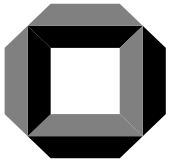

Universität Karlsruhe
Fakultät für Informatik
76128 Karlsruhe

Architektur vernetzter Systeme

Seminar Sommersemester 1996 & Wintersemester 1996/97

Herausgeber:
Arnd G. Grosse
Dietmar A. Kottmann

Universität Karlsruhe
Institut für Telematik

Interner Bericht 31/97

Zusammenfassung

Der vorliegende Interne Bericht enthält die Beiträge von Studenten zum Seminar „Architektur vernetzter Systeme“, das im Sommersemester 1996 und im Wintersemester 1996/97 am Institut für Telematik der Universität Karlsruhe stattgefunden hat. Themen waren dabei Replikationsverfahren im Mobilumfeld sowie Dienste und Konzepte in verteilten Systemen als auch alternative Ansätze wie Memory-Consictency-Modelle.

Abstract

This Technical Report includes student papers produced within small lessons called seminar of “Architektur vernetzter Systeme”. The seminar was held at the Institute of Telematics of the University of Karlsruhe in summer 1996 and winter 1996/97. The topics for discussion included mechanisms like replication especially in the field of mobile computing as well as services and concepts in distributed systems and also alternatives like memory-consistency-models.

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 1996 und im Wintersemester 1996/97 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.

In vernetzten Systemen gewinnt die Integration auch von mobilen Rechnern eine immer stärkere Bedeutung. Dieses stellt besondere Anforderungen an die Verfügbarkeit von Daten. Eine Lösung liegt in der Verwendung von spezifischen Replikationsverfahren, welche sich auch in einem mobilen Umfeld einsetzen lassen und in dem ersten Beitrag vorgestellt werden. Der zweite Beitrag hierzu beleuchtet den Einsatz von Agenten oder eines Data-Shipping-Ansatzes als alternative Verfahren zur Replikation.

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. Dieses ist Thema des dritten Beitrags. Der vierte Beitrag stellt den TINA-Ansatz vor, der die Erweiterung von Telekommunikationsnetzen um intelligente Dienste sich zum Ziel setzt.

Die nächsten drei Beiträge behandeln spezielle Thematiken aus dem Bereich der vernetzten Systeme. Der erste Beitrag befaßt sich mit Memory-Consistency-Modellen, die die Zusammenarbeit von mehreren Rechnern in einem vernetzten System an einem gemeinsamen Problem unter Beibehaltung von Konsistenzbedingungen ermöglichen. Der zweite Beitrag erläutert den Begriff des Web-Engineerings und die heutigen Probleme bei der Erstellung von WWW-Seiten oder Anwendungen. Der abschließende dritte Beitrag stellt Verteilmedien und deren Nutzung sowie Ansätze zu deren Optimierung vor.

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 Dezember 1997

Arnd G. Grosse Dietmar A. Kottmann

Replikation im Mobile Computing

Malte Peter

Kurzfassung

Dieser Beitrag handelt von einer Form des Datenmanagements im Mobile Computing - der Replikation von Daten. Abgekoppelte Operationen sind in diesem Zusammenhang der wichtigste Ansatz. Hier werden im verbindungslosen Zustand Zugriffe auf Server im Netz mit Daten im Cache des mobilen Computers emuliert. Die einzelnen Realisierungsalternativen von abgekoppelten Operationen werden vorgestellt, wobei der Trade-Off zwischen der Verfügbarkeit und der Konsistenzsicherung von Daten von großer Bedeutung ist. Schließlich werden mit Coda, Odyssey, Thor und Bayou Systembeispiele vorgestellt, die einen Einblick geben, wie solche abgekoppelte Operationen schon realisiert sind.

1 Einleitung

Tragbare Computer (Notebooks, PDAs - Personal Digital Assistents) sind heutzutage allgegenwärtig. In Verbindung mit einer schnurlosen Netzwerktechnologie bilden solche Computer eine ideale Hardware-Basis für Mobile Computing. Eine Grundvorraussetzung für das Mobile Computing ist die Fähigkeit, nicht nur konventionelle Stand-Alone-Anwendungen wie Textverarbeitung auszuführen, sondern Daten zu jeder Zeit und an jedem Ort zur Verfügung zu stellen. Die Mobilität stellt dabei allerdings ein großes Hindernis dar. Es gibt 3 grundlegende Fakten, die das Zugreifen auf verteilte Daten in einer mobilen Umgebung erschweren (siehe [SKM+93]):

- Mobile Geräte sind im Vergleich zu stationären Geräten (zum Beispiel Workstations) ressourcenärmer und teurer. Dabei spielen das Gewicht, die Größe und der Energieverbrauch eine Rolle.

- Mobile Geräte sind anfälliger für Verlust und Zerstörung. Sie werden nicht immer am gleichen Ort benutzt und können leicht liegengelassen werden. Dabei ist auch der Verlust der Daten mit eingeschlossen.

- Mobile Geräte haben viel schlechtere Netzwerk-Bedingungen. Eine Workstation kann normalerweise auf ein LAN oder WAN zurückgreifen. Ein Notebook im Hotel dagegen muß sich mit einer ISDN-Verbindung oder sogar nur mit einer GSM-Verbindung, die neben einer geringen Übertragungsrate auch plötzlich abbrechen kann, genügen. Typische Verbindungen von mobilen Computern zu Netzwerken sind Modems, Docking Stations oder einfach Disketten. Grundsätzlich kann man sagen, die Übertragungsrate ist gering und die Verbindung ist teuer und instabil. Manchmal kann sie auch nur vom mobilen Partner aufgebaut werden, da sein Standort dem stationären Partner nicht bekannt ist.

1.1 Replikation

Wegen den hohen Kosten und der Instabilität von drahtlosen Verbindungen, wechseln mobile Clients zwischen verbundenem und verbindungslosem Zustand. Man sagt auch, sie koppeln sich an das Netzwerk an bzw. ab. Das kann auch unfreiwillig passieren, zum Beispiel durch einen Netz-initiierten Verbindungsabbruch. Ist ein mobiler Computer vom Netz abgekoppelt, kann er nicht auf die anderen Rechner im Netzwerk zugreifen. Um trotzdem mit bestimmten Daten arbeiten zu können, müssen diese auf den mobilen Rechner repliziert (kopiert) werden.

Zur Vereinfachung nennt man den Rechner, von dem die Daten ursprünglich herstammen (meist ein Rechner im LAN) Server. Den Rechner, auf den die Daten repliziert werden (meist der mobile Rechner), nennt man Client. Durch die Anfälligkeit und den geringen Ressourcen der mobilen Computer liegt es nahe, den Server als wahren Speicherort der Daten zu belassen; der Client dient nur als zeitweiliger Cache, solange die Daten gebraucht werden. Die Bezeichnung „Cache“ wurde deshalb gewählt, weil der Client die Daten ja nur vorübergehend zwischenspeichert. Der physikalische Speicherort ist die Festplatte des Clients.

1.2 Anforderungen an Systeme im Mobile Computing

Systeme, die Replikation im Mobile Computing unterstützen, können sowohl Dateisysteme als auch Anwendungen sein, die auf ein verteiltes Dateisystem aufsetzen. Nach [SNKP95], [SKM+93] und [Hil95] müssen sie folgenden Anforderungen genügen:

- Aufgrund der schlechten Verbindungseigenschaften sollte die Abhängigkeit des Clients vom Server reduziert werden.

- Der Client sollte immer die richtigen Daten in den Cache kopiert haben, um auf einen Verbindungsabbruch vorbereitet zu sein.

- Idealerweise sollte die Mobilität völlig transparent für den Benutzer sein. Das heißt, der Benutzer kann genauso arbeiten, wie an seinem Computer im LAN. Im verbindungslosen Zustand sollten die Zugriffe auf den Server emuliert werden, so daß der Benutzer keinen Unterschied erkennen kann, ob eine Verbindung zum Server besteht oder nicht.

- Infolge der Replikation von Daten können Inkonsistenzen zwischen dem Original und der Kopie auftreten, falls beide schreibbar sind. Ein wichtiger Punkt im Mobile Computing ist es, mit diesen Inkonsistenzen klarzukommen.

Ein Ansatz, der diese Probleme behandelt, nennt man abgekoppelte Operationen. Er wird im nächsten Kapitel näher besprochen. In Kapitel 3 folgt dann eine Auflistung der Alternativen abgekoppelter Operationen. Vier Systembeispiele, die abgekoppelte Operationen unterstützen, werden in Kapitel 4 vorgestellt. Kapitel 5 gibt schließlich einen Ausblick auf die Systeme der nächsten Generation.

2 Abgekoppelte Operationen

Abgekoppelte Operationen (Disconnected Operations, siehe auch [Hil95], [SKM+93], [HH95], [GKLS94] und [Kot95]) ist ein Ansatz, der versucht, die in Kapitel 1.2 vorgestellten Anforderungen umzusetzen.

2.1 Konzept abgekoppelter Operationen

Die Grundidee ist folgende: Der Benutzer kopiert sich die Daten, mit denen er arbeiten will, vom Server (Sammlung), koppelt den Client vom Netz ab, führt die Anwendungen aus (Emulation) und überträgt die geänderten Daten schließlich zum Server (Reintegration), wenn die Verbindung wieder besteht (siehe auch Abbildung 1). Dabei bleiben die Daten auf dem Server bestehen, um es anderen Benutzern zu ermöglichen, ebenfalls mit den Daten zu arbeiten. Im Gegensatz zur direkten Benutzung eines Servers gibt es einige Vorteile:

- während der Client abgekoppelt ist, können andere Benutzer trotzdem auf die gleichen Daten zugreifen,
- es wird während der abgekoppelten Zeit keine teure Kommunikation benötigt,
- auch wenn keine Kommunikation möglich ist, kann der Client mit Daten des Servers arbeiten.

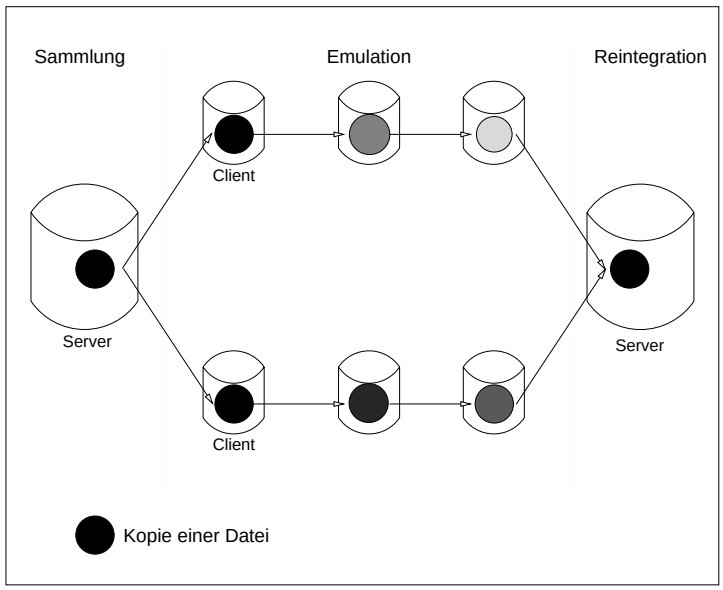

Abbildung1. Prinzip abgekoppelter Operationen

2.2 Probleme bei abgekoppelten Operationen

Die Emulation der Zugriffe auf den Server im abgekoppelten Zustand bringt einige Probleme mit sich:

Cache-Miss Ist ein Client abgekoppelt, kann er nur auf Daten zugreifen, die in seinem Cache (Hauptspeicher, Festplatte) sind. Will eine Anwendung auf Daten zugreifen, die nicht im Cache sind, spricht man von einem Cache-Miss. Darauf muß der Client eine Verbindung zum Server aufbauen, oder die Anwendung muß versuchen, ohne die Daten fortzufahren. Andernfalls blockiert die Anwendung.

Festschreiben von Dateien Das zweite Problem tritt beim Speichern von Dateien auf. Wenn der Client keine Verbindung zum Server aufbauen kann, können die Dateien nicht sofort zum Server übertragen werden. Solange müssen sie beim Client „vorübergehend" festgeschrieben werden. Festschreiben bedeutet, daß Dateien gespeichert werden, Transaktionen in einem Datenbanksystem ausgeführt werden usw. Erst wenn eine Verbindung zum Server wieder möglich ist, kann versucht werden, die Dateien endgültig beim Server festzuschreiben. Beim vorübergehenden Festschreiben bleibt die Wirkung also auf den Client beschränkt, wobei noch keine Aussage über die Wirkung des Festschreibens auf den Server getroffen werden kann.

Schreib-/ Lesekonflikte Beim Übertragen geändeter Dateien zum Server können zwischen den Versionen auf dem Server und den geänderten Versionen Schreib- und Lesekonflikte auftreten. Das Entdecken und auch das Lösen der Konflikte stellen Probleme von abgekoppelten Operationen dar, die gemeinsam betrachtet und gelöst werden sollten.

Es können zwei verschieden Arten von Konflikten auftreten ([DPS+94]):

- Schreib-Schreib-Konflikte: Ein Schreib-Schreib-Konflikt tritt auf, wenn zwei Clients die gleiche Datei verändern.
- Lese-Schreib-Konflikte: Ein Lese-Schreib-Konflikt tritt auf, wenn ein Client Daten verändert, die bereits veraltet waren, als er sie gelesen hatte.

Granularität der Daten Auch die Granularität der Daten spielt bei der Replikation eine Rolle (siehe [GKLS94]). Zum Beispiel besitzen Objekte einer OODB im Vergleich zu Dateien eines Dateisystems Eigenschaften, die andere Anforderungen an abgekoppelte Operationen aufdecken, weil

- OODB anders strukturiert sind, als Datei-Systeme: Objekte sind viel kleiner als Dateien, durchschnittlich ca. 100 Bytes. Zwischen Objekten bestehen im Gegensatz zu den strukturierten Namensräumen der Dateisysteme viele Verbindungen und Abhängigkeiten untereinander. Dadurch kann die Wirkung eines Konflikts eines Objekts sehr groß sein, weil sie sich auf viele andere Objekte auswirkt, die mit dem Objekt verbunden sind.
- OODB anders addressiert werden, als Datei-Systeme: Objekte werden durch Navigation und Queries addressiert. Adressierungen sind schwer vorauszusehen. Außerdem arbeitet der Benutzer mit atomaren Transaktionen, die serialisierbar und persistent sind. Dadurch kann sich ein Cache-Miss einer kleinen Datei auf die ganze Anwendung (bzw. Transaktion) auswirken.

2.3 Ablauf abgekoppelter Operationen

Abgekoppelte Operationen lassen sich nach [Kot95] in drei bzw. vier Phasen einteilen (siehe Abbildung 2):

Sammlung Anfangs befindet sich der Mobilrechner in der Sammlungsphase. Der Mobilrechner bereitet sich auf das Abkoppeln vor, indem er sich die Dateien repliziert, die er wahrscheinlich im abgekoppelten Zustand benötigen wird.

Emulation Wird der Client physikalisch abgekoppelt, beginnt die Emulationsphase: der Mobilrechner emuliert alle Zugriffe auf den Server mit lokalen Kopien im Cache.

Reintegration Wird er wieder physikalisch an das Netz angekoppelt, beginnt die letzte Phase (Reintegration). Die geänderten Dateien werden mit den Versionen auf dem Server synchronisiert.

Normalbetrieb Wenn der Client angekoppelt ist und keine Gefahr unangekündigten Abkoppelns besteht, ist er in keinen dieser Phasen, sondern in einem Zustand, der mit dem Normalbetrieb im Netz vergleichbar ist. Er ist voll angekoppelt und bereitet sich nicht auf ein Abkoppeln vor. Besteht allerdings das Risiko unangekündigten Abkoppelns, zum Beispiel bei einer instabilen GSM-Verbindung, ist der Mobilrechner nie in diesem Zustand, sondern muß immer mit einem ungewollten Abkoppeln rechnen.

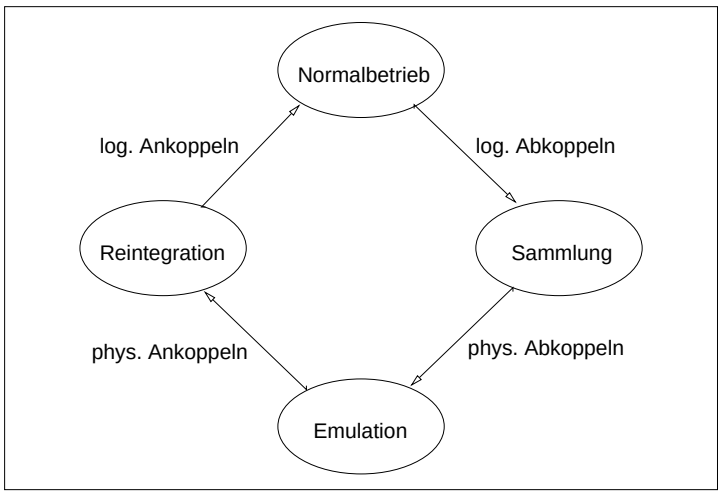

Abbildung2. Phasen abgekoppelter Operationen

3 Realisierungsalternativen bei abgekoppelten Operationen

In diesem Kapitel wird ein Merkmalsraum aufgespannt, der die Alternativen abgekoppelter Operationen beschreibt. Folgende Punkte werden betrachtet: Konsistenzsicherung,

Sammlung, Verbreitung von Updates und Konfliktbehandlung.

3.1 Konsistenzsicherung

Mit der Replikation von Daten sind immer zwei gegensätzliche Ziele verbunden: die Erhöhung der Verfügbarkeit und die Sicherung der Konsistenz. Deswegen ist die gundlegendste Frage die nach der Form der Konsistenzsicherung. Sie beeinflußt die anderen Punkte maßgeblich. Konsistenz beschreibt die Stabilität der Daten. Stark konsistente Daten sind stabil, das heißt, falls mehrere Kopien von Daten existieren, können keine Abweichungen untereinander auftreten. Starke Konsistenz kann erreicht werden, indem nur eine Kopie zugelassen wird, oder es muß dafür gesorgt werden, daß alle Kopien zu jeder Zeit, also auch in der verbindungslosen Zeit, up-to-date sind. Dadurch können zwar keine Konflikte auftreten, aber die Verfügbarkeit der Daten ist sehr eingeschränkt. Je schwächer die Konsistenz wird, desto mehr Abweichungen können zwischen verschiedenen Kopien einer Datei auftreten. Den Anwendungen werden so Daten zur Verfügung gestellt, die nicht garantiert aktuell sind, wodurch sowohl Schreib/Schreib- als auch Lese/Schreib-Konflikte auftreten können. Dafür steigt aber die Verfügbarkeit der Daten, weil sie sich leichter replizieren lassen. Somit steigt auch die Möglichkeit, abgekoppelte Operationen durchführen zu können.

Pessimistische Replikation Die pessimistische Replikation bringt die stärkste Form der Konsistenz. Will ein Benutzer eine Datei schreiben, wird sie für alle anderen gesperrt. Dies hat im Mobile Computing, wo ein Rechner längere Zeit vom Netz getrennt sein kann, wenig Sinn. Deswegen wird pessimistische Replikation dort nur dann eingesetzt, wenn der Client ausschließlich Leseberechtigung benötigt.

Pessimistische Replikation wird auch von verteilten Dateisystemen eingesetzt. Sie benutzen Netzwerk-Protokolle, um zu verhindern, daß Dateien im Cache nicht mehr aktuell sind. Zum einen gibt es Polling-Protokolle, wo der Client den Server über die Gültigkeit von Dateien erfragt (NFS). Zum anderen gibt es sogenannte „Callbacks“, wo der Server Tokens vergibt, die der Client zurückgeben muß. Für einen mobilen Client unterscheiden sich die beiden Protokolle erheblich: Polling erfordert nicht eine ständige Verbindung, bringt aber beträchtlichen Netzverkehr. Tokens dagegen erzeugen weniger Verkehr, setzen aber voraus, daß der Server jederzeit eine Verbindung aufbauen kann, falls sie noch nicht steht. Beide Möglichkeiten haben also erhebliche Mängel, wenn man mit hohen Verbindungskosten und schlechter Erreichbarkeit rechnen muß.

Optimistische Replikation Bei einer optimistischen Replikation werden Dateien nie gesperrt, sondern können jederzeit zum Lesen und Schreiben benutzt werden [HH95]. Ihre Konsistenz ist nur schwach. Deswegen müssen Operationen zur Konfliktentdeckung und -lösung bereitgestellt werden.

Snapshots Ein Snapshot ist eine Kopie von Daten und spiegelt diese zur Zeit seiner Erstellung wider [GKLS94]. Ist ein Snapshot einmal erstellt, hat er kein Bezug zu anderen

Kopien der Daten mehr: weder erhält er Updates, noch sendet er welche. Änderungen werden also nicht zurückgeschrieben. Die Daten können möglicherweise veraltet sein, trotzdem ist die Konsistenz eines Snapshots stark, weil er völlig unabhängig von anderen Kopien ist. Snapshots werden für Daten eingesetzt, mit denen zwar noch gearbeitet werden muß, die aber nicht mehr zurückgeschrieben werden müssen, zum Beispiel wenn Daten des letzten Quartals ausgewertet werden müssen. Außerdem werden Snapshots oft bei langer Verbindungslosigkeit, wo die Informationen rasch anwachsen, eingesetzt.

Konsistenz-Garantien Ein weiteres Problem der Konsistenzsicherung tritt auf, wenn ein Client auf mehrere Server schreiben darf. Falls nämlich ein Client seine Lese- und Schreiboperation auf verschiedene Server ausführt, die sich untereinander noch nicht synchronisiert haben, kann er eine inkonsistente Sicht auf die Daten bekommen. Führt er zum Beispiel zuerst eine Schreiboperation auf Server A aus und liest dann die gleiche Datei von Server B, während sich die Server A und B noch nicht aktualisiert haben, enthält die gelesene Datei nicht seine vorige Schreiboperation. Um das zu verhindern, kann man dem Client eine gewisse Konsistenz der Daten garantieren. Das könnten nach [TDP+91] zum Beispiel sein:

- Read your Writes: Leseoperationen beinhalten garantiert vorige Schreiboperationen; diese Garantie sichert die Konsistenz am stärksten.
- Monotonic Reads: Leseoperationen beinhalten garantiert die Schreiboperationen, die auch bei der vorigen Leseoperation berücksichtigt wurden.
- Writes follow Reads: Schreiboperation wird garantiert nach der Leseoperation, von der sie abstammt, verbreitet.
- Monotonic Writes: Schreiboperationen werden garantiert in der richtigen Reihenfolge verbreitet; diese Garantiesichert die Konsistenz am schwächsten.

Sie garantieren dem Client eine konsistente Sicht auf seine Daten, auch wenn er von verschiedenen Servern liest und schreibt, indem zu jeder Datei Versions-Vektoren gehalten werden, mit denen geprüft werden kann, ob die Server noch up-to-date sind.

3.2 Sammlung - welche Daten sollen in den Cache?

Um die richtigen Daten auszuwählen, gibt es mehrere Möglichkeiten:

- LRU-Strategie (Last Recently Used): Die simpelste Auswahl ist die Wahl der zuletzt benutzten Dateien. Diese müssen nicht mehr übertragen werden, da sie ja schon benutzt wurden und noch im Cache vorhanden sind. Es muß nur ihre Konsistenz überprüft werden. Will der Benutzer nur mit den Anwendungen fortfahren, die er während der Sammlungsphase gestartet hat, ist diese Strategie sinnvoll. Allerdings hat er kein Zugriff auf völlig andere Dateien und ist somit auf das Fortfahren der alten Anwendungen eingeschränkt.
- Benutzerdefinierte Auswahl: Bei dieser Strategie kann der Benutzer explizit angeben, welche Dateien er in den Cache haben möchte. Das können zum Beispiel seine

persönlichen Dateien sein: der Inhalt des Root- und des Mail- Verzeichnisses oder seine zuletzt bearbeiteten Dokumente. Weiterhin könnten das Dateien seiner typischen Applikationen sein, wie Editor, Compiler oder Entwicklungswerkzeuge. Ein drittes Beispiel wären bestimmte Datensammlungen: Source-Codes eines gerade zu bearbeitenden Projektes, Daten von Kunden, die besucht werden oder Dokumente, die nach dem Abkoppeln bearbeitet werden müssen. Der Benutzer sollte auch die Möglichkeit haben, Teile dieser Auswahl explizit während der Sammlungsphase anzugeben und so den Rechner speziell für sein Vorhaben nach dem Abkoppeln vorzubereiten.

– Abhängigkeit von Daten: Bei vielen Anwendungen, speziell bei Datenbanken, treten Abhängigkeiten zwischen Daten auf. Wird beispielsweise in einer objektorientierten Datenbank auf ein Objekt zugegriffen, ist es sehr wahrscheinlich, daß auch auf dessen referenzierte Objekte zugegriffen werden soll. Falls diese Objekte nicht in den Cache gelangt sind, ist meist ein Fortfahren nicht möglich. Deswegen ist es sehr wichtig, daß solche Abhängigkeiten erkannt werden. Da das Datei-System die Semantik der Dateien nicht kennt, sollte es den speziellen Anwendungen das Suchen nach Abhängigkeiten übertragen und dort sollte die Auswahl getroffen werden.

Für den Benutzer ist das Sammeln am effektivsten, wenn die oben genannten Strategien kombiniert werden. Sinnvoll wäre zum Beispiel die LRU-Strategie mit einer benutzerdefinierten Auswahl zu kombinieren (siehe [Kot95]).

3.3 Verbreitung von Updates

Genauso wichtig wie die Frage, welche Daten repliziert werden, ist die Frage, wann sie wieder zurückgeschrieben werden, sofern sie geändert wurden. Je geringer der zeitliche Abstand zwischen der Änderung und dem Rückschreiben wird, desto weniger Konflikte treten auf. Geschieht dagegen das Rückschreiben unabhängig von der Änderung, verringert der Client seine Abhängigkeit vom Server. Dadurch kann der Benutzer auch dann, wenn keine Verbindung zum Server möglich ist, weiterarbeiten und die Kommunikationskosten werden gesenkt, da die Kommunikation auf eine günstigere Zeit verlegt werden kann.

Synchrones Schreiben Verteilte Dateisysteme wie NFS oder AFS führen Schreiboperationen von Clients synchron aus. Das heißt, wenn ein Client eine Datei schreibt, werden die Änderungen sofort auf den Server übertragen. Im Mobile Computing ist das nur dann möglich, wenn der Client jederzeit eine Verbindung zum Server aufbauen kann.

Verzögertes Schreiben Die Änderungen werden asynchron übertragen, falls das Rückschreiben zeitlich unabhängig von der Änderung vorgenommen wird. Man spricht dann von verzögertem Schreiben. Dabei werden die Dateien auf der lokalen Festplatte des Clients vorübergehend gespeichert (siehe auch [HH95]).

Peer-To-Peer Anti-Entropie-Verfahren Dieses Verfahren ist ein Spezialfall von verzögertem Schreiben [DPS+94]. Es bedeutet, daß zwei beliebige Rechner, die miteinander kommunizieren können, ihre Aktualisierungen verbreiten. Danach haben beide Rechner die gleiche Version von der Datei, das heißt sie haben die gleichen Aktualisierungen in der gleichen Reihenfolge vorgenommen. Anti-Entropie kann inkrementell strukturiert sein, so daß auch Teilnehmer mit schlechter Verbindung nacheinander mit den verschiedenen Servern kommunizieren und so ihre Daten aktuell halten können. Das Peer-To-Peer Anti-Entropie-Verfahren ist für Clients prädestiniert, die Dateien von mehreren Servern repliziert haben.

3.4 Konfliktbehandlung

Mit dem Einsatz optimistischer Replikation können Konflikte zwischen verschiedenen Versionen einer Datei auftreten. Deswegen müssen in der Reintegrationsphase die verschiedenen Versionen synchronisiert werden, indem Konflikte gesucht und gelöst werden.

Bei der Konfliktentdeckung und -lösung gibt es prinzipiell zwei Möglichkeiten. Entweder wird die Semantik der Daten mit einbezogen oder nicht. Das Betrachten der Semantik hat den Nachteil, daß die Dateien nicht einheitlich behandelt werden können, so daß für jede Dateiart ein Lösungsalgorithmus zur Verfügung gestellt werden muß. Es hat den Vorteil, daß dabei der Inhalt der Daten betrachtet wird, wodurch viele Konflikte vermieden werden können. Da die Semantik von Daten nur Anwendungen kennen, die diese bearbeiten, wird bei der semantischen Konfliktbehandlung die Hilfe solcher Anwendungen benötigt.

Daß beim Vergleich zweier Dateien ohne Betrachtung der Semantik deutlich mehr Konflikte auftreten können, sieht man am Beispiel eines Terminkalenders. Angenommen eine Sekretärin und ihr Chef machen zeitgleich einen Termin aus, während das Notebook des Chefs nicht mit dem Computer der Sekretärin verbunden ist. Bei Wiederherstellung der Verbindung entdeckt man ein Konflikt, wenn man nicht den Sinn der konfliktbehafteten Datei kennt. Nur der Terminkalender kann entscheiden, ob der Konflikt zu lösen ist oder nicht. Zum Beispiel können zwei Termine, die ca. 1 Stunde dauern und 2 Stunden auseinander liegen, wahrgenommen werden, wenn sie beide im Büro stattfinden. Soll der eine Termin dagegen räumlich weit entfernt vom anderen stattfinden, können sie unmöglich beide wahrgenommen werden.

Konfliktentdeckung und -lösung hängen stark voneinander ab. Je simpler die Konfliktentdeckung gehalten wird, desto häufiger treten Konflikte auf und desto komplexer muß auch die Konfliklösung sein, um nicht jeden Konflikt abzulehnen. Wird dagegen bereits bei der Entdeckung die Semantik der Daten berücksichtigt, treten weitaus weniger Konflikte auf, die dann sehr kritisch sind und meist ohne Interaktion des Benutzers nicht gelöst werden können.

Die Konfliktbehandlung fängt bereits in der Emulationsphase an. Zum Beispiel werden zu einer Schreiboperation weitere Informationen gespeichert, wie ein Zeitstempel oder Informationen über die Version der Datei, von der die jetzt gespeicherte ursprünglich abstammt. Solche Informationen sind imminent wichtig, um in der Reintegrationsphase Konflikte zu lösen.

Konfliktentdeckung In der ersten Phase der Konfliktbehandlung muß nach Konflikten gesucht werden. Das heißt, die Unterschiede zweier Versionen einer Datei müssen entdeckt werden.

Konfliktentdeckung mittels Log-Dateien Bei der Konfliktentdeckung mittels Log-Dateien werden in der Emulationsphase alle Schreiboperationen auf den lokalen Daten durchgeführt und in einer Log-Datei protokolliert [Hil95]. Es gibt zwei atomare Operationen: INS - zum Einfügen - und DEL - zum Löschen. Die Log-Datei enthält zu jeder Änderung einen Zeitstempel, allerdings nicht zu einer globalen Zeit. Nach dem Ankoppeln werden dann die Log-Dateien auf den Server übertragen. Konflikte können erkannt werden, indem die Log-Dateien mit denen auf dem Server verglichen werden. Die Semantik der Daten wird nicht betrachtet. Da in den Log-Dateien keine Informationen über Leseoperationen enthalten sind - es werden ja nur Schreib- und Löschoperationen protokolliert) - ist es mit diesem Mechanismus nicht möglich, Lese/Schreib-Konflikte aufzudecken. Um abweichende Dateien zu identifizieren, kann ein Versions-Graph der Datei erstellt werden. Eine Replikation wird im Graphen als eine Teilung in 2 Wege dargestellt, eine Resynchronisation entsprechend als Zusammenfassung. Durch diesen Graphen kann der Benutzer leichter erkennen, wo der Konflikt aufgetreten ist.

Semantischer Vergleich Die Hilfe von Anwendungen kann bei der Konfliktentdekkung entweder darin bestehen, daß diese Anwendungen selbst nach Konflikten suchen, oder daß sie zu jeder Schreiboperation gewisse Anforderungen an die zu vergleichenden Dateien stellen. Ein Konflikt tritt auf, wenn bei Reintegration die Datei auf dem Server den Anforderungen nicht gerecht wird [DPS+94]. Zum Beispiel könnte bei einem Terminkalender so eine Anforderung zu einem Termineintrag sein, daß zu dieser Zeit an diesem Tag noch kein Termin eingetragen sein darf.

Konfliktlösung In der zweiten Phase der Konfliktbehandlung müssen nun die entdeckten Konflikte aufgelöst werden.

Wird ein Konflikt entdeckt, ist die simpelste Methode ihn zu lösen, die Schreiboperationen grundsätzlich abzulehnen. Das ist für abgekoppelte Operationen schlecht geeignet, da die Konflikte erst in der Reintegrationsphase entdeckt werden, was zeitlich weit entfernt vom vorübergehenden Schreiben auf die Festplatte des Clients liegen kann. Inzwischen könnten weitere Operationen ausgeführt worden sein, die auf die Schreiboperation aufsetzen, was zum Beispiel mit Log-Dateien erkannt werden kann. Diese Operationen wären dann alle ungültig. Tritt zum Beispiel in einer objektorientierten Datenbank bei einer Transaktion ein Konflikt bei einem Objekt auf, muß die komplette Transaktion, die aus vielen Schreiboperationen bestehen kann, zurückgesetzt werden, um die Konsistenz der Datenbank zu sichern. Desweiteren muß dem Benutzer das Mißlingen angezeigt werden; er sollte die Möglichkeit haben, die Transaktion basierend auf den frischen Daten erneut auszuführen.

Eine weitere Möglichkeit ist es, den Benutzer die Konflikte manuell lösen zu lassen. Das ist ebenfalls ungeeignet, da keine Aussagen getroffen wurden, ob der Benutzer bei der

Reintegration überhaupt anwesend ist. Es sollten also andere Maßnahmen vom Datei-System oder von den Anwendungen zur Konfliktlösung getroffen werden.

Applikationsabhängige Konfliktlösung Soll zur Konfliktlösung die Semantik der Daten einbezogen werden, benötigt man wie bei der Konfliktentdeckung die Hilfe der Anwendungen. Sobald ein Konflikt aufgetreten ist, wird ein anwendungs-spezifischer Konfliktlösungsmechanismus angestoßen, der versucht, den Konflikt transparent für den Benutzer zu lösen [SNKP95]. Die Wahl, welcher Mechanismus für welche Datei einzusetzen ist, kann anhand der Dateiendung oder an der Position in der Baumhierarchie erkannt werden, falls der Dateibaum hierarchisch aufgebaut ist.

Auch wenn der Konflikt nicht transparent zu lösen ist, wenn also trotzdem der Benutzer eingreifen muß, kann eine Anwendung hilfreich sein. Zum Beispiel kann sie die beiden Versionen einer Datei dem Benutzer nebeneinander anzeigen, so daß dieser leicht den Konflikt auflösen kann.

Mergeprocs Eine andere Art von abblikationsabhängiger Konfliktlösung ist der Einsatz von sogenannten Mergeprocs (siehe [DPS+94]). Jeder Schreib-Operation wird eine Prozedur angehängt, die bei Auftreten eines Konflikts ausgeführt wird. Sie liest die Kopie des Servers und versucht Alternativen zur Schreiboperation zu finden, führt diese aus und informiert den Benutzer. Tritt zum Beispiel bei einem Terminkalender ein Konflikt auf, weil zwei Termine sich überschneiden, kann die Merge-Prozedur Terminalternativen enthalten und einen noch freien Termin wählen.

4 Systembeispiele

In diesem Kapitel werden 4 Systembeispiele vorgestellt. Sie alle benutzen abgekoppelte Operationen, wägen aber auf unterschiedlicher Weise die einzelnen Lösungsalternativen ab, je nachdem für welchen Zweck sie konzipiert sind. Tabelle 1 gibt schließlich eine Übersicht über alle Systeme und deren Merkmale.

4.1 Das Coda-Datei-System

Coda ist ein verteiltes Datei-System und stammt von dem „Andrew File System“ (AFS) ab, das auch unter den Namen „Distributed File System“ im OSF DCE bekannt ist (siehe [SKM+93], [Kot95]). Dateizugriffe werden durch einen lokalen Cache-Manager ausgeführt, der auf jedem Client läuft und die Zugriffswünsche abfängt und ausführt. Solange der Client am Netz angekoppelt ist, wird der Cache mittels eines Callback-Mechanismus aktuell gehalten, indem der Server Schreibberechtigung an den Client vergibt, der sie wieder zurückgeben muß (siehe Kapitel 3.1). Das funktioniert nur solange, bis der Client vom Netz abgekoppelt wird. Dann führt Coda abgekoppelte Operationen aus, die auf optimistischer Replikation basieren.

