), der Server Console zur Verwaltung der Kanäle, dem BackWeb-Client und einer Scriptsprache.

Kanäle können bei BackWeb beliebige Daten umfassen: HTML-Seiten, Graphiken, Audio, Video, Programme usw. Dabei können öffentliche und private Kanäle angeboten und für einen Kanal verschiedene Nutzergruppen festgelegt werden, so daß Anwender je nach Gruppenzugehörigkeit unterschiedliche Daten bekommen.

Öffentliche Kanäle sind zum Beispiel Informationskanäle. Mitte 1997 wurden ca. 50 öffentliche Kanäle angeboten, darunter die Suchdienste Infoseek und Yahoo!, aber auch Nachrichten und Computer-Magazine. Nach eigenen Angaben wollte BackWeb bis Ende 1997 über 1000 Kanäle anbieten. Aktuelle Zahlen sind nicht bekannt.

Die Möglichkeiten der Software-Aktualisierung sind allerdings nicht mit denen von Castanet vergleichbar. So gibt es keine Möglichkeit, übertragene Programme kontrolliert auszuführen und bisher werden nur aktualisierte Daten übertragen wie bei McAfees Virenschutzprogramm.

Auch BackWeb setzt auf Unicast auf und wickelt die Datenübertragung wie bei Castanet beschrieben ab. Ein Proxy ist allerdings bisher erst angekündigt.

Zwischen Microsofts und BackWeb gibt es eine Vereinbarung, nach der die Version 4.0 des Internet Explorer einen BackWeb-Client enthält.

6.3 INRIAs WebCanal

INRIA (www.inria.fr) ist das nationale französische Forschungsinstitut für Informatik, das mit einem nach eigenen Angaben wettbewerbsfähigen Push-System aufwartet, das auf Multicast aufsetzt [Lia97]. Mit diesem Ansatz will man die in Abschnitt 4 beschriebenen Probleme lösen, um so eine zügigere Verbreitung der Push-Verfahren zu erreichen.

Dabei entwickelt man für die Datenübertragung keine eigenen Protokolle, sondern benutzt offene Standards, sofern sie existieren. Gegenwärtig werden diverse Standards zur Ankündigung neuer Kanäle, zur Verbreitung von Kanalinhalten und zur Datenübertragung (hier LRMP – Light-weight Reliable Multicast Protocol) benutzt.

INRIA sieht das Fehlen von Standards im Bereich der Push-Verfahren als den wichtigsten Grund für eine fehlende allgemeine Akzeptanz dieser Verfahren an. Dabei werden Versuche mancher Anbieter, die eigenen, noch dazu Unicast-basierten Verfahren zu standardisieren, als Schritt in die falsche Richtung angesehen, weil auf diese Weise keine offenen Standards geschaffen werden.

Im Kanal-Modell des WebCanal-Systems existieren zwei unterschiedliche Arten von Kanälen: Neben dem reinen *Info-Channel* gibt es den *Directory-Channel*, der niederratig Beschreibungen und Ankündigungen der Info-Channel überträgt. Dieser zentrale

Kanal soll Nutzern Informationen über verfügbare, geänderte und neue Kanäle liefern und somit eine Verbindung zu einem zentralen Server überflüssig machen.

Ein Kanal kann ähnlich wie bei BackWeb beliebige Inhalte haben. Nähere Informationen über Einschränkungen waren nicht verfügbar. Die Auswertung der übertragenen Daten soll dabei allein vom Nutzer abhängig sein. Das heißt, daß ein kontrolliertes Ausführen von übertragenen Programmen ebensowenig möglich ist wie bei BackWeb.

WebCanal ist das erste verfügbare Push-System, das auf Multicast aufsetzt und über die Verwendung offener Standards den Nutzer aus der Abhängigkeit bestimmter Anbieter befreit. Ein praktischer Nutzen ergibt sich allerdings erst dann, wenn genügend Anbieter von Push-Systemen ebenso wie Anbieter von Kanälen diesen Weg gehen. Bis dahin ist WebCanal lediglich eine Insellösung unter vielen. (Zu dieser Problematik s.a. Abschnitt 9.)

6.4 Kosten

Bisher sind die Kosten von Push-Systemen unerwähnt geblieben. Im allgemeinen ist der Client kostenlos verfügbar, wer jedoch einen eigenen Server installieren möchte, muß tief in die Tasche greifen: 10.000 Dollar und mehr sind keine Seltenheit, meist auch Staffelpreise in Abhängigkeit von der möglichen Anzahl der Kanäle, die mit diesem Server angeboten werden können. Bei Proxies gibt es sowohl kostenlose Angebote als auch kostenpflichtige und damit oft teure Lösungen.

7 Standardisierungsbemühungen

Wie bereits erwähnt, fehlen Standards für Push-Verfahren entweder oder werden von den Anbietern nicht benutzt, so daß Clients nur mit dem zugehörigen Server kommunizieren können.

7.1 Channel Definition Format

Bei den Kanälen bietet sich ein ähnliches Bild: Ein PointCast-Client beispielsweise ist nicht in der Lage, die Inhalte eines BackWeb-Kanals zu empfangen.

Um dem abzuhelfen, bemüht sich Microsoft derzeit, sein Channel Definition Format (CDF) [Ell97] beim WWW Consortium (W3C, www.w3c.org) als WWW-Standard anerkannt zu bekommen. CDF soll dazu dienen, Inhalte von Kanälen Anbieter-übergreifend nutzbar zu machen [Par97].

Der Kanal-Administrator erzeugt nach der Vorstellung von Microsoft hierbei für jeden Kanal ein CDF-Dokument, in dem beschrieben wird, welche Objekte zum jeweiligen Kanal gehören und in welchen Intervallen der Kanal auf Änderungen überprüft werden soll. Der Bezug zwischen CDF-Dokument und Objekt wird dabei durch die URLs der Objekte hergestellt. Damit benötigt der Nutzer nur noch den URL der zu einem Kanal gehörenden CDF-Datei, um auf dessen Inhalte zugreifen zu können.

Bei CDF handelt es sich um ein einfaches, textuelles Format, das auf SGML (Standard Generalized Markup Language) basiert, einer Obermenge des für die Beschreibung von WWW-Seiten verwendeten HTML.

Microsoft bietet mit CDF keinen wirklichen Fortschritt in der Push-Technik. Es handelt sich lediglich um eine Standardisierung, auf welche Weise Kanäle benutzt werden. Vielen Anbietern von Push-Systemen wäre dieser Standard außerdem nicht mächtig genug, und sie haben daher zwar Akzeptanz signalisiert, gleichzeitig aber eigene Erweiterungen angekündigt. Ob ein Standard in diesem Fall noch Sinn macht, sei dahingestellt.

Zudem ist dieser Standard auf das World Wide Web ausgerichtet und nicht allgemein einsetzbar, da die Übertragung beliebiger Inhalte nur durch URLs möglich ist. Auch bleibt generell fraglich, ob die Anbieter von Kanälen und Push-Systemen überhaupt ein Interesse an einem derartigen Standard haben, da sie dann ihre Kunden nicht mehr wie bisher an sich binden könnten (s. Abschnitt 9).

7.2 Open Software Distribution

Ein weiterer Standardisierungsvorschlag von Microsoft in Kooperation mit Marimba beschäftigt sich mit Software-Verteilung. Unterstützt wird der Vorschlag auch von Netscape und Lotus.

Open Software Distribution (OSD) beruht auf der Extensible Markup Language (XML) und soll helfen, automatisierte Software-Verteilung und -Aktualisierung im großen Stil zu ermöglichen. Ein OSD-Dokument beschreibt Software-Komponenten durch Strukturen, Versionen, Beziehungen usw.

Da es sich dabei noch um eine recht junge Idee handelt, waren ausführlichere Informationen zum Zeitpunkt der Recherche nicht zu bekommen.

8 Alternativen zu Push-Verfahren

Speziell innerhalb von Unternehmen sind Push-Systeme zur Informationsverteilung nicht unbedingt notwendig. Besonders, wenn es nur darum geht, textuelle Nachrichten an einen begrenzten Nutzerkreis zu verschicken, bewähren sich E-Mail-Listserver schon seit Jahren. Dabei wird eine Nachricht, die an alle bei einer Liste registrierten Nutzer gesendet werden soll, an den Server geschickt, der dann die Verteilung gemäß der eingetragenen E-Mail-Adressen übernimmt.

Bei diesem Ansatz wird allerdings, wie bei E-Mail üblich, an jeden Adressaten eine eigene Nachricht geschickt. Es ist auch möglich, an eine E-Mail andere Dateien anzuhängen, die dann vom Empfänger mit entsprechender Programme dargestellt und bearbeitet werden können. Das führt allerdings zu einem höheren Aufwand beim Nutzer, da dieser die Informationen mit zwei getrennten Systemen bearbeiten muß, die sich nicht integrieren lassen.

Per E-Mail lösen manche Anbieter von WWW-Seiten auch längst das Problem, wie man Interessenten von Änderungen der WWW-Seiten in Kenntnis setzt. Der Interessent füllt einfach ein Online-Formular aus, dem der Server die benötigten Daten entnimmt. Bei

einer Aktualisierung der WWW-Seiten werden alle registrierten Nutzer durch eine Mail auf die Änderung hingewiesen.

Für die Kommunikation innerhalb von Unternehmen kommt heute bereits vielerorts Groupware zum Einsatz. Dabei handelt es sich um Software, die Kommunikation und Koordination zwischen den Teilnehmern einer Gruppe ermöglicht. Die aktuellen Versionen von verbreiteten Groupware-Systemen wie beispielsweise Lotus Domino integrieren bereits Internet-Software. Domino kann die Ergebnisse von Datenbankenabfragen in HTML-Seiten umwandeln und ermöglicht so das Betrachten der Ergebnisse auf einem Browser. Sicherheitsfragen werden von den Marktführern längst gelöst: Verschlüsselung und Authentifizierung sind Teil des Systems. Auf diese Zusätze wird man bei herkömmlichen Push-Systemen noch lange warten müssen, aber für die Nutzer öffentlicher Nachrichtenkanäle dürfte Verschlüsselung ohnehin kein primäres Anliegen sein. In Unternehmen, besonders wenn die Daten über Extranets übertragen werden sollen, können Fragen der Sicherheit allerdings zum entscheidenden Kriterium werden.

9 Probleme mit Push-Systemen

Eines der wesentlichen Probleme wurde bereits mehrfach angesprochen und soll daher hier nicht wieder vertieft werden: die Belastung für die Netze. Zwar gibt es eine Lösung (Multicast), doch fehlen zur Umsetzung bisher die Infrastruktur und der Wille der System-Anbieter.

Push-Systeme binden die Nutzer zu sehr an den Anbieter des jeweiligen Systems. Die Entwicklung von Standards bzw. die Nutzung existierender könnte hier Abhilfe schaffen. Es ist aber eher so, daß die Anbieter ein Interesse daran haben, Nutzer an sich zu binden. Das trifft vor allem für diejenigen Anbieter von Push-Systemen zu, die gleichzeitig auch (kommerzielle) Informationskanäle anbieten. Wer Kanäle verschiedener Anbieter nutzen möchte, hat im Moment keine andere Wahl, als mehr als einen Push-Client auf seinem Rechner zu installieren. Eine Situation, die dem Nutzer alles andere als gerecht wird. Auch hilft das Ausweichen auf unabhängige Informationsanbieter nicht, denn diese haben sich mit der Bereitstellung ihrer Kanäle auch für eines oder wenige der verfügbaren Systeme entschieden.

Wie sehr der Markt im Moment umkämpft ist, zeigt sich in der Konzentration vieler Produkte auf das Internet bzw. WWW. In Intranets sind die meisten der verfügbaren Systeme nicht vorteilhaft anzuwenden, an ihrer Stelle ist eine einfache E-Mail-Liste oft effizienter (s. Abschnitt 8). Ernsthafte Anwendungen in Unternehmen werden zwar propagiert, zur Zeit steht aber ganz der einzelne Nutzer im Zentrum des Interesses.

Folgerichtig kommen alle Systeme zuerst als Windows95- oder WindowsNT-Version heraus. Teilweise beträchtlich später oder auch gar nicht folgen Macintosh- oder Unix-Versionen. Zwar ist Marimbas Castanet in Java geschrieben, doch ist die systemunabhängige Installation dadurch nicht in jedem Fall gewährleistet [Kir97]. Das Interesse der Anbieter am einzelnen Nutzer läßt sich auch durch den hohen Aufwand erklären, den die Bereitstellung der Kanäle bedeutet. Die Kosten für den Verwaltungsaufwand lohnen sich erst dann, wenn die Inhalte eine möglichst große Verbreitung erfahren. Dabei spielen

auch lukrative Abschlüsse über ebenfalls übertragene Werbung eine große Rolle.

Die Benutzerfreundlichkeit vieler Systeme läßt noch sehr zu wünschen übrig. Meistgenannter Kritikpunkt: die eingeschränkte Konfigurationsmöglichkeit des Client [Hei97]. So ist es bei Castanet beispielsweise nicht möglich, die zu übertragende Datenmenge zu beschränken. Andere Systeme weisen ähnliche Einschränkungen auf – teilweise ist die Aktualisierungsrate nur in engen Grenzen frei einstellbar.

Zur Zeit werden zwar Systeme und Verfahren in jedem Computer-Magazin diskutiert und verglichen, über die in den Kanälen angebotenen Inhalte liest man jedoch so gut wie nichts. Fast scheint es so, als habe man mit der Push-Technik eine Lösung gefunden, zu der man nun erst das Problem anbieten müsse. Ein Vergleich von tatsächlichen und propagierten Nutzungsmöglichkeiten sieht eher traurig aus: Wer alle zehn Minuten die neuesten Nachrichten oder die weltweiten Börsenkurse wissen möchte, wird gut versorgt. Über Nachrichten und Wetterdaten hinaus entwickeln sich erst langsam Nutzungsmöglichkeiten, so daß der Einsatz von Push-Systemen zur eigenen Information eher einer Spielerei gleicht.

Nicht ganz unbegründet ist die Befürchtung, die Nutzer könnten durch unkontrollierte Massensendungen, vornehmlich Werbung, geradezu überschwemmt werden. Viele Informationskanäle werden durch mitgelieferte, nicht filterbare Werbung kostenfrei gehalten, um beim Nutzer größere und schnellere Akzeptanz zu erreichen. Die mit der Kanalnutzung einhergehende Profile-Erstellung wird in Zukunft immer detaillierter werden, so daß Unternehmen durch Nutzung dieser Profiles zielgerichteter ihre Produkte bewerben könnten.