Angekoppelt ist Coda immer im Sammel-Zustand und nie im Normalbetrieb, weil in einem Datei-System eine Menge Dateien anfallen, die auf den Client repliziert werden

müssen und die bei plötzlichem Verbindungsverlust nicht fehlen dürfen (zum Beispiel Konfigurationsdateien, Systemdateien, ...). Deswegen können Dateien mit der LRU-Strategie gesammelt oder explizit angegeben werden.

Coda versucht, Konflikte mit Hilfe von Log-Dateien zu entdecken, die sehr optimiert sind, um sie so klein wie möglich zu halten. Wie in Kapitel 3.4 bereits erwähnt, können damit nur Schreib/Schreib-Konflikte ermittelt werden. Lese/Schreib-Konflikte werden nicht berücksichtigt. Tritt ein Konflikt auf, muß er manuell vom Benutzer beseitigt werden. Applikationsabhängige Konfliktlösung wurde zuerst nicht unterstützt, was durch die Tatsache begründet ist, daß es in einem Dateisystem eine Vielzahl von verschiedenen Dateien gibt. Außerdem haben Tests gezeigt, daß weniger als 1% der Zugriffe einen Konflikt hervorrufen, unter auschließlicher Berücksichtigung von Benutzerdateien waren es sogar nur 0,3% [SKM+93]. Mittlerweile gibt es zusätzlich einen anwendungsspezifischer Konfliktlösungsmechanismus namens ASR (Application Specific Resolver). Dateiverzeichnisse sind auch eine Art von Dateien und zwar die einzigsten, deren Semantik ein Dateisystem kennt. Tritt dort ein Konflikt auf, berücksichtigt Coda bei der Konfliktlösung die Semantik und versucht, den Konflikt automatisch zu lösen. Zum Beispiel tritt kein Konflikt auf, wenn in einem Verzeichnis eine Datei erstellt und eine andere gelöscht wird.

4.2 Odyssey

Odyssey ist der Nachfolger von Coda und hat einen großen Unterschied: es unterstützt anwendungsabhängige Konfliktlösung [SNKP95]. In Odyssey werden Dateien in einem hierarchischen Namensraum gespeichert, der sich aus typ-spezifischen Unterbäumen namens „Tomes“ zusammensetzt. Das heißt, Dateien des gleichen Typs werden in dem gleichen Tome gespeichert. Die verschiedenen Tomes können sich auch durch unterschiedliche Caching-Strategien und Konsistenzarten unterscheiden [SNKP95]. Die Lage einer Datei in einem Tome bestimmt somit deren Datei-Typ.

Der Cache-Manager ist bei Odyssey in zwei Teile unterteilt. Der eine Teil, „Viceroy“, führt typ-unabhängige Funktionen aus, wie zum Beispiel Authentifizierung. Der andere, „Warden“, ist dagegen für typ-spezifische Funktionen zuständig, zum Beispiel die Konfliktlösung. Dabei gibt es für jeden Dateityp solch einen Warden. Dadurch ist Odyssey leicht um bestimmte Dateiarten erweiterbar - lediglich ein neuer Warden muß hinzugefügt werden. Tritt in Odyssey ein Konflikt auf, wird eine vom Datei-Typ abhängige ASR-Komponente aufgerufen.

4.3 Die objektorientierte Datenbank Thor

Thor ist ein verteiltes, objektorientiertes Datenbank-System, das abgekoppelte Operationen unterstützt [GKLS94]. Der Benutzer erzeugt atomare Transaktionen, wobei jede einzelne viele Objekte benutzt. Die Implementation von Thor ist verteilt: Persistente Objekte liegen auf dem Server, Applikationen laufen auf den Clients. Kopien der persistenten Objekte können beim Client zwischengespeichert werden. Thor bietet eine zuverlässige und immer verfügbare Speichermöglichkeit für persistente Objeke auf dem Server. Es

kann mehrere Server geben. Zu einer Zeit gehört aber ein persistentes Objekt zu einem bestimmten Server. Objekte können zu anderen Servern migrieren. Sie sind auf einem Server geclustert, so daß die meisten Referenzen zu Objekten gehen, die auf dem gleichen Server liegen.

In der Sammlungsphase können Objekte mit der LRU-Strategie oder mittels der Query-Sprache der Datenbank ausgewählt werden. Der Benutzer kann Anfragen formulieren, die dann in der Datenbank gespeichert und bei Bedarf eingesetzt werden, so daß der Benutzer sie nicht laufend neu eingeben muß. Thor basiert auf optimistischer Replikation, Transaktionen können also im abgekoppelten Modus vorübergehend festgeschrieben werden.

In der Reintegrationsphase sollte in Thor anwendungsabhängige Konfliktentdeckung und -lösung eingesetzt werden, da eine Datenbank die Semantik der Objekte kennt. Für jedes Paar von Transaktionen könnten zum Beispiel Konflikt-Checks spezifiziert werden, die anhand der Datenstruktur der Objekte erkennen können, ob Abhängigkeiten zwischen den Änderungen vorliegen. Das ist aber noch nicht realisiert. Wie in Kapitel 3 erwähnt, kostet diese Konfliktentdeckung und -lösung weitaus mehr Zeit, als die bloße Betrachtung von Log-Dateien. Deswegen sollte Thor dem Benutzer auch die Möglichkeit bieten, nur bei bestimmten Objekten, wo häufig Konflikte auftreten, die anwendungsabhängige Konfliktbehandlung durchzuführen. Thor bietet mit Snapshots (siehe Kapitel 3.1.3) eine weitere Möglichkeit, abgekoppelte Operationen durchzuführen.

4.4 Bayou

Das Bayou-System ist eine verteilte Datenbank, die als Plattform für verschiedene verteilte Anwendungen dient [DPS+94]. Es ist speziell für mobile Computer gedacht, die nicht gerade ideale Netzwerkverbindungen haben, zum Beispiel Verbindungen mit Modem, Docking Station oder Diskette. Die Betonung liegt dabei wie bei Thor auf applikationsabhängiger Konfliktlösung und optimistischer Replikation. Um trotz der schlechten Netzwerkverbindung mobil mit Daten arbeiten zu können, führen Clients abgekoppelte Operationen aus, die auf optimistischer Replikation basieren. Eine Datei kann auf mehreren Servern liegen, ein Client kann von jedem dieser Server eine Kopie beziehen. Sowohl die Clients als auch die Server untereinander verbreiten Updates nach dem Peer-To-Peer Anti-Entropie-Schema. Um sicher zu sein, daß eine Änderung gültig ist, wird für jede Datei ein Primary-Server bestimmt, der Änderungen endgültig festschreiben kann.

In Bayou kann die Konfliktbehandlung deswegen anwendungsabhängig durchgeführt werden, weil die Anwendungen, die auf Bayou aufsetzen, die Semantik der Dateien kennen. Sie fügen zu jeder Schreiboperation folgende Parameter hinzu:

- Dependency-Set: enthält Anforderungen an die Datei auf dem Server, wann ein Konflikt auftritt. Er besteht aus einer Reihe von Abfragen und erwarteten Ergebnissen. Stimmen die Ergebnisse der Abfragen auf dem Server nicht mit den erwarteten Ergebnissen überein, liegt ein Konflikt vor.
- Mergeproc: wird ausgeührt, wenn ein Konflikt auftritt; enthält Alternativen zu der Schreiboperation (siehe Kapitel 3.4).

Eine Schreiboperation besteht als aus einem vorgeschlagenen Udate, einem Dependency-Set und einer Mergeproc. Die Verifikation des Dependency-Set und das Ausführen der Mergeproc und der Schreiboperation wird auf dem Server vollzogen.

Da Bayou das Lesen und Schreiben auf verschiedene Server unterstützt, können Inkonsistenzen zwischen den einzelnen Versionen der Server auftreten. Bayou bietet deswegen den Anwendungen sogenannten „Session"-Garantien an. Das sind Garantien, die eine konsistente Sicht auf bereits ausgeführte Operationen garantieren (siehe Kapitel 3.1).

Verschiedene Anwendungen haben unterschiedliche Ansprüche an die Konsistenz der Dateien im Cache. Dafür bietet Bayou den Anwendungen folgende Möglichkeiten, zwischen Konsistenz und Verfügbarkeit der Dateien abzuwägen:

- Auswahl einer Session-Garantie,
- Leseoperationen liefern auf jeden Fall konsistente Daten,
- Zeitparameter, die bestimmen, wann Daten auf jeden Fall aufgefrischt werden.

Tabelle1. Übersicht über die Systembeispiele

	Konsistenz-sicherung	Sammeln	Verbreitung von Updates	Konflikt-entdeckung	Konflikt lösung
Coda	Optimismus	LRU, explizit	verzögert	Log	manuell
Odyssey	Optimismus	LRU, explizit	verzögert	Log	ASR
Thor	Optimismus	LRU, Query	verzögert	Konflikt-Checks	ASR
Bayou	Optimismus Session-Garant.	LRU, explizit	anti-Entropie	Dependency-Set	Mergeproc

4.5 Vergleich der Systembeispiele

Wie man in Tabelle 1 sehen kann, beutzen alle Sytembeispiele optimistische Replikation. Dennoch unterscheiden sie sich in ihren Anwendungsgebieten erheblich. Coda ist das universellste System. Es benötigt keine Unterstützung von Anwendungen, so daß beliebige Datentypen repliziert werden können. Odyssey bietet mit den Tomes schon Datentyp-spezifische Methoden an. Somit kann man für bestimmte Datentypen spezielle Konfliktbehandlungsmethoden einsetzen; deren Erstellung stellt aber zunächst einen Aufwand dar. Die objektorientierte Datenbank Thor nutzt das Wissen über die Semantik der Objekte sehr gut aus, zum Beispiel für die Sammlung durch Queries. Es ist speziell für Datenbankanwendungen geeignet, so daß das Einsatzgebiet sehr beschränkt ist. Bayou dagegen dient lediglich als Plattform für Anwendungen. Es kann also ohne spezielle Anwendungen gar nicht genützt werden. Dafür stellt es weitaus mehr Funktionen zur Verfügung, als die anderen Systeme, wie zum Beispiel die Unterstützung von mehreren Servern oder die Wahl einer Konsistenz-Garantie.

Allgemein kann man sagen: Je spezieller ein System für ein Anwendungsgebiet ausgerichtet ist, desto transparenter wird das Ausführen abgekoppelter Operationen für den Benutzer, da durch speziell ausgelegte Sammlungs- und Konfliktbehandlungsmethoden weniger Cache-Miss und Konflikte auftreten; desto größer werden aber auch die Anforderungen an bestimmte Anwendungen, solche Methoden zur Verfügung zu stellen. Tabelle 2 gibt noch einen Überblick über die Anwendungsgebiete der vorgestellten Systeme.

Tabelle2. Anwendungsgebiete der Systembeispiele

System	Anwendungsgebiet
Coda	alle Dateien
Odyssey	alle Dateien, mit Unterstützung spezieller Anwendungen
Thor	Datenbankanwendungen
Bayou	spezielle Anwendungen
	Anwendungen mit gewissen Konsistenzansprüchen
	bei Zugriffen auf mehrere Server

5 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Form des Datenmanagements im Moblie Computing, die Replikation von Daten, besprochen. Abgekoppelte Operationen ist dabei der wichtigste Ansatz. Er wurde näher erläutert, wobei die unterschiedlichen Alternativen herauskristallisiert wurden. In diesem Zusammenhang ist vor allen Dingen die optimistische Replikation zu nennen. Sie wird von allen Systembeispielen, die abschließend vorgestellt wurden, unterstützt.

Die Replikation ist aber nur eine Möglichkeit des Datenmanagements im Mobile Computing. Andere Ansätze sind zum Beispiel der Client-Agent-Server-Ansatz oder der Data-Shipping-Ansatz (siehe auch [IB93], [Kot95]). Aber gerade bei längerer Verbindungslosigkeit spielen abgekoppelte Operationen eine wichtige Rolle. Optimistische Replikation gepaart mit einer anwendungsabhängigen Konfliktbehandlung bieten ideale Voraussetzungen für das Arbeiten bei schlechter Netzwerkverbindung. Diese Techniken werden auch die Systeme der Zukunft unterstützen. Solche Systeme müssen auch berücksichtigen, daß verschiedene Anwendungen sehr unterschiedliche Konsistenzanforderungen stellen. So sollte der Trade-Off zwischen Konsistenzsicherung und Verfügbarkeit von Daten in Form von verschiedenen Klassen, wie zum Beispiel die Session-Garantien in Bayou, gewählt werden können.

Datenmanagement in Mobile Computing: Agenten vs. Replikation vs. Data-Shipping

Daniel Weidner

Kurzfassung

Beim Entwurf von verteilten Systemen müssen heutzutage vermehrt Aspekte einbezogen werden, die den Einsatz mobiler Rechner und mobiler Telekommunikation berücksichtigen. Die vorliegende Arbeit gibt zunächst einen Überblick über die wichtigsten dieser Aspekte. Im Hinblick auf Datenmanagement im Mobile Computing werden anschließend die drei wichtigsten Paradigmen Data-Shipping, Replikation und Agenten vorgestellt und ihre spezifischen Eigenschaften erläutert. Im letzten Teil kommt es schließlich zu einer vergleichenden Betrachtung dieser Paradigmen, welche Unterstützung bei Entwurfsentscheidungen geben soll.

1 Einleitung

Was die heutige Informationslandschaft prägt, ist zum einen die große Verbreitung von Rechnern aller Art in vielen Bereichen des täglichen Lebens. Dies umfaßt sowohl stationäre Rechner wie PCs und Workstations verschiedener Leistungsklassen, als auch in zunehmendem Maße mobile Rechner, deren Gewicht, Dimension und Leistung ebenfalls einer breiten Palette angehört.

Zum anderen hat in den letzten Jahren die Vernetzung von Rechnern sowohl in LANs als auch in WANs stark zugenommen. Insbesondere durch drahtlose Netzwerke wird selbst den mobilen Benutzern ein umfassender und nahezu nahtloser Zugang zu Kommunikationsdiensten ermöglicht.

Der Vision des Mobile Computing – mit dem Ziel auf Informationen jederzeit und an jedem Ort zugreifen zu können – kommen all diese Entwicklungen sehr entgegen. Die Umsetzung in die Praxis stellt jedoch besondere Ansprüche an die Architektur von verteilten Systemen, die von heute verfügbaren Systemen noch nicht in ausreichendem Maße berücksichtigt werden. Insbesondere ist es notwendig, das Datenmanagement an die Gegebenheiten des Mobile Computing anzupassen, um einen möglichst günstigen Kompromiß zwischen Wirtschaftlichkeit und Qualität erzielen zu können.

In der vorliegenden Arbeit werden drei verschiedene Ansätze – Data-Shipping, Replikation und Agenten – für die Gestaltung und Optimierung von verteilten Systemen vorgestellt und miteinander verglichen. Abschnitt 2 gibt einen Überblick über die Eigenschaften des Mobile Computing, indem diese anhand von Kriterien eingegrenzt werden. Abschnitt 3 präsentiert das Data-Shipping, das auf dem herkömmlichen Client-Server-Modell basiert. In Abschnitt 4 soll der Ansatz der Replikation eingeführt werden, indem einige Mechanismen vorgestellt werden, die für das Mobile Computing wichtig sind. Abschnitt 5 zeigt, wie das Client-Server-Modell hin zu dem Client-Agent-Server-Modell

erweitert werden kann, und welchen Nutzen man aus dem Einsatz von Agenten bei mobilen Anwendungen ziehen kann. Die Gegenüberstellung der drei Ansätze findet in Abschnitt 6 statt, wo deren Stärken und Schwächen bei verschiedenen Gegebenheiten herausgestellt und Möglichkeiten der Adaptierung aufgezeigt werden sollen.

2 Eigenschaften des Mobile Computing

Mobile Rechner werden eingesetzt von Benutzern, die den Anspruch haben, auch unterwegs nicht auf die Vorteile der Rechnerunterstützung verzichten zu wollen. Dabei kann die Nutzung dieses Hilfsmittels sehr unterschiedlich ausfallen. An dieser Stelle kann prinzipiell zwischen drei verschiedenen Anwendungsausprägungen unterschieden werden:

- Im einfachsten Fall wird ein mobiler Rechner für Produktivanwendungen wie z.B. Textverarbeitung oder Tabellenkalkulation eingesetzt. Dazu wird die Standard-Software installiert und Dateien werden mit dem stationären Bürorechner über Disketten oder Datenkabel ausgetauscht. Die Rechner werden also ähnlich wie private stationäre Rechner behandelt, weshalb diese Ausprägung aus der weiteren Diskussion ausgeschlossen werden soll.

- Für die zweite Ausprägung werden bereits Möglichkeiten zur mobilen Datenkommunikation vorausgesetzt, aber bei den Anwendungen handelt es sich noch um solche, die auch auf dem Festnetz üblich sind. Als Beispiel sei die Verwaltung des Terminkalenders eines Managers genannt, wo der Manager unterwegs einen Mobilrechner verwendet, während der Rechner seiner Sekretärin an das Festnetz gekoppelt ist.

- Die dritte Ausprägung bilden spezielle "Mobil-Anwendungen", die eigentlich nur bei der Verwendung von mobilen Rechnern Sinn machen, wie z.B. Navigationssysteme für den Straßenverkehr in der Stadt oder die Steuerung einer Lkw-Flotte. Zu der mobilen Datenkommunikation kommen hier noch besondere Echtzeit-Anforderungen.

Wie gesagt, interessieren für das Mobile Computing lediglich die zwei letzgenannten Ausprägungen. Derartige Systeme können anhand ihrer Komponenten, die in Abbildung 3 dargestellt sind charakterisiert werden. Eine wichtige Komponente stellt natürlich das mobile Endsystem dar. Dieses kann über diverse Kommunikationskanäle mit verschiedenen Festnetz-Rechnern bzw. Servern in Verbindung treten. Zentrales Element ist die mobile Anwendung, die zusammen mit den Benutzern einem bestimmten Zweck dienen soll. Schließlich spielt auch das Netz, innerhalb dessen die Kommunikation stattfindet, eine entscheidende Rolle. Weitere Ausführungen zu den Eigenschaften des Mobile Computing sind in [Dav96] und [Kot95] zu finden.

2.1 Die mobilen Endsysteme

Mobile Endsysteme treten in einer großen Variantenvielfalt auf. Wichtige Kriterien sind die Energie-Reserven, die Hardware-Ressourcen und das Risiko der Zerstörung oder des Diebstahls.

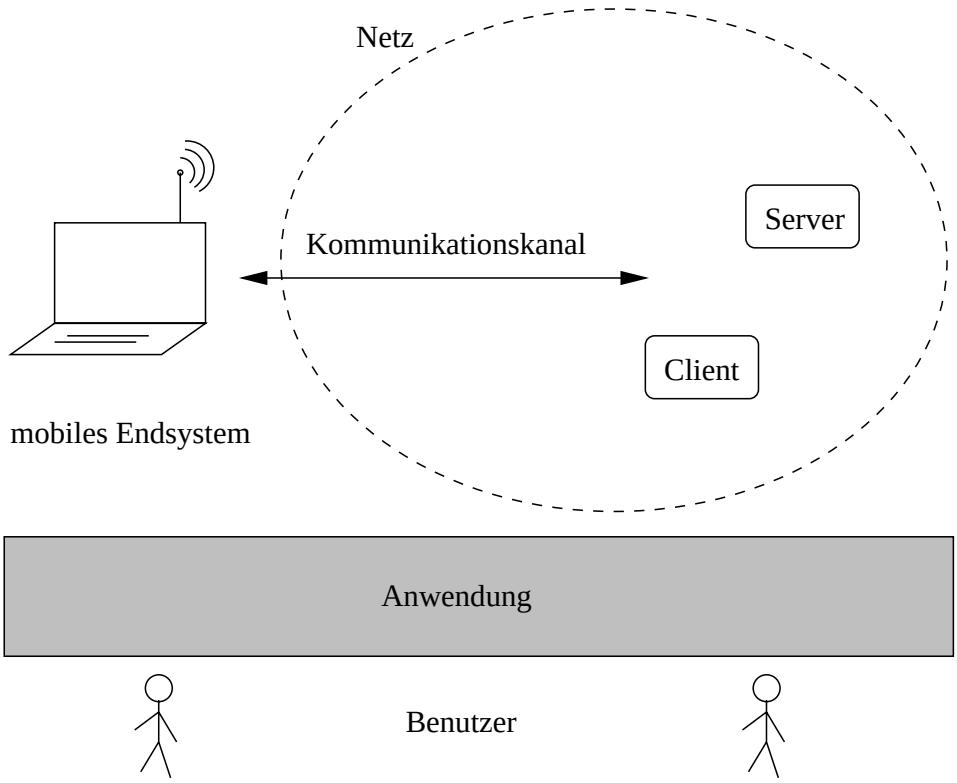

Abbildung3. Mobile Computing

Energie-Reserven: Bedingt durch die Tragbarkeit der Systeme, müssen diese über Batterien bzw. Akkus versorgt werden. Bei entsprechendem Energieverbrauch, der mit dem Umfang der Hardware-Ausstattung steigt, kommt es daher schnell zum Energie-Engpaß. Um diesem vorzubeugen, sind oft Sparzustände vorgesehen, deren Nutzung allerdings auch vom Benutzer bzw. von der Anwendung ermöglicht werden muß. Die Energie-Reserven können mit dem Aufenthaltsort variieren, z.B. wenn der Rechner in Gebäuden oder Fahrzeugen direkt an den Strom angeschlossen wird.

Hardware-Ressourcen: Die meisten mobilen Rechner sind mit weniger Speicherkapazität, kleinerer Rechenleistung und ärmerer Benutzer-Schnittstelle ausgestattet als entsprechende Festnetz-Rechner. Dies ist begründet durch Platzmangel, höhere Kosten für die tragbare Technologie und nicht zuletzt durch ein Energiespar-Bestreben beim Entwurf. Durch das Andocken in einem Büro und dem Anschluß von festen Bildschirmen, Tastaturen, oder weiterer Peripherie, können solche Begrenzungen allerdings zeitweise aufgehoben werden.

Zerstörung- bzw. Diebstahl-Risiko: Durch die Tragbarkeit, sind diese Risiken bei mobilen Rechnern wesentlich höher als bei Festnetz-Rechnern, folglich ist auch das Aufbewahren von Daten kritischer.

2.2 Die Kommunikationskanäle

Die Eigenschaften des Kommunikationskanals werden primär durch die verwendete Technologie zur mobilen Datenkommunikation geprägt. Wichtige Kriterien sind hier der Typ

des Netzanschlußes, die qualitativen Eigenschaften der Kommunikationsdienste (*QoS, Quality of Service*) sowie die vorliegende Kostenstruktur.

Typ des Netzanschlußes: Im Mobile Computing werden unterschiedliche Technologien zur mobilen Kommunikation eingesetzt. Die flexibelste ist die drahtlose Kommunikation, die aber aus Kostengründen nicht immer verwendet wird. In manchen Fällen begnügt man sich auch mit einem temporären Anschluß an das Festnetz, etwa über den Telefonanschluß im Hotel-Zimmer. Eine wichtige Eigenschaft der drahtlosen Kommunikation muß noch hervorgehoben werden, nämlich die Möglichkeit zum Broadcast (Senden einer Nachricht an alle Teilnehmer), die es machmal erlaubt, Anwendungen effizienter zu machen [WSD+95].

Quality of Service: Im Vergleich zur Festnetz-Kommunikation ist allgemein mit einer geringeren Leistungsfähigkeit zu rechnen, also geringere Bandbreite und erhöhte Latenzzeit. Die Leistungsfähigkeit unterliegt zudem erheblichen Schwankungen, die beispielsweise beim Wechsel zwischen verschiedenen Netzwerk-Zellen bei drahtloser Kommunikation auftreten. Zeitweise kann auch die Verfügbarkeit eines Kanals vollständig ausfallen, z.B. wenn sich das Gerät in einem Funkschatten befindet. Damit ist auch die Abbruchwahrscheinlichkeit sehr hoch, der bevorstehende Abbruch kann jedoch in vielen Fällen vorhergesagt werden, etwa wenn das Funksignal zusehends schlechter wird. Auch hier können je nach Aufenthaltsort Büro, freie Natur, Fahrzeug unterschiedliche Konnektivitätsmöglichkeiten gegeben sein, die einen besseren oder schlechteren *QoS* zulassen [Sat96].

Kostenstruktur: Die mobile Kommunikation ist meist mit erheblichen Kosten verbunden [HSW94]. Man unterscheidet zwischen zwei prinzipiellen Kostenmodellen, die zum einen die Verbindungsdauer und zum anderen die Menge der übertragenen Daten in Rechnung stellen. Das verbindungsorientierte Modell ist von Telefondiensten her bekannt, wobei die Verbindungszeit in feste Zeitintervalle zerlegt wird, deren Anzahl in Rechnung gestellt wird. Es muß also auch für die Wartezeiten bezahlt werden, in denen keine Kommunikation stattfindet. Anders bei dem nachrichtenorientierten Modell, wo für die Menge der abgeschickten Datenpakete bezahlt werden muß. Die Kosten sind jedoch generell höher als wenn es im verbindungsorientierten Kostenmodell jedesmal gelingen würde, die Verbindungszeit gut auszunutzen. Durch die Mobilität tauchen immer wieder Gelegenheiten auf, durch direkte Anschlußpunkte an das Festnetz günstiger zu kommunizieren, beim Ankoppeln in der Heimumgebung der Anwendung wird es sogar fast kostenlos.

2.3 Die Anwendung

Bei dem Entwurf oder der Optimierung von Systemen im Mobile Computing, müssen immer wieder die Eigenschaften der verwendeten Anwendungen betrachtet werden. Einige Eigenschaften werden durch das Pflichtenheft vorgegeben, andere sind nur schwer meßbar, weil sie von dem Verhalten und den Interessen der Benutzer abhängen [BI94]. Letzteres ist schwer modellierbar, aber durch das Sammeln von Erfahrungswerten können

Tendenzen ermittelt werden. Folgende Kriterien sollen an dieser Stelle angesprochen werden:

Zugriffs- und Änderungsraten: Die verschiedenen Datenelemente, die von der Anwendung verwaltet werden, weisen unterschiedliche Häufigkeiten des Zugriffs und der Veränderung auf. Je nach Anwendung können diese mehr oder weniger genau modelliert werden.

Voraussagbarkeit und Lokalität: Für viele Anwendungen ist es voraussagbar, auf welche Datenelemente in der nächsten Zukunft zugegriffen wird. Meist kann dies aus einem längeren Erfahrungszeitraum der Vergangenheit und einer entsprechenden Modellierung des Benutzerverhaltens abgeleitet werden. Zu diesem Aspekt gehört auch die allgemein typische Lokalität von Datenzugriffen.

Abgekoppelte Operation: In manchen Fällen wird gewünscht, daß das mobile Endsystem über einen längeren Zeitraum hinweg abgekoppelt, ohne Kommunikation, operieren kann, sei es um Kosten oder Energie zu sparen, oder weil zeitweise keine geeigneten Zugänge zum Netz zur Verfügung stehen.

Anforderungen an die Konsistenz: In anderen Fällen werden dagegen Echtzeit-Anforderungen gestellt, was die Konsistenz und Aktualität von Informationen in der gesamten Anwendung angeht [WSD+95].

Sicherheitsanforderungen: Hier stellt sich die Frage, inwieweit die übertragenen Daten gegen das Abhören und Verfälschen und der Zugang gegen unerwünschte Eindringlinge gesichert werden müssen.

Umfang der Daten: Bei multimedialen Anwendungen beispielsweise, können sehr umfangreiche Datenmengen anfallen. Dieser Aspekt muß im Zusammenhang mit der Wahl von geeigneten Kommunikationskanälen behandelt werden. Auch müssen Möglichkeiten der Datenkompression zur Qualitätsanpassung von Informationen untersucht werden.

Komplexe Dienste: Die von der Anwendung verwendeten Dienste können unterschiedliche Komplexität besitzen. Ein komplexer Dienst beansprucht viele verschiedene Datenquellen, es können auch umfangreichere Berechnungen vorkommen und durch die Zusammensetzung aus mehreren einzelnen Diensten treten Zwischenergebnisse auf, die zwischengespeichert und ausgewertet werden müssen. Die Steuerung solcher Vorgänge kostet Zeit und Ressourcen, die unterschiedlich zwischen den beteiligten Instanzen und Rechnern aufgeteilt werden können.

2.4 Die Netz-Infrastruktur

Anwendungen des Mobile Computing bedienen sich existierender Netz-Infrastrukturen bzw. werden erst durch die Bereitstellung bestimmter Kommunikationsdienste ermöglicht. Ziel ist es, möglichst weltweit einen günstigen Zugang zur Daten-Kommunikation zu haben. Das heißt die Kommunikation spielt sich innerhalb universeller Netze wie

z.B. dem Internet oder firmeneigenen Intranets ab und die Benutzer sind abhängig von diversen Dienstanbietern, die z.B. lokale Zugänge über drahtlose Verbindungen oder Telefonnetze erlauben.

Verbreitung standardisierter Dienste: Da es sich hier um heterogene und sehr weit verteilte Umgebungen handelt, stellt sich die Frage, welche Dienste an den verschiedenen Zugangspunkten der Netze zur Verfügung gestellt werden. Besonders wenn es um optimierende Mechanismen geht, die für das Mobile Computing wichtig sind, dann muß die Installation entsprechender Dienste eine große Verbreitung finden, um sie auch von überall her verfügbar zu machen. Eine große Rolle spielt hier auch die Standardisierung von Diensten, ansonsten muß die Anwendung sehr vielseitig sein, um mit verschiedenen Standards umgehen zu können.

Sicherheitsaskpekte: Durch die Benutzung globaler Netzwerke können sich zusätzliche Sicherheitslücken ergeben. Bei der Verwendung von fremden Zugangspunkten oder von Diensten bei unbekannten Dienstgebern muß sichergestellt werden, daß diese zuverlässig und korrekt arbeiten und die Sicherheitsanforderungen der Anwendung erfüllt werden. Da sich die Dienstgeber auch selbst schützen müssen, kann für einen mobilen Benutzer auch der Zugang zu Dienstleistungen erschwert werden.

3 Data-Shipping

Zur Strukturierung von verteilten Anwendungen wird heute häufig das Client-Server-Modell als konzeptionelle Grundlage verwendet. Eine verteilte Anwendung besteht hierbei aus einer Menge von Instanzen bzw. Prozessen, die als Diensterbringer (Server) oder als Dienstbenutzer (Clients) in Erscheinung treten können. Auf Anwendungsebene werden Protokolle bereitgestellt, die es einem Client erlauben eine anwendungsspezifische Dienstanforderung an einen üblicherweise entfernten Server zu stellen. Nach Erbringung des Dienstes durch den Server, gibt dieser dem Client eine Statusmeldung sowie eventuelle Berechnungsergebnisse zurück. In der Regel läuft dieser Vorgang synchron ab, d.h. der Client wartet nach der Dienstanforderung solange, bis er die Rückmeldung des Servers erhält.

Eine einfache Erweiterung dieses Modells stellt das Prinzip der Unterbeauftragung dar, wobei ein Server während der Diensterbringung seinerseits eine Dienstanforderung an einen weiteren Server stellt, dem er temporär als Client gegenübertritt. Durch das Fortsetzen dieses Prinzips wird eine beliebige Schachtelung entfernter Operationen ermöglicht. Es können also selbst sehr komplexe Dienste relativ einfach modelliert werden, indem sie auf eine Schachtelung einfacherer Dienste abgebildet werden.

Eine existierende verteilte Anwendung nach dem oben beschriebenen Client-Server-Modell läßt sich im Prinzip leicht auf die Nutzung mobiler Rechner erweitern: Die Server-Instanzen laufen wie gewohnt auf Festnetzrechnern, während auf den netzwerkfähigen Mobilrechnern die entsprechenden Client-Anwendungen installiert werden müssen.

Wie dies in Abbildung 4 angedeutet ist, befinden sich beim Data-Shipping alle Daten auf anderen Rechnern, vornehmlich auf Festnetz-Servern. Jedesmal, wenn die Anwendung

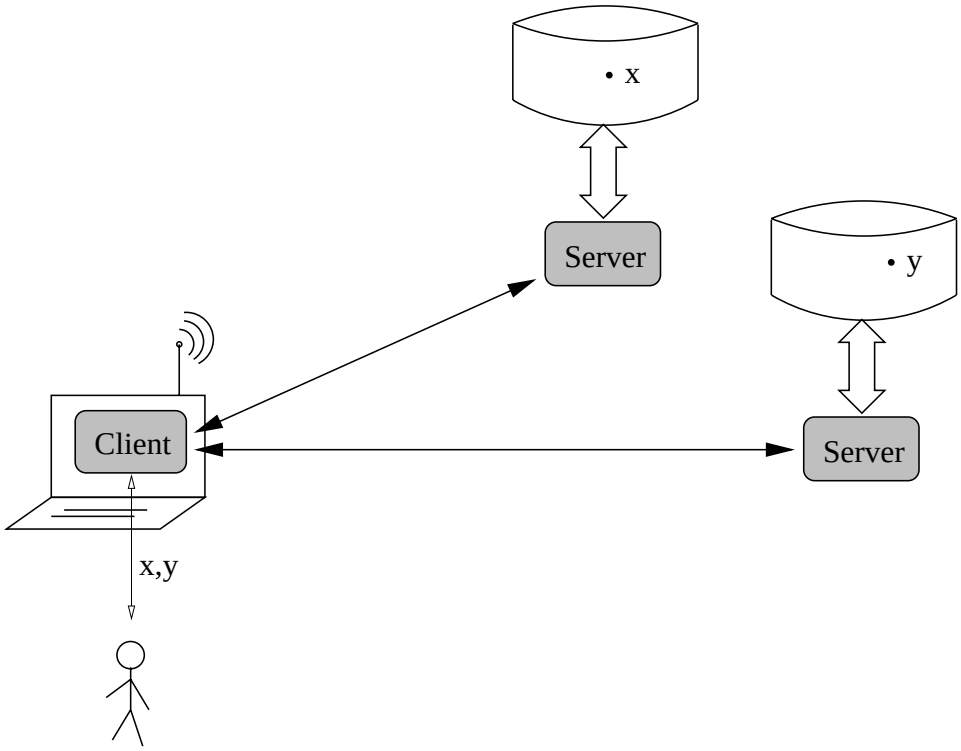

Abbildung4. Client-Server-Modell und Data-Shipping

auf bestimmte Daten zugreifen möchte, findet eine Anfrage an einen Server statt, der daraufhin die Daten verändern bzw. an den Client zurückliefern kann.

4 Replikation

Allgemein besteht der Ansatz der Replikation darin, Daten in mehreren Kopien innerhalb des Netzes zur Verfügung zu stellen. Im Fall des Mobile Computing bietet es sich insbesondere an, Kopien auf den mobilen Rechnern anzulegen.

Dabei werden mehrere Ziele verfolgt. Zum einen sollen die Kommunikationskosten reduziert werden, indem die Verteilung des Kommunikationsbedarfs über der Zeit verändert wird. Das heißt, es sollte nicht jeder Zugriff auf Anwendungsdaten systematisch einen Kommunikationsvorgang auslösen, wie dies beim Data-Shipping der Fall ist. Zum anderen soll die Möglichkeit zur abgekoppelten Operation gegeben werden, wo über bestimmte Zeiträume hinweg überhaupt keine Kommunikation zwischen mobilem Rechner und Server stattfindet und der mobile Benutzer trotzdem mit der Anwendung arbeiten kann. Ein weiteres Ziel ist die Beschleunigung des Zugriffes, um geringer Bandbreite und hoher Latenzzeit entgegenzuwirken. Gleichzeitig soll auch der Energie-Verbrauch reduziert werden, d.h. im mobilen Einsatz soll die energieintensive Kommunikation nach Möglichkeit vermieden werden und mögliche Sparzustände der Hardware sollen ausgenutzt werden.

Um diese Ziele erreichen zu können, müssen das System und die Anwendung einige wichtige Anforderungen erfüllen. Zunächst muß auf dem mobilen Rechner genügend Speicher vorhanden sein, um die notwendigen Kopien unterzubringen. Die Auswahl der anzulegenden Kopien ist um so schwerwiegender, je begrenzter dieser Speicherplatz ausfällt.

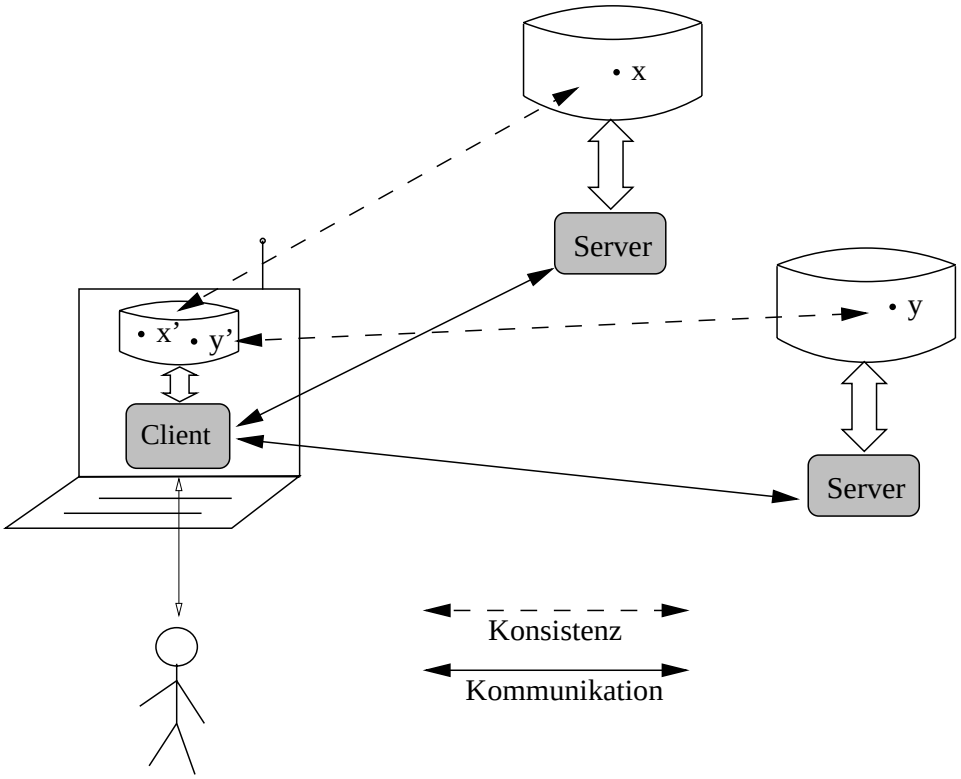

Abbildung5. Replikation

Hierbei spielt die Voraussagbarkeit und die Lokalität von Datenzugriffen eine wichtige Rolle, um effiziente Replikations-Strategien zu entwickeln. Wie dies in Abbildung 5 angedeutet ist, geht es also vor allem darum, daß erstens die richtigen Datenelemente repliziert werden und daß zweitens gewisse Konsistenzbedingungen zwischen den verschiedenen Kopien eingehalten werden [WSD+95].

Der Zugriff soll dabei möglichst transparent bleiben. Das gilt für den Benutzer, der die gewohnte Schnittstelle zur Anwendung, trotz Replikation, weiterbenutzen soll, aber auch den Anwendungsprogrammen soll idealerweise verborgen bleiben, wie der Zugriff tatsächlich ausgeführt wird. Dies setzt natürlich voraus, daß die Replikation auf Systemebene unterstützt und umgesetzt wird. Die Transparenz kann jedoch nicht vollständig sein, da in manchen Fällen Entscheidungen des Benutzers oder der Anwendung gefragt sein können [Sat96].

Abhängig von der Anwendungsart können sehr unterschiedliche Replikations-Verfahren zum Einsatz kommen. Anhand von zwei verschiedenen Anwendungsausprägungen sollen dazu im folgenden einige Ideen vorgestellt werden. Die erste Ausprägung stellt die verteilte Produktivanwendung dar, bei der längeres Arbeiten am abgekoppelten Mobilrechner und Modem-Kommunikation über das feste Telefonnetz üblich sind, während es sich bei der Zweiten um eine reine Mobil-Anwendung handelt, wo Echtzeit-Bedingungen in Verbindung mit drahtloser Kommunikation erfordert werden.

4.1 Verteilte Produktivanwendungen mit Mobilrechnern

Bei mangelnder Systemunterstützung wird ein Benutzer möglicherweise selbst dazu übergehen, manuell ein Replikationsverfahren anzuwenden, das seinen Bedürfnissen entspricht. Die typische Vorgehensweise ist die folgende: Der Benutzer lädt die benötigten Dateien vom Festsystem auf den Mobilrechner herunter, bevor er diesen vom Stromnetz und vom LAN abkoppelt. Dann arbeitet er mit den Kopien solange er unterwegs ist, und führt bei seiner Rückkehr schließlich ein entsprechendes Update auf dem Festsystem durch.

In [BCK95] ist von Studien die Rede, nach denen eine PC-Diskette genügend Platz bieten würde, um einem durchschnittlichen Benutzer mehrere Wochen lang die nötigen Informationen vorzuhalten. Es stellt sich natürlich die Frage, welche Dateien auf eine solche Diskette kopiert werden sollten und welche nicht, bzw. welche Dateien im Sinne einer Ausmusterungs-Strategie mit der Zeit wieder gelöscht werden sollten. All dies ist die Aufgabe von vorausschauenden Replikations-Verfahren.

Um voraussagen zu können, welche Daten der Benutzer im abgekoppelten Zustand brauchen wird, muß zuvor ein geeignetes Benutzermodell erstellt werden. Das kann geschehen, indem über längere Zeit Wissen über sein Verhalten gesammelt und ausgewertet wird. So können periodisch auftretende Zugriffe modelliert werden und zu jeder logischen Aufgabe, die der Benutzer bearbeitet, kann eine bestimmte Aggregierung der benutzten Dateien ausgemacht werden. Die Auswahl der zu replizierenden Dateien entsteht also durch die Kooperation zwischen dem Benutzer und dem System. Der Benutzer gibt lediglich grob an, welche Aufgaben er im abgekoppelten Zustand bearbeiten möchte. Das System bestimmt, welche diversen Dateien, Systemdateien, Hilfsprogramme und dergleichen dazu gebraucht werden, repliziert sie automatisch, und entlastet somit den Benutzer von dieser für ihn unübersichtlichen Arbeit.

Während das mobile System unterwegs und abgekoppelt ist, werden beliebige Lese- und Schreiboperationen sowohl auf Client- wie auch auf Server-Seite zugelassen. Es wird also eine optimistische Zugriffs-Strategie angewendet, bei der auftretende Konflikte erst bei der Reintegration, wenn die Daten des Mobilrechners mit denen des Festnetzes abgeglichen werden, erkannt werden können.

Um mit Konflikten sinnvoll umgehen zu können, muß die anwendungsspezifische Semantik der Dateiinhalte bekannt sein. Wenn eine Datei in zwei Kopien unterschiedlich verändert wurde, muß das auf der Ebene der Semantik nicht immer ein Konflikt darstellen, da die zwei Änderungen oftmals miteinander vereint werden können. Dagegen können durchaus semantische Inkonsistenzen zwischen zwei verschiedenen Dateien auftreten, deren Inhalte miteinander in Zusammenhang stehen, obwohl jeweils nur eine ihrer Kopien verändert wurde. Bei einer verteilten Terminkalender-Verwaltung beispielsweise, muß die Darstellung der Dateien bekannt sein, um feststellen zu können, ob zwei konkurrierende Schreibzugriffe tatsächlich eine Terminkollision bedeuten. Tritt eine Kollision auf, dann kann bei der Reintegration entweder eine Strategie Anwendung finden, die automatisch Ausweichtermine sucht, oder es muß jede Kollision dem Benutzer zur Auflösung vorgelegt werden.

In vielen Anwendungen muß der Benutzer als Entscheider in die Auflösung von Kon-

flikten einbezogen werden, also kann die optimistische Zugriffs-Strategie nur dann akzeptabel sein, wenn nicht zu viele Konflikte auftreten. In der Praxis zeigt es sich, daß diese in der Tat nur sehr selten auftreten. Dies belegen realistische Studien mit einem replizierten Unix-Dateisystem, von denen in [BCK95] berichtet wird. Bei Anwendungen des akademischen und wirtschaftlichen Bereichs wurde dort ermittelt, daß nur $0,0035$ Prozent aller Schreibzugriffe auf Dateien Konflikte erzeugten und nur $0,0018$ Prozent der neu erzeugten Dateien zu Namenskonflikten führten.

Normalerweise ist das Betriebssystemen nicht in der Lage die Dateiinhalte der Anwendungen zu interpretieren. Also müssen anwendungsspezifische Erweiterungen eingesetzt werden, die die semantischen Konflikte erkennen und, eventuell im Dialog mit dem Benutzer, diese auflösen. Eine weitere Möglichkeit stellt die Verwendung von Warteschlangen dar, in denen die Änderungsoperationen im abgekoppelten Zustand festgehalten werden und erst bei der Reintegration auf dem Partnerrechner ausgeführt werden, wobei dann Kollisionen direkt durch die Anwendung erkannt und entsprechend mit Hilfe des Benutzers aufgelöst werden können. Üblicherweise werden auch für Email- und Druck-Dienste im abgekoppelten Zustand Warteschlangen verwendet, denn es handelt sich dabei ebenfalls um eine Form der Replikation.

4.2 Mobil-Anwendung drahtlos und in Echtzeit

Betrachtet man Anwendungen, die speziell für das Mobile Computing zugeschnitten sind, dann liegt ein anderes Systemmodell zugrunde: Es besteht aus einer Population von mobilen Rechnern, die auf der Basis eines drahtlosen Netzwerks Informationen aus einer i.allg. auf dem Festnetz verteilten Datenbasis anfordern. Die Mobilrechner bewegen sich von Ort zu Ort und werden somit dynamisch unterschiedlichen Netzwerk-Zellen und Servern zugeordnet. Die angeforderten Informationen sind oft von ihrer Position abhängig, z.B. interessiert sich ein mobiles Navigationssystem für den Straßenverkehr nur für Stau- und Behinderungsmeldungen, die für den Verkehrsteilnehmer relevant sind. Die Mobilrechner verharren oft lange Zeit in ihrem Energiesparzustand oder sind ausgeschaltet, aber beim Einschalten sollen die Informationen möglichst schnell und aktuell verfügbar sein [WSD+95].

Dem Ziel des Energie- und Kostensparens steht in diesem Fall also eine wichtige Anforderung gegenüber, nämlich, daß Dateninkonsistenzen zwischen Mobilstation und Festnetz nur innerhalb eines stark begrenzten Zeitintervalls tolerierbar sind. Hier wird der Entwurf von Caching-Strategien entscheidend von der genauen Kenntnis bzw. Ausnutzung der Eigenschaften der mobilen Datenkommunikation beeinflußt. Zum Beispiel kann die dem Funk inhärente Möglichkeit des Broadcasting vom Festnetz aus genutzt werden, um leicht eine große Anzahl mobiler Funkteilnehmer auf dem Laufenden zu halten und gleichzeitig deren Energiereserven zu schonen. Entsprechende Algorithmen werden in [BI94] diskutiert und auf ihre Leistungsfähigkeit analysiert.

Bei den hier diskutierten Anwendungen kann die Modellierung der Benutzer-Interessen ebenfalls notwendig sein, um prediktives Caching zur Verbesserung der Zugriffszeiten zu verwenden. Da Änderungen der Daten sowohl auf Client- wie auch auf Server-Seite möglichst schnell konsistent gemacht werden sollen, stellt sich die Frage, für welche Da-

tenelemente eine Kopie auf dem mobilen Rechner anzulegen ist und für welche Datenelemente ein entfernter Zugriff auf dem Festnetz vorzuziehen ist. In [HSW94] wird dieser Frage nachgegangen, indem diese zwei Allokations-Schemas gegenübergestellt werden: Greift die mobile Anwendung häufig auf ein Datenelement x zu, dessen Änderungsrate im Server vergleichsweise niedrig ist, dann ist es günstig eine Kopie von x anzulegen. Ist der Zugriff auf x dagegen selten, und x wird im Server häufig verändert, dann ist es günstiger das Datenelement direkt vom Server anzufordern. Daraus wird dann eine dynamische Strategie abgeleitet, die in Abhängigkeit der Zugriffsstatistiken der nahen Vergangenheit zwischen diesen zwei Allokations-Schematas umschaltet.

Die Bewertung einer Caching-Strategie kann in Abhängigkeit des zugrundeliegenden Kostenmodells verschieden gut ausfallen. Beim verbindungsorientierten Modell kommt es darauf an, daß die Anzahl der Verbindungseinheiten minimiert wird und diese möglichst optimal ausgenutzt werden. Bei dem nachrichtenorientierten Modell geht es dagegen um die Minimierung der Menge übertragener Informationen.

5 Agenten

5.1 Motivation und Prinzip

Im herkömmlichen Client-Server-Modell wird die Abwicklung von komplexen Diensten direkt vom Client gesteuert, der die unterschiedlichen Elementar-Dienste aufruft, Zwischenergebnisse auswertet und weiterleitet, oder Server benutzt, die fest vorgegebene komplexe Dienste durch Unterbeauftragung anbieten. Wie schon im Zusammenhang mit Data-Shipping erwähnt, kann diese Vorgehensweise im Mobile Computing gravierende Nachteile mit sich bringen, weil die besonderen Eigenschaften der mobilen Endsysteme und der mobilen Kommunikation durch Clients, die für das Festnetz geschrieben und optimiert wurden, nicht berücksichtigt werden.

Abhilfe soll hier durch die Erweiterung zum Client-Agent-Server-Modell geschafft werden, mit dem die Asymmetrie, die zwischen Mobilrechnern und Mobil-Kommunikation einerseits und Festnetzrechnern und Festnetz-Kommunikation andererseits besteht, explizit unterstützt werden kann [Kot95]. In der Abbildung 6 ist das Prinzip einer Dienstabwicklung mit Agenten in drei Schritten dargestellt.

Im ersten Schritt wird ein durch den mobilen Client formulierter Ausführungsplan, der eine programmiersprachliche Beschreibung des komplexen Dienstes darstellt, an einen Festnetz-Server geschickt, der eine entsprechende Ausführungsumgebung zur Verfügung stellt. Nach diesem kurzen Kommunikationsvorgang kann der mobile Rechner wieder abgekoppelt werden, um Energie und Kommunikation einzusparen.

Im zweiten Schritt wird eine aktive Instanz, der Agent, erzeugt, der nun wie ein herkömmlicher Festnetz-Client den verschiedenen Servern gegenübertritt und den komplexen Dienst gemäß dem vorliegenden Ausführungsplan erledigt. Der Agent kann also die Ressourcen des Festnetz-Rechners benutzen um den Vorgang zu steuern, Zwischenergebnisse zu bearbeiten, Berechnungen durchzuführen, wobei er unter Umständen von wesentlich besserer Rechenleistung, schnellerer Hardware, kostengünstigerer und leistungsfähigerer Kommunikation profitieren kann.

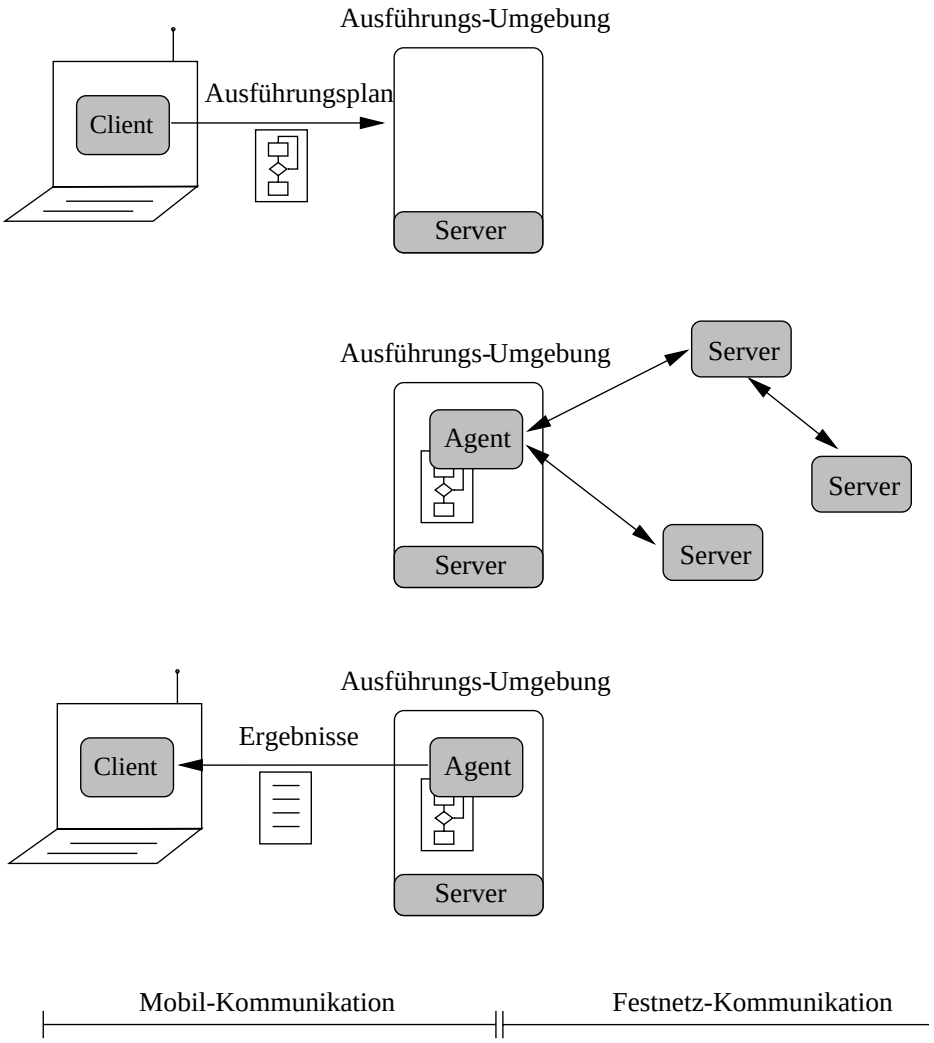

Abbildung6. Dienstabwicklung im Client-Agent-Server-Modell

Nachdem der Agent sein Ziel erreicht hat und die Ergebnisse vorliegen, meldet er diese im dritten Schritt an den mobilen Client zurück, indem er den Mobilrechner zum Beispiel aus einem wartenden Sparzustand aufweckt und die Daten in einer wiederum kurzen Kommunikationsphase überträgt. Im Gegensatz zum Data-Shipping handelt es sich also, vom Client aus gesehen, um eine asynchrone Dienstabwicklung, während der Agent als Vertreter des Clients auf dem Festnetz weiterhin synchrone Dienstaufrufe verwenden kann.

5.2 Funktionalität und Fähigkeiten der Agenten

Die Konzepte, nach denen Agenten-Systeme entworfen werden, können sehr unterschiedlich sein. Sie unterscheiden sich vor allem hinsichtlich ihrer Komplexität, die wiederum davon abhängt, welche Art von Aufgaben an einen Agenten gestellt werden sollen.

– Bei einfachen Systemen reicht es häufig, wenn ein Agent Operationen steuern kann, deren Ablauf vom Benutzer und von der Anwendung fest vorgegeben ist. Dazu gehört zum Beispiel das Formatieren und Ausdrucken eines umfangreichen Dokuments. In

diesem Fall ist die Steuerung sehr einfach, aber die Komplexität der Anwendung kann groß sein, somit können die Betriebsmittel des mobilen Endsystems stark entlastet werden.

- Bei der nächsten Entwicklungsstufe besteht die Aufgabe des Agenten darin, ein vorgegebenes Ziel zu erreichen, wobei durch Verhaltensregeln eine gewisse Strategie vorgegeben sein kann. Zum Beispiel, wenn ein Flug gebucht werden soll, dann muß der Benutzer u.a. angeben, welche Vorstellungen er hinsichtlich Preis, Leistung und Service hat, damit der Agent bei der Suche nach der richtigen Fluggesellschaft und dem richtigen Flug eine entsprechende Strategie anwenden kann. Hierbei steht also nicht von vorne herein fest, welche Server verwendet werden, und der Agent muß über Verzeichnisdienste die verschiedenen Fluggesellschaften finden und auch mit ihren möglicherweise unterschiedlichen Dienstleistungen zurechtkommen, um die notwendigen Informationen zu ermitteln und zu vergleichen. Bei dieser Ausprägung sind also das Ziel und die Strategie gegeben, der Agent benötigt jedoch einiges an Zusatzwissen und Verarbeitungslogik, um den Vorgang ordnungsgemäß steuern zu können.

- Es gibt noch weitergehende Einsatzkonzepte für Agenten, die in der Literatur ausgeführt werden [CGH+95]. So sind Agenten vorstellbar, die laufend im Netz unterwegs sind, indem sie sich auf autonome Weise von einer Ausführungsumgebung zur Anderen bewegen. Dabei halten sie Ausschau nach Informationen, die den Interessen des Benutzers entsprechen, wozu sie selbständig Strategien entwickeln und sich mit anderen Agenten austauschen.

Eine Eigenschaft haben jedoch all diese Agenten-Ausprägungen gemeinsam: Sie müssen während ihrer Ausführung ohne die interaktive Unterstützung eines menschlichen Benutzers auskommen. Sie müssen folglich die Fähigkeit besitzen, selbständige Entscheidungen zu treffen. Das fängt damit an, daß sie geeignet auf Fehlsituationen reagieren, die zur Laufzeit auftreten. Was ist zum Beispiel zu tun, wenn dem Drucker das Papier ausgeht? Solche und andere Fragen müssen jedenfalls beim Entwurf bedacht werden.

Um eine hohe Fehlertoleranz zu erreichen, muß der Agent in der Lage sein, die korrekte Durchführung einer Dienstleistung zu überprüfen oder nachzuweisen, bevor er in seiner Ausführung fortschreitet. Letztlich überträgt der Benutzer dem Agenten auch eine beträchtliche Portion Verantwortung, vor allem wenn elektronische Zahlungsmittel integriert und mitverwendet werden sollen.

5.3 Anforderungen an die Infrastruktur

Für die sinnvolle Anwendung des Client-Agent-Server-Modells im Mobile Computing ist eine entsprechende Unterstützung durch die Infrastruktur eine besonders wichtige Voraussetzung. Basis dafür ist erstens die Erarbeitung von Standards, mit deren Hilfe die Form der Agenten gestaltet wird und Betriebsmittel verfügbar gemacht werden. Zweitens müssen diese Betriebsmittel auch erreichbar sein, d.h. sie müssen als Netzwerk organisiert sein und dem mobilen Benutzer einen guten Zugang bieten. Schließlich gibt es wichtige Hilfsdienste, die in diesem Zusammenhang ebenfalls angeboten werden sollten.

Um die Aufgabe eines Agenten zu formulieren wird eine Programmiersprache benötigt und die Infrastruktur sollte dafür Standards bereitstellen. Eine einfache Möglichkeit besteht darin, eine bei praktisch jedem Betriebssystem vorhandene Skriptsprache direkt zu verwenden, wie z.B. Unix-Shellskripte. Für den durchschnittlichen Benutzer ist diese Lösung jedoch recht unübersichtlich und fehlerträchtig, weshalb sie eigentlich höchstens für Experten bei Mangel anderer Werkzeuge zu empfehlen ist. Ausserdem schränkt man sich damit in aller Regel auf eine bestimmte Plattform ein.

Die Aufgabe des Agenten sollte auf möglichst einfache und übersichtliche Art und Weise zu spezifizieren sein. Die Arbeit der Spezifikation teilen sich meist der Programmierer und der Anwender, wobei ein geübter Anwender unter Umständen beide Rollen übernehmen kann.

Der Programmierer wird dabei von einer Entwicklungsumgebung unterstützt, die es ihm erlaubt, in einer höheren Programmiersprache Agenten-Programme zu entwickeln. Ein wichtiges Ziel ist die Unabhängigkeit der Agenten von der Ausführungs-Plattform. Dies ist mit Hilfe von Abstraktionen möglich, die gängige Kommunikations- und Koordinations-Mechanismen kapseln, was oft mit objektorientierten Techniken verknüpft wird.

Das Ergebnis dieser programmierintensiven Phase ist die Bereitstellung von Agenten-Mustern, aus denen der Endanwender Varianten erstellen kann, indem er auf einfache Weise das gewünschte Muster auswählt und darin die gewünschten Parameter einsetzt.

Gemäß der verschiedenen Agenten-Ausprägungen entstehen unterschiedliche Anforderungen an die Mächtigkeit der verwendeten Sprache. Diese schlägt sich nieder in der Komplexität und Effizienz der Interpreter in den Ausführungsumgebungen. Sobald Ziele und Strategien formuliert werden sollen und die Agenten auf die Suche nach verteiltem Wissen geschickt werden, benötigt man neben den algorithmischen Sprachen auch Standards und Sprachen zur Wissensrepräsentation. Dazu gehören die entsprechenden Inferenzmaschinen, welche anhand von Methoden der künstlichen Intelligenz das Wissen weiterverarbeiten. Aber es muß auch die Schnittstelle zum Anwender bedacht werden, für den Vorgaben und Ergebnisse in einer für ihn leicht verständlichen Form gehalten werden müssen.

Mit Hilfe der beschriebenen Komponenten und Standards lassen sich also ganze Netzwerke von Agenten-tauglichen Betriebsmitteln mit Ausführungsumgebungen aufbauen, und dies wurde auch bereits durchgeführt. Ein Beispiel dafür ist Telescript, eine Sprache zur Formulierung von Agenten, mit der in den USA ein Netzwerk von Telescript-fähigen Interpretern aufgebaut wurde [Kot95]. Im Bereich der Wissensrepräsentation setzen sich Standards wie KIF oder KQML durch, die eine anwendungsunabhängige Verwendung ermöglichen [CGH+95].

Um für den mobilen Benutzer einen guten Zugang zu bieten, sind auch administrative Informationsdienste wichtig, wie zum Beispiel Verzeichnisdienste, die es erlauben die einzelnen Ressourcen über ihre jeweiligen Leistungsdaten oder Kosten zu selektieren.

Der Einsatz von Agenten, besonders in großen öffentlichen Netzen birgt einige Gefahren in sich, was die Sicherheit und die Vertraulichkeit angeht. Es besteht zum Beispiel die Gefahr, daß ein Agent auf kriminelle Art manipuliert wird, oder daß böswillige Viren-Agenten Schaden anrichten. Deshalb muß gerade hier auf die Hilfe von Verschlüsselungs-

und Authentisierungs-Diensten zurückgegriffen werden, da sonst die Vorteile der Agenten leicht zunichte gemacht werden können.

Insgesamt gesehen ist der Aufwand für die Infrastruktur beim Client-Agent-Server-Modell also recht hoch, aber je mehr Intelligenz und Homogenität in die Infrastruktur eingebracht werden kann, um so leichter gestaltet sich die Programmierung und die Anwendung von leistungsfähigen und nützlichen Agenten.

6 Den richtigen Ansatz auswählen

Es stellt sich nun die Frage, in welcher Situation welche der einzeln beschriebenen Ansätze und Verfahren eingesetzt werden sollen. Diese Frage ist im Allgemeinen nicht einfach zu beantworten, denn eine Fülle von Einflußgrößen müssen genau gegeneinander abgewägt werden, um optimale Lösungen anzustreben. Dazu kommt, daß etliche der Einflußgrößen sich dynamisch während des Betriebs ändern und daher eine adaptive Auslegung der Systeme benötigt wird.

Zunächst sollen die drei Ansätze Data-Shipping, Replikation und Agenten prinzipiell verglichen werden. Das geschieht in der folgenden Tabelle, wo anhand der bereits besprochenen Kriterien im Mobile Computing zusammengefaßt wird, wo deren Stärken und Schwächen auftreten. Dazu folgen noch genauere Erklärungen und Bemerkungen. Anschließend soll weiter auf die Adaptation eingegangen werden, insbesondere ist zu klären, wie weit eine transparente und automatische Anpassung gehen kann oder wann auch die Anwendung mitberücksichtig werden muß.

6.1 Stärken und Schwächen

	Data-Shipping	Replikation	Agenten
Energie-Engpaß	-	+	+
Knappe Speicherkapazität	0	-	+
Knappe Rechenleistung	0	0	+
Risiko Diebstahl/Zerstörung	+	-	+
Broadcast-Fähigkeit	0	+	0
Schlechter QoS	-	+	+
Teure Kommunikation	-	+	+
Kostenmodell Verbindung	-	+	0
Kostenmodell Nachricht	+	0	0
Zugriffs-Voraussagbarkeit	-	+	0
Abgekoppelte Operation	-	+	+
Niedrige rel. Änderungsrate	-	+	0
Hohe rel. Änderungsrate	+	-	0
Komplexe Dienste	-	0	+
Infrastruktur-Voraussetzungen	+	+	-

Beim Energie-Engpaß, also wenn das mobile Endsystem autonom mit Hilfe von Batterien operieren muß, dann ist Data-Shipping nachteilig, weil die Kommunikation nicht

optimiert ist und aufgrund der Asymmetrie in der drahtlosen Kommunikation für jeden Request etwas Energie verbraucht wird. Bei der Replikation dagegen, kann auf Kommunikation verzichtet werden und mit Hilfe von Agenten kann die Anzahl Requests reduziert werden.

Die knappe Speicherkapazität benachteiligt naturgemäß die Replikation, bei umfangreichen Zwischenergebnissen können auch beim Data-Shipping Probleme auftauchen. Die Lösung mit Agenten erlaubt es, genau die Informationsmenge in den Mobilrechner zu holen, die dort wirklich gebraucht wird.

Knappe Rechenleistung kann mit Einsatz von Agenten flexibel durch Rechenleistung des Festnetzes ergänzt werden.

Bei Diebstahl oder Zerstörung des mobilen Endsystems können beim Data-Shipping nur wenige kritische Daten verlorengehen, da sich der gesamte Datenbestand auf dem Server befindet. Anders, bei der Replikation, hier können je nach eingesetzter Datensicherungs-Strategie mehr oder weniger kritische Daten verlorengehen. Bei Agenten ist es ähnlich wie beim Data-Shipping, mit dem weiteren Vorteil, daß ein Teil der Datenhaltung (etwa Zwischenergebnisse bei komplexen Diensten) vom Agenten auf dem Festnetz übernommen wird.

Von der Broadcast-Fähigkeit profitieren lediglich einige Verfahren der Replikation.

Ein schlechter QoS, also niedrige Bandbreite, hohe Latenz und hohe Abbruchwahrscheinlichkeit verstärken die Probleme des Data-Shipping durch das Auftreten langer Wartezeiten. Replikationsverfahren können in diesem Fall ältere Kopien vorhalten, und beim Einsatz von Agenten muß dieser Kommunikationsflaschenhals nur kurzzeitig für das Senden des Agenten und das Empfangen seiner Ergebnisse überwunden werden.

Bei hohen Kommunikationskosten verhält es sich ähnlich, denn es kommt auch hier darauf an, den Kommunikationsbedarf zu reduzieren. Ein wichtiger Punkt ist hier aber das verwendete Kostenmodell. Das verbindungsorientierte Modell ist besonders für das Data-Shipping sehr ungeeignet, da eine Verbindung über längere Zeit aufrechterhalten werden muß, auch wenn nur kurze und weit auseinanderliegende Kommunikationsereignisse stattfinden. Bei Replikation und Agenten dagegen, steht schon bei Verbindungsaufbau fest, welche Daten übertragen werden müssen, und die Verbindungszeit kann damit entsprechend gut ausgenutzt werden. Im nachrichtenorientierten Modell hängen die Kosten nicht mehr von den Wartezeiten ab, für das Data-Shipping ist das ein klarer Vorteil. Für die Übertragung einer bestimmten Datenmenge sind die Kosten aber meist höher als wenn die gleiche Datenmenge bei guter Verbingszeit-Ausnutzung im verbindungsorientierten Modell übertragen wird. Die Nachteile und Vorteile für Replikation und Agenten können also von Fall zu Fall unterschiedlich sein.

Sind die Daten-Zugriffe für einen bestimmten Zeitraum voraussagbar, dann kann vor allem durch Replikation viel erreicht werden, sowohl in Bezug auf Kostenersparnis, wie auch Energieersparnis und Zugriffszeit-Optimierung. Das Data-Shipping kann aus dieser Information keinerlei Nutzen ziehen, Agenten im Prinzip auch nicht, man könnte sich aber vorstellen, durch "intelligente" Agenten etwas in dieser Richtung zu erreichen, dies würde aber auf eine Kombination von Agenten mit Replikation hinauslaufen.

Verlangt die Anwendung die Möglichkeit der abgekoppelten Operation, dann kann ledig-

lich die Replikation weiterhelfen, Data-Shipping ist in diesem Zustand nicht einsetzbar. Agenten können zwar vor dem Abkoppeln abgeschickt werden und in der Zeit des abgekoppelten Zustandes auf dem Festnetz operieren, sie lassen aber keine abgekoppelte Operation zu, in dem Sinne, daß in diesem Zustand keine Agenten abgeschickt werden können und es keinen Zugriff auf Ergebnisse gibt, dazu wären Ergänzungen um Replikations-Techniken notwendig.

Bei niedriger relativer Änderungsrate, also wenn der mobile Client häufig ein Datum liest, das im Server nur selten geändert wird, dann macht das Data-Shipping eine Vielzahl redundanter Datenzugriffe notwendig, durch Replikation kann der Kommunikationsbedarf in diesem Fall erheblich reduziert werden. Das Data-Shipping macht dagegen Sinn, wenn der umgekehrte Fall auftritt, nämlich wenn sich im Server ein Datenelement häufig ändert, das im mobilen Client nur selten benötigt wird, weil dann immer das Aktuellste übertragen werden kann, ohne unnötige Kommunikation zu erfordern. Bei Replikation kann es dagegen vorkommen, daß manche Daten häufig unnötig übertragen werden, entweder weil sie zu einer größeren Dateneinheit gehören oder weil das Verfahren versucht auf andere Gesichtspunkte, wie z.B. Zugriffszeit, hin zu optimieren. Für die Verwendung von Agenten hat dieses Kriterium keine Relevanz.

Wenn es um die Verwendung komplexer Dienste geht, dann geht es vor allem um die Steuerung des Ablaufs von Server-Zugriffen, d.h. bei diesem Kriterium spielt die Replikation eine untergeordnete Rolle. Klar benachteiligt ist hier das Data-Shipping, weil der mobile Client die Steuerung über den störanfälligen Mobil-Kanal durchführen muß, was entsprechend Energieverbrauch und Kosten verursacht. Außerdem fallen auf dem mobilen Rechner das Empfangen und das Verarbeiten von Zwischenergebnissen an, was ebenfalls Ressourcen erfordert und bei geringer Bandbreite und umfangreichen Zwischenergebnissen problematisch sein kann. Der Vorteil des Agenten-Ansatz ist es, daß diese gesamte Steuerung vom Agenten auf dem Festnetz übernommen wird, womit die oben aufgeführten Probleme meistens verschwinden. Durch den gezielten Einsatz komplexer Server für häufig gebrauchte Dienste kann die Situation für das Data-Shipping entschärft werden, der Agenten-Ansatz ist hier aber um einiges flexibler.

Die Voraussetzungen an die Netz-Infrastruktur sind beim Data-Shipping am geringsten. Prinzipiell muß lediglich eine Netzwerkstruktur vorausgesetzt werden, die den RPC unterstützt. Die breite Verfügbarkeit entsprechender Betriebssysteme und Entwicklungs-Werkzeuge spricht für die Verwendung dieses Client-Server-Modells. Für die Replikation wird der zusätzliche Aufwand oft in die Anwendung gesteckt ohne zusätzliche Netz-Dienste vorauszusetzen, weiterhin kann durch weitverbreitete Protokolle, wie z.B. SMTP und POP das abgekoppelte Arbeiten mit Emails sehr gut unterstützt werden. Um Agenten einsetzen zu können, müssen auf dem Festnetz allerdings entsprechende Annahme- und Ausführungs-Dienste vorhanden sein. Dazu gibt es aber heute noch keine etablierten Standards, so daß Agenten nur in einem relativ eingeschränkten Rahmen eingesetzt werden können. Außerdem muß Sicherheitsfragen erhöhte Aufmerksamkeit geschenkt werden, weil sich Agenten vor Manipulation und Server vor unerwünschten Eindringlingen schützen müssen. Für einen umfangreichen Einsatz von Agenten ist also die Organisation einer unterstützenden Infrastruktur sehr aufwendig.

6.2 Adaptive Systeme

Eine mobile Anwendung soll sich an eine Vielzahl von Situationen anpassen können. Wenn die mobile Komponente im heimischen Büro fest angedockt ist, dann gibt es keinen Energie-Engpaß mehr, es besteht die Möglichkeit auf leistungsfähige Peripherie zuzugreifen und die Kommunikation ist schnell und kostenlos. In dieser Situation können ausgefeilte Verfahren der Replikation und mit Agenten leistungsmindernd wirken und können ausgeschaltet werden, außerdem kann sich die Benutzeroberfläche große Bildschirme, gute Tastaturen und andere Eingabegeräte zunutze machen.

Im mobilen Einsatz muß eine intelligente Auswahl zwischen Data-Shipping, Replikation und Agenten getroffen werden. Die Qualitätsanpassung von Informationen stellt eine weitere Möglichkeit dar, sich an die Einschränkungen des mobilen Kontextes anzupassen. Diese ist vor allem für multimediale Daten wichtig, wobei je nach Informationstyp (Text, Bilder, Sprache, Video) unterschiedliche Kompressionsverfahren verwendet werden.