Speziell BackWeb zeigt bereits heute, was mit Push-Systemen möglich ist: Der Client liefert an den Server zurück, wie lange der Nutzer benötigte, bis er auf die Meldung einer Nachricht reagierte, wie oft er sich überhaupt mit dem Inhalt beschäftigt hat, wie oft komplett, ob und auf welche Werbung er ansprach. Schon fast zynisch mutet der im selben Dokument stehende Absatz über „private personalization“ an, die mit BackWeb problemlos und vertrauenswürdig möglich sei [Bac97].

10 Ausblick

Push-Verfahren stehen noch immer ganz am Anfang. Das bereits vorgestellte Multicast ist nur eine Richtung, in die sich Push-Verfahren in Zukunft entwickeln werden. Hier folgt eine Aufstellung interessanter Aspekte, die Push-Systeme verändern werden:

- Es gibt bereits erste Überlegungen zur Integration mobiler Nutzer und die Anpassung der Push-Verfahren an Netze niedriger Bandbreite [Uma97]. Wie wichtig speziell die Integration mobiler Nutzer sein wird, hängt aber in erster Linie von der allgemeinen Akzeptanz ab, die sich wiederum durch andere Aspekte entschiedet.

- Mit der „Personalisierung“ der Push-Systeme ist es heute noch nicht weit her. Neue Möglichkeiten bei Such- und Filterfunktionen sind angekündigt, insgesamt steckt die Profile-Generierung aber noch in ihren Kinderschuhen. Selbst auf dem

W3C-Workshop vorgestellte Verfahren sind noch keineswegs besonders fortschrittlich [KSK97].

- Die bei der Profile-Gewinnung anfallenden Informationen werden auf Server-Seite nicht nur zur Filterung des Datenstroms genutzt. Wer die Vorlieben und Abneigungen seiner Kunden kennt, kann sie direkter und mit speziell auf sie abgestimmter Werbung versorgen. Eventuell könnte sich auch ein Markt für Profiles entwickeln, wie es ihn heute schon bei bestimmten Versandgeschäften gibt. Offen und vermutlich auf der Strecke bleiben dabei natürlich Fragen zum Daten- und Persönlichkeitsschutz.

- Microsoft ist bestrebt, seinen Browser in die Betriebssystemoberfläche zu integrieren und so dem Nutzer eine einheitliche Sicht auf die lokalen Daten und das WWW zu geben. Dadurch würde auch das mitgelieferte Push-System zum Bestandteil der Oberfläche. Durch entsprechende Kooperationsverträge sichern sich dabei einige der System-Anbieter (z.B. BackWeb) einen deutlichen Marktvorteil gegenüber der Konkurrenz. Vorausgesetzt, auch das inhaltliche Angebot stimmt.

- Naht wirklich, wie so oft angekündigt, das Ende der Browser? Momentan scheinen Push-Systeme den Browsern nur im Unterhaltungssektor Konkurrenz zu machen – und da nur bedingt beim heutigen Stand der Technik. Nur die Bereiche, in denen großes Publikumsinteresse zu erwarten ist, werden in absehbarer Zeit verstärkt auf Push-Verfahren bauen. Bereiche, die man eher als spezialisiert bezeichnen könnte, werden zunächst nicht kosteneffizient mit Push arbeiten können (s. Abschnitt 9).

- Die Firma Intermind (www.intermind.com) hat im Push-Bereich das Kassieren von Lizenzgebühren als Einnahmequelle für sich entdeckt [Int97]. Auf ihren WWW-Seiten stellt sie den Umfang der von ihr beanspruchten Kommunikationspatente dar. Danach würde das sehr allgemein gehaltene Patent alle Vorgänge der Push-Techniken betreffen, bei denen ein Verteiler und ein Abonnent Kontrollstrukturen oder Meta-Daten austauschen, um eine dauerhafte Übertragung von Daten zu automatisieren. Die US-amerikanische Patentierungsbehörde hat bereits die Erteilung der Patente prinzipiell bestätigt. Intermind hat nun die Entwicklung eines eigenen Push-Systems zurückgestellt. Lizenzpflichtig könnten möglicherweise nicht nur die Hersteller von Push-Systemen, sondern auch die Distributoren und Anwender werden.

11 Abschließende Bemerkungen

Die Suche nach handfesten Hintergrundinformationen zu diesem Thema hat sich als äußerst schwierig erwiesen. Obwohl das Thema bereits Ende 1996 verstärkt an die Öffentlichkeit gebracht wurde, scheinen sich zum Großteil noch immer Medien wie PC Magazine oder PC Week damit zu beschäftigen. Dort werden aber lediglich einführende Artikel oder oberflächliche Produktbesprechungen angeboten. Diskussionen des Themas, so vorhanden, werden zumeist in übertriebener Form über eine goldene Zukunft der Push-Systeme bzw. das zugleich erwartete Ende der Browser geführt. Die von den Firmen selbst angebotene Literatur ist entweder dürftig (PointCast) oder stellt naturgemäß die Vorzüge des eigenen Produkts in den Vordergrund (bes. bei BackWeb).

Vertiefende Untersuchungen zu speziellen Aspekten finden sich nur am Rande, so zum Beispiel auf dem W3C Push Workshop im Herbst letzten Jahres. Hier wurden auch Themen wie „Push Technologies in Low Bandwidth and Highly Mobile Environments“ [Uma97] oder „WebCanal: Multicast Based Push“ [Lia97] vorgestellt. Zum erstgenannten Thema fand sich dann aber nur eine Zusammenfassung, während sich das letztgenannte auf eigenen WWW-Seiten präsentiert.

Ein ähnliches Bild in der hiesigen Informatik-Bibliothek: In der verfügbaren wissenschaftlichen Literatur konnte bis Ende 1997 keine Veröffentlichung gefunden werden, die sich mit dem Thema beschäftigt. Es gibt bisher erst eine unabhängige Untersuchung, die aber eher als Nebeneffekt eine Aussage über Push-Verfahren macht [GC97]. Auch hier besteht noch Bedarf an spezielleren Untersuchungen.

Zugriffsoptimierung in vernetzten Systemen durch verteiltes Caching

Matthias Winkel

Kurzfassung

Mit Hilfe von Caches wird versucht der rapide steigenden Menge von Informationsangeboten und der gleichfalls steigenden Anzahl von Anfragen – hervorgerufen durch die nahezu exponentielle Zunahme von Internet-Benutzern – Herr zu werden. Ein Cache legt hierbei häufig angeforderte oder für die nächste Operation notwendige Daten in einem eigenen Speicher ab. Durch die besondere Organisation dieses Speichers – z.B. durch Indextabellen – kann auf die darin enthaltenen Daten schnell zugegriffen werden. Beim Abrufen der Daten wird nun zuerst im Cache überprüft, ob die gewünschten Daten bereits in seinem Speicher vorhanden sind, und von dort an den Aufrufenden zurückgeliefert werden können.

In zweierlei Hinsicht soll hierbei der Zugriff auf Informationen im Internet optimiert werden: Erstens soll die Latenzzeit zwischen der Anforderung eines Dokumentes und dem Erhalt einer Antwort möglichst kurz sein. Zweitens ist die Belastung des Netzwerkes bei der Abwicklung solcher Anfragen zu minimieren.

Diese Ausarbeitung stellt die Technologien des verteilten Cachings anhand des Harvest-Systems vor, durch dessen Einsatz die obengenannten Optimierungsziele erreicht werden sollen.

1 Einleitung

In den vergangenen Jahren hat sich das WorldWideWeb als populärster Dienst im Internet herauskristallisiert. Die rapide steigende Anzahl an Benutzern hat zur Folge, daß sowohl die Menge an Informationsangeboten als auch die Anfragen zum Abrufen der angebotenen Informationen zunehmen. Dieses exponentielle Wachstum macht eine effiziente Nutzung der Informationen schon aufgrund ihrer unüberschaubaren Menge schwierig. Lange Antwortzeiten und eine erhebliche Auslastung des Netzwerkes durch die hohe Zahl der Anfragen sind die Konsequenzen. Desweiteren stellt schon das Auffinden einer Quelle, welche die gewünschten Informationen enthält, ein Problem dar [BDHUM98, BDH+95].

Die Entwickler des Harvest-Systems haben sich das Lösen bzw. das Optimieren dieser Probleme zum Ziel gemacht. Ihre Arbeit setzt die Entwicklungsgeschichte der Caches unter dem Gesichtspunkt *Internet* fort. So wurden zahlreiche vorhandene und neue Technologien zusammengeführt, um ein für das Internet abgestimmtes und seinen Bedürfnissen gewachsenes Cache-System zu erstellen.

Nach einer kurzen Erläuterung der grundsätzlichen Funktionsweise von Caches wird die Entwicklung der Cache-Technologien in Form eines kleinen historischen Abrisses behandelt. Das zweite und dritte Kapitel stellen dann das Harvest-System mit Schwerpunkt Objekt-Cache und den darin implementierten Technologien und Optimierungsmethoden.

1.1 Funktionsweisen von Caches

Das grundlegende Prinzip eines Caches ist einfach: Teile einer Gesamtheit von Daten werden näher dorthin gebracht, wo sie gebraucht werden sollen. Dies geschieht, indem der Cache die Daten aus externen Quellen sammelt und sie als Kopie zusammen mit einer Refernz auf ihre Herkunft in einem eigenen Speicher ablegt.

In einem Prozessor sorgt die Verwendung eines Caches zum Beispiel für einen um etwa eine Größeneinheit schnelleren Datentransfer zwischen Arbeitsspeicher und Prozessorbus, weil die für die Zwischenspeicherung im Cache zuständigen Speicherbausteine einen wesentlich schnelleren Zugriff erlauben als herkömmliche Speicherchips. Nach einem ähnlichen Prinzip, allerdings mit einer anderen Größe und mit einem anderen Zeitfakor arbeiten Caches, die in Verbinung mit Dateisystemen Verwendung finden. Hier werden häufig angeforderte Dokumente im Cache gespeichert, anstatt sie jedesmal von der deutlich langsameren Festplatte zu lesen.

Im Internet wird dieses Prinzip ein weiteres Mal erweitert: Stark frequentierte Dokumente werden dort in Caches bereitgehalten, und zwar in der Nähe derjenigen Stellen, wo sie häufig gebraucht werden. Aufgrund der Nähe bleibt die Zugriffszeit recht kurz, und das restliche Netzwerk wird nicht belastet.

Zwar ähnelt Caching im Internet vom Prinzip her der Verwendung von Caches in verteilten Dateisystemen, dennoch differieren die einzelnen charakteristischen Größen erheblich. Vor allem die allgemein hohen Latenzzeiten, die extrem große Anzahl der Benutzer und das Fehlen einer einheitlichen Referenzierung von Orten für Informationsquellen machten es notwendig, vorhandene Cache-Technologien zu überdenken und neue zu erschaffen.

1.2 Eine kurze Geschichte des Caches

Die Geschichte der Caches im Netz beginnt mit der Implementierung von Dokumentenspeichern auf Benutzerseite in der jeweiligen verwendeten Software, die zum "Durchstöbern" des Internets verwendet wird. Die Dokumente, die ein Benutzer im Laufe einer Internet-Sitzung anfordert, werden entweder temporär auf der lokalen Festplatte gespeichert oder im Arbeitsspeicher gehalten. Die Nachteile der alleinigen Verwendung dieser Technologie liegt auf der Hand, wenn man sich die Menge der unterschiedlichen Dokumente im Internet vor Augen führt. Die Wahrscheinlichkeit, daß auf ein Dokument zugegriffen wird, das sich bereits im lokalen Cache befindet, ist angesichts der immensen Bandbreite von verschiedenen Informationen äußerst gering.

Es scheint in der Natur des Internets zu liegen, nach Caches zu verlangen, die gleichzeitig von mehreren Benutzern verwendet werden. Diese Erkenntnis in Verbindung mit dem Wunsch nach einer Zugriffskontrolle auf das reichhaltige Informationsangebot führte zu den sogenannten *Proxy-Caches* (Abb. 15). Proxies erwarten eine Anfrage nach einem Dokument von einem Benutzer. Wenn dieses Dokument sich nicht im Proxy-Speicher befindet, so verhält dieser sich seinerseits als Client und fordert dieses Dokument aus der externen Quelle an, die der Benutzer bei seiner Anfrage mitangegeben hat[4]. Das Dokument wird dann im Proxy-Speicher abgelegt, sowie an den Benutzer weitergeschickt.

[4] z.B. in Form einer URL

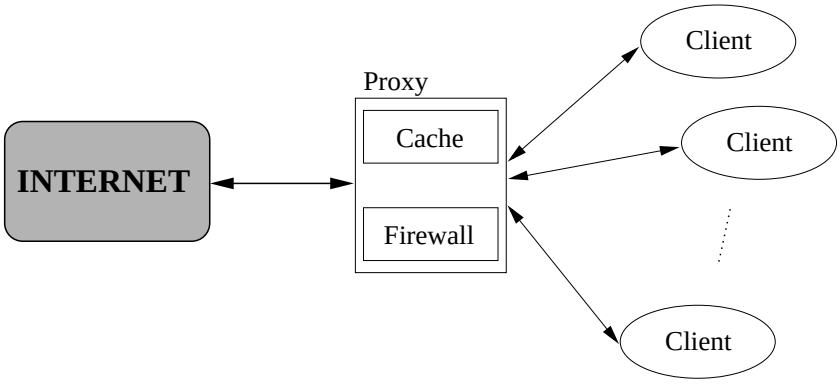

Abbildung15. Proxy-Cache

Wird das gleiche Dokument ein weiteres Mal angefordert, so kann der Anfragende direkt aus dem Cache heraus bedient werden.

Die Idee dabei ist, Proxies zum Beispiel zwischen dem Intranet einer Firma oder eines Providers und dem Internet zu plazieren mit dem Gedanken, daß die Mitarbeiter der Firma bzw. die Kunden des Providers vorwiegend ähnliche Interessen besitzen und demzufolge zu ähnlichen Themen Dokumente anfordern. Hierbei wird erhofft, daß schon aufgrund der gemeinsamen Interessenslage bestimmte Dokumente häufig wiederholt angefordert werden, die dann bereits im Proxy-Cache aufzufinden sind.
Die erste weit erhältliche und nach diesem Konzept funktionierende Software ist der CERN Web Server, welcher 1993 der Öffentlichkeit zugängig gemacht wurde[5][BBM+97].

Zusammen mit der immer größer werdenden Nachfrage nach Dokumenten, deren Inhalte sich häufig ändern, kam das Problem der Aktualität auf. Zwar unterstützten die Proxy-Caches die Funktion, zu einem Dokument eine Referenz auf einen Zeitpunkt mitabzuspeichern, ab dem das Dokument veraltet und neu von seiner Quelle anzufordern ist. Allerdings wurde dies von keinem Web-Server benutzt [BBM+97]. Deswegen wurden die Caches mit einer Heuristik ausgestattet, welche abschätzen sollte, wann ein Dokument veraltet ist und neu angefordert werden muß. Dies führte dazu, daß viele Server-Betreiber und Provider dazu übergingen, zu verhindern, daß Dokumente im Cache abgelegt werden. Sie sahen von ihren ursprünglichen Überlegungen ab und räumten der Aktualität Vorrang vor schnellen Zugriffszeiten ein.

Die Weiterentwicklung von Caches in der letzten Zeit führte zu Cache-Hierarchien: Die den Caches zugrunde liegende Software ermöglicht es, mehrere Caches zu einem hierarchischen Verbund zusammenzuschließen, wobei sich die Bandbreite von serverseitigen Caches, über Proxies, bis zu benutzerseitigen Caches erstreckt. Das Ziel ist die effektive Vergrößerung der gesamten Cache-Kapazität.

[5] Ähnliche, mehr proprietär ausgelegte Software wurde auch von anderen Herstellern etwa zur gleichen Zeit herausgebracht. Allerdings erreichten diese nicht den gleichen Verbreitungsgrad bzgl. Anzahl der Installationen.

2 Harvest

Einrichtungen und Dienste des Internets, wie zum Beispiel WWW oder Gopher, machen es sehr einfach, Informationen im Internet anzubieten. Die rasante Zunahme der Benutzer, der Datenmenge und der Bandbreite der Daten machen eine effektive Nutzung der angebotenen Informationen äußerst schwierig. So wird der Benutzer im wesentlichen mit folgenden Problemen konfrontiert [BDHUM98, BDMS94]:

- Relevante Information sind angesichts der riesigen Datenmenge und -bandbreite nur sehr schwer aufzufinden. Daraus resultierendes längeres Suchen des Benutzers in verschiedenen Quellen führt zu einer höheren Netzwerkbelastung.

- Zugriffe auf sogenannte populäre Informationen – das sind Informationen, die häufig von vielen Benutzern und deswegen oftmals gleichzeitig angefordert werden – führen im Netzwerk und auf dem Server zu Engpässen (Flaschenhalsproblem).

Das Harvest-System wurde hinsichtlich dieser Probleme konzipiert[BDHUM98]. Harvest besteht aus verschiedenen Werkzeugen zum Sammeln von Informationen aus verschiedenen Internet-Quellen, zur Indexerstellung über Inhalte gesamter Server, zum Durchsuchen und Replizieren dieser Indizes und zum Cachen von Objekten, die quer über das Netz der Netze verschickt wurden.

In diesem Kapitel wird das Harvest-System mit seinen Komponenten vorgestellt. Der Objekt-Cache wird ausführlich in Kapitel 4 behandelt.

2.1 Technischer Überblick

Das Harvest-System wurde mit dem Ziel entwickelt, möglichst effektiv Index-Informationen zu erstellen und zu verbreiten. Die Effizienz liegt vor allem in der Kombination von optimierter Software zum Sammeln und zum Verbreiten der Informationen. Insbesondere wird keine unnötige Last auf den Servern oder im Internet erzeugt durch den Verzicht auf komplexe Protokolle zum Übertragen kompletter Objekte zum Erstellen der Indizes. Desweiteren koordinieren Harvest-Systeme sich untereinander und arbeiten nicht gegeneinander.

Das Harvest-System besteht im wesentlichen aus folgenden Komponenten:

- Gatherer

- Broker

- Index/Search Subsystem

- Replicator

- Object Cache

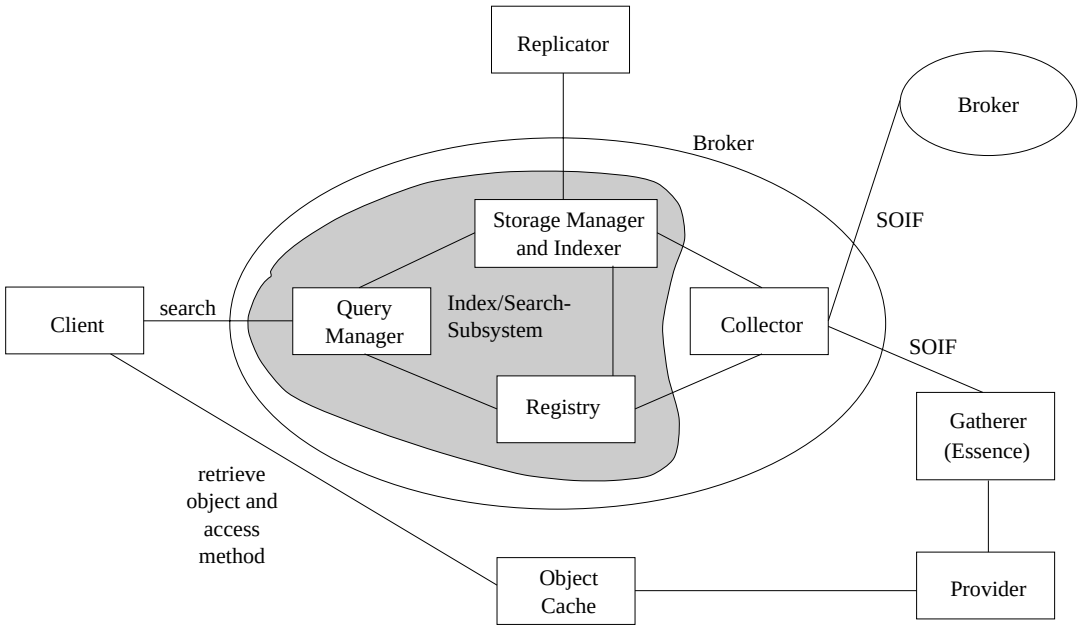

Abbildung16. Allgemeine Architektur des Harvest-Systems [BDHUM98]

Die grundsätzliche Architektur des Harvest-System veranschaulicht Abbildung 16.

Der *Gatherer* ist für den Einsatz auf dem Server des Providers vorgesehen. Durch seine Arbeit dort, werden bereits große Teile der Serverlast und des Netzwerkverkehrs eingespart. Es ist auch möglich den Gatherer über das Internet auf die Informationen eines Providers zugreifen zu lassen. Dies geschieht dann mittels der üblichen Protokolle wie FTP oder HTTP. Hierbei muß die sonstige Ineffizienz in Kauf genommen werden. Diese Art des Einsatzes von Gatherern ist für solche Systeme vorgesehen, auf denen die Harvest-Software nicht läuft.

Ein *Broker* sammelt die Informationen von mehreren Gatherern, um daraus Indizes zu erstellen, und sie über das Internet verfügbar zu machen. Es können auch die Informationen anderer Broker abgefragt und so mehrere Broker zu einer Kaskade zusammengefaßt werden.

Die Kommunikation zwischen einzelnen Harvestkomponenten erfolgt über ein spezielles Protokoll, dem sogenannten *SOIF* (Summary Object Interchange Format).

Zur Optimierung der Zugriffe auf häufig angeforderte Dokumente existieren die Komponenten *Replicator* und *Object Cache*: Mittels des Replicators können komplette Server oder Verzeichnisbäume gespiegelt und auf anderen Rechnern abgelegt werden. Im Object-Cache werden einzelne, immer wieder angeforderte Objekte gespeichert.

Die einzelnen Komponenten werden im folgenden detailliert vorgestellt.

2.2 Gatherer

Die Aufgabe des Gatherers ist es, Indexinformationen zu sammeln. Der Gatherer kann auf dem Server eines Internet-Providers laufen, um Zusammenfassungen über den Inhalt der dort angebotenen Informationen an den oder die Broker zu versenden. In periodischen

Zeitabständen überprüft er die Informationen auf Veränderungen und neu hinzugekommenes Material.

Unter Umständen ist das Erkennen von Änderungen nicht immer einfach. Insbesondere bei älteren WWW-Servern ist dies nicht möglich. Für diese Fälle errechnet der Gatherer eine Prüfsumme beim Herunterladen von einem Server. Wenn sich die Prüfsumme beim erneuten Herunterladen des Objekts geändert hat, so wurde dieses möglicherweise geändert. In diesem Fall übermittelt der Gatherer das neue Dokument an den Broker. Wenn der Gatherer allerdings die gleiche Prüfsumme ermittelt, so wird natürlich nicht das Dokument übermittelt, sondern lediglich die Information, daß die Daten des Brokers noch aktuell sind.[6] Diese Technik verhindert zwar nicht, daß Dokumente mehrfach vom Gatherer angefordert und heruntergeladen werden, was angesichts der Tatsache, daß der Gatherer eigentlich auf dem Server läuft, auf dem sich auch die zu sammelnden Daten befinden, durchaus akzeptabel ist. Dadurch entsteht nämlich keine zusätzliche Netzlast. Dafür wird allerdings erreicht, daß der Broker Dokumente nicht unnötigerweise mehrfach herunterlädt, was eine zusätzliche Netzlast bedeuten würde.

Der Gatherer ist außerdem in der Lage, die notwendigen Informationen für eine Zusammenfassung in Abhängigkeit des vorliegenden Objekttyps zu erstellen. Dies geschieht mittels der Unterkomponente *Essence*, welche den Dateityp erkennt – zum Beispiel anhand der Endung des Dateinamens – und dann die notwendigen und für diesen Typ charaketeristischen Informationen herauszieht. So kann der Gatherer unter anderen komprimierte tar-Dateien entpacken oder Titel und Autor eines LaTex-Dokumentes ermitteln. Hierdurch wird erreicht, daß die Daten auf einen relevanten Teil reduziert werden. Bei einem HTML-Dokument bedeutet dies, daß die Tags zur Formatanweisung (z.B. `<font>`, `<type>`, etc.) herausgefiltert werden. Übrig bleibt das eigentlich Wesentliche, nämlich der Inhalt des Dokumentes, der zur Indexerstellung weiter verwendet wird.

Für viele gängige Objekttypen sind entsprechende Vorgehensweisen bereits in Essence vorprogrammiert (z.B. für Tex, HTML, etc.). Desweiteren ist es dem Benuter möglich, die Typenerkennung seinen Wünschen anzupassen. Er kann weitere Objekttypen hinzufügen, oder das Herausziehen der relevanten Daten bei vorhandenen Objekttypen abändern.

Die Zusammenfassungen der Objekte werden in einem lokalen Cache abgelegt und können von dort in einem einzigen, komprimierten Stream abgeholt werden, was deutlich schneller ist, als für jedes Objekt eine eigene Abfrage zu formulieren.

Indem der Gatherer also vorgefilterte und inkrementell erweiterbare Indexinformationen in einem komprimierten Datenstrom zur Verfügung stellt, wird die Effizienz der Netzwerkübertragungen deutlich gesteigert: Die Server- und Netzwerkauslastung werden um mehrere Größenordnungen reduziert.

2.3 Broker

Broker stellen eine Schnittstelle zur Verfügung, mit der gesammelte Informationen von Benutzern oder anderen Brokern abgefragt werden können. Der Broker bezieht seine

[6] Es kann evt. auch ein Zeitpunkt übermittelt werden, an dem das nächste Mal die Aktualität dieser Daten überprüft werden soll.

Informationen entweder von einem oder mehreren Gatherern oder von andern Brokern. Er enthält somit nicht mehr das ursprüngliche Objekt, sondern nur noch die relevanten Teile davon, welche die Gatherer zuvor herausgefiltert haben.

Jedes zu indizierende Objekt wird mit einer Prüfsumme und einem Zeitstempel versehen. Veraltete Dokument werden mittels eines Garbage Collectors entfernt bzw. neu angefordert.

Der Broker ist wie der Gatherer in der Lage, seinen Index inkrementell zu erweitern. Die Daten holt er dabei in periodischen Abständen von den Gatherern. Mittels der Prüfsumme und den IDs der Gatherer kann der Broker doppelt vorhandene Dokumente entdecken, ohne sie vorher komplett von irgendwoher beziehen zu müssen.

Aus Sicht des Benutzers ist die Hauptaufgabe des Brokers, Suchanfragen zu bearbeiten. Diese werden an die Index/Search-Subkomponenten des Brokers weitergeleitet.

Zur Kommunikation mit den Gatherern oder anderen Brokern wird das SOIF-Protokoll – Summary Object Interchange Format – verwendet. Das SOIF überträgt die Daten der Objekte als Attribut-Werte-Paare. Hierbei ist Attribut die Bezeichnung der Kategorie einer gewünschten Information (z.B. Inhalt, Titel, Autor, Thema, etc.) und Wert die Information selbst. Die Verwendung anderer gängiger Protokolle, etwa HTTP, würde einen unnötigen Overhead bedeuten, da nur die relevanten Daten der Objekte übertragen werden und zum Beispiel Formatanweisungen wie Schriftart, Schriftgröße etc. nicht mehr vorhanden sind [BDHUM98].

2.4 Index/Search Subsystem

Um den verschiedenen Such- und Indexanforderungen entsprechen zu können, legt Harvest mit dem Index/Search-Subsystem eine allgemeine Schnittstelle zwischen Broker und Indizierung fest. Diese Index/Search-Subkomponente bearbeitet Suchanfragen, die von Benutzern oder von anderen Brokern stammen.

Die beim Broker eingegangenen Suchanfragen aus dem Netz werden an das Index/Search-Subsystem weitergeleitet. Dieses beantwortet die Anfrage mit einer Liste von Objekt-IDs zusammen mit den vom Anfragenden gewünschten SOIF-Attributen.

Der *Query-Manager* erstellt aus den Daten eine formatierte Ausgabe, die vom Benutzer gelesen oder von den anderen Brokern weiterverarbeitet werden kann.