Aus der Sicht der Systemunterstützung gibt es eine breite Palette an Anpassungsstrategien. Diese reicht von einem vollständigen *laisser-faire*, wo die Anwendung die gesamte Verantwortung für die Anpassung trägt, bis hin zu kompletter Anwendungstransparenz der Anpassung. Es bleibt also die Wahl, wieviel Intelligenz und damit Software-Aufwand jeweils in die Anwendung oder in das unterstützende System eingebaut werden soll. Meist wird hier der Mittelweg gegangen, also ein System realisiert, daß die Verantwortung für die Anpassung im Dialog mit einer adaptiven Anwendung teilt.

Die vollständig transparente Anpassung ist zwar wünschenswert aber nicht immer machbar. Das soll an folgendem Beispiel gezeigt werden. Bei der übertragung von Video-Sequenzen zu einem Mobilrechner soll eine Anpassung an eine veränderliche Bandbreite vorgesehen werden. Handelt es sich bei der Anwendung um eine Video-Anzeige-Tool, dann ist es sinnvoll bei einer sinkenden Bandbreite die Kompressionsrate zu erhöhen oder gar auf Schwarz-Weiß Darstellung umzuschalten, um die Echtzeit-Darstellung zu bewahren. Ist die Anwendung jedoch ein Video-Editier-System, dann ist diese Bildqualitätsdegradierung keineswegs erwünscht, sondern hier müssen die Bilder halt mit langsamerer Geschwindigkeit aber ohne Qualitätsverlust übertragen werden.

Ein solches System, das adaptive Anwendungen unterstützt, verwaltet und beaobachtet die veränderlichen Parameter, die für die Anpassung relevant sind und handelt mit dem Client Bereiche aus, in denen eine bestimmte Strategie gefahren werden kann. Beim herausfallen aus diesem Bereich, wird der Client benachrichtigt und es kann ein neuer Bereich ausgehandelt werden. Sinnvoll ist dabei die Unterstützung durch einen Agenten auf der Festnetzseite. Dieser hat die Möglichkeit, dort die verfügbaren Ressourcen auszukundschaften und so eine flexible Optimierung zu realisieren.

7 Ausblick

Im Rahmen dieser Arbeit wurden mehrere Möglichkeiten des Datenmanagements im Mobile Computing aufgezeigt und miteinander verglichen. Optimaler Entwurf und optimale Konfiguration hängen auf komplexe Weise von einer Vielzahl von Parametern und

Randbedingungen der mobile Anwendung ab. Während die Hardware rasant weiterentwickelt wird und zunehmende Verbreitung findet, ist und bleibt es eine Herausforderung die dazu passenden Softwaresysteme zu schaffen und einzusetzen. Die Entwicklung geht dahin, daß die Endsysteme zur Entlastung des Anwenders möglichst einfach zu bedienen sein sollen. Dabei soll jedoch die Flexibilität nicht verlorengehen, da eine genaue Konfigurierbarkeit im Mobile Computing sehr wichtig ist. Um dies zu erreichen, werden in Zukunft intelligente und zuverlässige adaptive Systeme gefragt sein.

Trading

Tobias Gerndt

Kurzfassung

In er Architektur der verteilten Systeme spielt das Trading eine wichtige Rolle. Trading hat in den letzten Jahren immer mehr an Bedeutung gewonnen und hat die herkömmlichen Client-Server-Modelle dahingehend erweitert, daß eine zusätzliche Instanz, der Trader, die Verwaltung der verfügbaren Dienste übernimmt und bei Dienstanfragen der Clients als Vermittler auftritt.

Ziel des Tradings ist es, dem Benutzer ein Tools zur Verfügung zu stellen, das einen transparenten Zugriff auf einen Dienst ermöglicht, der von einem Server angeboten wird, der unter Optimalitätsbedingungen ausgewählt wurde. Gleichzeitig sollte diese Vermittlung fehlertolerant sein und zum Lastausgleich beitragen.

In dieser Arbeit werden die Merkmale des Tradings aufgezählt, die zur Erbringung oben genannter Aspekt wichtig sind. Anhand dreier Systembeispiele werden aktuelle Entwicklungen und Begriffe erläutert. Abschließend gibt es einen Vergleich existierender Systeme.

1 Einleitung

1.1 Nameserver

Den Begriff des Nameservers findet man sehr häufig im Zusammenhang mit dem *Client-Server-Modell*. In diesem klassischen Modell ruft ein Client einen Dienst auf, der vom zugrundeliegenden Netzwerk als Nachrichtenpaket zum Server weitergeleitet, dort ausgewertet und schließlich mit dem Rücktransport der Ergebnisse zum Client beendet wird. Die Lokalisierung der Server geschieht über eindeutige physikalische Adressen. Um dem Client den Umgang mit den benutzerunfreundlichen Server-Adressen zu erleichtern, die zur Lokalisierung und zum Aufruf der Server benötigt werden, werden diese unter *logischen Namen* registriert. Die Instanz, die die Abbildung vom logischen Namen auf eine physikalische Adresse durchführt, ist der *Nameserver*. Durch ihn ist es für den Client viel einfacher, einen Server zu identifizieren.

In manchen Systemen (Grapevine, Clearinghouse) helfen Gruppennamen, sämtliche Server unter einem logischen Namen zu ermitteln, falls es mehrere gleichwertige Server in einem verteilten System gibt. Dadurch hat der Client die Möglichkeit zur geeigneten Auswahl eines Servers. Beim zellenbasierten DCE-*Cell Directory Service* (CDS) hat man u.a. die Möglichkeit, Server mit gleicher Dienstschnittstelle zu gruppieren. Bei nicht-automatischem Binden kann der Client vom CDS durch sukzessives Anfragen nach Server-Verweisen, sog. *Binding Handles*, sämtlicher Server ermitteln, die den gewünschten Dienst bereitstellen.

1.2 Trader

Allen Namensverwaltungssystemen ist gemein, daß sie letztlich nur eine Abbildungsfunktion bereitstellen, die eine Objektspezifikation auf eine Menge von potentiell verfübaren

Objekt-IDs abbildet. Der Client wählt aber letztendlich anhand der Informationen des Nameservers einen Server aus. Beim *Trading* wird versucht, eben diesen Auswahlprozeß transparent zu gestalten.

Hierzu übergibt der Client dem Trader lediglich eine Beschreibung des benötigten Dienstes. Als Vermittlungseinheit hat der Trader nun seinerseits die Aufgabe, die verfügbaren Server-Objekte zu identifizieren, die die geforderte Dienstleistung erbringen, und die Auswahl eines Objektes zugunsten des Clients zu optimieren. D.h. der Trader ist in der Lage, nicht nur anhand statischer, sondern auch anhand dynamischer Attribute zu optimieren.

Man muß sich jedoch bewußt sein, daß der Übergang von einem Nameserver zu einem Trader fließend ist, und es nicht immer möglich ist, die beiden genau voneinander abzugrenzen.

1.3 Vergleich: Nameserver - Trader

In Tabelle 3 werden die wichtigsten Unterschiede zwischen einem Nameserver und einem Trader aufgelistet. [Kel93]

Kriterium	Nameserver	Trader
Attributierte Anfragen	möglich	ja
Server-Auswahl	nein	ja
Auswahloptimierung	nein	ja
Server-Transparenz	nein	möglich

Tabelle3. Vergleich zwischen Nameserver und Trader

2 Trading-Dienst in verteilten Systemen

2.1 Basismodell des Tradings

Das Basismodell eines Traders ist in Abbildung 7 dargestellt. Server, die ihre Dienstleistungen im offenen System anbieten wollen, lassen ihre Dienstangebote unter Angabe von Diensttyp und Diensteigenschaften durch spezielle *export*-Anweisungen beim Trader registrieren (1). Wenn ein Client auf einen Dienst zugreifen möchte, dann richtet er sich mit seiner Dienstanfrage an den Trader (2), welcher die Anfrage prüft, ob ein von einem Server exportierter Dienst den Anforderungen des Clients genügt (3). Wenn diese Prüfung positiv ausfällt, initiiert er das Binden zwischen Client und (dem ausgewählten) Server (4). Falls mehrere Server den Anforderungen des Clients genügen, wählt der Trader selbständig einen Server aus. Falls kein passender Server gefunden werden kann, muß der Trader die Dienstanfrage zurückweisen oder einen anderen Trader kontaktieren.

Ein Trader kann folgende Grundfunktionalitäten aufweisen:

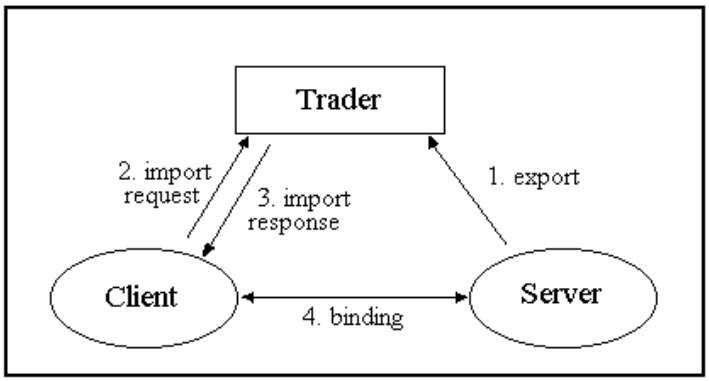

Abbildung7. Basismodell des Tradings

- Servern wird ermöglicht, ihre Dienstangebote dem Trader zu exportieren.
- Clients können Dienste oder nur Informationen über Dienste vom Trader importieren.
- Die Dienstangebote von Servern können über mehrere Trader aufgeteilt werden, um z.B. die Belastung eines einzelnen Traders zu reduzieren (Föderation).
- Der Trader unterstützt die Kooperation zu anderen Tradern, um z.B. eine Dienstanfrage, die nicht erfüllt werden kann, nicht gleich zurückzuweisen (Kooperation).

Hier sei nochmals auf die wichtigste Funktionalität eines Traders hingewiesen. Im Gegensatz zu Nameservern oder Directory-Diensten ist ein Trader wesentlich flexibler, was den Vermittlungsprozeß angeht. Beim Trader wird eine Dienstanfrage nicht nur wie eine Art Datenbankabfrage verwaltet, sondern es werden auch Dienstqualitäten eines Servers berücksichtigt, die sich z.T. dynamisch ändern (z.B. Drucker-Warteschlangen). Außerdem ist er in der Lage, bei mehreren verfügbaren Dienstanbietern nach verschiedenen Kriterien zu optimieren.

2.2 Dienstspezifikation

Der *Diensttyp* eines Dienstes legt fest, welche Dienstleistung ein Server offeriert bzw. welchen Dienst der Client als Dienstbenutzer erwarten kann. Mit ihm wird die Schnittstelle beschrieben, die für den Zugriff auf den Dienst bekannt sein muß. Dazu gehören die Operationen, die der Server anbietet, und die Datentypen der Ein- und Ausgabeparameter. Damit man die Dienstleistungen, die in einem System verfügbar sind, unterscheiden kann, muß für jede neue Dienstschnittstelle ein neuer Diensttyp definiert werden, der dann im System eindeutig ist. Die Dienste eines Diensttyps weisen deshalb immer die gleiche Grundfunktionalität auf. Diese kann sich jedoch bzgl. ihrer Eigenschaften und Qualitäten deutlich unterscheiden.

Um auch in größeren, heterogenen Umgebungen die Spezifikation von Dienstschnittstellen zu ermöglichen, werden zur Unterstützung der Dienstangebote einheitliche Schnittstellenbeschreibungen benötigt. Eine der bekanntesten ist die *Interface Definition Language* (IDL) des OSF/DCE.

Entscheidend für die Mächtigkeit und Flexibilität des Trading-Mechanismus ist es, die exportierten und importierten Dienste möglichst genau zu beschreiben. Es ist nicht ausreichend, sie nur durch den Dateityp zu beschreiben. Deshalb werden die Dienstleistungen durch eine *Attributliste* ergänzt. Man unterscheidet hier zwischen statischen und dynamischen Attributen. Bei der Einteilung, um welche Attribute es sich handelt, betrachtet man die Änderungsrate eines Dienstattributs bzgl. der Zugriffsfrequenz.

Als Beispiel kann man einen Printserver betrachten. Die Arbeitslast (Queue) und die Bearbeitungsdauer eines Auftrags gehören zu den dynamischen Attributen, da sie sich nach jedem Zugriff verändern. Wohingegen die Postscript-Fähigkeit und der Druckertyp (Matrix, Laser) zu den statischen Attributen gehören, weil sie dauerhafte und garantierte Qualitätsmerkmale repräsentieren.

Verwaltet werden die exportierten Daten in einem speziellen Verzeichnis, dem *Dienst-Directory*. Es stellt eine Sammlung einzelner Dienstangebote dar, die durch Information über administrative, organisatorische und strukturelle Beziehungen zwischen den Angeboten ergänzt wird. Wegen ihrer hohen Änderungsrate bedürfen die dynamischen Attribute einer besonderen Aufmerksamkeit. Um zeitliche Inkonsistenzen zu vermeiden, werden in Trading-Systemen häufig Mechanismen zur Leistungsoptimierung eingesetzt, die in Abschnitt 2.4 genauer betrachtet werden.

2.3 Dienstauswahl

Falls einem Trader nach einer Dienstanfrage eines Clients mehrere Server zur Auswahl stehen, die die gestellten Anforderungen erfllen, dann muß er in der Lage sein, anhand von Auswahlkriterien und -strategien zu optimieren.

Beim *Auswahlkriterium* unterscheidet man zwischen individueller und sozialer Optimierung:

- Bei der *individuellen Optimierung* hängt die Auswahl nur von der initiierenden Dienstanfrage ab. Alle anderen, gleichzeitig eintreffenden Anfragen spielen bei der Entscheidung keine Rolle.

- Bei der *sozialen Optimierung* wird versucht, die Entscheidung unter einem globalen Gesichtspunkt zu treffen. Unter Kenntnis aller gleichzeitig auftretenden Dienstanfragen eines oder mehrerer Trader wird eine optimale Lösung für alle Anfragen gesucht.

Auch bei den *Auswahlstrategien* gibt es mehrere Alternativen, die auch in Mischform auftreten können:

- Bei der *First Choice-Strategie* entscheidet sich der Trader für den Server, der als erster die gestellten Forderungen erfüllt.

- Bei der *Random Choice-Strategie* werden den exportierten Diensten und damit den Servern Wahrscheinlichkeiten zugewiesen, die vom Trader verwaltet werden. Genügen mehrere Server den Anforderungen des Clients, so wird einer zufällig gemäß dieser Wahrscheinlichkeit ausgewählt.

– Bei der *Cyclic Choice-Strategie* werden die Server mit gleichwertigen Diensten vom Trader in eine Liste eingeordnet, die dann zyklisch abgearbeitet wird.

– Wenn eine Dienstanforderung vom Trader bereits einmal vermittelt wurde, so kann bei der *Conservative Choice-Strategie* der Vermittlungsprozeß und die damit anfallende Verzögerung umgangen werden, indem der Trader die vorher getroffene Entscheidung übernimmt.

– Wenn bei mehreren geeigneten Dienstangeboten nur eine Untermenge der Server bei der endgültigen Auswahl berücksichtigt wird, dann handelt es sich um die *Probing-Strategie*.

– Stellt der Client sehr hohe Ansprüche an den geforderten Dienst, die insbesondere den aktuellen Server-Zustand betreffen, dann muß nach der *Best Choice-Strategie* verfahren werden.

2.4 Performanz

Beim Trading kommt es nicht nur auf Transparenz und Flexibilität beim Auswahlprozeß, sondern auch auf Geschwindigkeit an. Unter Geschwindigkeit verstehen man hier die Dauer der Verzögerung, die mit einer Dienstanfrage verbunden ist. Insbesondere stellt sich dieses Problem bei der Erfassung der dynamischen Attribute eines Servers.

– Der ungünstigste Fall tritt auf, wenn ein Trader sofort nach Auftreten des Bedarfs seine Information direkt vom Server erfragt, da die Informationen z.T. für den weiteren Auswahlprozeß irrelevant sind. Eine Möglichkeit zur Verbesserung liegt in der *gestaffelten Auswertung*, deren Ziel es ist, die Anzahl der Zugriffe dadurch zu minimieren, daß man den Zeitpunkt des Zugriffs so weit wie möglich hinauszögert.

– Um die Server mit Zugriffen auf ihre dynamischen Attribute nicht unnötig zu belasten, kann auch ein spezieller, u.U. verteilt realisierter *dedizierter Zustand-Server* eingesetzt werden.

– Ein weiteres Problem liegt in der Zeitspanne zwischen der Akquirierung dynamischer Daten und dem eigentlichen Binden zwischen Client und Server. Um zu verhindern, daß zwei Clients gleichzeitig einen Server wegen dessen schwacher Auslastung zugewiesen bekommen, der jedoch anschließend durch diese Auftragskollision überlastet wäre, setzt man *Reservierungstechniken* ein, die einem Trader für eine angemessene Zeitspanne die erfragten dynamischen Server-Attribute garantiert. Falls ein Server eine weitere Zustandsanfrage erhält, so muß er in seiner Antwort die Reservierung mit berücksichtigen.

– Mit dem *Konzept des lokalen Caching* ist es möglich, einmal erfragte Informationen unter Festhaltung des Zugriffszeitpunkts lokal in einem Cache-Speicher zu halten. Erfolgt nach dem ersten Zugriff relativ bald ein zweiter, so kann der Wert aus dem Cache bei ausreichender Aktualität wiederverwendet werden, ohne daß erneut auf

den Server zurückgegriffen werden muß. Der Aktualitätstest wird durch spezielle Aktualitätsattribute unterstützt, die sowohl die Änderungsrate des Attributs als auch seine anwendungstechnische Relevanz berücksichtigen.

2.5 Kooperation und Föderation

In großen, offenen verteilten Systemen verwaltet jeder Trader eine Domäne. Angesichts der Angebotsvielfalt ist es wünschenswert, daß es eine Interaktion zwischen Tradern gibt. Man unterscheidet zwei Formen der Zusammenarbeit von Tradern. [KS95]

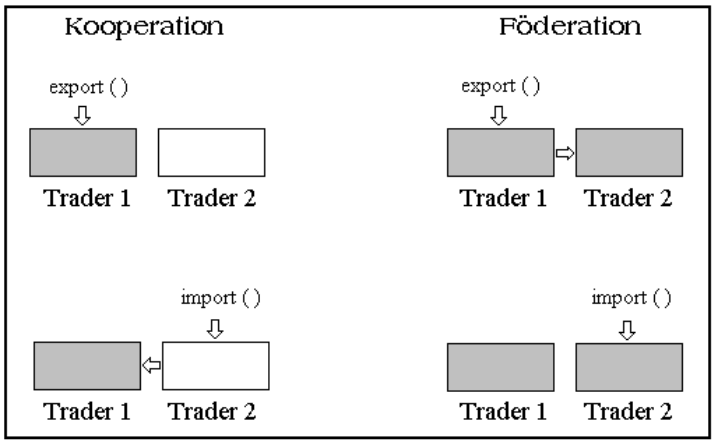

Abbildung8. Kooperation und Föderation

Beim *Konzept der Kooperation* kann ein Trader, der eine Dienstanfrage eines Clients nicht erfüllen kann, diese an einen anderen Trader weiterleiten. Wenn dieser Trader die Anfrage befriedigen kann, übermittelt er den gefundenen Server mittelbar über den ersten Trader oder direkt an den Client. Ebenfalls ist es moglich, daß der zweite Trader rekursiv weitere Trader mit der Vermittlung beauftragt. Wenn eine Dienstanfrage allerdings über mehrere Trader läuft, und ein Trader am Ende dieser Schlange einen Server vermittelt, auf den der Client keinen Zugriff hat, da dieser sich in einer benachbarten Domäne befindet, kann es zu Problemen kommen. Man versucht, diese Zugriffsprobleme mit speziellen Zugriffskontrollmechanismen in den Griff zu bekommen.

Eine weitere Möglichkeit der Zusammenarbeit zwischen Tradern bietet das *Konzept der Föderation*, welches v.a. bzgl. der Laufzeit günstiger ist. Bei der Föderation schließen sich Trader zusammen, um so ihr Dienstangebot zu erweitern. Hierzu exportieren die Trader untereinander bestimmte Teile ihres Dienstangebots. Die Föderation sollte vier wesentlichen Anforderungen gengen:

- Autonomieprinzip: Möglichkeit, der Föderation jederzeit beizutreten oder sie zu verlassen.
- Separationsprinzip: externe Zugriff auf einen Dienst nur über die Föderation möglich.
- Heterogenitätstransparenz: die Heterogenität der Trader bleibt dem Client verborgen.

– Lokationstransparenz: die Lokation der Föderation bleibt dem Client verborgen.

Im Rahmen der ODP-Standardisierung wird über ein konkretes Modell der dezentralen Föderation diskutiert. Hier handeln die Trader der Föderation einen sogenannten *Föderationskontrakt* aus, der aus dem Import- und dem Export-Kontrakt besteht. Der Import-Kontrakt kennzeichnet im wesentlichen die über den entfernten Trader verfügbaren Dienste, während der Export-Kontrakt eines Traders den Umfang der von entfernten Tradern erlaubten Zugriffe auf die eigene lokale Dienstdatenbasis beschreibt.

2.6 Management

Die *Managementdienste* beeinflussen die Entwicklung des Trading in immer stärkerem Maße. Sie haben die Aufgabe, Informationen über das verteilte System und über die darin plazierten Komponenten abzulegen oder im Bedarfsfall bereitzustellen. Die Struktur der hierzu benötigten Daten und ihr internes Zugriffsprotokoll bleiben der Anwendung jedoch verborgen. Das Managementsystem übernimmt damit die Aufgabe, die Kooperation zwischen den verschiedenen autonomen Komponenten eines offenen verteilten Systems zu unterstützen, also zwischen den Tradern, dem Directory-Service, den Servern und den Clients.

Für die Managementunterstützung des Tradings gibt es folgende Aufgaben:

– Informationsakquisition: Die Trader benötigen Informationen über die Server. Während im statischen Fall eine lokale Datenbasis ausreichend ist, muß das Managementsystem Dienste bereitstellen, die die Informationen bzgl. der dynamischen Attribute zusammenstellen und an den Trader weiterleiten. Außerdem müssen sie sicherstellen, daß diese die erforderliche Aktualität besitzen.

– Typenmanagement: Die Menge der verfügbaren Diensttypen, die die bereitgestellten Dienstleistungen eindeutig kennzeichnen, kann sich zur Laufzeit ändern. Das Typmanagement hat für das Hinzufügen und Entfernen von neuen Diensttypen zu sorgen. V.a. muß die globale Konsistenz einer Diensttypdefinition gewahrt bleiben.

– Fehlermanagement: In verteilten Systemen ist es besonders wichtig, daß die vermittelten Dienste zuverlässig sind. Das Fehlermanagement protokolliert das Ausfallverhalten von Systemkomponenten und integriert ihre statistisch erwiesene Verfügbarkeit mit in die Auswahlentscheidung.

– Föderationsmanagement: Hier werden die Systemdienste bereitgestellt, die bei der Föderation den beteiligten Tradern erlauben, eine Föderation aufzubauen, diese jederzeit zu modifizieren und sie auch wieder zu verlassen. Wesentlich ist hierbei die Überprfung der Autorisierung und Authentisierung der beteiligten Trader, so daß ein Trader nur Server vermitteln darf, über die er explizit verfügen darf.

– Accounting-Management: Da im offenen Dienstemarkt die Anbieter verschiedener Dienste miteinander konkurrieren, müssen die Kosten, die durch die Erbringung der

Dienstleistung entstehen, verwaltet und berechnet werden. Dazu werden vom Managementsystem Operationen gefordert, die die Serverkosten erfragen und diese bei Beanspruchung verrechnen können.

– Shutdown-Management: Des weiteren werden Systemdienste benötigt, die das Aufbauen und das geregelte Abbauen von Trader-Beziehungen und das Beenden der Tätigkeit eines Traders ermöglichen.

Einige Ansätze sehen den Trader als eigenständige Anwendung an, die auf spezielle Management-Dienste zugreift. Während in anderen Ansätzen der Trader dem Managementsystem zugeordnet wird. Er unterstützt insbesondere das *Konfigurationsmanagement* mit dem Ziel, die Komponenten des verteilten Systems zu koordinieren.

2.7 Fehlertoleranz

Ein neuer Trend ist, den Trading-Mechanismus zur Erbringung von Fehlertoleranz einzusetzen. Hauptaugenmerk ist hierbei nicht, den Trading-Dienst fehlertolerant zu gestalten, sondern den durch das Trading vermittelten Dienst fehlertolerant anbieten zu können. Voraussetzung dafür ist jedoch die klare Entkopplung von Client und Server, die durch das Client-Service-Modell ermöglicht wird. Der Client darf beim Zugriff auf einen Dienst keinen Server mehr direkt kontaktieren. Mehr dazu wird im Abschnitt 3.1.1.

3 Vergleich verschiedener Systeme

3.1 CYGNUS

Das CYGNUS-System wurde mit dem Ziel entwickelt, den Client vom Server klar zu entkoppeln. Man ist von der klassischen Client-Server-Architektur abgekommen und hat ein *Client-Service-Modell* erstellt. In diesem Abschnitt wird dieses neue Modell vorgestellt, und der Dienst-Akquirierungs-Mechanismus beispielhaft erläutert. [CR94]

Client-Service-Modell Sinn des Client-Service-Modells ist es, die Fähigkeiten eines Servers von den konkreten Server-Schnittstellenspezifikationen zu trennen. Dieser Mechanismus wählt Server aus, paßt unterschiedliche Server-Schnittstellen an und behandelt Zugriffsfehler. Man erhält die Identität des Servers für Rechnungen, verfeinerte Dienstspezifikationen und für Fehlerberichte bzgl. der Server. Dadurch wird die Entwicklung, der Gebrauch und die Instandhaltung von Client-Server-Software in großen heterogenen verteilten Systemen mit vielen eigenständigen Servern erleichtert. Die folgende Aufstellung gibt Aufschluß über die Unterschiede zwischen dem Client-Server-Modell und dem Client-Service-Modell:

Client-Server-Modell:

– Client fordert Client-Server-Binding an

– Client greift mittels Server-abhängigem Protokoll auf den Dienst zu

– Client gibt das Client-Server-Binding frei

– Server-Identität ist festgesetzt, wenn das Client-Server-Binding aufgebaut ist

– Client setzt sich mit Abbrüchen des Client-Server-Bindings auseinander

Client-Service-Modell:

– Client fordert Client-Service-Binding an

– Client greift mittels Server-unabhängigem Protokoll auf den Dienst zu

– Client gibt das Client-Service-Binding frei

– Server-Identität ist festgesetzt, wenn das Service-Server-Binding aufgebaut ist

– Client setzt sich mit Abbrüchen des Client-Service-Bindings auseinander

– Client muß sich nicht mit Abbrüchen des Service-Server-Bindings auseinandersetzten

– Dienst-Zugriffs-Aufruf gibt Server-Identität an den Client zurück

Das Client-Service-Modell unterscheidet drei verschiedene Schnittstellen. Ein Server bietet in einer m:1-Abbildung m verschiedene Dienstschnittstellen an, die in einer verteilten Datenbank verwaltet werden. Ein Client hingegen erwartet in einer 1:n-Abbildung n verschiedene Dienstleistungen, die in CYGNUS gegen die übliche Konvention als Diensttypen bezeichnet werden. Sie werden durch Attributlisten spezifiziert, die die statischen Eigenschaften der benötigten Dienstleistungen festlegen. Die Trading-spezifischen Aufgaben, d.h. die Zuordnung der Diensttypen zu den angebotenen Dienstschnittstellen, geschieht in einem *Dienst-Agenten*, welcher lokal beim Client ansässig ist.

Software-Architektur Die Server exportieren ihre Schnittstellenspezifikationen an verschiedene *Server-Interface Directory-Server*, auf die dann später der *Service Specification Interpreter* (dt. Anfrage-Agent) zugreift. Das *Service Acquisition System* (dt. Dienst-Agent) versteckt die Realisierungsdetails vor dem Client, ist aber im Client-Rechner plaziert. Und dies aus zwei Gründen:

– Die Links zu den Clients bleiben intakt, auch wenn das Netzwerk zusammenbricht.

– Server sind autonome Instanzen, auf denen keine zusätzliche Software laufen soll.

Die *Runtime Library* isoliert den *Client-Code* vor den Details des Service Acquisition System, welches aus 3 Komponenten (SAM, SSI, SAA) besteht und unterstützt somit die Portabilität des Client-Codes. Der *Service Acquisition Manager* (dt. Agent-Manager) beantwortet Anfragen von Clients und erzeugt SAM und SSI für eine *Service Acquisition Session*. Die *Service Specification Interpreter* (dt. Anfrage-Agent) analysieren die Dienstspezifikationen, bestimmen das Service-Server Binding und aktivieren lokal die SAAs. Außerdem helfen sie den *Service Access Monitors* (dt. Logger) bei Zugriffsfehlern

auf Client-Seite. *Service Access Agents* (dt. Dienst-Agenten) passen heterogene Server-Schnittstellen an und implementieren Server-abhängige Fehlertoleranz-Algorithmen, damit existierende Server nicht für CYGNUS-Clients modifiziert werden müssen. Die gesamte CYGNUS Software-Architektur für verteilte heterogene Systeme kann man Abbildung 9 entnehmen:

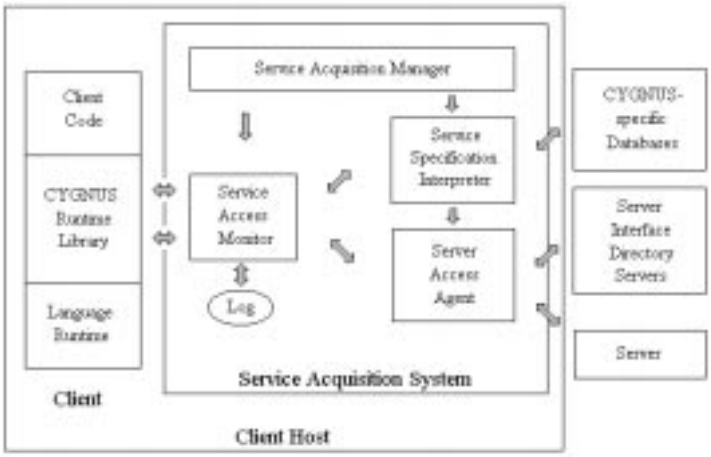

Abbildung9. CYGNUS Software-Architektur

Ein Vorteil der Dienst-Spezifikation im CYGNUS-System ist, daß Benutzer von der Last befreit sind, zu wissen, wo der Empfänger sitzt und welche Art von Kommunikationsmedium er besitzt. Veranschaulicht am Beispiel:

((CONTEXT, messager), (SENDER, Bob), (RECEIVER, Allen), (ACCESS_INTERFACE, send)). Bob schickt an Allen eine Textnachricht über das Kommunikationsmedium, das Allen im Moment benutzt(egal, ob Telefon, Fax oder PC)!

Dienst-Akquirierungs-Phase In diesem Abschnitt wird ein Beispielablauf der Dienst-Akquirierungs-Phase aufgeführen, unterteilt in die vier Abschnitte Dienst-Anforderung, Dienst-Zugriff, Dienst-Rekonfiguration und Dienst-Beendigung:

1. Dienst-Anforderung:

   Die Dienst-Anforderung beginnt, indem der Client den Service Acquisition Manager kontaktiert und einen Request-Port fordert. Der Agent-Manager erzeugt einen SAM und einen SSI und gibt einen IPC-Port zurück. Danach wartet er wieder auf die nächste Dienstanfrage. Der SAM erstellt eine Verbindung zum SSI und zum Client. Er akzeptiert die Dienstspezifikations-Nachricht des Clients und schickt sie wortgetreu an den SSI.

   Nachdem ein SAA wegen des angeforderten Dienstes durch den SSI aktiviert worden ist, baut der SAM eine Verbindung zum SAA auf. Schließlich wird noch ein Dienstzugangsport erzeugt und dem Client mitgeteilt.

2. Dienst-Zugriff:

Nach der Client-Service-Binding Anforderung gibt der Client die Dienstoperation am Dienstzugangsport ab und wartet auf Ergebnisse. Wenn der SAM ein *invocation request* erhält, gibt er es weiter an den SAA, der die Anforderung in einen oder mehrere Aufrufe an verbundene Server übersetzt. Die Bestätigung der Ausführung vom SAA gibt der SAM an den Client weiter und speichert u.U. die Anforderungen und Ergebnisse in einer Art Logbuch.

3. Dienst-Rekonfiguration:

   CYGNUS benutzt zwei Recovery-Algorithmen, um die Clients sicherer gegenüber Netzwerk- oder Serverausfällen zu machen:

   Algorithmus 1 ist vom Server abhängig, der gerade "benutzt" wird. Bei transaktionsbasierten Servern kann der SAA seine Arbeit unterbrechen und dann fortfahren, wenn der Server wieder hochgefahren ist. Hier kann der Server-Zustand korrekt wiederhergestellt werden.

   Algorithmus 2 ist ein Server-unabhängiger "logging & replay"-Algorithmus. Beim Server- oder SAA- Absturz kann der SAM alle gespeicherten Dienstanforderungen mittels eines neuen SAA an einen neuen Server schicken. Falls sich die neue Server-Schnittstelle von der alten unterscheidet, kann der SAA die Requests konvertieren. Dies ist nur möglich, da die SAAs so konzipiert sind, daß sie Dienstzugriffs-Requests in einem Server-unabhängigem Format erhalten.

   Es gibt 3 mögliche Verbindungsabbrüche im CYGNUS Architektur-Modell:

   - Die Verbindung zwischen dem Server und dem SAA kann abreißen, z.B. falls der Server abstürzt.

   - Die Verbindung zwischen dem SAM und dem SAA wird abgebaut, falls z.B. nach einer bestimmten Zeitspanne kein Ergebnis eintrifft.

   - Die Verbindung zwischen SAM und dem SSI kann zusammenbrechen, falls z.B. zu viele Anforderungen für SAAs eingehen.

   Um diese Verbindungen robuster gegen Abbrüche zu machen, gibt es immer Sicherheitsabfragen vom SAM an den SSI, falls es Probleme mit dem SAA gibt. Als Rettungsmaßnahmen gibt es dann komplette Wiederholungen aus dem Logbuch des SAM.

4. Dienst-Beendigung:

   Eine Sitzung wird beendet, indem der Client die "Service-Termination"-Operation in der Runtime Library aufruft. Wenn der SAM eine solche Nachricht erhält, gibt er sie an den SAA weiter, baut die Verbindung mit dem SSI ab (falls sie noch aktiv war) und beendet sich selbst, nachdem er einige Aufräumarbeiten erledigt hat (z.B. Logbuch). Der SAA terminiert, nachdem er den Remote-Server informiert hat.

   Am Ende, nachdem sich SAM und SAA terminiert haben, beendet sich auch der SSI.

3.2 RHODOS

Der RHODOS-Trader ermöglicht den Zugriff auf Objekte in homogenen, verteilten Systemen mit größtmöglicher Autonomie und Flexibilität für den einzelnen Benutzer.

Der Hauptgrund, warum man die bereits entwickelten Methoden und Algorithmen für gleichzeitigen Objektzugriff im 1-Personen-System im verteilten Fall nicht nutzen kann, ist, daß man nie die volle Information über alle Objekte im System hat. Dieses Problem kann durch die Einführung von Tradern gelöst werden, die für den Benutzer transparent auf Objekte lokal oder entfernt zugreifen.

In dem hier vorgestellten System wird die Lokationstransparenz durch attributierte Namen erreicht. Ein Objekt wird hierbei nicht durch einen einzelnen Namen identifiziert, sondern durch eine Reihe typisierter Attribut-Wert-Paare, die das Objekt genauer auszeichnen. Dadurch wird der Abbildungsprozeß flexibler, da nun oft auch eine unvollständige Information über ein gesuchtes Objekt für die Selektion ausreicht. [NG94]

Trading Architektur Trading kann als Exportieren und Importieren angesehen werden. Die gemeinsame Nutzung ist abhängig von den Export-Bedingungen des Exporteurs und dem Einverständnis des Importeurs. Dafür muß jedes Objekt nach einem passenden Namensschema benannt sein, und diese Objektbeschreibungen müssen den Tradern bekannt sein. Um Objekte aus verschiedenen Systemen unterscheiden zu können, gibt es das Konzept der *naming domains* (dt. Namens-Bereiche). Ein RHODOS-Trader besteht aus zwei Servern:

- einem Nameserver, der alle Namensoperationen (create, change, delete, ...) an den Objekten ausführt und

- einem Trader, der die Trading-Operationen (export, import, withdrawal) bereitstellt.