Das Index/Search-Subsystem emöglicht es zum Beispiel, daß ein Broker zwar das komplette Literaturverzeichnis einer Bibliothek beinhaltet, ein anderer Broker aber nur Teile daraus – beispielsweise nur Bücher eines bestimmten Themas – herunterlädt.

Für Harvest wurden zwei Index/Search-Subsysteme entwickelt: *Glimpse* und *Nebula*. Während *Glimpse* mit speicherplatzoptimierten Indizes arbeitet und die interaktive Eingabe von Suchanfragen zuläßt, unterstützt *Nebula* schnelle Suchläufe und komplexe Suchanfragen. *Glimpse* ist diejenige Version, die in der Harvest-Quellcode-Distribution mit ausgeliefert wird.

Die ebenfalls im Broker enthaltene *Registry* verwaltet Vermerke über die gerade im Objekt-Cache befindlichen Daten. Diese Informationen stehen auch dem Index/Search-

Subsystem innerhalb des Brokers zur Verfügung, so daß bei einer Suchanfrage sofort ermittelt werden kann, ob die gewünschten Daten im Object-Cache vorhanden sind.

2.5 Replicator

Der Replicator kopiert gesamte Verzeichnisbäume oder gar komplette Server. Er besteht aus dem *Mirror-Daemon* (Mirror-d) und dem *Flood-Daemon* (Flood-d).

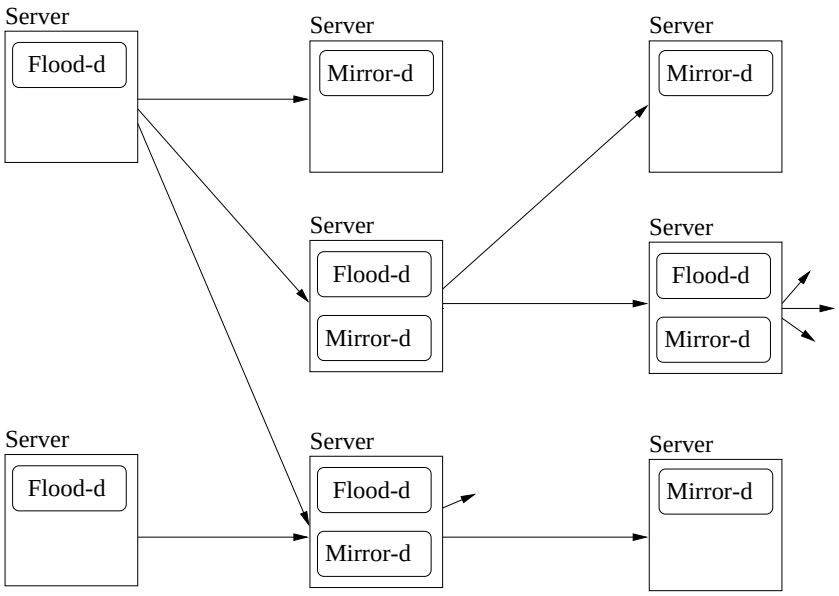

Abbildung17. Hierarchie replizierter Server

Ein Flood-d gehört zu jedem zu kopierenden Verzeichnisbaum bzw. Server. Er arbeitet gruppenorientiert. Das bedeutet, daß Flood-d seine Objekt und Update-Informationen von einem Mitglied der Gruppe zu einem anderen entlang eines bestimmten Graphen sendet, den er selbst festlegt und verwaltet. Den Weg in einem Graphen, entlang dem die Daten übermittelt werden, kann der Flood-Daemon selbst festlegen, indem er die Netzwerkbandbreite zwischen ihm und anderen Flood-Daemons, die auf Rechnern seiner Gruppenmitglieder laufen, mißt. So kann der für die Netzwerkbandbreite günstigste Pfad zur Datenübertragung gewählt werden.

Ein Flood-Daemon kann mehrere Gruppen bzw. Mirror-Daemons beliefern. Umgekehrt kann ein Mirror-Daemon seine Daten von mehreren Flood-Daemons erhalten. Dadurch ist es möglich, komplette Hierarchien aufzubauen (Abb. 17). Voneinander ansonsten unabhängige Gruppen werden miteinander verbunden, indem sie mehrere gemeinsame Mitglieder besitzen, oder einzelne Mitglieder der verschiedenen Gruppe vom gleichen Flood-d bedient werden.

Auf jedem Server, der von Flood-d beliefert wird – also zu jeder Kopie des Verzeichnisbaums – existiert ein Mirror-Daemon. Dieser schickt periodisch Statusmeldungen zu seinen unmittelbaren Nachbarn in der Gruppe, um neue Informationen zu entdecken, die Flood-d nicht ausliefern konnte. Gründe dafür könnte ein über längere Zeit überlastetes Netzwerk, ein Fehler auf dem Server oder gar ein Fehler im Flood-d-Prozess sein.

3 Object Cache

Diese Kapitel widmet sich dem Object Cache des Harvest-Systems. Insbesondere die implementierten Techniken zur Optimierung werden vorgestellt.

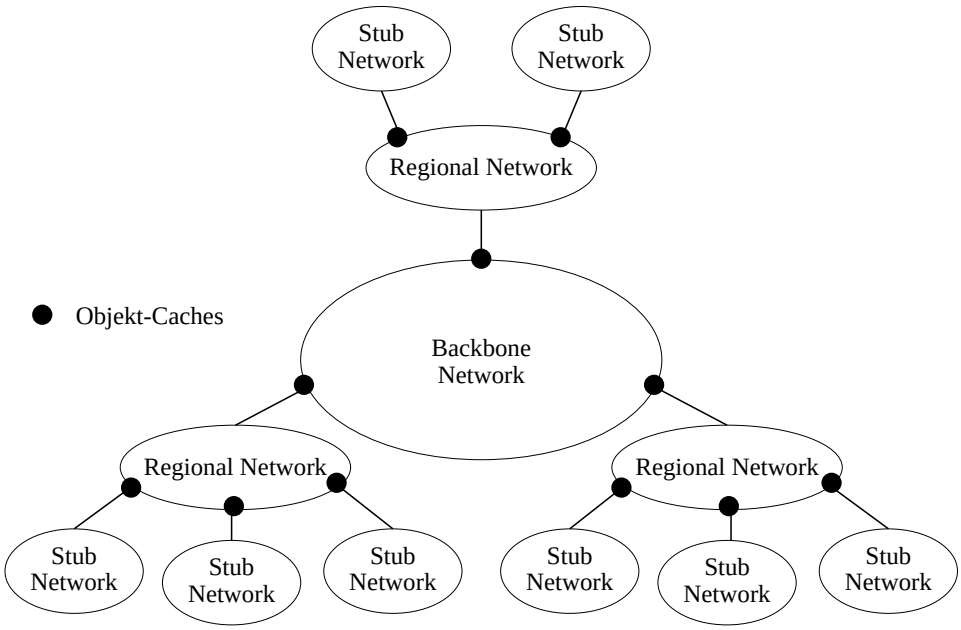

Abbildung18. Hierarchische Anordnung der Objekt-Caches

3.1 Cache-Hierarchie

Um das schon angesprochene Flaschenhalsproblem, das durch häufiges Anfordern von populären Dokumenten entsteht, zu bewältigen, enthält das Harvest-System einen hierarchisch angeordneten Objekt-Cache (Abb. 18).

Die Notwendigkeit eines verteilten Cache-Systems für das Internet erkannten die Entwickler von Harvest zu einem Zeitpunkt, an dem vorwiegend FTP und Electronic Mail (eMail) als cache-lose, verteilte Quasi-Dateisysteme benutzt werden. Es sind im Wesentlichen zwei Gründe, welche die Harvest-Entwickler angeben: Erstens, weil Menge und Größe der Objekte, die am häufigsten angefragt werden, durch das fortlaufende Hinzufügen neuer Informationdienste wachsen, kann ein Objekt-Cache Antwortzeiten und Netzwerkauslastungen verringern: Nach eigenen Angaben der Entwickler könnte ein Fünftel des gesamten NSFNET-Backbone-Verkehrs vermieden werden, wenn nur die FTP-Daten in Caches abgelegt werden würden [CDN+95]. Zweitens, das Netzwerk wird durch den Einsatz von Caches vor Clients geschützt, die immer wieder versuchen dasselbe Objekt abzufragen. Möglicherweise liegt in diesem Fall sogar ein Fehler im Client vor. Zwar werden durch eine Cache-Hierarchie solche Fehler nicht behoben, aber dafür werden sie isoliert und das Netzwerk vor ihnen abgeschirmt [BDMS94].

Harvest-Administratoren können einen Baum von zusammenarbeitenden Caches – sogar über größere Entfernungen hinweg – aufbauen. Die Caches stehen entweder in einer Eltern- oder in einer Nachbarbeziehung zueinander.

Jeder Cache, der eine Anfrage nach einem Objekt erhält, das nicht in ihm gespeichert ist, entscheidet unabhängig darüber, ob das Objekt von seinem Ursprungs-Server, vom Vater-Cache oder von einem der Nachbar-Caches geholt wird. Hierfür verwendet Harvest das *Resolution Protocol*. Seine Arbeitsweise erklärt der folgende Absatz.

Zunächst wird die URL, die auf das gewünschte Objekt verweist, überprüft. Wird hierbei in der URL ein Teil-String aus einer vom Adminstrator frei konfigurierbaren Liste gefunden, dann holt Harvest das Objekt direkt von seinem Heimat-Server. Dadurch wird erreicht, daß Objekte, die ständigen Änderungen unterliegen oder eine lokale URL besitzen nicht im Cache-Verbund abgelegt werden. Sich stndig ndernde Objekte können zum Beispiel durch CGI-Skripte dynamisch generierte WWW-Seiten sein. Solche Objekte, von denen zu erwarten ist, daß sie bei jedem Abrufen einen anderen Inhalt besitzen, werden sinnvoller Weise jedes Mal von Ihrer ursprünglichen Quelle angefordert. Auch brauchen lokal vorhandene Objekte, die sich also auf dem gleiche Rechner oder im gleichen Firmennetz wie der Cache befinden, nicht im Cache-Speicher abgelegt werden.

Andernfalls, wenn der Cache eine Anfrage nach einem Objekt erhält, das nicht in seinem Speicher liegt, schickt er Meldungen [7] an seine Nachbarn und an seinen Vater. Diese Meldungen werden parallel abgesetzt. Der Cache erhält das Objekt von demjenigen Server, von dem als erstes eine positive Antwort empfangen wird, von dem also die kleinste Latenzzeit gemessen wurde.

Zusätzlich kann Harvest so konfiguriert werden, daß derjenige Server, der die ursprüngliche Quelle des Objektes ist, auch im Resolution Protocol berücksichtigt wird. In diesem Fall sendet der Cache zusätzlich zu den Meldungen an seinen Vater und an seine Nachbarn eine "Hit"-Meldung über UDP an den Echo Port des Ursprungsservers. Dieser schickt die Meldung als Echo an den Cache zurück. Wenn der Cache das Echo seiner eigenen Meldung empfängt, sieht diese für ihn so aus, wie eine positive "Hit"-Meldung eines der anderen Caches in der Hierarchie. Möglicherweise wird dieses Echo noch vor einer anderen Meldung empfangen, was dazu führt, daß das verlangte Objekt von seinem Heimatserver angefordert wird. Dies geschieht dann, wenn der Ursprungsserver des verlangten Objekts "näher" am anfragenden Cache ist als sein Vater oder seine Nachbarn. Andererseits kann die Verbindung auch so "langsam" sein, daß das UDP-Echo nicht sofort empfangen wird. Der Cache wartet dann nicht auf Echo, sondern wertet es einfach als "Miss", wenn nach dem Absenden der Meldung an den Heimat-Server zwei Sekunden vergangen sind, ohne daß ein Echo registriert wurde.

Der anfragende Cache erhält also das Objekt über einen Nachbarn, über den Vater oder über den Heimatserver, je nachdem von wem er zuerst eine UDP-"Hit"-Meldung empfangen hat. Wie auch immer, der Cache wartet nicht ab, bis das gesamte Objekt in den Speicher übertragen wurde, sondern sobald er die ersten Daten empfangen hat, werden diese an den anfragenden Client übermittelt [CDN+95].

3.2 Unterstützte Protokolle

Zur Kommunikation mit den Caches stehen drei Protokolle zur Verfügung.

[7] Es handelt sich dabei um Remote Procedure Calls.

Das *Encapsulating Protocol* ermöglicht den Datenaustausch zwischen den Caches zur Fehlererkennung mittels Check-Summen und digitalen Signaturen. Eltern-Caches können außerdem durch dieses Protokoll Time-To-Live-Angaben von Objekten an ihre Kinder weitergeben.

Das *Connectionless Protocol* definiert die Vater-Kind-Beziehungen zwischen den Caches. Aus Gründen der Effizienz können Eltern-Caches damit auch kleinere Objekte an ihre Kinder übertragen, ohne eine TCP-Verbinung aufbauen zu müssen.

Von den meisten WWW-Browsern wird das *Proxy-http Protocol* unterstützt. Dadurch können von den meisten Clients mittels der bekannten Standardprotokolle – Gopher, FTP, HTTP – Objekte aus dem Cache angefordert werden [CDN+95].

3.3 Eindeutige Referenzen auf Objekte

Wie bereits in der Einleitung erwähnt, identifiziert eine URL ein Objekt nicht eindeutig. Ein WWW-Server zum Beispiel kann eine Postscript-Datei als Text übermitteln, falls auf Client-Seite kein Postscript-Interpreter installiert ist. Erst die URL zusammen mit dem MIME-Header macht ein Objekt tatsächlich eindeutig: MIME ist die Abkürzung für "Multipurpose Internet Mail Extension" und wurde ursprünglich zur Übertragung von Multimedia-Daten via EMail entwickelt. Später wurde es in HTTP integriert, um Typ und weitere Meta-Daten von Objekten zwischen Client und Server übermitteln zu können. Durch die Hinzunahme von MIME zur URL stehen Informationen darüber zur Verfügung, wie die Daten in dem Objekt zu interpretieren sind.

Allerdings hat dies noch nicht ausreichende Verbreitung, weshalb die Entwickler dafür sorgten, daß beim Vergleich von Objekten, um diese in der Cache-Hierarchie aufzufinden, die MIME-Header nicht übereinstimmen müssen [CDN+95].

3.4 Negatives Caching

Harvest unterstützt auch sogenanntes *negatives Caching*:

Wenn ein DNS-Lookup fehlschlägt, weil zum Beispiel die angegebene URL nicht existiert, wird dies im Cache gespeichert. Wenn nun wieder eine Anfrage nach einem Objekt mit der gleichen (fehlerhaften) URL erfolgt, meldet der Cache sofort den Fehler zurück, ohne noch einmal einen DNS-Lookup durchzuführen[8]. Nach einer gewissen Zeit wird die DNS aus dem negativen Cache entfernt.

Sollte eine Anfrage nach einem Objekt einen Fehler verursachen, wird auch diese URL für eine bestimmte, vom Administrator voreingestellte Zeit in den Cache gelegt. Innerhalb dieser Zeit werden Anfragen nach diesem Objekt direkt vom Cache selbst mit einer entsprechenden Fehlermeldung beantwortet. Es werden also keine Ressourcen vergeudet, mit häufigen und vergeblichen Versuchen den nicht zu erreichenden Ursprungsserver zu kontaktieren [CDN+95].

[8] Diese Vorgehensweise wird z.B. von Domain-Name-Servern nicht unterstützt.

3.5 Parallelität und I/O-Behandlung

Alle Anfragen an den Cache werden von einem einzigen Prozeß bearbeitet. Dadurch wird die Umschaltzeit (größenordnungsmäßig mehrere Millisekunden), die benötigt wird, um von einem Prozeß auf einen anderen umzuschalten, eingespart. Dies ermöglicht eine schnellere Abarbeitung, als jede Anfrage von einem eigenen Prozeß bearbeiten zu lassen.

In Harvest wurde eine eigene Abstraktionsschicht für Zugriffe auf das Netzwerk und auf das Dateisystem implementiert, basierend auf einer BSD-Select-Loop. Die Entwickler geben hierfür außer der besseren Effizienz noch einen weiteren Grund an, der in die Überlegung zur Implementierung einer solchen Abstraktionsschicht einbezogen wurde; nämlich die bessere Portierbarkeit auf andere UNIX-Systeme.

Die einzige Ausnahme ist das Anfordern von FTP-Objekten. Falls eine Anfrage nach einem FTP-Objekt einen "Cache-Miss" erzeugt und das FTP-Objekt von einem Server geholt werden muß, erzeugt Harvest dafür einen eigenen FTP-Prozess. Diese Vorgehensweise wurde von den Entwicklern aus Gründen der Einfachheit bezüglich der Implementierung gewählt. Offenbar ist es wegen der Komplexität des File Transfer Protocols schwierig, es in die BSD-Select-Loop zu integrieren.

Das Schreiben von Daten auf den Festplattenspeicher des Caches sowie die Ausgaben an einen Anfragenden erfolgen nichtblockierend (*non-blocking I/O*). Das bedeutet, daß sobald die ersten Bytes eines Dokumentes den Cache erreichen, diese sofort weitergeschickt werden. Der anfragende Benutzer oder Client erhält also schnellstmöglich seine gewünschten Daten und muß nicht erst warten, bis sie vollständig im Cache vorliegen.

Intern arbeitet Harvest unter der Verwaltung eines nicht-preemptiven Schedulers, was dazu führt, daß auf aufwendiges Datei-Locking verzichtet werden kann [CDN+95].

3.6 Speicherverwaltung und Cache-Organisation

Sämtliche Meta-Daten (URL, Time-To-Live(TTL), Anzahl der Referenzen, Verweise auf das Dateisystem und verschiedene Flags) werden im virtuellen Speicher abgelegt[9]. Dies ist auch mit den am häufigsten angeforderten Objekten möglich, wenn eine entsprechende Option in der Konfiguration des Harvest-Cache gesetzt wird. Wenn die Speicherbelegung im virtuellen Speicher für "heiße" Objekte eine bestimmte, konfigurierbare Grenze überschreitet, werden Objekte daraus entfernt, bis eine voreinstellbare untere Grenze erreicht ist. Die Objekte bleiben im eigentlichen Cache vorhanden. Es werden lediglich ihre Repräsentationen aus dem virtuellen Speicher gelöscht.

Der Cache arbeitet nach dem *Write-through-Prinzip*. Alle Objekte, die sich im virtuellen Speicher befinden, existieren auch auf der Festplatte. Das bedeutet schreibende Operationen auf den virtuellen Speicher wirken sich auch sofort auf die entsprechenden Objekte auf der Festplatte aus.

Der Objekt-Cache ist als *Hash Table* organisiert. Den Schlüssel bilden die URLs. Im Cache befindliche Objekte bleiben so lange dort, bis ihre TTL überschritten ist, und

[9] Nach Angaben der Entwickler benötigen die Meta-Daten für 1,000,000 Objekte etwa 60-80 MBytes Speicherplatz.

sie von einem internen Mechanismus entfernt werden. Wenn der Cache eine Anfrage auf ein veraltetes WWW-Objekt erhält, erneuert er die TTL des Objekts und fordert das Objekt mittels des HTTP-Protokolls unter der Angabe von "`get-if-modified`"[10] neu an [CDN+95].

3.7 Diskmanagement

Harvest verwaltet mehrere Festplatten und versucht die Objekte gleichmäßig darauf zu verteilen: Auf jeder Festplatte werden hundert durchnumerierte Verzeichnisse erzeugt. Mittels einfacher Rotation durch die Verzeichnisse und über die Festplatten beim Erzeugen neuer Objekte werden diese auf die Verzeichnisse und auf die Festplatten verteilt. Nehmen wir an, ein Cache besteht aus vier Festplatten und darin befinden sich eine Million Objekte, so sind in einem Verzeichnis durchschnittlich 2500 Objekte gespeichert.

Sobald der belegte Festplattenspeicherplatz eine konfigurierbare obere Grenze überschreitet, schaltet der Cache in den sogenannten *Garbage Collection Mode*. In diesem Modus werden jeweils nach einer bestimmten Anzahl von abgearbeiteten Anfragen die ältesten Objekte von Festplatte gelöscht. Sobald der belegte Speicherplatz unter eine untere Grenze sinkt, verläßt der Cache den Garbage Collection Mode und arbeitet normal weiter. Sollte der belegte Speicher sogar ein festgelegtes Maximum überschreiten, werden sofort die ältesten Objekte aus der nächsten Reihe des Hash Tables gelöscht [CDN+95].

4 Ausblick

Die weitere Verbreitung und damit der Erfolg von Harvest wird nicht zuletzt durch seine hervorragende Konzeption, die sich nah an den Bedingungen und Eigenschaften des Internets orientiert, vorangetrieben, sondern auch durch die freie Verfgbarkeit der Harvest-Software.

4.1 Aktueller Stand

Das Harvest-Caching-Projekt der Harvest Developer Group ist in der Zwischenzeit zu einem kommerziellen Produkt mit Support unter dem Namen "cached" geworden und liegt in der version 2.0 vor. Die letzte Version der frei verfügbaren Software heißt "cached 1.4 pl3" und ist über `http://harvest.cs.colorado.edu/` zu beziehen. Ihre Weiterentwicklung erfolgt unter dem Namen "Squid" beim National Laboratory for Applied Network Research [SSvLD97, CDN+95, Sch96].

4.2 Ziel und Entwicklung

Den Squid-Cache-Betreibern schwebt eine sinnvolle Neustrukturierung der vorhandenen Squid-Caches – bisher nur als einfaches Netzwerk miteinander verbunden – vor. Bestehende Ressourcen (WiN-Leitungen, Festplattenkapazitäten, etc.) sollen dabei so optimal wie möglich genutzt werden.

[10] Das bedeutet, das Objekt wird nur dann übermittelt, wenn es sich tatsächlich geändert hat

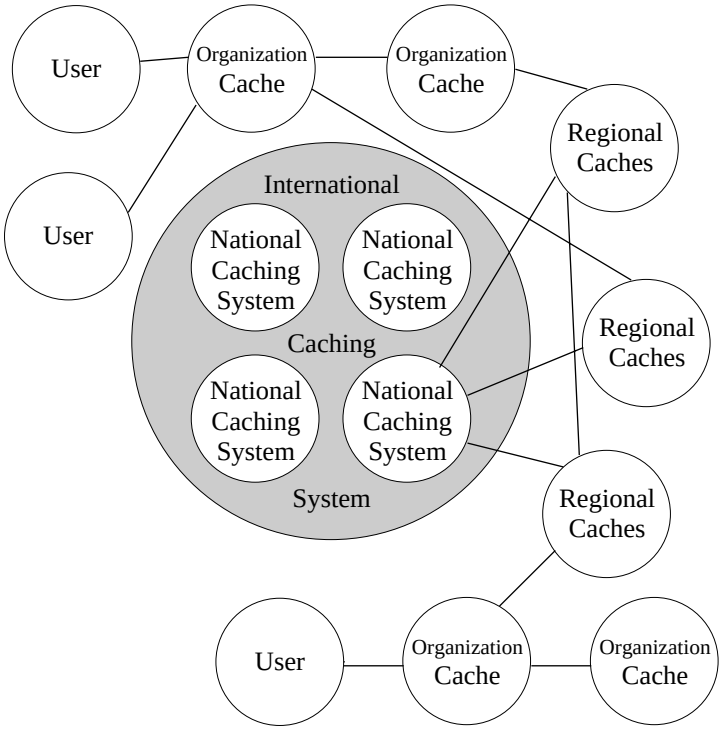

Abbildung19. Globale Cache-Hierarchie (geplant)

Das Ziel ist eine globale hierachische Cache-Struktur, wie in Abbildung 19 skizziert: Hierbei verfügt jeder Benutzer über einen lokalen Cache, der mit dem Cache (z.B. Proxy) des übergeordneten Instituts verbunden ist. Alle Institute besitzen wiederum den Cache der nächsthöheren Institution als Vater, der selbst an regionale Caches angeschlossen ist (einzelne Instituts-Caches gehören zum Cache einer Universität, die wiederum am regionalen Rechenzentrum angeschlossen ist). Die regionalen Caches gehören zu einem nationalen Cache-Verbund. Und alle diese nationalen Einrichtungen bilden zusammen ein internationales Cache-System. Es sollen sowohl Vater-Sohn-Beziehungen zwischen den Caches verschiedener Ebenen bestehen als auch Nachbarschafts-Beziehungen zwischen den Caches einer Ebene.

Derzeit befindet sich eine Netzstruktur im Aufbau, in welcher bestimmte große Caches an den Knoten des B-WiN untereinander verknüpft werden, und die Zugriffe auf die internationale Cache-Struktur ermöglicht[11]. Proxy-Caches einzelner Institutionen benutzen diese Netzstruktur. Außerdem bestehen zwischen vielen dieser Caches bereits Nachbarschaftsbeziehungen [SSvLD97].

5 Zusammenfassung

Zusammenfassend sind noch einmal die Eigenschaften aufgelistet, die das Harvest-Cache-System charakterisieren und ihm Vorteile gegenüber anderen Caching/Proxy-Programmen einbringen[SSvLD97]:

[11] Stand 3.2.1997

- Zukunftsweisendes Konzept für nationale und internationale Ebenen statt isolierter Systeme
- Einbettung in bestehende hierarchische Ordnungen (Regionale Ebene, Landes-, Bundesebene) durch Nachbarschafts- und Eltern-Kind-Beziehung
- Durch konsequent angewandte Optimierungsverfahren sehr kurze Antwortzeiten und reduzierte Server-/Netzlast. Im Vergleich zum CERN-Cache ist Harvest 10-100mal so schnell laut Angaben der Entwickler.
- Jeder Objekt-Cache innerhalb einer Cache-Hierarchie arbeitet autonom, das bedeutet z.B., daß kein Cache Einfluß darauf nehmen kann, welche Objekte ein anderer Cache in seinem Speicher ablegt.

Verteilte Anwendungen auf dem WWW

Markus Manck

Kurzfassung

Die aktuelle Entwicklung im Softwarebereich tendiert zu verteilten Anwendungen. Durch das Internet setzen sich WWW-Browser als bevorzugte Benutzeroberfläche immer mehr durch. Dadurch ist eine Synergie der traditionellen HTML-basierten Informationsbereitstellung und bisheriger verteilter Programmiersysteme zu einem gemeinsamen Konzept naheliegend. Die Entwicklung von dynamischen HTML läuft noch größtenteils seperat ab. Im Bereich der verteilten Programmiersysteme konkurrieren die CORBA-Spezifikation der OMG und Microsofts DCOM immer stärker. Durch Java rückt eine Vereinigung von HTML und CORBA/DCOM näher. Java RMI selbst ist ein Spezialansatz für Java und läuft somit gegen den Trend zur Sprachunabhängigkeit. Die Eigenschaften von CORBA, DCOM und Java RMI, sowie deren Integrationsfähigkeit in das WorldWideWeb stehen im Mittelpunkt der folgenden Ausführungen.

1 Einleitung

Mit der boomhaften Verbreitung des World Wide Web in allen Gebieten des alltäglichen und geschäftlichen Lebens wird die Forderung von einheitlicher Software in einer einheitlichen Ablaufumgebung, welche dem Nutzer jederzeit und jederorts zur Verfügung steht, und zwar auf jeder Plattform, immer lauter. Der Nutzer ist der großen Anwendungen, welche ihn zu immer schnelleren und damit teureren Rechner nötigt, müde. Der Trend geht zu kleinen Anwendungen, die schnell sind und möglichst wenig kosten. Diese Anwendungen sollen immer auf dem neuesten Stand sein und ohne Aufwand oder lange Wartezeiten erweiterbar und zu updaten sein. Manche Anwendungen sollen von dem Nutzer unter Umständen auch nicht komplett erworben werden, sondern gemietet und pro Nutzung bezahlt werden. Gleichzeitig sollen diese Anwendungen in ihrer Mächtigkeit und Komfortabilität nicht schlechter sein als die bisherigen Großpakete. Eine weitere Forderung ist es die gewohnte Software unter verschiedensten Plattformen nutzen zu können. Doch was bedeuten diese Forderungen? Es bedeutet, daß sich die Anwendungen von plattformabhängigen Gesamtsystemen zu plattformunabhängigen verteilten Anwendungen verändern. Der Aspekt der Plattformabhängigkeit liegt auf der Hand, doch warum verteilte Anwendungen? Mit Hilfe verteilter Anwendungen ist der Wunsch nach kleinen Anwendungen, die jedoch nichts von den Fähigkeiten großer Produkte einbüßen einfach zu erfüllen. Die Teile einer Gesamtanwendung, die der einzelne Nutzer benutzt, werden eine GUI mit nutzerspezifischer Funktionalität darstellen, welche nicht nutzerspezifische Arbeiten an spezielle, auf ihre jeweilige Aufgabe zugeschnittene Anwendungsteile delegieren. Dadurch erreicht man kleine Oberflächen, die einfach zu updaten und zu warten sind, aber durch Kommunikation mit anderen Programmen nichts an Funktionalität verlieren. Außerdem wird es ermöglicht, diese Anwendungsteile in einer für alle Anwendungen gleichen Umgebung zu starten und sie via WWW zu betreiben. Doch für solche verteilte Anwendungen bedarf es ausgereifter und zuverlässiger Konzepte.