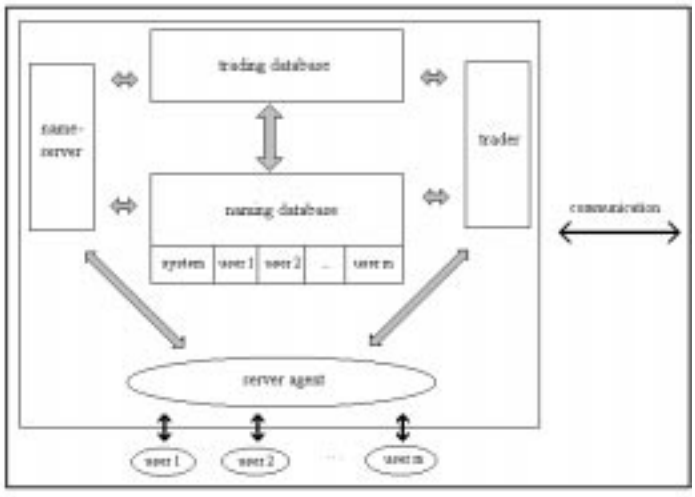

Abbildung10. RHODOS-Trader Architektur

Die Trader können untereinander kooperieren und den Benutzern erlauben auf Objekte zuzugreifen, die vom RHODOS-System verwaltet werden. Die Objekte werden in drei Gruppen unterteilt:

1. *private objects* gehören dem einzelnen Benutzer.
2. *system objects* vom RHODOS-Betriebssystem.
3. *shared objects* sind Objekte, die vom Benutzer oder Systemverwalter exportiert wurden und nun von anderen Benutzern importiert werden können.

Ein häufig anzutreffender Begriff ist der der *domains*. Eine Domäne ist ein abstrakter Begriff für den Geltungsbereich eines Objekts. Man unterscheidet zwischen *user subdomain*, *system subdomain* und *trading subdomain*. Die Vereinigung aller Bereiche ist die *RHODOS domain*. Je nachdem zu welchem Bereich ein Objekt gehört, kann es für den Benutzer unsichtbar, sichtbar aber nicht benutzbar oder sichtbar und zugänglich sein. Ein Objekt ist genau dann unsichtbar, wenn es sich nur in der user subdomain oder der system subdomain befindet. Es ist dann sichtbar, wenn es sich auch noch in der trading subdomain befindet. Um Objekte zu verwalten, benötigt man spezielle Trading-Operationen:

- export *attribute [, attribute]; domain [, domain]; export conditions;*
- import *attribute [, attribute];*
- withdrawal *attribute [, attribute]; domain [, domain];*

Als Parameter kann man Attribute oder eine Attributliste angeben. Bei der export-Operation kann der exportierende Benutzer so die Eigenschaften seines Objekts genau beschreiben, und beim Import kann ein Benutzer das gewünschte Objekt spezifizieren. Der domain-Parameter gibt an, in welche Domäne ein Objekt exportiert bzw. aus welcher es zurückgezogen werden soll. Die Informationen, die nötig sind, um attributierte Namen auf Objekte abzubilden, werden in einer Namensdatenbank gehalten. Die Informationen über die Export-Bedingungen, Namen der Importeure etc., also alle Daten, um Objekte miteinander zu teilen, stehen in der Trading-Datenbank. Wie in Abbildung 10 zu erkennen, haben der Systemverwalter und jeder Benutzer ihre eigenen Teilbereiche in der Namensdatenbank, wo sie ihre privaten Objekte abgelegt haben, die damit für andere unsichtbar sind.

In der Trading-Datenbank befinden sich alle Objekte der trading-domain. Jedes Objekt, das sich dort befindet, hat verschiedene Sektionen, in denen Objektinformationen, Export/Import-Bedingungen und Daten über Importeure stehen:

Trading-Dienst Implementierung Dieser Abschnitt befaßt sich damit, wie beim Trading Lokationstransparenz erreicht wird, und wie die Kooperation zwischen den Tradern abläuft.

Wenn ein Benutzer auf ein bestimmtes Objekt zugreifen will, dann muß er es anhand einer Attributliste spezifizieren und wendet sich anschließend mit dieser Import-Operation an

Section 1	Section 2	Section 3
attribute name	exported to	imported by
...	export conditions 1	import conditions
access rights	...	...
owner domain	exported to	imported by
reference to a system name	export conditions n	import conditions m

Tabelle4. Datenbankeinträge für lokale Objekte

Section 1	Section 2	Section 3
attribute name	import user 1	imported from
...	...	import conditions
access rights	import user n	
reference to a database entry		

Tabelle5. Datenbankeinträge für entfernte Objekte

den Trader. Der beauftragte Trader durchsucht erst die lokale Trading-Datenbank nach einem passenden Objekt. Findet er dort keines, dann kontaktiert er einen entfernten Trader. Der entfernte Trader durchsucht ebenfalls seine lokale Datenbank. Falls er erfolgreich ist, und ein Benutzer dieser RHODOS-domain sein Objekt exportiert hat, dann importiert der lokale Trader dieses Objekt in seine eigene trading-domain und führt nun die Import-Operation des Benutzers aus. Nach Abschluß der Import-Prozedur kann der Benutzer auf das Objekt zugreifen, als ob es ein lokales wäre. Man erreicht damit eine völlige Lokationstransparenz, da der Anwender nicht weiß, ob das Objekt, auf das er zugreift, lokal oder entfernt ist.

Abschließend folgt noch einen kurzen Überblick über die Hauptkomponenten des RHODOS-Systems und deren Funktionen, über die man bei der Implementierung Bescheid wissen sollte.

- Der *User-Agent* läuft auf jedem Rechner im verteilten RHODOS-System. Er kommuniziert direkt mit dem Benutzer-Prozeß. Er erhält die Requests vom lokalen Benutzer und gibt sie an den Request-Manager des Traders weiter, erhält Antworten von ihm und leitet diese an den Benutzer weiter.

- Der *Communication-Manager* erhält Nachrichten von entfernten Tradern, leitet sie weiter an den Request-Manager, erhält von ihm Antworten und übergibt die Ergebnisse wieder an den Sender.

- Der
  Request-Manager bearbeitet Anfragen vom User-Agent oder vom Communication-Manager. Je nachdem um welche Art Anfrage (Namens- oder Trading-Operation) es sich handelt, ruft er den Naming-Manager oder den Trading-Manager auf.

– Der *Naming-Manager* kümmert sich um Namensoperationen bzgl. der Attribute oder der Objekte.

– Der *Trading-Manager* sorgt für das korrekte Ausführen der Trading-Operationen.

– Der *Datenbank-Manager* kümmert sich sowohl um die Namens- als auch um die Trading-Datenbank. Mit dem verteilten RHODOS-Betriebssystem hat man zeigen können, daß es möglich ist, einen Trading-Dienst aufzubauen, mit dem man Objekte in einem großen homogenen System mit anderen Benutzern teilen kann. Grundlegende Merkmale sind hierbei die Kooperation von Tradern und die Export-/Import-Operationen der Objekte. Dadurch ist man in der Lage, eine fast vollständige Information über allen verfügbaren Objekte zu haben.

3.3 DRYAD

Da in einem verteilten System auch Benutzer mit geringen Ressourcen Zugriff auf andere Dienste haben sollten, versucht man mit dem Trading-Mechanismus ein bequemes und effizientes Tool dem Endbenutzer zur Verfügung zu stellen, mit dem er in der Lage ist, direkt auf entfernte Dienste zuzugreifen. Zugleich will man ein System, das fehlertolerant ist und zum Lastausgleich beiträgt.

Der folgende Abschnitt beschäftigt sich mit *implizitem Trading* und der Implementierung des DRYAD Trading-Systems.

Implizites Trading Die Idee, die hinter dem impliziten Trading steht, basiert auf der Tatsache, daß alle Serverreferenzen auf Referenzen auf Files reduziert werden können. Genauso kann man die Anforderung eines Files auch etwas abstrakter als eine Dienstanforderung bezeichnen. Unter einem Dienst kann man entweder eine Programmausführung oder die Wiedergewinnung von Daten aus einem File verstehen. Bevor ein Programm oder ein entfernter Aufruf ausgeführt werden kann, muß ein ausführbares Programm vom Betriebssystem gelesen werden. Ressourcen, wie z.B. Laufwerke oder auch Prozesse, werden häufig als Files im Betriebssystem dargestellt. Folglich kann die Vermittlung von Servern als eine neue Art von Filesystem betrachtet werden. Die Hauptfunktion des Filesystems ist es, die Namen zu interpretieren und dem anfordernden Prozeß den Inhalt des benannten Files bereitzustellen.

Im DRYAD-System wurde ein spezieller File-System-Manager-Prozeß geschaffen, den man *Daphne* genannt hat. Daphne hat v.a. die Aufgabe in dem virtuellen Filesystem aufzupassen, was Server und was Dateiinhalte sind. Der Benutzer sieht nur Programme oder Daten, auf die er zugreifen kann. Den Zugriff erledigt Daphne mit Hilfe eines Traders. Diesen nicht leicht zu verstehenden Sachverhalt sollte man sich an einem Beispiel verdeutlichen:

Wenn ein Benutzer auf ein File X zugreifen will (`read X`), dann sieht der Manager-Prozeß den Namen X und interpretiert ihn. Nachdem er die Benutzer- und die System-Anforderungen überprüft hat (`read user-/system-requirements`), in denen einige Auswahlkriterien stehen, generiert Daphne eine Dienstanforderung an den Trader

(`request service`). Nachdem eine Antwort eingetroffen ist, schickt er diese sofort an den Client, als ob es der Inhalt des Files wäre.

Durch implizites Trading ist es möglich, auch alte Applikationen mittels Trading zu nutzen ohne sie zu modifizieren. Wenn die Daphne-Files in einer Computer-Filesystem sichtbar sind, dann erscheinen alle Dienste in einem Netzwerk als lokale Dienste.

Eine weitere praktische Hilfe leistet implizites Trading bei Systemumgebungen. Bisher war es normal, daß ein Benutzer seine Workstation selbständig konfigurieren mußte, indem er eine Art Startup-File auf seine Bedürfnisse zugeschnitten hat. Wenn er jedoch einen anderen Computer benutzen mußte, war er gezwungen, alles wieder mühsam anzupassen und sich mit den Problemen der neuen Umgebung auseinanderzusetzen (Window-Manager, Screen-Größe, Soundunterstützung, Farbparameter, etc.). Wenn man jedoch das Startup-File mit Hilfe des Traders generiert, könnten die speziellen Eigenschaften der Umgebung automatisch an die Fähigkeiten des Systems angepaßt werden. [KK94]

Implementierung des Trading-Systems Im DRYAD-Forschungsprojekt hielt man sich stark an den Trading-Dienst, wie er im ODP Referenz-Modell beschrieben worden ist. Die Implementierung des Trading-Protokolls basiert direkt auf dem *Remote Procedure Call* (RPC). Sowohl die Trading-Datenbank als auch das Filesystem bauen auf den RPC auf. Die ISO hat Trading nur in Bezug auf sein externes Verhalten definiert. Intern gibt es leichte Unterschiede zwischen den einzelnen Realisierungen. Trotzdem verhalten sich alle Trading-Objekte deterministisch. Für das Management und die Beschreibung des Trading-Verhaltens gibt es ein Regelwerk. Dieses Regelwerk hat drei Bereiche:

- Richtlinien für Importeure, Exporteure und Trader
- Richtlinien für das Importieren und Exportieren.
- Richtlinien für lokale Operationen, Sicherheit und Föderation

Abschließend sei noch erwähnt, daß die Hauptkomponenten der DRYAD Trading-Software die Trader-Objekte, eine Bibliotheksschnittstelle für C-Programmierer und ein grafisches Management-Tool für den Administrator sind. [Kut96]

3.4 Sonstige Systeme

In Tabelle 6 werden weitere verteilte System, die den Trading-Mechanismus benutzen, miteinander verglichen:

Kriterium	ANSA-Trader	Prokrust. Trader	V-System	MELODY	CYGNUS
Attributierung	statisch	statisch	statisch und dynamisch	statisch und dynamisch	statisch
Föderation	ja	ja	ja	ja	implizit
Kooperation	ja	ja	nein	nein	nein
Verbindungsart	direkt	direkt	direkt	indirekt	indirekt
Fehlertoleranz	keine	keine	keine	Transaktion, Checkpoints	Logger
Trading-Strategie	Random-, Best Choice	Random-, Best Choice	Best-Choice	Variante von Best Choice	Conservative Choice
Auswahlkriterium	individuell	individuell	individuell	individuell	individuell
Informations-verwaltung	Subtyping	Subtyping	X.500, MSS	Subtyping	verteilte Datenbank
Mechanismen zur Laufzeitoptimierung			gestaffelte Auswertung, Probing	gestaffelte Auswertung, Leistungszahl, Caching	

Tabelle6. Vergleich verschiedener Trading-Systeme

TINA - Die Zukunft liegt in "intelligenten" Netzen

Marie-Luise Schneider

Kurzfassung

TINA steht für Telecommunications Information Networking Architecture. Dieser Beitrag behandelt die grundsätzlichen Konzepte der TINA-Architektur und soll damit die Antwort auf die Frage liefern, was eigentlich ein TINA-System ist. Dazu werden speziell die Unterabteilungen der TINA-Architektur: Netzwerkarchitektur, Softwarearchitektur, Dienstarchitektur und Verwaltungsarchitektur vorgestellt.

1 Einleitung

Die Probleme, mit denen die Telekommunikationsindustrie heute konfrontiert werden, sind weltweit die gleichen. Es geht darum, auf Netzen mit ständig wechselnden Technologien kosten-effektiv und schnell neue Dienste anbieten zu können, die an dem Ort, wo sie angeboten werden sollen, den Bedürfnissen der Verbraucher ganau entsprechen. Deshalb wurden schon 1990 von industrieller Seite, die TINA-Workshops initiiert. Aus diesen kleineren Gruppen, die alle an ähnlichen Problemen arbeiteten, wurde dann unter Führung von Bellcore, BT, NTT und anderen großen Telekommunikations- und Rechnerunternehmen das TINA-Konsortium gebildet, das seit Januar 1993 in Red Bank, New Jersey ständige Forschungsarbeit leistet. Unterstützt wird das Konsortium von 35 großen Unternehmen und Organisationen aus Europa, dem pazifischen Raum und Nord Amerika durch die Finanzierung von 45 ständigen Forschern im TINA Core Team oder durch die Durchführung von Hilfsprojekten in den eigenen Forschungszentren. Im Dezember 1993 konnten die ersten Konzepte der Architektur vorgelegt werden, die bis heute noch durch Beispielimplementierungen validiert und verfeinert werden. Darüber hinaus werden auch noch Erweiterungen um zusätzliche Telekommunikationsaspekte wie Echtzeitanforderungen angefügt. Ein weiterer Teil der Arbeit der Team-Mitglieder bezieht sich auf Ausarbeitungen von Demonstrationen für die Unternehmensmitglieder, sowie auf die Zusammenarbeit und Beeinflussung von wichtigen Standardisierungsorganisationen.

Die TINA-Architektur stellt eine Menge von Konzepten und Prinzipien zur Verfügung, die auf die Spezifikation, den Entwurf, die Implementierung, die Installierung und Ausführung von *Software* für Telekommunikationssysteme angewandt werden sollen. Die Spannweite an verschiedenen Diensten, die von einem TINA-System den Systembeteiligten zur Verfügung gestellt werden soll, ist möglichst groß und reicht von stimmbasierten, interaktiven Multimedia- und Informationsdiensten bis zur Verwaltung von Diensten. Ein Telekommunikationssystem aus Sicht von TINA, stimmt nicht notwendigerweise mit administrativen Grenzen zusammen, d.h. daß ein TINA-System aus Hardware- und Software-Komponenten von verschiedenen Netzanbietern und Deinstanbietern bestehen

soll. TINA-Systeme sollen auch fähig sein, mit anderen schon existierenden Systemen (wie Intelligent Networks oder Telecommunication Management Networks) zusammenzuarbeiten, die nicht TINA-konform sind. Bezüglich der Software gibt es in der TINA Architektur zwie Hauptprinzipien:

- verteiltes Rechnen (distributed computing) und
- Objektorientierung

Um die Problemstellungen besser einteilen zu können, wird in der TINA-Architektur zwischen vier Unterarchitekturen unterschieden (vgl. [CM95]):

Netzwerkarchitektur Konzepte und Prinzipien zum Entwurf, Spezifikation, Implementierung von Transportnetzwerken (enthält Übertragung und Vermittlung).

Softwarearchitektur Konzepte und Prinzipien zum Entwurf und Erstellung von verteilter Software und softwareunterstützender Umgebungen.

Dienstarchitektur Konzepte und Prinzipien zum Entwurf, Spezifikation, Implementierung und Verwaltung von Telekommunikationldiensten.

Verwaltungsarchitektur Konzepte und Prinzipien zum Entwurf, Spezifikation, Implementierung von Softwaresystemen, die benutzt werden, um Dienste, Ressourcen, Software und unterliegende Technologien zu verwalten.

Diese Unterarchitekturen müssen zusammen ein konsistentes Konzept bilden und beeinflussen sich deswegen gegenseitig. So müssen die allgemeinen Softwarearchitekturkonzepte auch auf die Software der Dienst-, Netzwerk- und Verwaltungsarchitektur angewandt werden, um durch diese Vereinheitlichung interoperable und portable Software zu garantieren. Andererseits müssen die Dienste der Managementsysteme nach den Prinzipien der Dienstarchitektur erstellt werden.

Die Seminararbeit folgt in ihrem Aufbau dieser Gliederung in Unterarchitekturen, nachdem zunächst die äußere Erscheinungsform eines TINA-Systems vorgestellt worden ist.

2 Grundlegendes Gerüst für TINA-Systeme

Die Kernstruktur der Telekommunikationssoftware eines TINA Systems besteht aus den Telekommunikationsanwendungen, die die Fähigkeiten des Systems implementieren, und der verteilten Prozeßumgebung (distributed processing environment) *DPE*, die die verteilte Ausführung der Telekommunikationsanwendungen unterstützt.

Diese Software wird nun in ein größeres Konzept eingebettet, das durch Schichtung strukturiert wird. Auf der untersten Schicht dieser Struktur befinden sich die Hardware-Ressourcen wie Prozessor, Speicher und Kommunikationsgeräte. Darüber liegt eine Software-Schicht, die das Betriebssystem, Kommunikationssoftware und andere unterstützende Software enthält, wie sie in Rechnersystemen vorkommen. Diese Schicht

wird die ursprüngliche Rechner- und Kommunikationsumgebung (native computing and communications environment) *NCCE* genannt. Darüber liegt die DPE-Schicht und darauf die Anwendungsschicht. Auf jeder dieser Ebenen soll Zusammenarbeit mit nicht TINA-konformen Systemen möglich sein. Jede dieser allgemein beschriebenen Ebenen kann nun noch genauer untersucht werden. Die NCCE z.B. ist keine homogene Schicht, sondern wird aus einer Menge von untereinander verbundenen Rechnerknoten gebildet, die verschiedene Hardware und Software Technologien unterstützen können.

Die Aufgabe der DPE ist es, für die Anwendung eine technologieunabhängige Sicht auf die NCCE zu liefern, um die technologieabhängigen Aspekte der Anwendung zu minimieren, und so den Entwurf, die Wiederverwendbarkeit und Portabilität zu vereinfachen. Dazu muß die DPE auch die Verteilung vor der Anwendung verbergen. D.h. die Anwendungen sind zwar als Menge von interagierenden Objekten auf die Knoten verteilt, aber bei ihrem Entwurf muß wegen der Unterstützung der DPE kein Wissen über die Lage der Teile eingebracht werden. Dies ermöglicht die DPE durch Verteilung der Objekte auf das System und Ausführung entfernter Operationen. Die genaue Ausstattung und Implementierung der NCCE gehört nicht zum Aufgabenbereich der TINA-Architektur. Dazu gehört wie gesagt nur die DPE und die Telekommunikationsanwendungen. Aber es gibt festgesetzte Berührungspunkte zwischen DPE und NCCE:

1. Auf einem Knoten, auf dem sich eine DPE und TINA-Anwendungen befinden, kann auch andere Software existieren. Diese Existenz spielt jedoch für TINA keine Rolle.

2. Die DPE benötigt zwar Transportfunktionen und hat bestimmte Anforderungen an deren Funktionsfähigkeit, aber wenn die Transportfunktionen der NCCE diesen Anforderungen nicht genügt, muß die restliche Funktionalität in der DPE Schicht selbst implementiert sein.

3. Einige Knoten können mit speziellen Hardwareressourcen ausgerüstet sein, wie video processing boards, switch fabrics und Sprach-Aufnahme bzw. -Abspiel-Geräten. Die Software für solche Geräte wird aber nicht als Teil der NCCE betrachtet, weil sie keine allgemeinen Rechner- oder Kommunikationsfunktionen darstellen. Diese Komponenten bilden also sozusagen einen eigenen Stack neben dem TINA-Stack, auf dessen Software die TINA-Anwendungen aber zugreifen können.

Wie das Bild zeigt, werden die Anwendungen selbst wieder unterteilt in die Elemente, Ressourcen und Dienste. In dieser Einteilung folgt TINA einem TMN-Prinzip (vgl. [DMB+94]):

1. *Die Elementschicht*
   In dieser Schicht befinden sich Objekte (sogenannte Elemente), die atomare Einheiten von physischen oder logischen Ressourcen repräsentieren, die zur Allokierungskontrolle, zu Benutzungs- und Verwaltungszwecken dienen. Beispiele sind switching fabric und Übertragungsausrüstung. Elemente, die eine physikalische Geräte vertreten, müssen gemäß der Softwarearchitektur konzipiert sein, damit andere TINA-Software mit ihnen in einer TINA-konsistenten Art interagieren kann. Elemente, die physikalische Geräte repräsentieren, sind natürlich technologieabhängig. Trotzdem sollte der

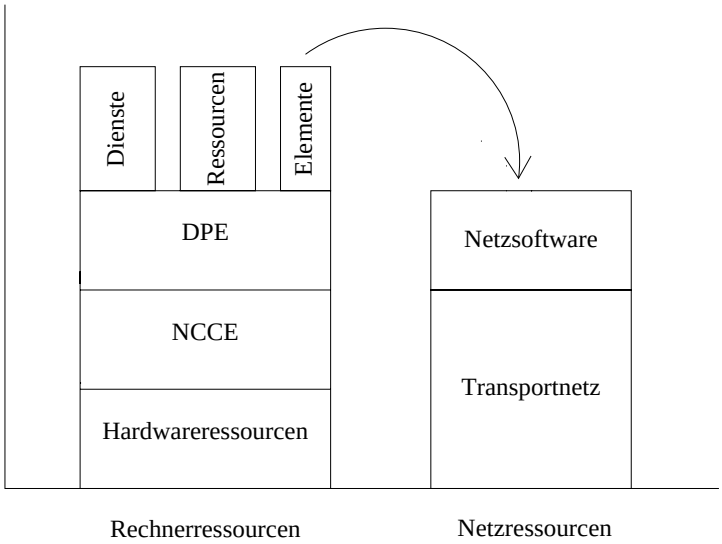

Abbildung11. Das Transportnetz in der Umgebung eines DPE-Knotens

Gebrauch von Standardelementrepräsentationen Herstellerunabhängigkeit bieten. Die Objekte der Elementschicht sind individuelle Ressourcen, da sie miteinander nicht direkt interagieren können und Beziehungen zwischen den Elementen in dieser Schicht nicht dargestellt werden. Die Elemente sind aus Verwaltungssicht nur verwaltete Objekte (Definition s. Abschnitt 4.3), die nur die Dinge repräsentieren, die verwaltet werden sollen.

2. *Die Ressourcenschicht*
   Diese Schicht enthält nun die Objekte, die Mengen von Elementen und ihre Beziehungen manipulieren können. Sie bietet der Dienstschicht eine abstrakte Repräsentation der Elemente.

3. *Die Dienstschicht*
   Die Objekte der Dienstschicht können zu einem bestimmten Dienst gehören oder dienstunabhängig sein. Die ersteren umfassen die Logik, Daten und Verwaltungskapazität für ihren Dienst, die zweiten generische Zugriffs-, Kontroll- und Verwaltungskapazitäten die auf alle Dienste anwendbar sind.

3 Netzwerkarchitektur

Der Zweck dieser Netzwerkarchitektur (Network Architecture) ist es, eine Menge generischer Konzepte vorzustellen, mit denen ein Transportnetz in einer technologieunabhängigen Art beschrieben werden kann, und die die Mechanismen zum Aufbau, zur Modifikation und zum Abbau von Netzverbindungen bieten. Die Netzwerkarchitektur definiert Abstraktionen, mit denen die Ressourcenschicht arbeiten kann, indem einerseits eine stark abstrahierte Sicht auf Netzverbindungen den Diensten angeboten werden kann, und andererseits generische Beschreibungen von Elementen definiert sind, die für bestimmte Technologien und Produkte spezialisiert werden können.

Die TINA Netzarchitektur geht von der ITU-T Empfehlung G.803 aus, die eine Netzstruktur bestehend aus Partitionen und Schichten vorschreibt. Ein Netz ist demnach in Unternetze partitioniert. Diese Partitionierung kann sich rekursiv fortsetzen, bis man auf der Ebene von Schltern und digitalen Verbindungselementen angekommen ist. Schichten können als eigene Netze mit kompatiblen Ein- und Ausgaben aufgefaßt werden. Zwischen Schichten kann eine Client/Server-Beziehung bestehen, indem ein Verweis von einem Unternetz zu einem anderen in der Clientschicht durch eine Ende-zu-Ende-Verbindung in der Serverschicht realisiert wird. Um diese Netze beschreiben zu können, existiert ein Network Resource Information Model *NRIM*, das Netze als Zusammenschlüsse von Unternetzen, Verbindungen und Endpunkten beschreibt. Das Prinzip der Verbindungsgraphen zielt im Gegensatz zur detaillierten Beschreibung im NRIM auf die dienstorientierte Sicht der Verbindungen, wie sie der Benutzer braucht, ab. Verbindungsgraphen bestehen aus Knoten (die Rechnerknoten darstellen), Ports (die Netzzugangspunkte darstellen) und Linien, die die Ports verbinden (Netzverbindungen darstellend). (vgl. [CM95])

4 Softwarearchitektur

Eines der grundsätzlichsten Prinzipien für die Konstruktion von Software in TINA Systemen ist die Objektorientierung, die gegenüber der funktionalen Dekomposition einen wesentlichen Vorteil bietet: die bessere Unterstützung der Wiederverwendbarkeit von Software und Spezifikationen. Denn eines der ausgesprochen wichtigen Ziele des TINA-Konsortium ist es, Dienstkomponenten zu definieren, die wiederverwendet, spezialisiert und benutzt werden können, um neue Dienste zu erstellen. Die Eigenschaften der Objektorientierung wie Vererbung, Spezialisierung und Komposition sind also notwendige Instrumente, um die beschriebenen Architekturmerkmale zu unterstützen. Um eine konsistente Anwendung objektorientierter Prinzipien zu garantieren, greift die TINA-Architektur auf das RM-ODP descriptive model (s. dazu [ISO95a]) zurück, das vollständig ist und eine konsistente Menge von Konzepten anbietet. Auch in anderer Hinsicht lehnt sich TINA an das RM-ODP (dieses Mal das prescriptive model; s. dazu [ISO95b]) an. Um ein komplexes System zu beschreiben, spezifiziert man es unter fünf Gesichtspunkten:

1. *Unternehmensgesichtspunkt* (enterprise viewpoint): Zweck und Zusicherungen für das System, Aufgaben der Systembeteiligten.

2. *Informationsgesichtspunkt* (information v.): Bedeutung und Ziel des Systems.

3. *Zusammensetzungsgesichtspunkt* (computational v.): Dekomposition des Systems in eine Menge interagierender Objekte.

4. *Verteilungsgesichtspunkt* (engineering v.): Benötigte Infrastruktur, um die Verteilung des Systems zu unterstützen.

5. *Technologiegesichtspunkt* (technologie v.): Benötigte Technologie.

Da der ODP Standard für allgemeine Systeme gedacht ist, wird er in der TINA Softwarearchitektur ausdrücklich im Hinblick auf Telekommunikationssysteme verfeinert und angepaßt. Dabei gilt der Hauptaugenmerk der TINA Forschung den Informations-, Zusammensetzungs- und Verteilungsaspekten.

4.1 Modellierungskonzepte bezüglich des Unternehmensgesichtspunkts

Das Unternehmensmodell beschreibt das System aus Sicht der Organisation und der Menschen, die das System benutzen und betreiben. Diese Konzepte umfassen:

- Rollen, die beschreiben, welche Art von Aktivität die einzelnen Beteiligten im System übernehmen.
- Anforderungen, die die Funktion und Kapazität des Systems beschreiben.
- Verpflichtungen, die die Beteiligten übernehmen.
- Zusicherungen, die Einschränkungen an die Benutzung und das Betreiben des Systems darstellen.

Über Verpflichtungen und Zusicherungen gibt es in der Architektur jedoch noch keine Aussagen.

4.2 Modellierungskonzepte bezüglich des Informationsgesichtspunkts

Diese Konzepte stellen ein Gerüst für Informationsspezifikationen dar, also Spezifikationen, die beschreiben sollen, welches Problem das System lösen soll, wie es benutzt und verwaltet werden kann. Die Konzepte enthalten Aussagen über:

- informationstragende Einheiten = Informationsobjekte
- Klassifikation der Informationsobjekte in Objekttypen
- Beziehungen zwischen Einheiten
- Einschränkungen und Regeln, die ihr Verhalten steuern, einschließlich der Regelnq für Konstruktion und Destruktion

Die Informationsspezifikation beschäftigt sich ausschließlich damit, was das System tun soll, und nicht damit, aus welchen Teilen die benötigte Software besteht (s. Zusammensetzungsaspekt), und auf welche Knoten diese verteilt werden müssen (s. Verteilungsaspekt). Als Notation wird ein Quasi-GDMO-GRM (Guidelines for the Definition of Managed Objects, General Relationship Model) benutzt, zur graphischen Darstellung dient OMT (Object Modelling Technique).

4.3 Modellierungskonzepte bezüglich des Zusammensetzungsgesichtspunkts

Diese Konzepte sollen ganz allgemein folgenden Anforderungen genügen (s. [NDSC95]):

- Konzepte für die Beschreibung von Interaktionen zwischen Anwendungsqkomponenten sollen definiert werden;
- die Konzepte sollten sowohl diskrete als auch kontinuierliche Kommunikation wie Audio-, Videokommunikation umfassen;
- es müssen Konzepte existieren, um Verwaltungsinteraktionen zwischen Softwareeinheiten beschreiben zu können;
- die Konzepte sollen funktionale und auch nicht-funktionale Aspekte von verteilten Anwendungen betrachten. Nicht-funktionale Aspekte sind solche, die mit der Dienstqualität, Konfiguration und Aspekte der Unternehmenszusicherungen (Accounting- und Sicherheitszusicherungen) in Beziehung stehen.
- die Konzepte sollen die Entwicklungsmöglichkeit von verteilten Anwendungen unterstützen, so daß Anwendungskomponenten, die unabhängig aufgegeben und erneuert werden können, definiert werden;
- die Konzepte sollen ein Gerüst für die Spezifikation von Anwendungsschnittstellen liefern, wobei die Spezifikationen Aussagen über Struktur und Semantik der Schnittstellen liefern sollten;
- und die Konzepte sollen, wo immer möglich, auf existierenden Standards und Ergebnissen basieren.

Die Modellierungskonzepte folgen wieder stark den Vorgaben des RM-ODP prescriptive model (s. [ISO95b] und [NDSC95]).

1. Eine verteilte Anwendung ist zusammengesetzt aus Objekten, die miteinander interagieren.

   Dieser Punkt ist in TINA genauso realisiert.

2. Ein Objekt stellt seine Fähigkeiten über eine oder mehrere Schnittstellen anderen Objekten zur Verfügung. Bei TINA wird noch konkretisiert, daß die Menge von Fähigkeiten eines Objekts in Untergruppen gegliedert wird, die jeweils einen Dienst darstellen.

3. Interaktion zwischen Objekten geschieht entweder durch Operationsinitiierung und -beantwortung oder durch Informationsflüsse, wobei jeder Fluß ein unidirektionaler Bitstrom ist.

   Wegen dieser Definition unterteilt TINA Schnittstellen in operationale Schnittstellen und Stromschnittstellen, die die oben definierten Interaktionstypen unterstützen.

4. Um die dynamische Konfigurierung eines Objekts zu vereinfachen, wird das Trading-Konzept benutzt. Ein Objekt, das eine bestimmte Kapazität benötigt, muß keine Verbindung zu einem Objekt haben, das diese Kapazität bereitstellt, sondern kann erst während der Ausführung ein geeignetes Objekt mithilfe eines Traders suchen.

   In TINA sind darüber hinaus noch allgemeinere Arten aufgezeigt, wie man eine Verbindung (Voraussetzung zur Benutzung einer Schnittstelle) zu einem anderen Objekt herstellen kann.

   – Erzeugung eines Objekts, das diese Schnittstelle besitzt

   – Gewinnung der Referenz als Argument oder Ergebnis einer Interaktion mit einem anderen Objekt. Letzteres kann das Schnittstellen-anbietende Objekt selbst oder ein anderes sein.

5. Transaktionen mit ACID-Eigenschaften werden initiiert, indem bestimmte Operationen sogenannte Transaktionsoperationen aufgerufen werden.

   In TINA sind deshalb ganz allgemein Operationen in drei Kategorien unterteilt:

   – transaktionsinitiierende Operationen (transaction initiation operation)

   – transaktionsverbindende Operationen (transaction join operation)

   – Operationen, die nichts mit Transaktionen zu tun haben (non-transaction operation)

   Dabei können aber nur Anfrageoperationen, wie sie die operationalen Schnittstellen zur Verfügung stellen, als transaktionsinitiierende bzw. -verbindende Operationen verwendet werden.

6. Um Anwendungen in die Lage zu versetzen, neue Arten von Interaktionen wie Multicast-Gruppen-Kommunikation zu definieren, die nicht innerhalb des Basisinteraktionsmodell aufgebaut werden, und selbst Kontrolle über diese Interaktionen auszuüben, werden Explizit-Binder-Objekte im Unterschied zu einfachen Binder-Objekten definiert.

   Implizites Binden ist in TINA folgendermaßen definiert: implizite Bindungen werden, wie der Name schon andeutet, implizit von der Infrastruktur zur Verfügung gestellt, d.h. durch grundlegende Mechanismen der DPE und des unterliegenden Transportnetzes. Dabei müssen die beteiligten Objekte diese Bereitstellung nicht explizit anfordern. Wünschen die Anwendungen aber eine eigene Kontrolle der Verbindung, um z.B. neue Parteien dynamisch hinzufügen, entfernen oder zwischen Verbindungsaufbau und Interaktion unterscheiden zu können, dann braucht man eine explizite Bindung. Diese ist z.B. notwendig, um die Dienstanforderungen einer Bindung zu erfüllen, die Stromschnittstellen zur Übertragung von Multimediaströmen verbindet. Eine solche Bindung wird als Binder-Objekt modelliert, das die Bindermechanismen kapselt und Operationen zur Kontrolle der Bindung liefert.