Den Weg zu verteilten Anwendungen im WWW bahnen zwei verschieden Philosophien, einerseits eine Erweiterung traditionellen HTML's um Funktionalität, andererseits die Verwendung des WWW als graphische Schnittstelle durch verteilte Programmiersysteme.

Auf verschiedene Konzepte von verteilten Anwendungen im WWW soll nun im folgenden eingegangen werden.

2 Das Komponentenmodell

Um eine große Anwendung in viele kleine aufzuspalten ist eine neue Art der Softwaregestaltung notwendig. Große Applikationen, die ihre eigene Oberfläche und Funktionalität komplett enthalten, sind dazu nicht geeignet. Stattdessen wird eine Philosophie benötigt, welche es erlaubt kleine, in ihrer Funktionalität beschränkte Softwarekomponenten in einem gemeinsamen Rahmen zusammenzustellen, also eine Gesamtanwendung in einer Art Baukastensystem zusammenzusetzen. Dieser gemeinsame Rahmen soll unabhängig von der Art der Komponenten sein. Es existieren nun mehrere solche Ansätze solcher Compound Document Frameworks (CDF). Ein solches Framework besteht aus einem immer gleichen Frontend, welches als Container oder Dokument mit verschiedenen Arten von Inhalten, wie zum Beispiel Text, Tabellen, Bilder, etc, gefüllt werden kann. Zu jeder Art von Inhalt gehören nun verschiedene Softwarekomponenten, die diesen Inhalt manipulieren können und sich in dem allgemeinen Frontend integrieren. So könnte eine Anwendung zum Beispiel aus Komponenten für einen Datenbankzugriff, für Datenverarbeitung und Datenrepräsentation bestehen und zusammengesetzt zur Sichtung der Daten einer Datenbank dienen.

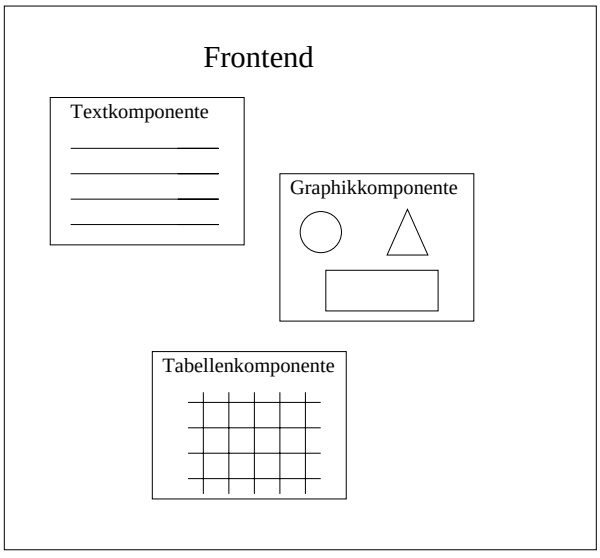

Abbildung20. CDF-Konzept.

Auf diese Art erreicht man eine Anwendung, die aus vielen Teilen besteht. Diese Teile sollten auch jederzeit aus verschiedenen Quellen, auch dem WWW ergänzt werden können. Implementationen solcher CDF's stellen OpenDoc oder Microsoft OLE dar. Obwohl bereits verschiedene Frontends entwickelt wurden, zum Beispiel Apples CyberDog,

hat sich durch die Verbreitung des WWW, bei dem Anwender der Wunsch eines Webbrowsers als Universal Frontend durchgesetzt. Die Entwicklung von Browsern hin zu einem Universal Frontend ist aktuelle Entwicklung.

3 HTML

Ursprünglich wurde die Hypertext Markup Language (HTML) zur reinen Repräsentation von Informationen in Gestalt von Bild und Text im Internet entwickelt. HTML hat einen rein beschreibenden Character und beinhaltet somit keine eigene Funktionalität, entspricht also nicht dem vorhin eingeführten Komponentenmodell. Ein HTML-Dokument kann durch einen HTML-konformen Clienten (Browser) dargestellt werden. Durch die unabhängige Betreuung von HTML durch das World Wide Web Consortium (W3C) einerseits und dem Boom des WWW in den letzten 5 Jahren andererseits, hat sich HTML als allgemeiner Standard durchgesetzt. HTML-konforme Browser sind inzwischen auf jedem größerem Betriebssystem vorhanden, womit HTML an keine Plattform gebunden ist. Um ein HTML-Dokument von einem Server auf Anfrage zu einem Clienten zu transportieren, wird das Hypertext Transfer Protocoll (HTTP) benutzt. Die Sicherheit der Datenübertragung kann durch Protokollerweiterungen wie SSL oder S-HTTP gewährleistet werden. Dieses Protokoll ist zustandslos, das heißt ein Browser schliesst nach erfolgreich bearbeiter Anfrage an einen Server die Verbindung zu diesem und öffnet sie erst wieder bei einem neuen Request. Auf der Serverseite werden, außerhalb der Bearbeitung einer Anfrage, keine Daten über oder von dem Clienten gehalten.

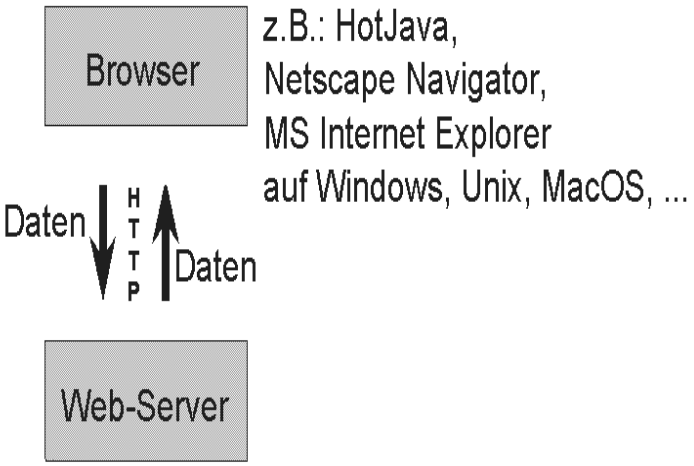

Abbildung21. Traditionelle HTTP-Client/Server-Architektur.

Mit der zunehmenden Verbreitung des WWW's wurde jedoch schnell der Wunsch wach, das WWW nicht nur zu Darstellung und Transport von Dokumenten zu nutzen, sondern auch funktionale Aspekte zu berücksichtigen. Eine weit verbreiteter Ansatz war das Einführen von Common Gateway Interfaces (CGI).

Dieser Mechanismus bietet einem WWW-Server die Möglichkeit die Anfrage eines Clients an eine andere Anwendung weiterzureichen. Nachdem diese Anwendung die Anfrage

bearbeitet hat, generiert der Server ein neues HTML-Dokument, welches das Ergebnis der fremden Anwendung enthält und liefert diese an den Clienten. Parameter und Zustände können über die 'Seitenadresse', die URL (Uniform Resource Locator), an eine CGI-Anwendung übergeben werden oder direkt über HTTP an den Server geschickt werden und dann von CGI-Anwendungen ausgelesen werden. Diese beiden Verfahren der Parameterübergabe, werden als GET- bzw. POST-Mechanismus bezeichnet. Beiden Verfahren gemeinsam ist die Tatsache, daß Zustände oder Parameter für jeden Seitenzugriff übergeben werden und bei einem erneuten Zugriff nicht erhalten bleiben. Durch den CGI- Mechanismus wurde zwar die Möglichkeit geschaffen, Anwendungen über das WWW zu starten, Client und Server haben jedoch weiterhin statischen Character.

Ein weiterer Schritt Server und Client mit Funktionalität zu füllen, ist die Verwendung von Scriptsprachen wie JavaScript oder VBAScript, welche in die HTML-Dokumente eingefügt werden und durch eigenständige Komponenten auf Server- oder Clientseite abgearbeitet werden.

Durch diese CGI-Erweiterungen ist es möglich den Server um sessionabhängige Zustände zu erweitern (z.B. mit Microsofts Active Server Pages), welche auch außerhalb eines einzelnen Seitenzugriffes erhalten bleiben und sich somit von der Übergabe von Parametern in der URL unterscheiden. Für komplexere Anwendungen ist diese Art der Interaktion jedoch nicht ausreichend.

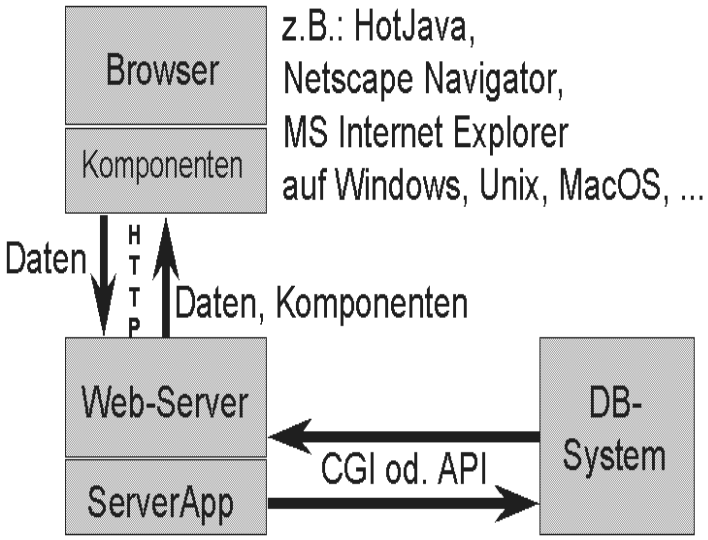

Abbildung22. Erweiterte HTTP-Client/Server-Architektur.

Server-API's (Application Interface), zur zustandsbasierten Programmierung von Serverapplikationen, werden von den Herstellern der verschiedenen Webserver angeboten und sind in aller Regel nicht miteinander kompatibel. Sowohl CGI-basierte als auch API-basierte Serverapplikation zeichnen sich durch allgemein schlechte Flexibilität und Performanz aus. Zusammenfaßend ist HTML ein Dokumentenformat, welches um wenige funktionale Aspekte erweitert wurde, jedoch für komplexere Anwendungen auf dem

WWW nicht geeignet ist.

4 Verteilte Anwendungen

Um Anwendungen in verteilten Systemen zu entwickeln benötigt man eine Softwareinfrastruktur, welche die Verteilung von Objekten und die Interoperabilität zwischen Objekten gewährleistet.

Für das Design verteilter Anwendungen gibt es verschiedene Ansätze. Der bekannteste Ansatz ist eine Zweiteilung in Client und Server, wobei je nach der Menge der enthaltenen Funktionalität ein Client beziehungsweise Server als 'fat client/server', bei viel enthaltener Funktionalität, oder als 'thin client/server', bei wenig enthaltener Funktionalität, bezeichnet wird. Sind verschiedene Dienste zwischen Client und Server nicht direkt durch diese realisiert, zum Beispiel bei Datenbankrepräsentationen, sondern durch zusätzliche Anwendungen, so spricht man von einer Multitierarchitektur.

Die gebräuchlichste Architektur ist die sogennante ThreeTier-Architektur, bei welcher sich eine komplette Anwendung in drei Ebenen gliedert, zum Beispiel in Komponenten zur Datenrepräsentation, Datenverarbeitung und Datenhaltung.

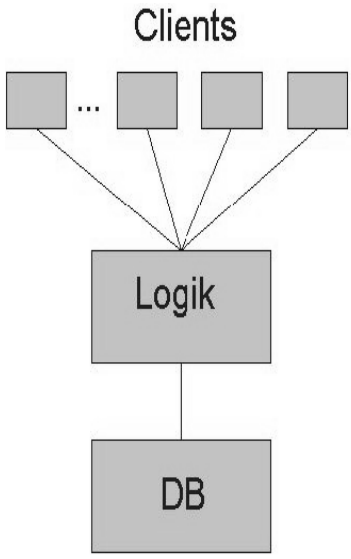

Abbildung23. Eine ThreeTier-Architektur.

Um solche Multitierarchitekturen zu realisieren, werden oft Verteilungsplattformen verwendet, welche zwischen dem jeweiligen Betriebssystem und der eigentlichen Anwendung liegen und Kommunikation und Datentransport zwischen verschiedenen Anwendungen ermöglichen. Diese Verteilungsplattformen werden oft als Middleware-Plattformen bezeichnet.

Bei der Betrachtung solcher Middleware lassen sich leicht Unterschiede und Gemeinsamkeiten in verschieden Bereichen erkennen. Besonderes Augenmerk gilt bei der folgenden Betrachtung von vier verschiedenen Ansätzen von Middleware der Sprachunabhängig-

keit der Ansätze, der Art der Schnittstellenbeschreibung, der Codemobilität, dem Exceptionhandling, dem Kommunikationsprotokoll und eventuellen zusätzlichen Diensten der Middleware.

4.1 CORBA

CORBA steht für Common Object Request Broker Architecture und ist eine Spezifikation einer Middleware-Plattform, welche von der Object Management Group (OMG) festgelegt wurde. Es existieren mehrere kommerzielle Object Request Broker (ORBs), welche auf dieser Spezifikation basieren. (Orbix,ObjektBroker,Visigenic,...)

CORBA unterstützt derzeit die Sprachen C, C++, Smalltalk, Ada, Cobol und Java.