Die TINA-Architektur folgt dem Vorbild von TMN, wenn sie die Schnittstellen in Dienst- und Verwaltungsschnittstellen einteilt. Dadurch erscheint ein Objekt unter zwei verschiedenen Blickwinkeln. Die Dienstschnittstelle liefert die Sicht eines Dienstanbieters und

die Verwaltungsschnittstelle die eines verwalteten Objekts. Das Objekt, das eine Verwaltungsschnittstelle zur Verfügung stellt, wird demnach verwaltetes Objekt (managed object) und das Objekt, was sie benutzt Verwalter (manager), genannt. Diese Begriffe gelten natürlich nur bezüglich der jeweiligen Schnittstelle. Eine Dienstschnittstelle kann durch eine operationale Schnittstelle oder eine Stromschnittstelle realisiert werden, wohingegen eine Verwaltungsschnittstelle nur den Typ einer operationalen Schnittstelle besitzen kann. Für Objekte und Schnittstellen existieren Spezifikationsschablonen, die in der von Programmiersprachen unabhängigen Notationsart TINA-ODL (Object Definition Language) beschrieben werden. Einen weiteren Architekturbaustein bilden die "Gruppen". Allgemein können Gruppen als Objekte mit einer Menge von Zusicherungen (policies) aufgefaßt werden. D.h. daß Objekte in TINA nur dann zusammengefaßt werden dürfen, wenn ihnen bestimmte Zusicherungen zugefügt werden. Alle Gruppeneigenschaften existieren nicht implizit, sondern müssen explizit durch Zusicherungen ausgedrückt werden. Bestimmte Eigenschaften wie Kapselung, Existenzabhängigkeiten, Lokalität, die in einem verteilten System oft gebraucht werden, sollten als Zusicherung vordefiniert werden, damit Standardschablonen als Hilfe für Entwickler entworfen werden können. Die Beziehung zwischen Objekt und Gruppe zeigt das OMT-Modell:

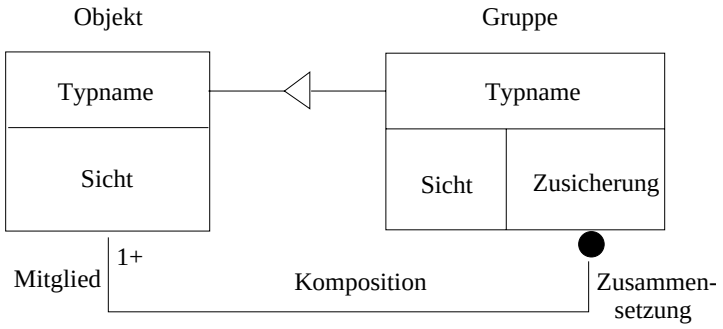

Abbildung12. Die Beziehung zwischen Objekt und Gruppe

4.4 Modellierungskonzepte bezüglich des Verteilungsgesichtspunkts

Diese Konzepte bieten ein Gerüst, um den Einsatz einer Anwendung zu beschreiben, sowie ein Gerüst für die Organisation einer abstrakten Infrastruktur der sogenannten verteilten Prozeßumgebung (distributed processing environment, DPE). Die DPE unterstützt alle softwarebasierten Anwendungen und kapselt die Verteilungsmechanismen, die Sicherheitsprüfer (security checkers), die Monitoren für Dienstqualität (quality of service monitors) und die Prozeß- sowie Kommunikationsressourcenmanager, die vor den Designern der softwarebasierten Anwendungen verdeckt werden sollen. Aus dessen Sicht (beim Entwurf sollen nur die Zusammensetzungsaspekte eine Rolle spielen) stellt die DPE eine homogene Infrastruktur dar, die die Komplexität und Heterogenität der benutzten Netz- und Rechnerressourcen verdeckt.

Um Softwarekomponenten nun durch die DPE ausführen zu lassen, müssen sie zu Orts- und Aktivierungseinheiten (= Cluster) gebündelt werden. Diesen Clustern müssen jeweils Rechnerressourcen (= Capsules, Kapseln) zugeordnet werden. Kommunikation in-

nerhalb eines Clusters wird dem Betriebssystem überlassen, wohingegen Kommunikation zwischen verschiedenen Clustern durch einen Kanal modelliert wird, der sich mit Leiten/Umleiten (marshall/unmarshall), protokollspezifischer Nachrichtenbehandlung und Konsistenzüberprüfungen für Bindungen befaßt (dieser Kanal wird durch das DPE-Kerntransportnetz realisiert).

Die DPE selbst besteht aus drei Teilen: DPE-Kern (kernel), Kerntransportnetz (kernel transport network) und den DPE-Diensten (service, manchmal auch server). Der DPE-Kern bietet Unterstützung bei der Kontrolle der Objekt-Lebenszyklen und der Kommunikation zwischen Objekten. Die Kontrolle über die Objekt-Lebenszyklen schließt die Fähigkeiten ein, ein Objekt zur Laufzeit zu kreieren (instanziieren) und zu zerstören. Die Interobjekt-Kommunikation unterstützt Mechanismen zum Aufruf von Operationen, die von den operationalen Schnittstellen entfernter Objekte angeboten werden. Ein entferntes Objekt ist ein Objekt in einem anderen Cluster. Auch wenn sich die Cluster auf demgleichen Rechner befinden, existiert dennoch die Möglichkeit, daß ein Cluster auf einen anderen Rechner zur Laufzeit migriert. Also dürfen keine Annahmen über den Ort der Cluster gemacht werden, und die Intercluster-Kommunikation sollte immer den Aufruf entfernter Mechanismen verwenden. Der DPE-Kern bietet grundlegende, technologieunabhängige Funktionen wie sie auch die meisten Betriebssysteme anbieten: die Möglichkeit Programme ablaufen und miteinander kommunizieren zu lassen. Der DPE-Kern soll auf allen Knoten installiert sein, die überhaupt eine DPE enthalten.

Um die Kommunikation zwischen entfernten Objekten zu erleichtern, kommunizieren die DPE-Kerne der verschiedenen Rechner miteinander. Diese Kommunikation wird durch das Kerntransportnetz (KTN) bewerkstelligt. Das KTN erzeugt eine technologieunabhängige Sicht auf die Kommunikationsmöglichkeiten der ursprünglichen Rechner- und Kommunikationsumgebung des jeweiligen DPE-Knotens. Das KTN ist ein virtuelles Netz, das logisch verschieden vom Transportnetz ist.
Die DPE-Dienste unterstützen durch ihre operationalen Schnittstellen die Ausführung und Kommunikation der Objekte zur Laufzeit. Beispiele sind: Trading und Namensdienste, die zur Laufzeit die Lokalität und Identität von Schnittstellen entfernter Objekte bestimmen, Benachrichtigungsdienste, die das Verschicken von Nachrichten zu relevanten Softwareeinheiten erleichtern, Sicherheitsdienste, die garantieren, daß nur autorisierte Interaktion zwischen Objekten stattfindet. Die Unterscheidung zwischen DPE-Kern und DPE-Diensten existiert, um grundlegende Kapazitäten, die auf jedem DPE-Knoten installiert sein müssen, von Fortgeschritteneren unterscheiden zu können, die nicht überall installiert sein müssen.

4.5 Modellierungskonzepte bezüglich des Technologiegesichtspunkts

Das Technologiekonzept in TINA besteht gerade darin, keine bestimmte Technologie vorzuschreiben, sondern es den Systemingenieuren zu überlassen, welche Kommunikationsprotokolle sie am besten z.B. für die Anforderungen an die Dienstqualität geeignet halten.

5 Dienstarchitektur

Die Konzepte der Dienstarchitektur (service architecture) sollen generell folgenden Zielen dienen (vgl. [BG95]):

- eine einheitliche Herangehensweise an Entwurf und Verwaltung jeglicher Art von Diensten wie z.B. Multiparteien-, Multimedia- und Träger-Diensten.
- Dienste sollen im Hinblick auf Zeit und Technologie entwicklungsfähig sein. Im Hinblick auf die Zeit sollen Dienste änderbar sein, um neuen Anforderungen wieder zu genügen. Im Hinblick auf die Technologie sollten Dienste trotz heterogener oder u.U. neuer Technologien anwendbar bleiben.
- Anpassungsfähigkeit an sich ändernde Märkte.
- Zusammenarbeit mit schon existierenden und nicht-TINA-konformen Systemen und Diensten, wie Intelligent-Network-basierten Diensten.
- Reibungslose Zusammenarbeit in einer Multi-Lieferanten/Betreiber Umgebung.
- Unterstützung der Mobilität, d.h. Zugriff auf Dienste, auch wenn der Benutzer von einem Ort oder von einem Terminal(-Typ) zu einem anderen wechselt.
- Dienstkunden (subscriber) sollten ihre Dienste teilweise selbst verwalten dürfen.
- unabhängige Entwicklung von Diensten und Netzressourcen. Dadurch wird die schnelle Einführung neuer und die einfache Modifikation existierender Dienste erleichtert.
- Unabhängigkeit von bestimmten Transporttechnologien, erlaubt es, den gleichen Dienst auf verschiedenen Transportinfrastrukturen anzubieten (einen Videokonferenzdienst über ein ATM-Netz oder N-ISDN).
- effiziente Nutzung von Netzressourcen, die nur wenn sie gebraucht werden, und dann nach Verhandlungen (negotiation) zugewiesen werden.

Um Diensteigenschaften wie Offenheit und Interoperabilität zu garantieren, werden Dienste nach dem Universal Service Component Model (USCM) entworfen: Das USCM spiegelt die grundlegenden Separationsprinzipien, die oben erwähnt wurden, wider. Dazu geht es von vier Sektoren aus, dem Dienstinneren und drei Sektoren, die das Dienstinnere umlagern. Diese sollen die folgenden Eigenschaften haben:

- der Benutzungssektor (usage sector) bietet Schnittstellen für externe Komponenten und Dienste, die den Dienst benutzen und kontrollieren, d.h. der Dienst ist aus dieser Sicht ein Server, auf den Klienten über den Benutzungssektor zugreifen.
- der Verwaltungssektor (management sector) bietet die Logik und die Datenverwaltung, um die Initialisierung, Konfigurierung, das Accounting und andere Verwaltungsfunktionen zu kontrollieren, d.h. der Dienst ist aus dieser Sicht ein verwaltetes Objekt, auf das der Verwalter über die Schnittstelle, die durch den Verwaltungssektor zur Verfügung gestellt wird, zugreifen kann.

– der Abhängigkeitssektor (substance sector) enthält die interne Repräsentation der externen Komponenten, die der Dienst benutzt, d.h. der Dienst ist aus dieser Sicht ein Kunde, der über seinen Abhängigkeitssektor (der die Schnittstellen des Benutzungssektors der Serverdienste aufruft) auf andere Dienste zugreift.

– das Innere besteht aus der Logik und den Daten, die die eigentlichen Operationen des Dienstes definieren. Es kann selbst wiederum aus einer oder mehr Komponenten bestehen. Jede einzelne Komponente definiert Teile der Dienstoperationen, ohne inhärentes Wissen über die Umgebung.

Das USCM-Paradigma ist auch mit dem Zusammensetzungsgesichtspunkt der Softwarearchitektur verträglich. Ein softwarebasierter Dienst besteht demnach aus Objekten (s. 4.3). In diesem Fall verlangt das USCM-Paradigma, daß jedes Objekt genau einem USCM Sektor zugeteilt werden kann. Diese Unterteilung unterstützt die Wiederverwendbarkeit von Komponenten (z.B. ganzen USCM-Sektoren) sowie die Konsistenz und Verwaltung der Dienste.

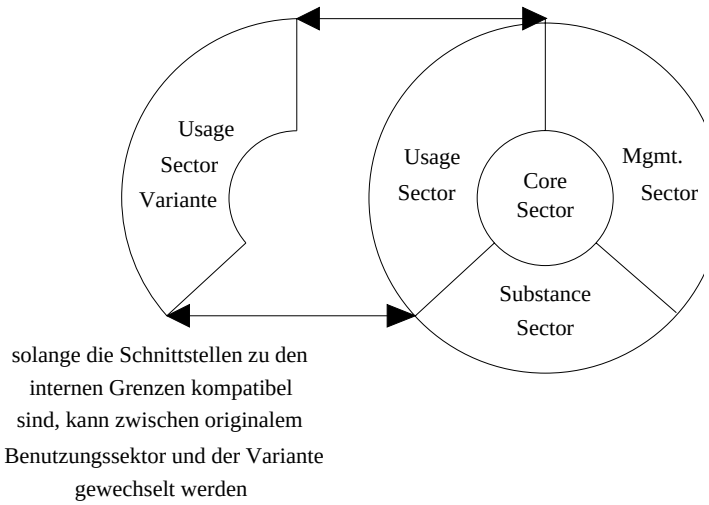

Abbildung13. USCM und ein Beispiel zur Sektorenersetzung

Die Dienstarchitektur beschreibt zusätzlich Sitzungs- (session) und Zugangskonzepte (access), wobei sich die Sitzungskonzepte um Dienstaktivitäten und zeitliche Beziehungen kümmern und die Zugangskonzepte um Benutzer- und Terminalbeziehungen mit Netzen und Diensten. Eine Sitzung ist allgemein als zeitliche Periode definiert, während derer eine Aktivität ausgeführt wird, um ein Ziel zu erreichen. Es werden vier Sitzungstypen unterschieden: Dienst-, Benutzer-, Kommunikations- und Zugriffssitzung.

5.1 Das Sitzungskonzept

Eine Dienstsitzung ist eine einzelne Aktivierung eines Dienstes. Sie verbindet die Benutzer eines Dienstes miteinander, damit sie miteinander interagieren können und Einheiten miteinander teilen können, wie z.B. ein Dokument. Eine Dienstsitzung wird durch einen Dienstsitzungsmanager (service session manager) repräsentiert. Dieser bietet zwei Arten

von operationalen Schnittstellen an. Die erste ist eine generische Sitzungskontrollschnittstelle, die es dem Benutzer erlaubt, sich Sitzungen anzuschließen oder sie zu verlassen. Bei einigen Diensten kann sie auch noch die Möglichkeit bieten, die Sitzung zu unterbrechen und wiederaufzunehmen. Die zweite bietet dienstspezifische Operationen, die durch die Fähigkeiten des jeweiligen Dienstes begründet sind.

Eine Benutzersitzung enthält die Information über die Aktivitäten des Benutzers und die Ressourcen, die für die entsprechende Dienstsitzung allokiert wurden. Beispiele für die gespeicherte Information sind die gesammelten Rechnungen des Benutzers, seine Daten über eventuelle Unterbrechungen und Wiederaufnahmen der Dienstsitzung oder dienstspezifische Informationen, wie die aktuell editierte Seitennummer bei einem verteilten Dokumenten-Editier-Dienst. Wenn ein Benutzer sich einer Dienstsitzung anschließt, wird gleichzeitig eine Benutzersitzung eingerichtet. Sie wird zerstört, wenn der Benutzer die Dienstsitzung wieder verläßt. Die Dienstsitzung enthält Verweise auf die Benutzersitzungen und stellt dadurch eine gruppenorientierte Sicht her.

Eine Kommunikationssitzung ist eine dienstorientierte Abstraktion von Verbindungen im Transportnetz. Eine Kommunikationssitzung enthält Informationen über die Verbindungen einer bestimmten Dienstsitzung, wie den Kommunikationspfaden, den Endpunkten und den Charakteristiken der Dienstqualitäten. Eine Kommunikationssitzung ist nur erforderlich, wenn ein Datenströme zwischen den Softwarekomponenten benötigt werden. Die Kommunikationssitzung wird durch ein Softwareobjekt, den sogenannten Kommunikationssitzungsmanager repräsentiert.

Eine Zugriffssitzung beinhaltet Informationen über die Anbindung eines Benutzers an ein bestimmtes System und seine Einbindung in Dienste. Er kann in viele Dienste zur selben Zeit eingebunden sein und die Zugriffssitzung protokolliert die Daten hierüber.

Der Zweck dieser Sitzungskonzepte ist es, die verschiedenen Aufgaben voneinander zu trennen, und damit die Verteilung von Aufgaben zu erleichtern. Die Trennung von Dienst- und Zugriffssitzung erlaubt es, daß die Methoden und Technologien, mit denen verschiedene Benutzer den Zugang bewerkstelligen, verschieden sein können und, daß der Benutzer den Ort wechseln kann, während der Dienst weiterlaufen kann. Die Trennung der Benutzer- von der Dienstsitzung erlaubt die Verteilung von Funktionalität und Zustandsdaten, wobei die Benutzersitzung eine lokale und die Dienstsitzung eine Gruppenansicht liefert. Es wird dadurch auch das Unterbrechen und Wiederaufnehmen eines Dienstes ermöglicht. Die Trennung von Dienst- und Kommunikationssitzung führt zur Trennung der Dienstaktivitäten von den Verbindungen, die zu ihrer Aufrechterhaltung benötigt werden. Das soll erreicht werden, weil nicht jede Dienstsitzung eine Verbindung beansprucht (stattdessen kann sie durch die operationalen Schnittstellen, die die DPE bietet, aufrechterhalten werden). Auch wenn sie das Transportnetz benutzt, liegt keine Eins-zu-Eins-Beziehung zwischen denen vor, die im Rahmen dieses Dienstes eine Kommunikationssitzung in Anspruch nehmen. Das könnte z.B. der Fall sein bei einer Kooperation von drei Teilnehmern an einem Dokument, wofür eine Dienstsitzung existiert, um das Dokument zu editieren, und zwei der Benutzer eine zusätzliche Audioverbindung aufgebaut haben, um die Änderungen zu diskutieren.

5.2 Das Zugangskonzept

Um dem Benutzer einen flexibleren Zugang zu Diensten bezüglich des Ortes und des Terminaltyps zu ermöglichen, unterscheidet man Benutzer- und Terminalzugang. Dazu sind zwei Arten von Agenten vordefiniert: Benutzeragent und Terminalagent.

Ein Benutzeragent repräsentiert einen Benutzer. Zu seinen Aufgaben gehört es, Anforderungen zum Aufbau von Dienstsitzungen bzw. zum Anschluß an existierende Dienstsitzungen entgegenzunehmen. Diese Anforderung kann sogar von einer anderen Dienstsitzung selbst ausgehen. Darüberhinaus werden im Benutzeragenten alle Signale und Nachrichten, die mit Diensten in Verbindung stehen, empfangen und bearbeitet.

Ein Terminalagent ist ein Objekt, das für die Repräsentation eines Terminals verantwortlich ist, d.h. er ist verantwortlich, die präzise Ortsangabe eines Terminals zu erhalten. Zwei Beispiele dafür bieten die Ermittlung des Zugangspunkts, an dem ein portabler Rechner angeschlossen ist, und die Bestimmung, in welcher Zelle sich ein mobiles Telefon aktuell befindet.

Um Zugang zu einem Dienst zu erhalten, müssen Benutzer ihren Benutzeragenten mit einem Terminalagenten verbinden. Das kann innerhalb eines Login-Vorgangs, der einen Dienstzugriff ermöglichen soll, stattfinden. Ein Benutzer kann auch gleichzeitig mit mehreren Terminals verbunden sein, z.B. wenn er in einer Videokonferenz einen Terminal und ein Telefon benutzt. Genauso kann auch ein Terminal gleichzeitig mit mehreren Benutzern verbunden sein, z.B. wenn bei einem Treffen alle Benutzer mit ihren Benutzeragenten an das Telefon im Sitzungsraum angeschlossen sind. Bei ankommenden Sitzungsanforderungen muß der Benutzeragent bestimmen, zu welchem Terminal die Ankündigung weitergeschickt werden soll. Wenn der Benutzer gerade auf das System zugreift, kann der Benutzeragent die Nachricht zu einem der benutzten Terminals senden und der Benutzer wählt dann aus, welches Gerät der Benutzeragent endgültig auswählen soll. Ansonsten muß der Benutzeragent einen Terminal auswählen, dessen Terminalagent den Benutzer dann alarmieren kann. Diese Auswahl des Benutzeragenten kann durch eine Vereinbarung oder Präferenz des Benutzers oder durch eine Fehlerbehandlung gelenkt werden. Ein Benutzer könnte z.B. mit seinem Benutzeragenten vereinbart haben, von Montag bis Freitag jeweils zwischen 9 und 17 Uhr das Bürotelefon auszuwählen, außerhalb dieser Zeit das private Telefon und einen voice mail service, wenn niemand erreichbar ist.

6 Verwaltungsarchitektur

6.1 Computing Management

Das Computing Management umfaßt die Verwaltung der Rechner, Plattformen und der Transportmöglichkeiten, die die verteilte Umgebung darstellen, in der die TINA-Anwendungen arbeiten. Die Verwaltung der Software, die in dieser Umgebung abläuft, gehört aber auch zu dem Bereich des Computing Managements, so daß diese Verwaltungsklasse unterteilt werden kann in (vgl. [dlFW94]):

1. *Softwaremanagement*, bestehend aus Einsatz, Konfiguration, Instanziierung, Aktivierung und Deaktivierung der Anwendungssoftware. Dabei wird hier nur der Softwa-

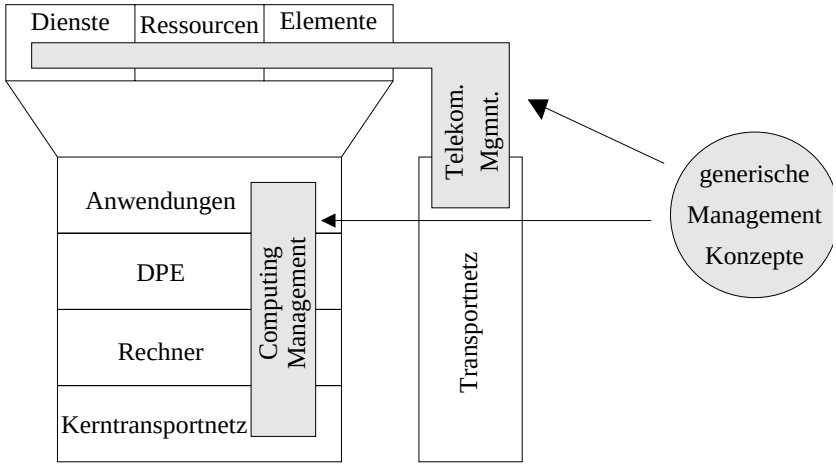

Abbildung14. Zwei Verwaltungstypen

reaspekt der Anwendung berücksichtigt und nicht, was die konkrete Aufgabe der Anwendung ist.

2. *Infrastrukturmanagement*, bestehend aus
   - Management der DPE
   - Management der Rechnerumgebung (sonst auch Rechnersystem genannt)
   - Management des Kerntransportnetzes

Für die Durchführung der Verwaltungsaktivitäten und die Erweiterung der Prinzipien des Computing Management ist die TINA-Softwarearchitektur zuständig.

6.2 Telekommunikationsverwaltung

Die Telekommunikationsverwaltung (telecommunication management) schließt die Verwaltung des Transportnetzes, die Verwaltung der Anwendungen, die dieses Netz benutzen und kontrollieren sowie die Verwaltung der Dienste ein. Es liegt jedoch im Aufgabenbereich der Dienst- und Netzwerkarchitektur, die Verwaltungsaktivitäten anzuwenden und auch die generischen Prinzipien der Verwaltungsarchitektur zu erweitern oder zu spezialisieren. Die generischen Verwaltungsprinzipien bestehen aus der funktionalen Separation in Fehler- (fault), Konfigurations- (configuration), Abrechnungs- (accounting), Durchsatz- (performance) und Sicherheits- (security) Management, wie sie auch schon für OSI Managementsysteme gelten. Diese Teilbereiche müssen in der Dienst-, Netzwerk- und Softwarearchitektur jeweils noch spezialisiert werden. Z.B. sind in der Abrechnungsverwaltung Konzepte für Objekte, für die Rechnungen erstellt werden können, definiert. Diese Konzepte sind jedoch unabhängig von dem Typus des in Rechnung gestellten Objekts. In der Dienstarchitektur wird das generische Abrechnungsmodell noch um Konzepte der Gebühren und Rechnungen erweitert. Die generischen Konzepte können in den einzelnen Architekturen auch unterschiedlich neu untergliedert werden. Z.B. gehört in der Dienstarchitektur zum Konfigurationsmanagement auch noch das Subskriptionsmanagement, das sonst nirgendwo gebraucht wird. In der Netzwerkarchitektur gehört dafür aber die Verbindungsverwaltung zum Konfigurationsmanagement.

7 Zusammenfassung

Die vorgestellte Architektur hat sich zum Ziel gesetzt, wichtige Standards wie TMN, IN, OMG und ODP in ihre Konzepte einzubinden. Das hat dazu geführt, daß die TINA-Architektur keine grundlegend neuen Lösungsansätze hervorgebracht hat, sondern schon vorhandene und für gut befundene Lösungen im Hinblick auf Telekommunikationssysteme ausgebaut und zu einer Architektur vereinigt hat. Diese Architektur muß nun noch vor allem im Bereich der Verwaltung für die Softwarearchitektur und bei anderen Details vervollständigt werden. Es wurden natürlich auch schon Projekte angegangen, um diese Theorie in die Praxis umzusetzen und dadurch zu validieren. Teile dieser Arbeiten konnten bei Demonstrationen für die Mitgliedsfirmen (z.B. eine Demonstration bei der Telecom '95) vorgeführt werden. Diese Forschungsarbeit, die auch von allen großen europäischen Telekommunikationsgesellschaften unterstützt wird, stellt die Voraussetzung für den Beginn des Informationszeitalters auch für die Allgemeinheit dar, da TINA vor allem die Breitbandtechnologie (mit ihren Anwendungen wie Video-on-demand oder Videokonferenz) und die Mobilität im Netz (Mobilfunk und mobile Rechnerknoten) unterstützt. Desweiteren sollen auch noch migration paths von Intelligent Networks und Telecommunication Management Networks zu TINA ausgearbeitet werden , um einen langsamen aber stetigen Übergang zu globalen Telekommunikations Informations Netzen zu erreichen.

Memory Consistency Modelle

Boris Moser

Kurzfassung

Dieser Artikel beschreibt Speicherkonsistenzmodelle und erklärt, weshalb solche Konsistenzmodelle überhaupt notwendig sind. Außerdem werden die Unterschiede der Konsistenzmodelle z. B. in der Performance erläutert. Da Speicherkonsistenzmodelle, die sehr effizient sind, nicht gerade programmiererfreundlich sind, werden noch Transformationsregeln vorgestellt. Sie ermöglichen es, Programme, die einem strengen Konsistenzmodell genügen, und daher leicht zu programmieren sind, auf schwächer konsistenten Systemen effizienter ablaufen zu lassen.

1 Einleitung

Verteilte Systeme (vgl. Abbildung 15) ermöglichen es, eine Gesamtaufgabe zu lösen, indem jeder Rechner des verteilten Systems eine Teilaufgabe löst. Da die Rechenleistung von mehreren Rechnern zur Verfügung steht, kann eine Lösung schneller erreicht werden. Was ist nun bei der Programmierung zu beachten? Ist ein verteiltes System genauso leicht zu programmieren wie ein Einzelplatzrechner?

Da bei verteilten Systemen mehrere Rechner zusammenarbeiten, stellt die Regelung des

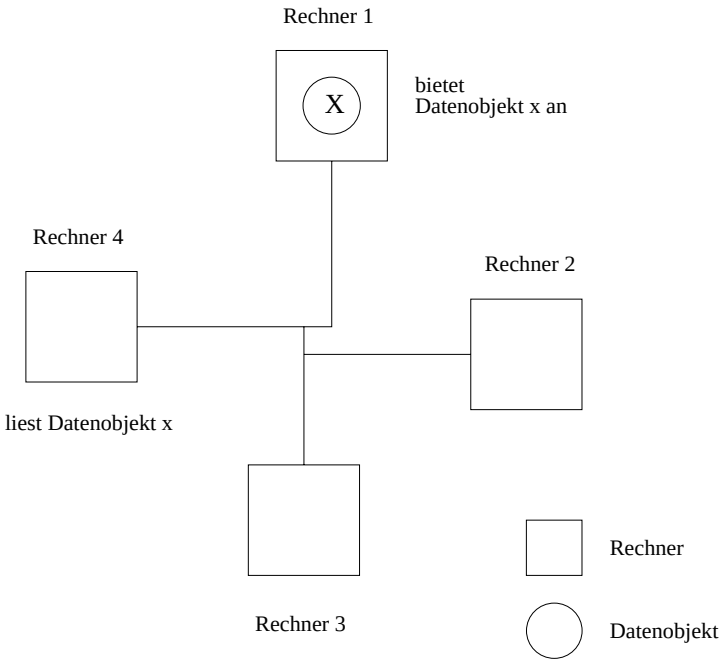

Abbildung15. Verteiltes System

Speicherzugriffs kein einfach zu lösendes Problem dar. Bei Parallelrechnern hat man sich intensiv mit der Lösung des Problems beschäftigt und kann also auf schon entwickelte Modelle zurückgreifen. Ein DSM System (distributed shared memory system) ist ein

MIMD System mit physisch verteiltem Speicher, der global adressierbar ist. Die Kommunikation geschieht über ein Verbindungsnetzwerk. Nach [Tan95] existieren bei DSM Systemen konventionell eine oder mehrere Kopien von Speicherdaten, die nur gelesen werden und nur eine Kopie, die nur geschrieben wird. Wird eine Schreib-Kopie von mehreren Prozessen gemeinsam benutzt, so bildet sie einen Flaschenhals.
Ein Erhöhen der Performance kann erreicht werden, wenn mehrere schreibbare Kopien gestattet sind. In Abbildung 16 ist zu sehen, daß Rechner 1 das Datenobjekt x im Speicher hält und die Rechner 2-4 Kopien des Datenobjektes x in ihren lokalen Speichern haben. Das ist natürlich von Vorteil, weil der Zugriff auf einen lokalen Speicher schneller erfolgt als auf einen entfernten Speicher eines anderen Rechners. Nur greift Rechner 2 schreibend auf die Kopie zu. Woher wissen nun Rechner 3 und 4, daß sich eine Kopie und damit x geändert hat und ihre Kopien ungültig sind?
Man kann mit mehreren Kopien zwar die Performance verbessern, aber es taucht eine neue Frage auf. Wie können die Kopien konsistent gehalten werden? Das Konsistenzproblem ist vor allem sehr zeitintensiv, wenn die Kopien im Speicher verschiedener Rechner sind, weil der Nachrichtenaustausch bei einem Rechnernetz viel langsamer ist als bei einem lokalen Speicherzugriff. Folglich läßt die Performance nach.
Da Daten nicht immer konsistent sein müssen, hat man sich entschlossen, keine perfekte Konsistenz zu gewährleisten, sondern auf schwächere Konsistenz auszuweichen, um die Performance zu verbessern.
Nun noch einige Definitionen, die für spätere Argumentationen benötigt werden.

Definition 1 *Man spricht von einer sequentiellen Ausführung, wenn ein Programm in der durch das Programm gegebenen Reihenfolge der Befehle ausgeführt wird.*

Definition 2 *Man spricht von einer Ausführung außerhalb der Programmreihenfolge, wenn ein Programm nicht sequentiell ausgeführt wird.*

Definition 3 *Sprungvorhersage (branch predection) bedeutet, daß das Sprungziel eines Sprungbefehls aus Erfahrungswerten bestimmt wird.*

Definition 4 *Sprungvorhersage mit spekulativer Ausführung bedeutet, daß ein Programm nach einem Sprungbefehl ausgeführt wird, ohne das wirkliche Sprungziel zu kennen.*

Definition 5 *Ein Branch-Befehl ist ein Sprungbefehl in Assemblersprache.*

Schwächere Konsistenzmodelle führen Befehle außerhalb der Programmreihenfolge aus und das in Abschnitt 4 vorgestellte Rahmenwerk verwendet noch die Sprungvorhersage mit spekulativer Ausführung. Jedoch können diese Optimierungen das Speichermodell komplizieren, da der Programmierer, wenn z. B. ein Programm außerhalb der Programmreihenfolge ausgeführt wird, nicht weiß, in welcher Reihenfolge die Befehle abgearbeitet werden.
Ein Konsistenzmodell umfaßt gewisse Richtlinien, die zwischen Software und Speicher eingehalten werden sollten. Hält sich der Programmierer an diese Regeln, so ist der korrekte Programmablauf gewährleistet. Hält sich der Programmierer nicht an diese Regeln, so kann der korrekte Programmablauf nicht gewährleistet werden.

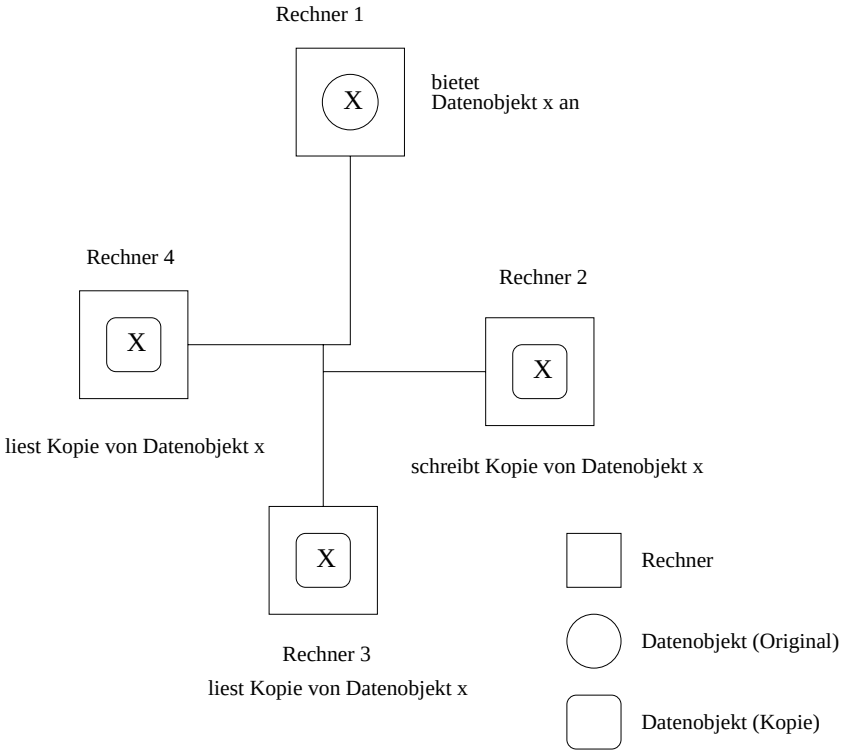

Abbildung16. Verteiltes System mit Kopien

2 Probleme

Bei DSM Systemen können im Prinzip zu jedem Zeitpunkt mehrere Zugriffe auf denselben Speicher gleichzeitig erfolgen.

Definition 6 (Data-Race) *Ein Data-Race entsteht, wenn zwei oder mehr Speicherzugriffe die gleiche Speicherstelle betreffen und die Speicherzugriffe in einer anderen Ordnung erscheinen als sie initiiert worden sind.*

Insbesondere ist ein Data-Race dann verheerend, wenn mindestens ein Speicherzugriff ein Schreibzugriff ist und keine Synchronisation unter den Zugriffen erfolgt. Ein Data-Race führt dazu, daß bei gemeinsamem Zugriff auf eine Variable nicht eindeutig gesagt werden kann, welcher Wert gelesen wird. Der Nachweis, ob ein Programm data-race-free ist, ist NP-vollständig.

Definition 7 *Ein Programm ist data-race-free, wenn keine der sequentiellen Ausführungen einen Data-Race enthält. [ACFW93]*

In Abbildung 15, ist ein verteiltes System mit vier Rechnern zu sehen, wobei Rechner 1 ein von den Rechnern 2-4 benötigtes Datenobjekt x besitzt. Greifen nun die Rechner 2-4 lesend auf x zu, dann ist zwar ein Data-Race entstanden, der aber keine verheerenden Auswirkungen mit sich bringt. Greift jetzt Rechner 2 schreibend auf Datenobjekt x zu, dann taucht ein Problem auf, weil nicht bekannt ist, welchen Wert Rechner 3 und Rechner 4 lesen – den alten Wert von Datenobjekt x oder den neuen?
Im folgenden bedeutet $R(x)$ ein Lesen und $W(x)$ ein Schreiben der Speicherstelle x.

Beispielsweise ist $R(x)1$ ein Lesen der Speicherstelle x mit dem Wert 1 und $W(x)0$ ein Schreiben von 0 in x.

Abbildung 17 zeigt ein Beispiel für einen Data-Race. Prozeß 1 greift schreibend und

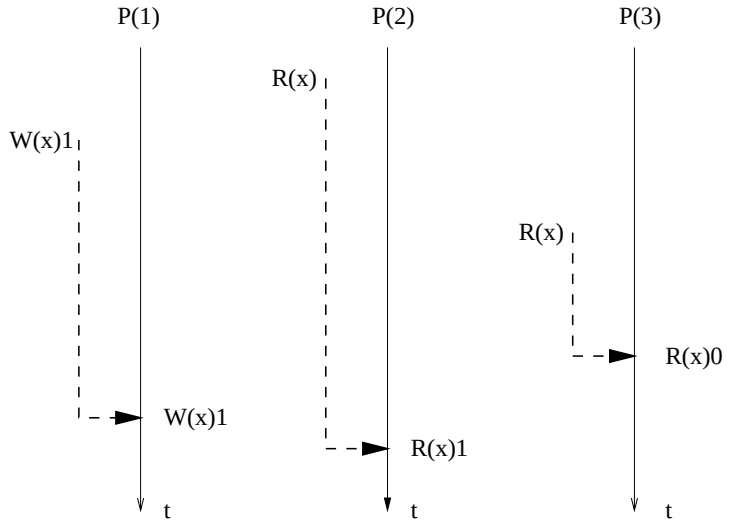

Abbildung17. Data-Race

Prozesse 2 und 3 greifen lesend auf Datenobjekt x zu. Veranschaulicht wird die zeitliche Abfolge der Operationen mit Hilfe eines Zeitstrahls. Man kann damit erkennen, wann eine Operation initiiert wurde, und wann auf das Datenobjekt zugegriffen wird. Die Operationen auf der linken Seite des Zeitstrahl, sind die initiierten und die auf der rechten Seite sind die ausgeführten. Betrachten wir das Beispiel nun genauer:
Die Leseoperation von Prozeß 2 müßte eigentlich den alten Wert von Datenobjekt x liefern, da die Leseoperation vor der Schreiboperation von Prozeß 1 initiiert worden ist. Jedoch greift die Leseoperation erst nach der Schreiboperation auf das Datenobjekt x zu und liefert daher den neuen Wert von x. Ähnlich sieht es mit der Leseoperation von Prozeß 3 aus. Die Leseoperation wird nach der Schreiboperation von Prozeß 1 initiiert, aber greift vorher auf x zu und lierfert daher den alten Wert von Datenobjekt x anstatt des neuen Wertes.
Der Speicherzugriff geschieht in der Regel über ein meist mehrstufiges Verbindungsnetz und damit mit einer durch das Netz gegebenen Latenzzeit, die je nach Zugriffspfad erheblich variieren kann. Zugriffe, die durchaus in der richtigen Reihenfolge initiiert worden sind, können sich überholen, wodurch die vorgeschriebene Ordnung der Speicherzugriffe und damit die Semantik der Programmausführung verändert wird.