Um eine Methode eines anderen Objektes aurufen zu können, definiert ein CORBA-Objekt eine Schnittstelle, welche aus einer Menge von Methoden des Objektes besteht. Diese Schnittstelle wird mit Hilfe einer Interface Definition Language (IDL) definiert. Diese Sprache ist eigenständig und von anderen Programmiersprachen unabhängig. Das bedeutet daß die Implementation in verschiedenen Sprachen erfolgen kann. Eine Schnittstelle kann vererbt werden, jedoch nur die Schnittstelle, nicht deren Implementation. Die Implementation muß bei der Vererbung auf Vererbungsmechanismen der benutzten Sprache zurückgreifen. Das heißt die Wiederverwendbarkeit auf Implementationsebene ist eingeschränkt. Eine Schnittstelle wird durch ihren einen Pfad angesprochen, welcher mithilfe eines Naming Services auf eine Objektadresse abgebildet wird. Die IDL wird von einem IDL-Compiler compiliert, welcher dann sogenante Stubs bzw Skeletons erzeugt. Ein Stub ist ein Programmstück in der Zielsprache des Clients, welches für die Verarbeitung eines Methodenaufrufs auf der Clientseite verantwortlich ist. Ein Skeleton hingegen ist dem Server zugeordnet und für die Verarbeitung eines Methodenaufrufs auf Serverseite verantwortlich.

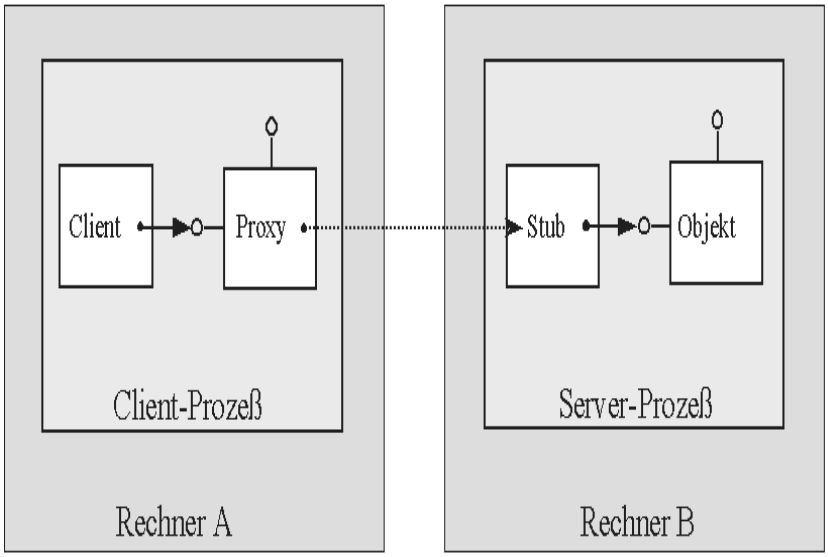

Abbildung24. Proxy und Stub

Beim Erzeugen eines CORBAobjektes, werden Stub bzw. Skeleton zu den Objekten gelinkt.

Durch die verschiedenen Implementierungen der CORBA-Spezifikation, ist es unter Umst-änden schwierig, Code von einer Objektimplementation zu einer anderen zu übertragen. Eine Implementationsvererbung ohne erneutes Übersetzung des Codes ist nicht möglich. Die CORBA Spezifikation sieht ein Exceptionhandling im Implementationscode vor.

Wird ein CORBA-Server gestartet, so wird er bei dem zuständigen ORB registriert, jeder ORB verfügt dazu über einen eigenen Hintergrundprozess. Nach der Registrierung kann ein Server durch einen Methodenaufruf eines Clients automatisch gestartet werden.

Desweiteren bietet CORBA ein Dynamic Invocation Interface, über welches unbekannte Schnittstellen und Methoden beschrieben und aufgerufen werden können.

Zur Kommunikation zwischen verschiedenen ORBss existiert seit CORBA2.0 ein eigenes Protokoll, das Internet Inter-ORB Protocol (IIOP), welches die Kompatibilität zwischen verschiedenen ORBs gewährleistet.

CORBA bietet dem Nutzer bislang 15 sogenannte CORBAServices, welche durch die OMG vollständig spezifiziert, im allgemeinen jedoch noch nicht von allen ORB-Herstellern implementiert sind. Die wichtigsten Services sind der schon beschriebene Naming Service, der Event Notification Service, welcher für die Einbindung von Ereignissen zuständig ist, der Lifecycle Service, der die Lebensdauer von Objekten steuert, der Security Service, der Externalization Service, welcher alle Zustände eines eines Serverobjektes an einen Client übermittelt, sowie der Transaction Service.

Die weiteren Services sind Persistent Object Service, Property Service, Query Service, Relationship Service, Time Service, Licensing Service, Trading Service, Concurrency Service und Collection Service.

Desweiteren bietet die OMG, je nach Einsatzgebiet der ORBs, Spezifikationen zu anwendungsspezifischen Problemen, die sogenannten CORBAFacilities an.

4.2 DCOM

DCOM steht für Distributed Component Object Model und ist ein Produkt von Microsoft. Bisher steht DCOM nur auf Windows NT 4.0 Rechnern und in einer Betaversion der Software AG auf Solaris und Linux verfügbar. DCOM unterstützt zur Zeit C, C++, VBA (Visual Basic for Applications) und Java.

Ähnlich wie CORBA besitzt DCOM eine IDL. Bei DCOM kann ein Objekt mehrere Schnittstellen besitzen, welche wiederum jede Methoden besitzt. Um einem Client alle Schnittstellen darzulegen, besitzt jedes DCOM-Objekt das Interface IUnknow, welches unter anderem die Methode QueryInterface besitzt.

Wie auch bei CORBA werden Stubs und Skeletons erzeugt, welche in einer Dynamic Link Library (dll) abgelegt werden. Auf diese DLL müßen sowohl Client- als auch Serverobjekte im folgenden Zugriff haben.

Die Schnittstellen eines DCOM-Objects werden durch Universal Unique Identifier

(UUID) eindeutig bezeichnet, wodurch Namenskonflikte auch bei Namensgleichheit ausgeschlossen sind.

Wie auch bei CORBA, ist ein Verfahren zum dynamischen Aufruf von Schnittstellen über die spezielle Schnittstelle IDispatch vorhanden.

Bei DCOM können sowohl die Schnittstellen, als auch die Implementation vererbt werden. Dazu wird das Vererbungsverfahren der Aggregation benutzt, welches darin besteht, ein Objekt zu kreieren, das andere Objekte einschließt.

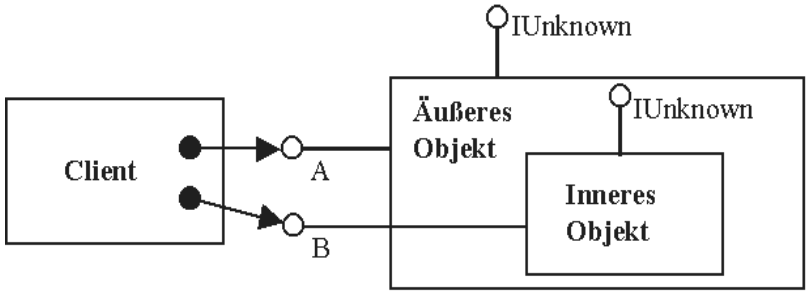

Abbildung25. Aggregation

Dieses Verfahren ist insofern interessant, als daß eine Art der Implementationsvererbung vorliegt, ohne den Implementationscode neu übersetzen zu müßen.

Ein DCOM-Objekt kann vollständig an einen Client transferiert werden, indem der Client das Objekt erst als Binärstream empfängt und dann registriert.

Ein Server ist erst dann für einen Clienten zugänglich, wenn er in der Windows-NT Registrierdatenbank eingetragen wurde. DCOM-Objekte verfügen über keine eigene Fehlerbehandlung und geben immer nur eine Instanz des vordefinierten Datentyps HRESULT zurück, was bedeutet, daß es unter DCOM kein Exceptionhandling gibt.

Die Kommunikation zwischen DCOM-Objekten wird entweder über HTTP oder ein DCOM eigenes, binäres Protokoll abgewickelt.

DCOM bietet über die Registry einen verteilten Naming Service, sowie Mechanismen zur Persistenz und Ereignissteuerung. Weitere spezifische Services sind nicht vorhanden.

4.3 JAVA und JAVA RMI

Java an sich ist keine Middlewareplattform wie CORBA und DCOM, sondern eine Programmiersprache. Durch Verwendung von JAVA RMI jedoch kann es gleiche Aufgaben wahrnehmen. Doch zunächst eine kurze Einführung zu Java.

Java ist eine einfache, zu großen Teilen an C++ erinnernde objektorientierte Programmiersprache der Firma Sun Microsystems. Der interessante Aspekt von Java ist, daß ein Javaprogramm zu einem Bytecode compiliert wird, welcher über das WWW transportiert werden kann. Dieser Bytecode wird von einem Programm namens Java Virtual Machine (JVM) interpretiert. Diese JVM ist inzwischen auf fast allen Plattformen vorhanden, womit der Java an sich den Status einer plattformunabhängigen Sprache erwirbt. Bei dem

Ausführen des Bytecodes durch die JVM, wird durch diese die Sicherheit des jeweiligen Systems durch Einschränkungen der Systemfunktionen gewährleistet, man sagt die JVM arbeitet in einer 'sandbox'. Java eignet sich besonders für Anwendungen innerhalb des WWW, da ein Javaprogramm nicht nur standalone auf einem WWW-Server ausgeführt werden kann, sondern über HTTP an einen Client transferiert werden und dort in einem Browser mit integrierter JVM ausgeführt werden kann.

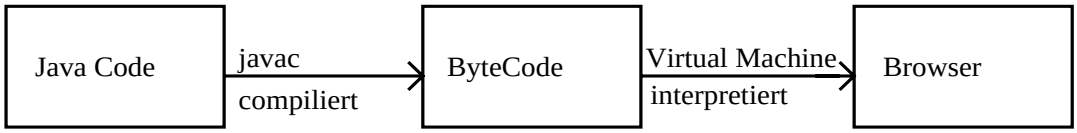

Abbildung26. Java

Damit erreicht man mehrere angenehme Effekte:

- Man hat die Möglichkeit einen Clienten während deiner Laufzeit in seiner Funktionalität zu erweitern, indem man einfach ein Javaprogramm von einem Server lädt.
- Man hat ein ablauffähiges Programm auf einem Browser und damit eine Anwendung auf dem WWW.
- Alte Softwareversion können einfach durch einen Besuch einer Website geupdatet werden, indem die neue Version direkt von einem Server herunterlädt.
- Anwendung müssen nicht mehr mühsam auf verschiedene Plattformen portiert werden.
- Man erhält mit dem Browser seiner Wahl eine einheitliche Ablaufumgebung für seine Anwendungen.
- Java-Programme können in den verschiedenen Kontexten wiederverwendet werden.

Java hat jedoch auch Nachteile. Dadurch daß der Bytecode interpretiert wird ist Java vergleichsweise langsam. Dieser Effekt wird durch sogenannte Just in Time Compiler (JIT) etwas gemildert. Diese JITs übersetzen den plattformunabhängigen Javacode in plattformabhängigen, optimierten Code. Ein weiterer Nachteil von Java ist, daß Java keine Mechanismen zum Methodenaufruf in anderen Objekten besitzt. Dies bedeutet, daß ein Javaprogramm, nachdem es einmal von einem Server heruntergeladen wurde lokal abläuft.

Java bietet also eine plattformunabhängige Sprache mit denen Anwendungen auf dem WWW realisiert werden können. Eine Interoperabilität zwischen den verschieden Anwendungen auf mehreren Rechnern ist jedoch nicht gewährleistet.

Dieses jedoch wird mit JAVA RMI erreicht, welches in das Java Development Kit 1.1 aufgenommen wurde.

Sun hat mit dem Java Devolpment Kit 1.1 (JDK), das bisherige Java um einen Mechanismus zum Methodenaufruf in entfernten Objekten (Remote Method Invocation - RMI) erweitert.

RMI unterstützt auschließlich Java.

Dazu wird auf eine eigene IDL verzichtet und stattdessen das javaeigene Interfacemodell benutzt. Ein Interface, welches RMI zur Verfügung stehen soll muß public sein und von einer speziellen Klasse namens java.rmi.Remote erben. Der Java-Compiler erzeugt dann aus diesem Interface Stubs und Skeletons. Vererbung ist im Rahmen der Java-Vererbungsstrategie möglich. Die Addressierung der Interfaces erfolgt über einen eigenen Nameserver, welcher eine RMI-Registry verwendet.

Durch die ausschliessliche Unterstützung von Java ist RMI-Code sehr einfach in verschieden Implementationen einsetzbar.

Zur Fehlerbehandlung existiert eine eigene Exceptionklasse. Ein RMI-Sytem verfügt desweiteren über eine Remote Reference Schicht, welche zur Fehlerbehandlung und Steuerung der Objekterzeugung dient. Eine Transport-Schicht ist für die Kommunikation zwischen Objekten zuständig. Das zugrunde liegende Protokoll ist zur Zeit TCP, soll jedoch zukünftig das IIOP sein.

RMI bietet einen Naming Service und einen LifeCycle Service.

Zusammenmfaßend ist zu sagen, daß sich CORBA als Verteilerplattform auf heterogenen Systemen empfiehlt, DCOM auf reinen NT-Umgebungen jedoch den Vorzug zu geben ist, da keine Inkonsistenzen zwischen verschiedenen ORBs zu befürchten sind und das Betriebssystem DCOM unterstützt. Außerhalb der Microsoftwelt ist DCOM nicht vertreten. RMI sollte noch mit Vorsicht betrachtet werden, da es gerade erst dem Beta-Stadium entwachsen ist.

5 Web-Middleware-Integrationen

In den vorhergegangen Abschnitten wurde gezeigt, daß Java zwar plattformunabhängig ist und mit den meisten Browsern eine Ablaufumgebung besitzt, jedoch in bezug auf Interoperabilität einschneidende Mängel besitzt. CORBA dagegen setzt einen komfortablen, offenen Standard in bezug auf Interoperabilität und Verteilung, hat jedoch unter Umständen den Nachteil, daß ein zu einem Server korrespondierender Client je nach gewählter Sprache nicht plattformunabhängig ist, gleiches in verschärftem Maße für DCOM, da sowohl Client als auch Server zusätzlich auf die Betriebssysteme Windows NT und Windows 95 festgelegt sind. JAVA RMI kennt diese Problematik nicht, ist jedoch auf die Sprache Java festgelegt. Nun liegt es nahe als Zielsprache für Clients von vorneherein Java, und als Ablaufumgebung einen Browser zu wählen, was im Falle von JAVA RMI und CORBA zu einem plattform- und betriebssystemunabhängigen Clienten führt. Im Falle von DCOM ist der Client zwar verteilt in bezug auf die Hardware weitgehend plattformunabhängig jedoch auf die microsofteigenen Betriebssysteme fixiert.