3 Konsistenzmodelle

Wie bereits erwähnt wurde, geben Speicherkonsistenzmodelle an, was der Programmierer zu beachten hat, damit die Speicherzugriffe korrekt ausgeführt werden. Nach [Mos93] haben Simulationen gezeigt, daß schwächere Konsistenzmodelle die Performance um 10 bis 40 Prozent verbessern können. Nicht zu vernachlässigen ist außerdem das Problem

der Netzbelastung. Verwendet man ein strenges Konsistenzmodell, das oft Konsistenz gewährleisten muß, so müssen häufiger Speicherkopien konsistent gehalten werden als bei einem schwächeren Konsistenzmodell, bei dem nur an wenigen Synchronisationspunkten Konsistenz gewährleistet wird. [Tan95]

3.1 Strikte Konsistenz (strict consistency)

Das strengste Konsistenzmodell wird strikte Konsistenz genannt und ist so definiert: Ein Lesezugriff auf eine Speicherstelle x liefert den Wert, den der zeitlich letzte Schreibzugriff in Speicherstelle x geschrieben hat.
Dieses Konsistenzmodell ist natürlich dasjenige, das sich ein Programmierer wünscht. Voraussetzung für dieses Konsistenzmodell, das sagen die Worte „der zeitlich letzte Schreibzugriff“, ist die Existenz von globaler, absoluter Zeit. Daher ist es möglich, daß sich – je nach Entfernung der Rechner und je nach Zeitunterschied zwischen Lese- und Schreibzugriff – ein Signal mit mehrfacher Lichtgeschwindigkeit ausbreiten müßte und daher gegen physikalische Gesetze verstößt.
Zur Verdeutlichung dieser Argumentation diene Abbildung 18. Es sind zwei Rechner zu sehen, die mit einem gewissen Abstand zueinander aufgestellt sind. Ereignisse, die auf den Rechnern stattfinden, können sich höchstens mit Lichtgeschwindigkeit ausbreiten. Dies ist durch von den Rechnern ausgesandte Lichtstrahlen dargestellt. Da sie sich erst im Punkt T treffen, können Ereignisse, die zum Zeitpunkt 0 stattfinden, frühestens zum Zeitpunkt T geordnet werden. Ein zum Zeitpunkt 0 auf Rechner 1 geschriebener Wert kann zudem frühestens zum Zeitpunkt $2T$ auf Rechner 2 gelesen werden. Nicht jedoch wie gefordert zum Zeitpunkt ε mit $0 < \varepsilon < 2T$. Da strikte Konsistenz aus diesem Grund nicht einfach zu realisieren ist, wird sie im folgenden nicht näher betrachtet.

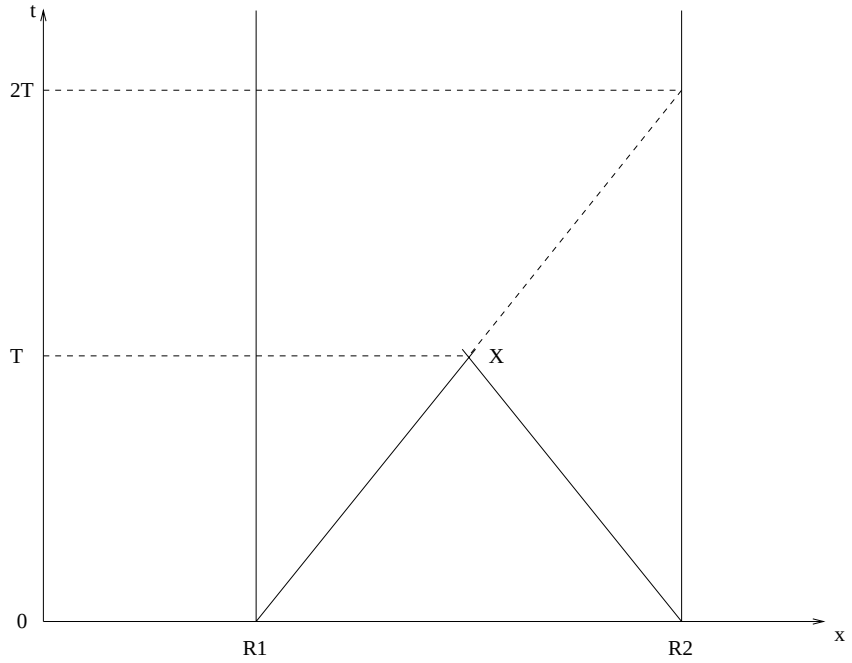

Abbildung18. Zwei Ereignisse

3.2 Sequentielle Konsistenz (sequential consistency)

Man spricht von sequentieller Konsistenz, wenn das Ergebnis einer beliebigen Berechnung dasselbe ist, als wenn die Operationen aller Prozessoren auf einem Uniprozessor in einer sequentiellen Ordnung ausgeführt würden. Dabei ist die Ordnung der Operationen der Prozessoren die des jeweiligen Programms.
Aufgrund dieser Definition stellt die sequentielle Konsistenz die korrekte Ordnung der Speicherzugriffe innerhalb eines Konstrollfadens sicher. Jedoch kann man sich nicht auf eine bestimmte Ordnung von Zugriffen auf gemeinsam benutzte Datenobjekte durch parallele Kontrollfäden verlassen. Für diese Ordnung muß der Programmierer mit geeigneter Synchronisation des Datenzugriffs sorgen (z. B. Definition kritischer Bereiche).

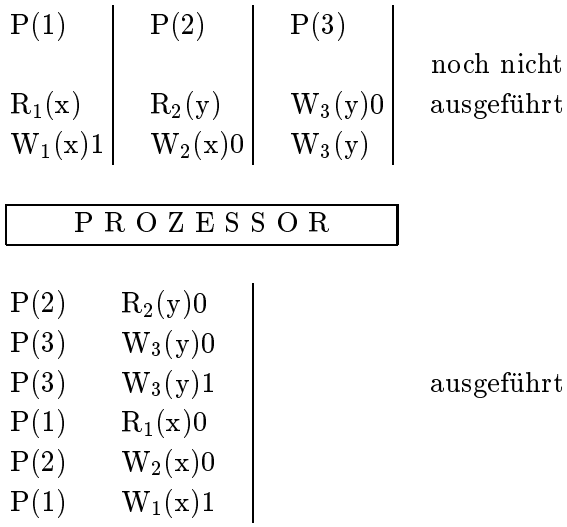

Abbildung19. Beispiel

Abbildung 19 veranschaulicht die Definition der sequentiellen Konsistenz. Gezeigt wird eine Verschachtelung der Operationen von drei Prozessen und wie sie von einem Prozessor sequentiell ausgeführt werden könnten. Ein System, das sequentiell konsistent ist kann durchaus Operationen in mehreren Reihenfolgen ausführen. Abbildung 20 zeigt zwei mögliche Beispiele. Im Beispiel a) ist zu sehen, daß der erste Lesebefehl von Prozeß 2 vor dem Schreibbefehl von Prozeß 1 ausgeführt wird. Und im Beispiel b) werden beide Lesebefehle nach dem Schreibbefehl ausgeführt. Eine Abarbeitung der Befehle, so daß der erste Lesezugriff eine 1 und der zweite Lesezugriff eine 0 liefert, kann nicht vorkommen, da jeder Prozeß sequentiell ausgeführt wird. Abbildung 21 zeigt die Ordnung der Zugriffe bei der Ausführung.
Für Programmierer ist dieses Konsistenzmodell natürlich sehr angenehm, weil die möglichen Ausführungen von Operationen noch überschaubar sind. Jedoch ist der Kommunikationsaufwand erheblich, da jeder Prozeß die gleiche Sequenz der Speicherzugriffe sehen muß. Bei einem Rechnernetz mit vielen Rechnern und einem großen Netzdelay ist das ein gravierender Faktor, der für die schlechte Performance verantwortlich ist.

a)

P(1):	W(x)1	
P(2):	R(x)0	R(x)1

b)

P(1):	W(x)1	
P(2):	R(x)1	R(x)1

Abbildung20. Beispiel

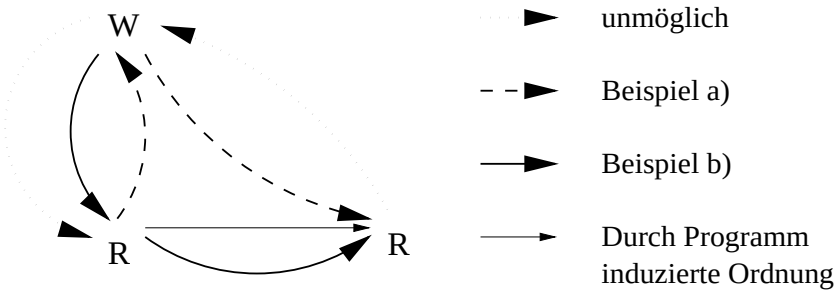

Abbildung21. Ordnung der Zugriffe

3.3 Schwache Konsistenz (weak consistency)

Weiß der Programmierer, daß in gewissen Phasen der Programmausführung eine Verletzung der Zugriffsordnung toleriert werden kann, so kann er die Forderung nach einer sequentiellen Konsistenz aufgeben und sich statt dessen mit einer schwachen Konsistenz begnügen, um die Effizienz der Programmausführung zu steigern. Schwache Konsistenz heißt, daß die Konsistenz des Speicherzugriffs nicht mehr zu allen Zeiten gewährleistet ist, sondern nur zu bestimmten vom Programmierer in das Programm eingesetzten Synchronisationspunkten.

Definition 8 *Ein Lesezugriff durch Prozeß P_i heißt zu einem bestimmten Zeitpunkt bezüglich Prozeß P_k ausgeführt, wenn ein Schreibzugriff durch P_k den Wert, den P_i durch den Lesezugriff auf dieselbe Adresse erhält, nicht mehr beeinflussen kann.*

Definition 9 *Ein Schreibzugriff durch Prozeß P_i heißt zu einem bestimmten Zeitpunkt bezüglich Prozeß P_k ausgeführt, wenn ein Lesezugriff durch P_k den Wert, der von P_i auf dieselbe Adresse geschrieben worden ist, erhält.*

Für eine schwache Konsistenz müssen nur noch die folgenden drei Bedingungen erfüllt sein:

- Bevor ein Schreib- oder Lesezugriff bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Synchronisationspunkte erreicht worden sein.
- Bevor ein Synchronisationspunkt bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreib- oder Lesezugriffe ausgeführt worden sein.
- Die Synchronisationspunkte müssen sequentiell konsistent sein.

Der Preis für die durch schwache Konsistenz erzielbare Leistung besteht darin, daß der Programmierer sich jetzt nicht mehr auf eine vom System gewährleistete sequentielle Ordnung der Speicherzugriffe verlassen kann, sondern selbst dafür verantwortlich ist, durch richtige Wahl der Synchronisationspunkte für die Einhaltung der Programmsemantik zu sorgen.
Der Kommunikationsaufwand ist bei schwacher Konsistenz nicht so groß wie bei sequentieller Konsistenz, da erst Konsistenz gewährleistet wird, wenn ein Synchronisationspunkt ausgeführt wurde. Außerdem sehen alle Prozesse nur noch die Zugriffe auf die Synchronisationsvariablen in der gleichen Reihenfolge.

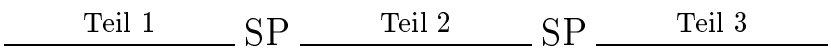

Abbildung22. Prozeß mit Synchronisationspunkten

P(1):	W(x)1	W(x)2	SP
P(2):	R(x)1	R(x)2	SP
P(3):	R(x)2	R(x)1	SP

Abbildung23. Beispiel

Abbildung 22 zeigt einen Prozeß, der zwei Synchronisationspunkte enthält. Die Synchronisationspunkte garantieren, daß alle vorhergehenden Operationen ausgeführt wurden. In welcher Reihenfolgen die Befehle zwischen zwei Synchronisationspunkten ausgeführt werden, ist nicht eindeutig. Sie können sich durchaus überholen.
Im Beispiel in Abbildung 23 haben sich die Prozesse 2 und 3 noch nicht synchronisiert. Es ist daher möglich, daß die Prozesse 2 und 3 die Schreibzugriffe von Prozeß 1 in umgekehrte Reihenfolge sehen. Erst nachdem die jeweiligen Synchronisationspunkte erreicht sind, ist sichergestellt, daß die Lesezugriffe die gleichen Werte liefern. Abbildung 24 zeigt die Ordnung der Befehle in diesem Beispiel. Im Unterschied zur Sequentiellen Konsistenz ist jetzt ein Zyklus in der Ordnung der Befehle möglich.

3.4 Release Konsistenz (release consistency)

Dieses Konsistenzmodell beruht auf der Unterscheidung zwischen synchronisierenden und nichtsynchronisierenden Speicherzugriffen. Synchronisierende Zugriffe sind solche, bei denen durch das Verriegeln oder Entriegeln eines Speicherbereichs eine bestimmte Zugriffsordnung hergestellt werden kann. Nichtsynchronisierende Zugriffe sind solche, bei denen es durch die Natur des Algorithmus gleichgültig ist, wer bei konkurrierenden Zugriffen das Rennen macht, d. h. in welcher Reihenfolge die Zugriffe wirklich stattfinden. Innerhalb der synchronisierenden Zugriffe wird zwischen sperrenden und freigebenden Zugriffen (acquire und release) unterschieden.
Für die Release Konsistenz gelten die folgenden Bedingungen: [Gil93]

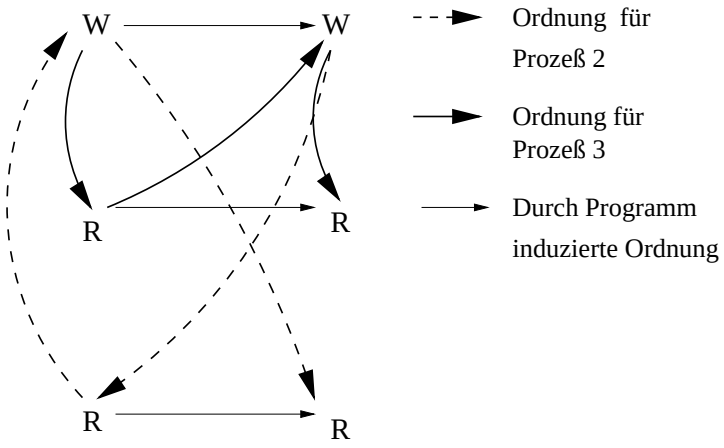

Abbildung24. Ordnung der Zugriffe

- Bevor ein Schreib- oder Lesezugriff auf ein gemeinsames Datenobjekt bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden sperrenden Zugriffe (acquire) ausgeführt worden sein.
- Bevor ein freigebender Zugriff (release) bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreib- oder Lesezugriffe auf ein gemeinsames Datenobjekt ausgeführt worden sein.

Ähnlich wie die schwache Konsistenz ersetzt die Release Konsistenz die vollständige Ordnung der Lese- und Schreibzugriffe durch die wesentlich schwächere Ordnung von Synchronisationspunkten und bürdet damit dem Programmierer die Aufgabe auf, durch richtiges Einsetzen der Synschronisationspunkte für einen korrekten Programmablauf zu sorgen. Diese zusätzliche Mühe des Programmierers bringt aber einen Leistungsgewinn von 10 bis 40 Prozent [Mos93].

Abbildung25. Prozeß mit acquire und release Zugriffen

P(1):	Acq(L)	W(x)1	W(x)2	Rel(L)
P(2):		Acq(L)	R(x)2	Rel(L)
P(3):	R(x)1			

Abbildung26. Beispiel

Abbildung 25 zeigt einen Prozeß, der aus drei Teilen besteht und der zweite durch acquire und release Zugriffe umschlossen ist. Die Definition der Release Konsistenz sagt, daß alle einem acquire nachfolgenden Zugriffe erst nach diesem ausgeführt werden dürfen.

Zusätzlich darf ein release erst ausgeführt werden, wenn alle vorhergehenden Operationen ausgeführt sind. Innerhalb der acquire und der release Zugriffe (Teil 2) können sich die Operationen auch überholen. Zu beachten ist, daß die Befehle aus Teil 3 in Teil 2 vorgezogen und die Befehle aus Teil 1 bis Teil 2 verzögert werden können.
In Abbildung 26 sehen wir eine mögliche Sequenz von Speicherzugriffen. Es ist garantiert, daß der Lesebefehl von Prozeß 2 eine 2 liefert, wenn der acquire Zugriff nach dem acquire Zugriff des Prozesses 1 initiiert wird. Auch dann, wenn der acquire von Prozeß 2 vor dem release von Prozeß 1 erfolgt. Dann wird der acquire nämlich so lange verzögert bis der release von Prozeß 1 ausgeführt wurde. Wann Speicherzugriffe von Prozessen ausgeführt werden, die nicht durch ein acquire und ein release umschlossen sind, ist nicht eindeutig. In diesem Beispiel wurde der Lesezugriff von Prozeß 3 nach dem ersten und vor dem zweiten Schreibzugriff des ersten Prozesses ausgeführt.
Abschließend sei noch bemerkt, daß ein acquire alle lokalen Kopien aktualisiert und ein release die geänderten Daten an alle beteiligten Rechner schickt. Im Vergleich zur sequentiellen Konsistenz haben wir einen Performancegewinn, da nur bei acquire und release Zugriffen synchronisiert wird.

3.5 Hybrid-Konsistenz (hybrid consistency)

Bei Hybrid-Konsistenz (siehe [Fri94]) gibt es zwei Arten von Operationen: strenge (strong) und schwache (weak) Speicherzugriffe. Auf den strengen Operationen besteht eine globale Totalordnung, d. h. jeder Prozeß sieht die gleiche Ordnung der strengen Operationen. Die Hybrid-Konsistenz sagt jedoch weder etwas über die Ordnung der schwachen Operationen aus noch wie sie mit den strengen verschachtelt sind.
Der Kommunikationsaufwand hängt bei diesem Konsistenzmodell davon ab, wieviele Operationen als streng markiert sind. Sind im Verhälnis sehr viele strenge Operationen vorhanden, so ist der Kommunikationsaufwand groß, da ja jeder Prozeß diese Ordnung sehen muß.

a)

P(1):	SW(x)1		
P(2):	SR(x)1	WR(x)1	WR(x)0

b)

P(1):	SW(x)1		
P(2):	SR(x)0	WR(x)1	WR(x)0

Abbildung27. Beispiel

In den beiden Beispielen in Abbildung 27 wird zwischen strengen und schwachen Operationen unterschieden. $SW(x)$ bedeutet, daß ein Schreibzugriff als streng markiert und $WR(x)$ bedeutet, daß ein Lesezugriff als schwach markiert ist.
Die Beispiele zeigen zwei mögliche Ausführungen, die hybrid konsistent sind. Auffallend ist, daß in beiden Beispielen der zweite Lesezugriff eine 0 liefert. Das kann aber durchaus sein, da nur die als streng markierten Operationen sequentiell konsistent sein müssen. Schwache Operationen können auch vorgezogen, verzögert und außerhalb der Programmreihenfolge ausgeführt werden. Abbildung 28 zeigt die Ordnung der Zugriffe während der

Ausführung. Innerhalb des zweiten Prozesses ist keine Ordnung, da unter den schwachen Operationen keine Ordnung existiert.

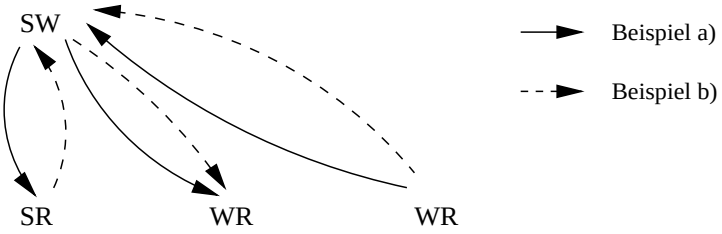

Abbildung28. Ordnung der Zugriffe

3.6 Vergleich der Konsistenzmodelle

Zu sehen ist in Abbildung 29, daß je schwächer das Konsistenzmodell ist desto größer ist die Performance. Aber desto komplexer ist auch das Speichermodell. Die Ordnung von Weak Consiatency und Release Consistency ist [Tan95] entnommen. Eine andere Ordnung wäre denkbar. Wenn nämlich ein schwach konsistentes Programm weniger Synchronisationspunkte hat als ein release konsistentes, dann läuft ein schwach konsistenten Programm effizienter ab als ein release konsistentes. Diese Abbildung ist nur als Skizze zu verstehen. Der lineare Verlauf ist willkürlich gewählt.

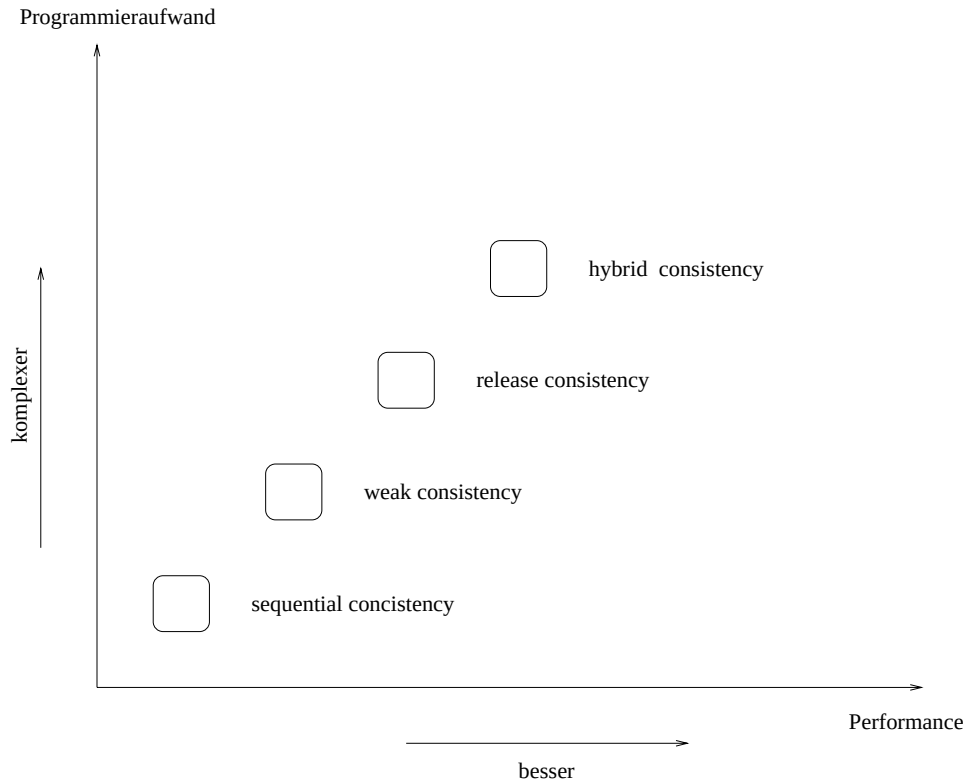

Abbildung29. Vergleich der Konsistenzmodelle

4 Transformationen

Schwache Konsistenzmodelle bieten eine bessere Performance als strenge. Unglücklicherweise ist es wesentlich schwieriger, ein Programm für einen hybrid konsistenten Speicher zu schreiben als für einen sequentiell konsistenten. Da wäre es doch schön, wenn es geeignete Transformationsregeln gäbe.
[ACFW93] haben sich diesem Problem gewidmet. Aufbauend auf ihren Überlegungen

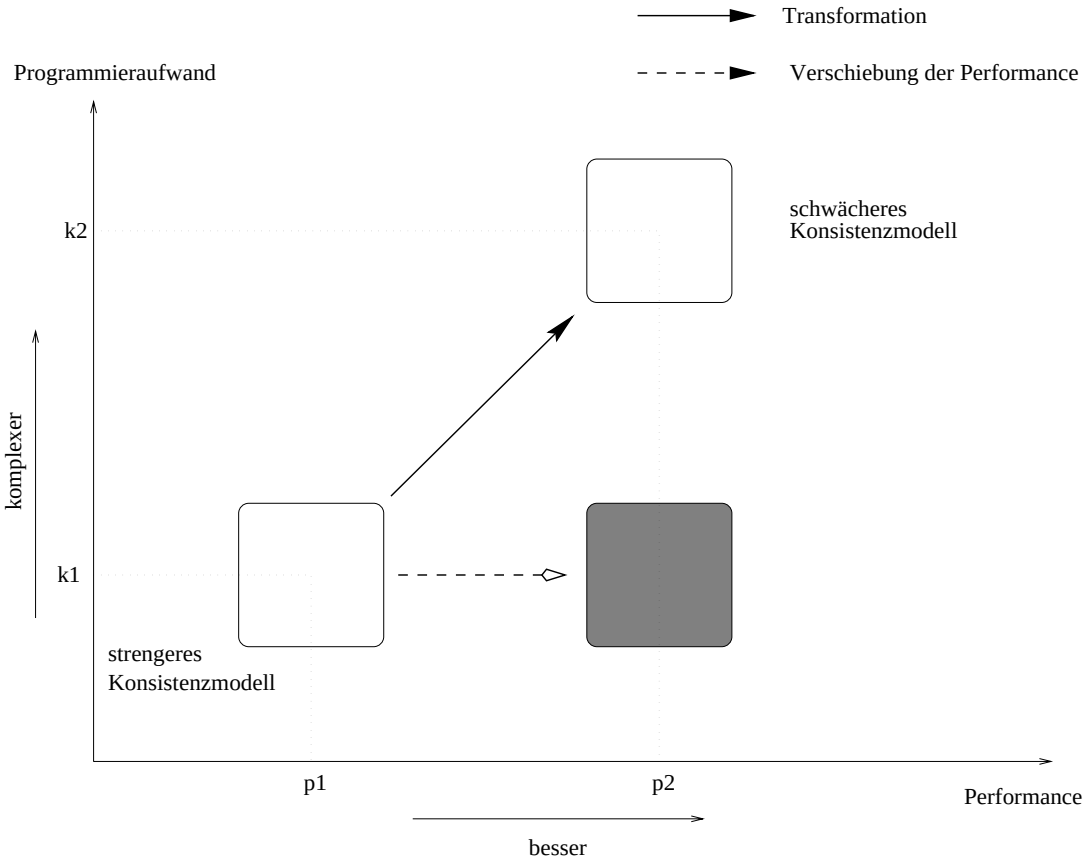

Abbildung30. Transformation

und den Transformationsregeln (siehe weiter unten) ist es möglich, ein Programm für ein strenger konsistentes System zu entwickeln, und es auf einem schwächer konsistenten System ablaufen zu lassen. Für einen Programmierer hat das den Vorteil, daß er mit einem weniger komplexen Speichermodell arbeitet. Durch die Transformationen verbessert sich aber die Performance bei gleichbleibender Komplexität des Speichermodells. In Abbildung 30 sehen wir die Vorgehensweise einer Transformation. Als erstes wird mit der Komplexität k_1 und der Performance p_1 programmiert. Danach ermöglicht die Transformation eine Performance nach p_2 und einen daraus resultierenden Performancegewinn von $p_2 - p_1$ bei gleichbleibender Komplexität von k_1 anstatt k_2.
[ACFW93] stellen in ihrer Arbeit ein Rahmenwerk vor, das die Grundlage für formale Definitionen von Speicherkonsistenzbedingungen nicht-sequentieller Speicherzugriffe bildet (sequentielle Konsistenz, schwache Konsistenz, Hybrid-Konsistenz und Release Konsistenz).

Ein Interface, das die Bedingungen umsetzt, befindet sich zwischen dem Programm und dem System und ist architekturunabhängig. Das Rahmenwerk bildet ein System, das

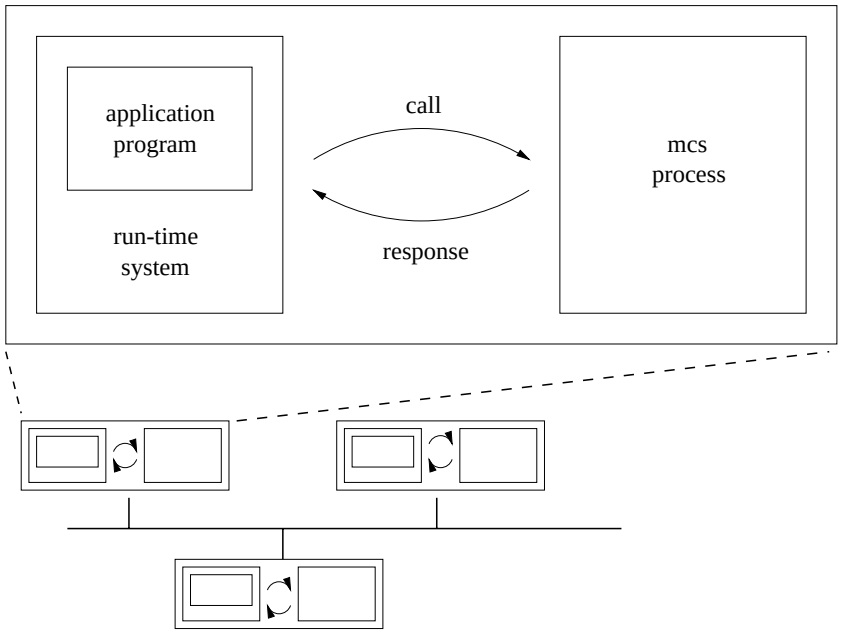

Abbildung31. Ein Node

aus einer Ansammlung von Nodes besteht. Jeder Node besteht aus einem Application Program, einem Memory Consistent System (MCS) Process und einem Run-Time System (vgl. Abbildung 31). Ein Application Program besteht aus Speicherzugriffsbefehlen und bedingten Verzweigungsbefehlen. Der MCS Process implementiert die gemeinsamen Datenobjekte, auf die das Application Program zugreifen kann. Das Run-Time System führt die Speicherzugriffe aus, indem es mit dem lokalen MCS Process kommuniziert. Außerdem kann das Run-Time System Befehle out of order ausführen und trifft Sprungvorhersagen die auch spekulativ ausgeführt werden. Ein Speicherzugriff des Run-Time System besteht aus zwei Teilen: einem Call zum MCS und einem Response vom MCS. Ein Event ist ein Call, ein Response oder ein Sprungbefehl. Ein Run ist eine Abfolge von Events, so daß auf jeden Call sein korrespondierender Response folgt.
Das Rahmenwerk bildet die theoretische Grundlage für die Beweise der folgenden Sätze, und damit für die Anwendbarkeit des Transformationsgedankens.

Satz 1 *Jeder hybrid konsistente Run eines Programms, in dem alle Schreibzugriffe als streng und alle Lesezugriffe als schwach markiert sind, ist sequentiell konsistent.*

Ein einfacher Weg, Satz 1 zu benutzen, ist bei einem sequentiell konsistenten Programm, jeden Schreibzugriff als streng und jeden Lesezugriff als schwach zu markieren. Dann läuft es auch auf einem hybrid konsistenten System.
Um eine noch effizientere Transformation mit Satz 1 durchzuführen, ist es sinnvoller, alle Schreibzugriffe in einem kritischen Abschnitt als streng und alle anderen Operationen als schwach zu markieren.

Satz 2 *Jeder hybrid konsistente Run eines Programms, in dem alle Schreibzugriffe als*

schwach und alle Lesezugriffe als streng markiert sind, ist sequentiell konsistent.

Satz 3 *Jeder hybrid konsistente Run eines Programms, das data-race-free ist, ist sequentiell konsistent.*

Satz 3 ist besonders hilfreich, da es einem Programmierer verdeutlicht, daß jedes Programm, das data-race-free ist, sich auf einem hybrid konsistenten Speicher verhält wie ein sequentiell konsistentes.

5 Nachwort

In diesem Artikel wurden Speicherkonsistenzmodelle vorgestellt und gezeigt, daß strenge Konsistenzmodelle leichter zu programmieren sind, jedoch nicht so effizient sind wie schwache Konsistenzmodelle. In Abschnitt 4 haben wir dann gesehen, daß es mit bestimmten Transformationsregeln möglich ist, ein sequentiell konsistentes Programm zu entwickeln, und es mit verbesserter Performance auf einem hybrid konsistenten System ablaufen zu lassen. Nicht unerwähnt soll jedoch bleiben, daß bei dem sehr theoretischen Rahmenwerk nicht immer klar ist, wie eine dort definierte Konsistenzbedingung zu implementieren ist.

Web-Engineering

Oliver Reiniger

Kurzfassung

Auf Grund der Kommerzialisierung des Internet und der sich daraus ergebenden Anwendung der Web-Techniken in Unternehmen, wird es notwendig die Entwicklung von Web Anwendungen aus der chaotischen Phase der Anfangszeit in einen definierten und kontrollierbaren Vorgang zu verwandeln. Hierzu dienen sowohl die klassischen Methoden der Softwaretechnik, die an die speziellen Anforderungen des Web angepasst werden müssen, als auch die Verwendung neuer Methoden und Werkzeuge.
Die Anforderungen, die an Web-Applikationen und ihre Entwicklungswerkzeuge gestellt werden, sowie derzeit übliche Techniken und Werkzeuge sind Gegenstand dieses Beitrags. Darüber hinaus soll auf neuere Entwicklungen und deren Konsequenzen eingegangen werden.

1 Motivation

Nachdem das Internet jahrelang als eine Spielwiese von ein paar eingefleischten Fans galt, hat sich die Kommerzialisierung immer mehr durchgesetzt. So sieht ORACLE z.B. eine Entwicklung "vom Modewort zur Schlüsseltechnologie" [MM96]
Ein weiterer Aspekt ist, daß Internetanwendungen noch vor wenigen Jahren fast ausschließlich für UNIX-Plattformen verfügbar waren. Mittlerweile haben aber auch die PC's Ihren Internetanschluß gefunden. Nicht zuletzt deswegen, weil die Marktführer IBM und Microsoft diesen expandierenden Markt entdeckt haben.
Interessanterweise sehen viele der renomierten Anbieter von Entwicklungswerkzeugen die große Entwicklungsmöglichkeit der Web-Technologie nicht im weltumspannenden Internet sondern im Mikrokosmos des Intranet - dem (firmen-)internen Netzwerk. So meint ORACLE: "Zwar liefern Internet und World Wide Web die spektakulärsten Schlagzeilen, aber die eigentliche Dynamik dieser Technologie geht zukünftig vom Intranet aus [...]"[MM96]. Diese Annahme wird durch die Tatsache gestützt, daß das WWW ursprünglich für das Intranet des europäischen Kernforschungszentrums CERN entwicklet wurde.

2 Web-Engineering

2.1 Zielsetzung

Schon der Begriff Web-Engineering lehnt sich stark an die Softwaretechnik an. Damit ist klar, daß auch die Inhalte, die hinter diesem Schlagwort stecken mit den Aufgaben des Software-Engineerings einiges gemeinsam haben. D.h. Gegenstand des Web-Engineering sind Softwareprojekte im Bereich des Web mit der Zielsetzung einer effizienten Implementierung von korrekten, sicheren und leicht wartbaren Applikationen mit günstiger

Ressourcenausnutzung.
Da – wie in Kapitel 1 ausgeführt – die Dienste des Internet immer mehr kommerziell genutzt werden, ist für eine Firma, die im Internet präsent sein möchte, zusätzlich von Bedeutung, wieviele der firmeneigenen Personal- und Rechnerressourcen über welchen Zeitraum für diese Aufgabe benötigt werden. Somit sind auch die Disziplinen Projektplanung und Kostenschätzung Gegenstand des Web-Engineering.
Um diesen Anforderungen gerecht zu werden, bedarf es gewisser Voraussetzungen im Bereich der Entwurfs- und Entwicklungswerkzeuge sowie gewisser Vorausleistungen von Seiten der Dienstprimitive aus der Betriebssystem- und Sprachumgebung. Besondere Aufmerksamkeit muß dabei auch der Verschlüsselung von Daten gewidmet werden.