Falls es die Performance erlaubt, ist es desweiteren möglich auch die DCOM-Server in Java zu entwerfen und somit ein komplett plattformunabhängiges verteiltes System zu entwerfen, wobei auch hier die Betriebssystemeinschränkungen für DCOM gelten.

Wie sieht nun eine verteilte Anwendung im WWW aus? Unabhängig von der gewählten Middleware wird sich eine verteilte Anwendung in mindestens zwei Bereiche gliedern. Ein

Bereich ist die Darstellung der Anwendung im WWW, welche idealerweise durch einen, in einem Browser ablaufenden, Java-Clienten realisiert werden kann. Ein zweiter Bereich ist die Serverlandschaft, welche von den Clienten benutzt wird und in allen von der entsprechenden Middleware unterstützten Programmiersprachen implementiert werden kann.

Um diese Gliederung zu verdeutlichen ein kleines Beispiel. Die zu realisierende Anwendung soll eine Sicht auf eine Datenbank ermöglichen und aus einer Oberflächenkomponente, einer Datenverarbeitungskomponente und einer Datenhaltungskomponente bestehen.

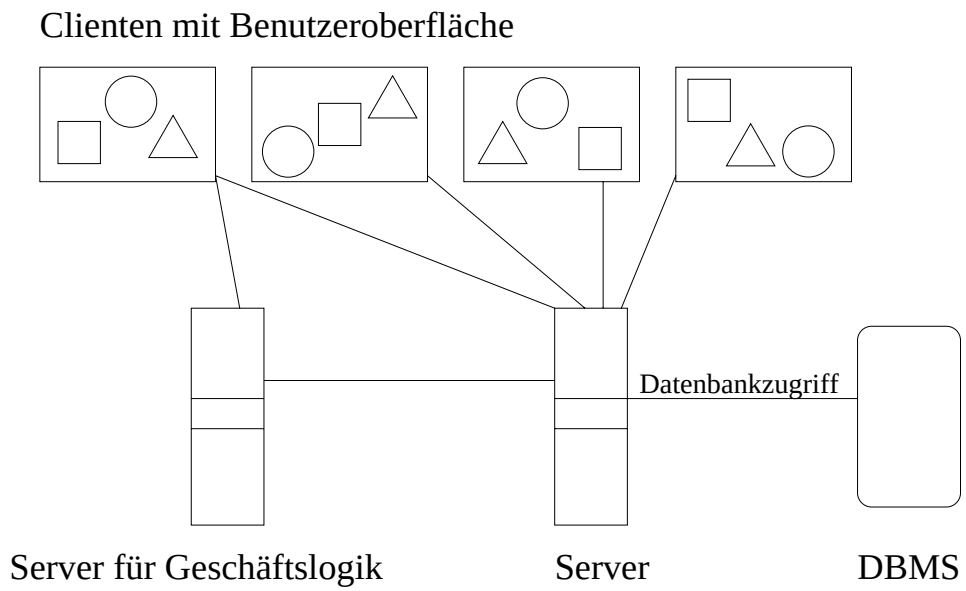

Abbildung27. Architekturschema

Wird die Anwendung mit CORBA realisiert, so wird zunächst ein Client benötigt, welcher in einem Browser ablaufen kann. Dazu ist aufgrund der momentanen Browsertechnologie lediglich Java geeignet. Desweiteren wird auf der Clientseite ein ORB benötigt. Die datenverarbeitende Komponente, welche die sogenannte Businesslogik implementiert, benötigt einen ORB, kann aber ansonsten in jeder, von CORBA unterstützen Sprache implementiert werden. Die Kommunikation zwischen Clienten und Server erfolgt über IIOP. Der Zugriff auf die Datenbank kann entweder über den CORBA-eigenen QueryService geschehen, oder über einen sprachtypischen Mechanismus.

Unter DCOM kann der Client entweder in Java oder, soweit ein Microsoft-Browser verwendet wird, mit Hilfe jeder von DCOM unterstützten Sprache implementiert werden. Der Server mit der Businesslogik kann in jeder von DCOM unterstützten Sprache realisiert werden. Die Kommunikation erfolgt hier entweder über DCOM oder HTTP. Auf eine Datenbank kann entweder über ODBC (Open Database Connectivity) oder über OLE DB und ADO (Active Data Objects) geschehen. Voraussetzung für jede Komponente ist die Verwendung von Windows 95/98 oder Windows NT.

Kommt JAVA RMI zum Einsatz, so müssen sowohl der Client, alsauch der Server mit der Businesslogik in Java realisiert werden. Zur Kommunikation zwischen Clienten und Server kommt IIOP zum Einsatz. Für den Zugriff auf die Datenbank liegt es nahe JDBC

(Java Database Connectivity) zu verwenden.

Zur Erstellung verteilter Anwendungen im WWW, gibt es in jedem Middlewarebereich eigene, kommerzielle Produkte, zum Beispiel OrbixWeb von IONA bei CORBA, InterDev bei DCOM oder Java Workshop von Sun.

6 Ausblick

Die Zukunft verteilter Anwendungen auf dem WWW liegt meiner Meinung nach bei einer Kombination von Verteilerplattformen mit plattformunabhängigen Sprachen, wie im Beispiel von OrbixWeb. Besonders geeignet ist CORBA und Java, wobei es wünschenswert ist Java weiter zu beschleunigen und CORBA um einen Standard zu erweitern welcher den Transfer eines CORBA Serverobjektes zu einem Client ermöglicht. Dies bedeutet zwar eine Abweichung von CORBA von einer reinen Spezifikation zu einer teilweisen Implementation, bringt CORBA jedoch den Vorteil Serverobjekte je nach Bedarf und Häufigkeit der Nutzung direkt an einen Client zu binden und damit die Performance zu erhöhen. Von der OMG wurde in diesem Bereich eine CORBA Mobile Agent Facility als Lösungsansatz in einem Request For Proposal vorgestellt. Dieser Ansatz verwendet die Technologie der Mobilen Agenten. DCOM bleibt ein interessanter Mitbewerber unter den verschiedenen Verteilerplattformen, zumal die angekündigte Erweiterung zu COM+ und Portierung auf weitere Fremdsysteme eine Erweiterung des Einsatzgebietes bietet.

Sowohl CORBA als auch DCOM sind sowohl im industriellen Bereich, sowie im universitären Bereich schon im Einsatz und finden immer mehr Verbreitung.

Insgesamt ist die Tendenz zu verteilten Anwendungen auf dem WWW dank der steigenden Verwendung von Intraneten, auch in eher ungewöhnlichen Gebieten wie der industriellen Anlagensteuerung steigend.

Abbildungsverzeichnis

Literatur

[Bac97] BackWeb Technologies. BackWeb - A Cooperative Architecture for a Flexible „Push-Pull" Broadcasting Solution. *Whitepaper, BackWeb Techn., 2077 Gateway Place, Suite 500, San Jose, CA 95110, USA*, März 1997.

[BBM+97] M. Baentsch, L. Baum, G. Molter, S. Rothkugel und P. Sturm. World-Wide-Caching. The Application-Level View of the IEEE. *IEEE Communications Magazine*, Jun 1997, Seite 107–178.

[BC95] K. Bharat und L. Cardelli. Migratory Applications. In *Second International Workshop on Mobile Object Systems, Linz, Austria*. Springer-Verlag, 1995.

[BDH+95] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwartz und Duane P. Wessels. Harvest: A Scalable, Customizable Discovery and Access System. In *Technical Report CU-CS-732-94*. Department of Computer Science, University of Colorado, Boulder, August 1994 (revised Match 1995).

[BDHUM98] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy und Michael F. Schwartz Udi Manber. The Harvest Information Discovery and Access System. *Computer Networks and ISDN Systems* (28), 1998, Seite 119–125.

[BDMS94] C. Mic Bowman, Peter B. Danzig, Udi Manber und Michael F. Schwartz. Scalable Internet Resource Discovery: Research Problems and Approaches. *Communications of the ACM* **8**(37), Aug 1994, Seite 98–107.

[Bic96] M. Bichler. Hyperbase - World Wide Web: Basis für betriebliche Anwendungen? *iX* **8**(8), August 1996, Seite 44–50.

[CDN+95] Anawat Chankhunthod, Peter B. Danzig, Cuck Neerdeals, Michael F. Schwartz und Kurt J. Worrel. A Hierachical Internet Object Cache. In *Technical Report CU-CS-766-9*. Computer Science Departments: University of Southern California, University of Colorado - Boulder, 1995.

[CGH+95] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris und G. Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, Oktober 1995, Seite 34–49.

[Cha96] David Chappell. *ActiveX und Ole verstehen*. Microsoft Press. 1996.

[Ell97] C. Ellerman. Channel Definition Format (CDF). *WorldWideWeb Consortium (W3C), Submission*, März 1997.

[GC97] J. Graham-Cumming. Hits and Miss-es: A Year Watching the Web. *Technical Report, Optimal Networks Inc., Palo Alto, CA, USA*, 1997.

[Gra95] R. Gray. Agent Tcl: A transportable agent system. In *J. Mayfield and T. Finin (Eds.): Proc. of the CIKM WS on Intelligent Information Agents, 4th Int. Conf. on Information and Knowledge Management (CIKM 95), Baltimore, Maryland.* 1995.

[Gri97] R. H. Grimes. *Professional DCOM Programming*. Wrox Press Inc. 1997.

[Hei97] T. Height. net impressions – push. *NetGuide Magazine Online, CMP Media Inc.*, Mai 1997.

[Int97] Inc. Intermind. About Intermind's Channel Communication Patents. *Intermind Inc.*, 1997.

[JJ97a] V. Johnson und M. Johnson. How IP Multicast works. *Whitepaper, IP MMulticast Initiative (IPMI)*, 1997.

[JJ97b] V. Johnson und M. Johnson. IP Multicast Backgrounder. *Whitepaper, IP MMulticast Initiative (IPMI)*, 1997.

[JRS95] D. Johansen, R. Renesse und F. Schneider. An Introduction to TACOMA Distributed System 1.0. *Tech. Report 95-23, Dept. of Computer Science, University of Tromso, Norway*, Juni 1995.

[Kel93] L. Keller. Vom Name-Server zum Trader — Ein Überblick über Trading in verteilten Systemen. *Praxis der Informationsverarbeitung und Kommunikation (PIK)* **16**(3), 1993, Seite 122–133.

[Kir97] Ch. Kirsch. Halb zog sie ihn ... – Programm-Aktualisierung per Netz. *iX 8*, August 1997, Seite Seite 96–103.

[KSK97] T. Kamba, H. Sakagami und Y. Koseki. Automatic personalization on push news service. *W3C Push Workshop, Peabody, Massachusetts, USA*, September 1997.

[Lia97] T. Liao. WebCanal White Paper. *INRIA, Frankreich*, Juni 1997.

[LM82] M. Livny und M. Melman. Load balancing in homogenous distributed systems. In *Proc. ACM Comput. Networking Peformance Symposium*, 1982, Seite 47–55.

[Mar97] Inc. Marimba. Castanet White Paper. *Marimba Inc., Sherman Avenue, Palo Alto, CA 94306, USA*, 1997.

[Mer97] Bernhard Merkle. RMI: Verteilte Java-Objekte. *iX* **12**(12), Dezember 1997, Seite 130–135.

[MMTH95] G. Di Marzo, M. Muhgusa, C. Tschudin und J. Harms. The Messenger Paradigm and its Impact on Distributed System. *ICC'95 Workshop on Intelligent Computer Communications*, Juni 1995.

[Par97] H. Partovi. Application Standards for "Pushing" Context and Streaming Media. *W3C Push Workshop, Peabody, Massachusetts, USA*, September 1997.

[Poi97] Inc. PointCast. PointCast Network Tour. *PointCast Inc., 501 Macara Ave., Sunnyvale, CA 94086, USA*, 1997.

[PS97] H. Peine und T. Stolpmann. An Introduktion to Mobile Agent Programming and the Ara System. *ZRI Reportl, Dept. of Computer Science, Univ. of Kaiserslautern*, 1997.

[Pud97] Arno Puder. Verteilte Objekte: DCOM versus CORBA. *iX* **8**(8), August 1997, Seite 44–51.

[RJ97] D. Reed und K. Jones. Pushing Push: Advancing the Features of Channel Communications. *W3C Push Workshop, Peabody, Massachusetts, USA*, September 1997.

[RO96] Jeri Edwards R. Orfali, Dan Harkey. *The Essential Distributed Objects Survival Guide*. Wiley. 1996.

[RO97] Jeri Edwards R. Orfali, Dan Harkey. *Instant CORBA*. Wiley. 1997.

[Sch96] Oliver Schade. Gemeinsam sind wir schneller. *iX* (8), 1996, Seite 124–129.

[SSvLD97] O. Schade, Walter Steiner, F. v. Leitner und A. Döring. *Der Squid Cache*. http://www.usf.uni-kassel.de/ system/cache.htm. 1997.

[TWW92] V. Tschammer, A. Wolisz und M .Walch. The Performance of Multiple Traders Operating in the Same Domain. In *IEEE Workshop on Future Trends of Distributed Computing Systems*, Taipeh, Taiwan, 1992. Seite 122–128.

[Uma97] A. Umar. Push Technologies in Low Bandwidth and Highly Mobile Environments. *W3C Push Workshop, Peabody, Massachusetts, USA*, September 1997.

[V2.97] OrbixWeb V2.01. *IONA Whitepaper*. IONA Technologies Ltd.,Dublin,Ireland. 1997.

[VR96] The Netscape ONE Development Vision und Product Roadmap. *Netscape Whitepaper*. Netscape Communications Corp.,Mountain View,CA. 1996.

[Whi96] J. White. Mobile Agents. In *J. Bradshaw (Ed.): Software Agents*. AAAI Press, 1996.

[WT91] A. Wolisz und V. Tschammer. Some Performance Aspects of Trading Service Design. In *Proceedings of the Tenth Anual Joint Conference of the IEEE Computer and Communications Societies, Networking in the 90s*, Nr. 2, April 1991, Seite 0919–0928.

[WT93] A. Wolisz und V. Tschammer. Performance aspects of trading in opden diszributed systems. *Computer Communications* **16**(5), Mai 1993, Seite 277–287.

[You96] E. Yourdon. Java, the Web, and Software Development. *IEEE Computer* **8**(8), August 1996, Seite 25–30.