2.2 Werkzeuge

Waren in den 60er bis hinein in die 70er Jahre im Bereich Softwaretechnik noch die Umsetzung von Problemen in mathematische Algorithmen gefragt, so wurde mit der Entwicklung von immer mächtigeren Softwareprodukten eine Abstrahierung von der Rechnerebene in eine Entwurfsebene nötig. Der Bereich der Datenbanken, als einer der ältesten Softwaredisziplinen, spielte hier mit der Entwicklung von E/R-Diagrammen eine wichtige Vorreiterrolle.
Heute, im Zeitalter der objekorientierten Programmierung, hat sich diese Abstrahierung im Bereich des Systementwurfs (ob jetzt in Notation von Booch oder Rambough sei dahingestellt) noch verstärkt. Der Einsatz von rechnergestützen, visualisierenden Entwurfswerkzeugen - wie z.B. Rational Rose - hat sicher die Qualität von Softwareprodukten positiv beeinflußt.
Deshalb wäre der Einsatz ähnlicher Werkzeuge im Bereich des Web-Engineerings nicht nur wünschens- sondern auch empfehlenswert (vgl. [Gel96]). Auf Grund der extrem ausgeprägten Verweisstruktur der Web-Sites ist ohne eine akribische und detailgetreue Modellierung des Systems eine spätere Wartung nur schwer wenn nicht sogar gänzlich unmöglich. Außerdem wird durch eine Planungsphase, in der das System genau spezifiziert wird, ein späterer "Wildwuchs" durch einfache straight-forward Programmierung vermieden.
Zur Implementierung von Web-Anwendungen sollte eine ganze Reihe von Werkzeugen zur Verfügung stehen. Angefangen von einem HTML-Editor für ein ansprechendes Layout von Web-Seiten, bis hin zur automatischen Generierung des Applikationscodes aus den Modellierungsdaten. Desweiteren sollten diese Entwicklungswerkzeuge einen hohen Komfort bieten. D.h. sie sollten vom eigentlichen Programmtext abheben, hin zu einer Visualisierung desselben. Im Bereich der Datenbankanwendungsentwicklung zeigt PowerBuilder, wie so etwas aussehen könnte.
Ein Problem, das beim Web-Engineering wesentlich massiver Auftritt als in der herkömmlichen Softwareentwicklung, ist die Rasanz, mit der sich im Web Lage und Inhalt von Informationen ändern. Schon bei relativ einfachen HTML-Seiten muß bei entsprechenden externen Verweisen eine Aktualisierung im Wochenrhythmus ins Auge gefaßt werden. In ähnlicher Weise ändern sich die Daten bzw. Anwendungen, die man der Öffentlichkeit zugänglich machen möchte. Außerdem ist die Enwicklung bei den

Web-Browsern und anderen WWW-Werkzeugen noch lange nicht abgeschlossen. Derzeit sind Neuerscheinungen bzw. Überarbeitungen in monatlicher Folge keine Seltenheit und damit der Zwang eigene Web Applikationen anzupassen oder zu überarbeiten durchaus vorhanden. Eine gute Konfiguartionshaltung sowie eine ausgereifte Versionskontrolle (vgl. hierzu [You96]) wird damit unumgänglich.
Eine Integrierung dieser oben genannten Werkzeuge zusammen mit den entsprechenden Umgebungsanbindungen - auf die im nächsten Abschnitt näher eingegangen werden wird - in eine einheitliche Entwicklungsumgebung, in der die Komponenten compiliert und gebunden und die Applikationen getestet werden können, würde die Effektivität noch weiter erhöhen.

2.3 Voraussetzungen

Neben der Entwicklung der oben genannten Entwicklungswerkzeuge müssen auch noch andere Voraussetzungen geschaffen werden, um hochwertige Produkte im Web zu etablieren. Dies fängt beim Vorhandensein einer einheitlichen Benutzschnittstelle für jegliche Art der Webapplikation an und endet bei einer günstigen Auslastung von Client und Server bei der Ausführung einer solchen Web-Anwendung. Ebenso muß eine einfache Einbindung der klassischen Internetdienste wie Mail, News oder FTP möglich sein.
Darüber hinaus verlangt die Vielschichtigkeit von Web-Anwendungen eine Integration von Netz-, Betriebssystems- und Datenbankkonstrukten in eine entsprechende Programmiersprache [You96]. Wobei hier auch konzeptionelle Probleme zu lösen sind. So ist es z.B. ein Diskussionsgegenstand, ob mengenorintierte SQL-Ausdrücke in eine solche Sprache eingebettet werden oder ob man sich an den äquivalenten Konstrukten der objektorientierten Welt orientiert, denen eine Listenform zugrunde liegt.
Ein ebenso wichtiger Aspekt ist das Laufzeitverhalten der mit dieser Sprache entwickelten Anwendung. Die Optimierung desselben muß sich an folgenden Tatsachen orientieren: Ausgehend von einer Client-Server-Architektur muß versucht werden, die Auslastung von Client und Server so zu gestalten, daß ein Großteil der Leistung vom Client erbracht wird, da eine hohe Auslastung des Servers duch eine Vielzahl von Clients angenommen werden kann. Desweiteren sollte ein möglichst geringer Datenaustausch zwischen Client und Server stattfinden, da schon heute Datenübertragungsraten von weniger als 200 Byte/sec im Internet keine Seltenheit mehr sind. Diese Zahl kann sicher jeder bestätigen, der in letzter Zeit versucht hat, sich Informationen oder neue Gerätetreiber über das Internet zu besorgen. Da das Internet immer noch unaufhörlich wächst und die Zahl seine Teilnehmer noch zunehmen wird, ist eine weitere Verringerung der Datenübertragungsraten anzunehmen.
Im Intranet dagegen liegen diese Werte momentan sicher noch um Einiges besser. Allerdings sind sie im Vergleich zum Zugriff auf lokale Daten immer noch nicht konkurrenzfähig. Sollte außerdem das Intranet in dem Maß an Bedeutung gewinnen, wie dies von manchen erwartet wird, ist mit einem Angleich an die Internetwerte zu rechnen. Es gilt also in zwei Richtungen zu optimieren.
Eine der wichtigsten Voraussetzungen um den Web-Anwendungen den Durchbruch zu sichern, liegt in der Integration von Sicherheitsmechanismen. Dabei scheinen sich Me-

chanismen in Form von Application Program Interfaces (API) oder auch Protokollerweiterungen im Bereich der OSI-Schichten anzubieten (vgl. [TB95]). Eine Ansiedlung der Authentisierungs- und Chiffriermechanismen auf dieser relativ niedrigen Ebene würde einen möglichen (Viren-)Angriff auf die Anwendung erschweren sowie ein Abhören sensibler Daten mit an Sicherheit grenzender Wahrscheinlichkeit verhindern.
Zusammenfassend läßt sich sagen, daß das Web-Engineering das Ziel verfolgt,

- korrekte,
- sichere und
- leicht wartbare

Applikationen zu entwickeln, die sich zudem durch eine

- geringe Ressourcenauslastung

auszeichnen.

Zu diesem Zweck müssen dem Anwendungsentwickler *visualisierende* Werkzeuge

- zum rechnergestützen Entwurf
- zur automatischer Generierung des Codes (zumindest teilweise) und
- zur Versionskontrolle und -verwaltung

zur Verfügung gestellt werden.

Diese Werkzeuge lassen sich allerdings nur dann sinnvoll einsetzen, wenn

- eine einheitliche Entwicklungsumgebung und Benutzerschnittstelle zur Verfügung stehen,
- klassische Internetdienste in diese integriert werden und
- eine auf die speziellen Anforderungen des Web zugeschnittene Programmiersprache vorhanden ist.
- Der Kommunikationsaufwand zur Ausführung der Applikationen muß – zumindest für das Iternet – minimiert,
- eine Auslastungsbalance zwischen Client und Server hergestellt und
- Sicherheitsmechanismen auf niedriger z.B. Betriebssystemebene zur Verfügung gestellt werden.

3 Die Technik

3.1 Hypertext Markup Language

Die Hypertext Markup Language (HTML) (dt.: Hypertext Auszeichnungssprache) ist eine Textbeschreibungssprache. D.h. das Aussehen von Dokumenten, die mit ihr Hilfe erstellt wurden, sind stark von der Umgebung abhängig, in der sie betrachtet werden. So sind z.B. die Größe und das Aussehen von Überschriften vom jeweilgen Browser abhängig, da diesen kein Format sondern nur eine Hirarchiestufe zugeordnet wurde.
Ähnlich wie das Hypertext Transfer Protocol (HTTP) ist HTML so alt wie das Web selbst. HTML hat diverse Entwicklungsstufen und Erweiterungen auf seinem Weg vom Schweizer Atomforschungszentrum CERN über das National Center for Supercomputing Applikations (NCSA) hin zu Netscape und Microsoft hinter sich.
Heute liegt HTML in einem offiziellen Standard mit Versionsnummer 3.2 vor. Daneben gibt es mehrere herstellerspezifische Erweiterungen, die sich meist als paarweise inkompatibel erweisen. Zu den Standardbefehlen von HTML gehört neben den Ausdrücken für einfache Formatzuweisungen, Verweisen und eingebundenen Grafiken auch Tabellen, Bedienknöpfe (in begrenzter Anzahl) und klickbare Grafiken. Darüber hinaus steht ein Vorrat an logischen Operationen u.ä. zur Verfügung.
HTML ist derzeit das Standardhilfsmittel um statische Seiten zu erstellen und daran wird sich in absehbarer Zeit nichts ändern, da keine Alternativen in Sicht sind.

3.2 Datenbankanbindungen

Eine der einfachsten Möglichkeiten von einer statischen, reinen HTML-Seite mit fixem Inhalt zu einer interaktiven Seite mit variablem Inhalt zu gelangen, ist eine Anbindung an eine Datenbank, aus der der Benutzer über das WWW Informationen beziehen möchte. Dabei gibt es mehrere Möglichkeiten, auf die ich im folgenden näher eingehen werde.
Die Gemeinsamkeit zwischen diesen Lösungsansätzen ist die Tatsache, daß alle auf relationalen Datenbanken aufbauen. Dies ist deshalb sinnvoll, da alle relationalen Datenbankmanagementsysteme die Abfragesprache SQL gemeinsam haben. Diese wurde von International Standardisation Organisation (ISO) aus Kompatibilitätsgründen standardisisert. Allerdings wurde dieser Standard von den einzelnen Herstellern zu individuellen SQL-Dialekten erweitert, so daß man nur auf eine gemeinsame Schnittmenge von SQL und eine geringe Anzahl von proprietären Befehlen - die sogenannte Open Database Connectivity (ODBC) - zurückgreifen kann (vgl. [Bag96]).

Funktionsprinzipien Die klassische Methode, eine WWW-Anwendung an eine Datenbank anzubinden ist, wie in Abbildung 32 zu sehen, über eine zentrale Schnittstelle, das Common Gateway Interface (CGI). Dabei interagieren der Web-Server und das Informationssystem vollkommen getrennt über das CGI-Programm. Es handelt sich hier um die Kopplung zweier autarker Systeme. Datenbank und Web-Server werden getrennt eingerichtet und gewartet [Bic96].
Möchte der Web-Client Informationen aus der Datenbank abfragen, so wird die entsprechende Anfrage über den Server und das CGI an die Datenbank übermittelt. Dieser

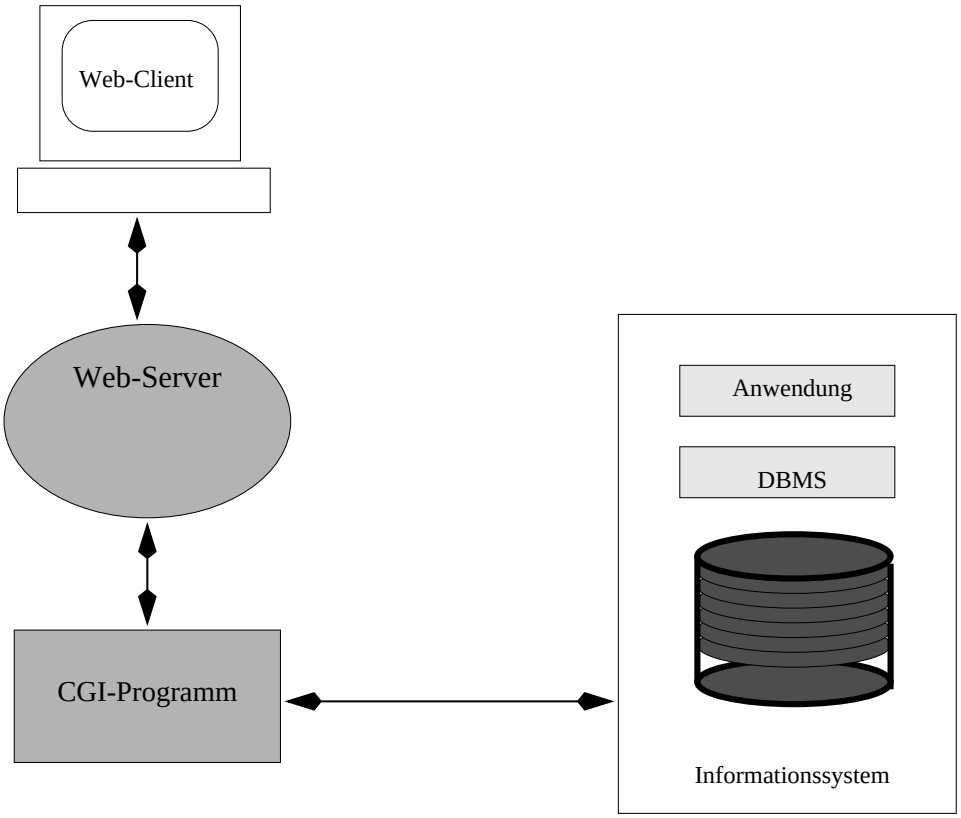

Abbildung32. Datenbankanbindung über Common Gateway Interface (CGI)

bearbeitet die Anfragen, beantwortet sie und leitet sie auf umgekehrten Weg zurück. Eine Bearbeitung der Anfrage findet, abgesehen von einfachen Konvertierungen, weder beim Web-Server noch beim CGI statt.
Diese Interface stammt allerdings aus der Frühzeit des World Wide Web und hat außerdem den Nachteil nicht besonders ineffizient zu sein.
Eine neuere Entwicklung ist die Integration von Web- und Datenbankserver. Als Beispiel sei hier die ORACLE-Web-Anbindug genannt. Ihr Aufbau läßt sich wie folgt skizzieren:
Ein Oracle 7 Datenbankserver wird in den Web-Server integriert. Auf diesem Web-Server setzen sowohl die Web-Entwicklungs und Generierungswerkzeuge als auch die Datenbankentwicklungsprogramme auf [MM96]. Damit ist zwar das lauffähige System integriert, die Entwicklung der Anwednungen läuft aber immer noch zweigleisig.
Eine Anfrage des Web-Client geht als HTTP-Anfrage beim Web-Server ein, der nach einer geeigneten Abfrage der Datenbank die HTML-Seite generiert und dem Client zur Verfügung stellt. Der wesentliche Unterschied zu herkömmlichen CGI-Anbindungen besteht darin, daß das CGI in den Web-Server integriert ist.
Ein konzeptionell neuer Ansatz ist mit Hilfe von Java möglich, auf das in Kapitel 3.3 noch näher eingegangen wird. Wie in Abbildung 34 zu sehen, greift der Client mit Hilfe entsprechender Datenbanktreiber direkt auf die Datenbank zu.
Diese einfache Anbindung hat allerdings den Nachteil, daß sie auf Grund der Sicherheitsdefizite, die Java im derzeitigen Entwicklungsstadium und auch ganz prinzipiell (siehe Kapitel 3.3) hat, so nicht umgesetzt werden kann. Eine etwas kompliziertere aber auch sicherere Lösung (vgl. 35 stellt sich wie folgt dar:

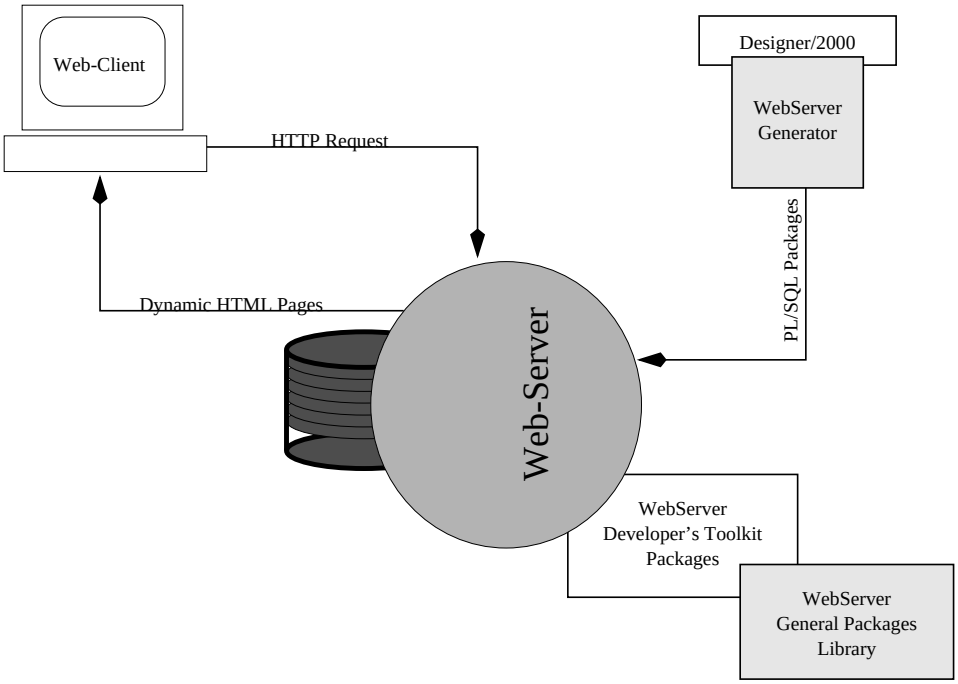

Abbildung33. Oracle Web-Anbindung

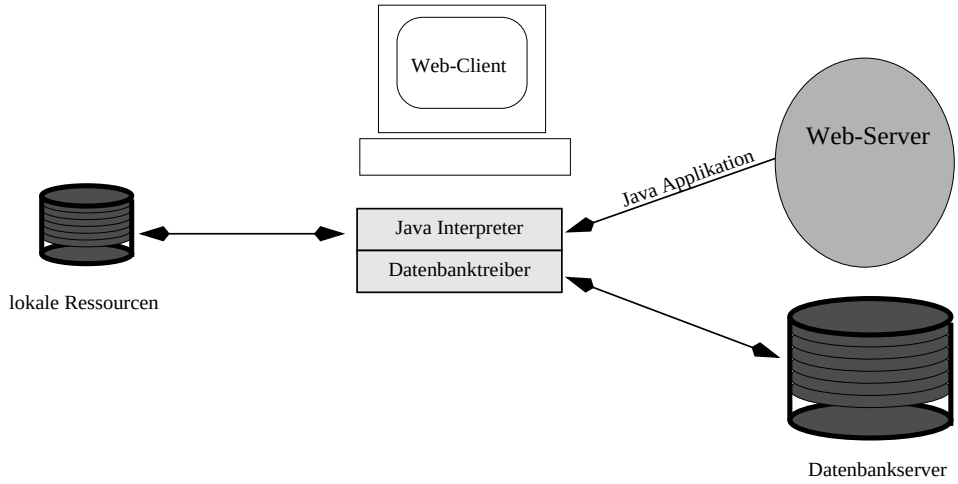

Abbildung34. Direkte Anbindung mit Java

In dieser Konfiguration greift die Java Applikation über Gateway-Server auf die Datenbank und direkt auf HTTP-Server zu. Dies hat wie bei Abbildung 34 den Vorteil, daß die Rechnerleistung für die Aufbereitung der Datenbankanfrage und Datenbankantwort vom Client erbracht wird und der Datenbankserver nur die reine Datenbankanfrage bearbeitet.
Die Gateway-Server übernehmen hier Sicherungaufgaben für die Datenbank.

3.3 Java

Java ist derzeit das Schlagwort, das im Bezug auf das Internet und Netzanwendungen insgesamt durch die Fachzeitschriften die Runde macht. Es handelt sich hier "um das

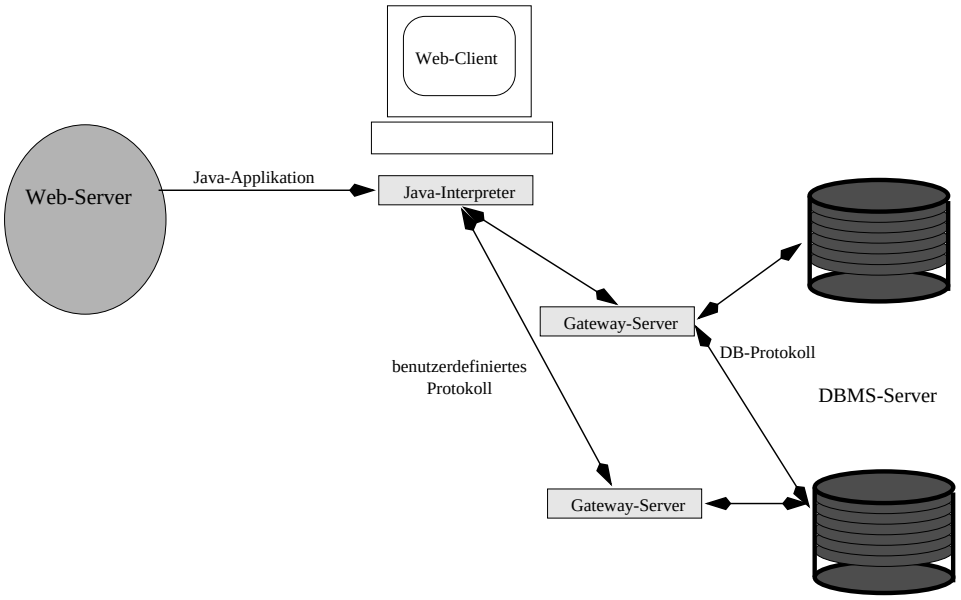

Abbildung35. Datenbank Anbindung mit Java über Gateway

Pardigma, das die EDV-Industrie wie kaum eine andere Softwaretechnologie in Atem hält"[Bac96]. Dabei gehen die Einschätzungen des Durchsetzungsvermögens dieser Programmiersprache, wenn man denn von Java als einer Programmiersprache sprechen kann, weit auseinander. Während die einen von einer reinen Modeerscheinung sprechen, sehen die anderen in Java schon das "Turbopascal der 90er" [Kir96]. Allerdings ist klar, daß "kein Internet-Programmierer Java mehr ingorieren kann"[Bac96] und das nicht erst seitdem praktisch alle führenden Firmen der Software-Branche zu den Lizenznehmern von Java gehören.

Prinzip Der Punkt in dem sich Java von herkömmlichen Programmiersprachen unterscheidet ist, daß in Java geschriebene Anwendungen nicht direkt in Maschinencode übersetzt werden, sondern in einen Byte-Code, der dann mit Hilfe eines Interpreters zur Ausführung gebracht wird, oder aber auch in HTML-Dokumente eingebunden werden kann.
Wie in Abbildung 36 zu sehen, greifen die Interpreter, wie z.B. ein Web-Browser, bei der Ausführung des Java-Byte-Codes auf Java Basisklassen zu, die lokal vorhanden sein müssen. Diese wiederum verwenden ausführbaren Code und können dann auf dem entsprechenden Betriebssystem zur Anwendung gebracht werden.
Die Vorteile dieser Technik liegen klar auf der Hand: Java-Applikationen sind Plattformunabhängig. D.h. eine Java Applikation kann auch auf einem mit einem Java-Chip ausgestatteten Haushaltsgerät zur Ausführung bebracht werden. Der Weg über die Zwischenstufe des Byte-Codes hat aber auch Nachteile. Derzeit erhältliche Java-Interpreter sind bei der Ausführung eine Javaanwendung etwa um den Faktor 33 langsamer als ein vergleichbares C-Programm [Kra96]. Allerdings sollen hier Just-in-Time-Compiler, die demnächst verfügbar sein sollen und den Byte-Code bem Laden des Programms zur Laufzeit in ausführbaren Code übersetzen, einen beträchtlichen Performancegewinn bringen.

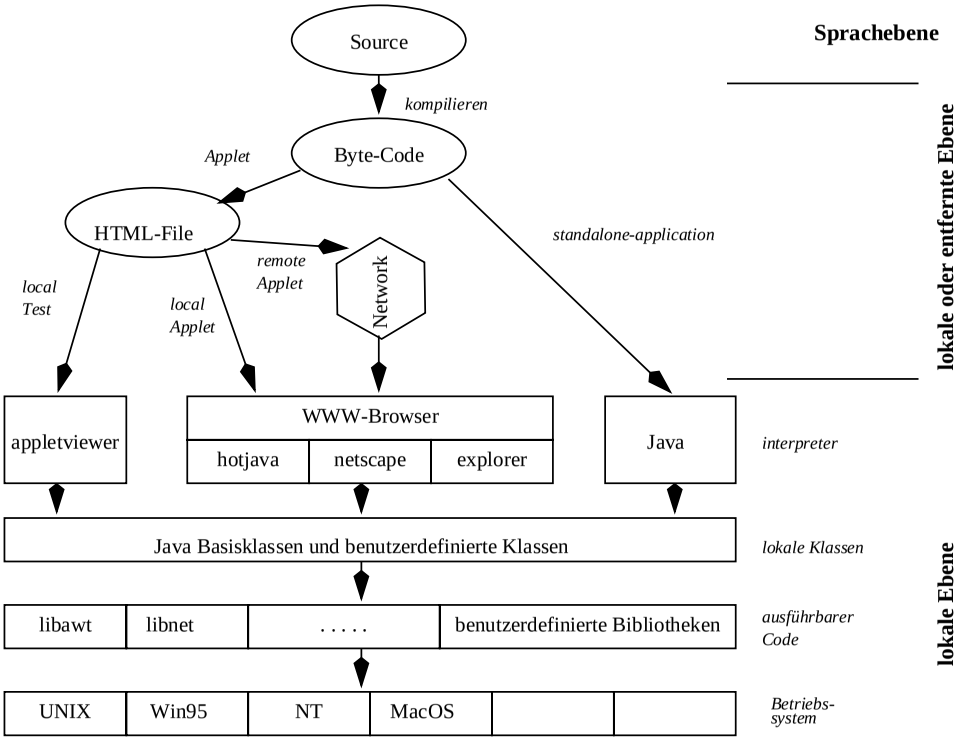

Abbildung36. Java Basis-Architektur

Konzeptionell ist Java eine objektorientierte Programmiersprache. Java ist klassenbasiert, unterstützt eingeschränkte Polymorphie, dynamisches Binden sowie Einfachvererbung. Syntaktisch erinnert Java stark an C++ wobei "viele Möglichkeiten von C beziehungsweise C++ aber in Java nicht übernommen wurden, um die Sprache einfach zu halten"[Kra96]. Im Gegensatz zu C++ bzw C ist Java eine typbasierte Programmiersprache.
Daneben hat Java einen Vorrat an grafischen Methoden. "Die Zeichenoperationen in Applets gleichen einem GKS (Grafic Kernel System)"[Bac96]. Außerdem können Java-Programmteile echt parallel ablaufen.

Anwendungsmöglichkeiten Die Anwendungsmöglichkeiten Javas sind breit gefächert. Grundsätzlich kann man zwischen Standalone-Applikationen und Applets unterscheiden. Standalone-Applikationen sind "klassische" lokale Applikationen wohingegen Applets meist von einem (entfernten) WWW-Server in einem WWW-Browser auf Clientseite geladen werden. Der Browser übernimmt hierbei die Aufgabe des Interpreters und stellt eine geeignete Ablaufumgebung zur Verfügung. Applets können aber auch so implementiert werden, daß sie zwar von einem entfernten System über den Browser geladen werden, dann aber ohne Browser als Standalone-Applikationen ablaufen.
Die zusätzlichen Möglichkeiten, die Java in Verbindung mit einem einfachen HTML-Browser ermöglicht sind relativ groß. So können Applets z.B. weitere Toplevel-Fenster (außerhalb des Browsers) öffnen und komplexe grafische Oberflächen der Präsentationsschicht realisieren [Kra96]. Ebenso können Aufgaben der Applikationsschicht in Applets verlagert werden, wodurch die Systemressourcen des Client stärker genutzt werden (vgl. 2.3).

Neben diesen clientseitigen Anwendungen gibt es auch die Möglichkeit, Java-Applikationen serverseitig zu vewenden. Diese Servlets können einfache Dienste wie z.B. die eines CGI (vgl. hierzu Abbildung 32) bis hin zu komplexen Aufgaben eines Datenbankservers übernehmen.

Probleme Einen schwachen Punkt hat auch Java. Es ist der Sicherheitsaspekt, der vor allem nach der "DNS-Attacke" vom Mai 96, viel diskutiert wurde. Mit dem Laden eines Applets, über dessen Herkunft man nicht genau informiert ist, kann man sich nämlich leicht ein trojanisches Pferd ins Haus holen. Das sind Anwendungen die unter dem Mantel einer nützlichen Applikation gewissen unerwünschten Nebentätigkeiten nachgehen, die die Sicherheit des Systems bzw. die Integrität der Anwendungen unterminieren. Dies ist besonders deshalb kritisch, weil Java-Applets praktisch auf jeder Systemebene Aufgaben übernehmen und somit ihre Fähigkeiten auch leicht mißbrauchen können.
Hier fehlt es derzeit noch an Konzepten, um dem Benutzer eines Applets zusichern zu können, daß das Applet auch genau das tut, was es vorgibt zu tun. Allerdings wird diese Problematik von manchen Autoren (vgl. [Kra96]) im Intranet als nicht besonders emminent angesehen, da hier davon ausgegangen werden könne, daß die Herkunft der Applets sichergestellt werden kann und man damit für ihr Verhalten garantieren könne.
Natürlich sind auch schon heute Sicherheitsmechanismen in Java implementiert. So existiert in jeder Ablaufumgebung genau eine Instanz der Klasse Security-Manager, die - auf Kosten der Laufzeit - die Zugriffe auf lokale Systemzugriffe kontrolliert. Auf diese Art und Weise kann Applets eine bestimmte Art des Systemzugriffs erlaubt oder verweigert werden. Dieses Konzept halte ich allerdings für zu grob, da aus der Art des Systemzugriffs noch nicht abgelesen werden kann, welche Absichten dahinter stecken.

4 Heutige Realisierungen

4.1 Entwicklungs-Werkzeuge für HTML-Seiten

Die immer noch wichtigste Anwendung im Web sind HTML-Seiten. Dabei sind dies keine wissenschaftlichen Abhandlungen über ein aktuelles Thema mehr. Es sind u.a. Anzeigen oder Produktankündigungen von Firmen und anderen kommerziell orientierten Organisationen. Mit der Art der Texte hat sich auch der Personenkreis gewandelt, der diese Seiten erstellt. Waren dies früher technisch versierte (wissenschaftliche) Mitarbeiter so sind dies heute "Mitarbeiter, die aus grafischen Berufen stammen"[JB96].
Natürlich kann man auch heute noch Web-Seiten mit Hilfe eines einfachen Editors - z.B. vi - entwerfen. Allerdings entsprechen diese Seiten nicht ganz den derzeitigen Standards und diese Art der "Entwicklungsumgebung" läuft auch unseren in 2.1 aufgestellten Forderungen nach einer einheitlichen und komfortablen Entwicklungsumgebung zuwieder. Ebenso ist das Generieren von Web-Sieten aus bestehenden Dokumenten vieler Textverarbeitungsprogramme möglich. Viele dieser Programme haben mittlerweile eine entsprechende Erweiterung. Allerdings sind viele dieser Konverter qualitativ nicht auf einem

besonders hohen Stand und zum anderen wird an das Aussehen einer WWW-Seite andere Anforderungen gestellt als dies bei einem normalen Dokumnet der Fall ist.
Als ein Beispiel für einen recht komfortablen HTML-Editor - denn Editoren gibt es mittlerweile so viele, daß die Verweise auf entsprechende Editoren schon ganze Suchindizes im Web füllen - kann man HoTMetal PRO 3.0 heranziehen. Es handelt sich hierbei um einen reinen Editor. Da die Darstellung von HTML-Seiten - wie in Kapitel 3.1 erwähnt - vom jeweiligen Browser abhängt, ist die Notwendigkeit eines WYSIWYG-Werkzeugs (What You See Is What You Get) nicht unbedingt gegeben.
HoTMetal behandelt die HTML-Befehle auf einfache Weise. Diese können als Marken in den Text eingefügt werden. Dies vehindert unschöne Tippfehler. Außer der Möglichkeit Texte neu zu erstellen, können Dokumente herkömmlicher Textverarbeitungsprogramme, wie z.B. Word, importiert werden. Daneben können Grafiken verschiedenster Formate importiert und bearbeitet werden.
Die Möglichkeiten eines HTML-Werkzeugs sollten allerdings über das reine Erstellen von Seiten hinausgehen. Wenn es darum geht eine Site zu verwalten, sollten z.B. Verweise zwischen den Seiten in einer übersichtlichen Form dargestellt und bearbeitet werden können. Ein Werkzeug das dies - zumindest teilweise - kann ist Frontpage von Microsoft. Hier kann man sich alle Seiten einer Web-Site sowie zwischen den Seiten bestehende Verweise grafisch oder als Inhaltsverzeichnis anzeigen lassen. (vgl. hierzu auch [JB96] und [Beh96])

4.2 Warenhauskatalog im WWW (trabstec AG Tübingen)

Wie in Abschnitt 3.2 erwähnt, gibt es verschiedene Arten der Datenbankanbindung eines WWW-Systems. Die hier vorgestellte Anbindung ist CGI-Script basiert (siehe Abbildung 32). Das System bietet einen vollständigen Produktkatalog inclusive Bestellformular für interne und externe Benutzer. Dabei treten neben den herkömmlichen Problemen wie Laufzeitverhalten u.ä. Schwierigkeiten mit der Datensicherheit auf (vgl [Win96]).
Mit Hilfe eines ausgeklügelten dreistufigen Authentisierungssystems können so den Kunden Tabellen mit speziellen Rabattsätzen zur Verfügung gestellt und Mitarbeitern Präferenzlisten mit häufig georderten Artikeln zusammengestellt werden.
Die Funktionsweise des Systems ist relativ simpel und entspricht dem in Abbildung 32 dargestellten Sachverhalt. Nach Authentisierung des Benutzers und der Eingabe der Selektionskriterien, wird eine enstprechende Datenbankabfrage an die Datenbankserver weitergeleitet und die HTML-Seite mit Hilfe eines SQL-Scripts generiert.
Daß solche Anwendungen ihre Vorteile haben, liegt auf der Hand. Im Vergleich zu einem herkömmlichen Papierkatalog, der in unterschiedlich Ausführungen, in Form von Sprache, Währungsangaben und Inhalt gedruckt werden muß, kann hier die Datenbank auf die Herkunft der Datenbankanfrage reagieren und die Web-Seite mit einfachen Mitteln in Deutsch, Englisch oder Französisch mit Preisangaben in der jeweiligen Landeswährung generieren.

4.3 Anwendungsentwicklung

Im Bereich der Applikationserstellung sind in letzter Zeit vor allem Werkzeuge zur Erstellung von Java Applikationen auf den Markt gekommen. Beispielhaft sind hier Produkte wie Symantec-Café [Kun96] oder Java WorkShop [Kir96], die in Punkto Benutzerfreundlichkeit und Komfort dem Microsoft Produkt BackOffice Konkurrenz machen sollen. Bei beiden Paketen handelt es sich um Programmieroberflächen, die auf verschiedenen Plattformen zum Einsatz kommen können, da sie selbst in Java geschrieben wurden.
Der Unterschied zu einem Grundjavapaket besteht darin, daß die Editoren um Funktionen und Anzeigen erweitert wurden, die die Vererbung vereinfachen bzw. übersichtlich machen. So verfügt Java WorkShop z.B. über einen Klassen-Browser mit dessen Hilfe man sich die Methoden der gewünschten Klasse anzeigen und auswählen kann, welche man verwenden möchte. Im Programmtext des aktuellen Programms wird dann automatisch der entsprechende Methodenaufruf eingefügt.
Darüber hinaus verfügen beide Entwicklungsumgebungen über die Möglichkeit, fehlerhaften Code zu debuggen. Dies geschieht mit üblichen Break-Points und Inspekt-Variablen die dann in einem separaten Fenster angezeigt werden.
Neben diesen eher herkömmlichen Programmierumgebungen gibt es auch ein etwas anders Produkt auf dem Markt. Der CyberAgent[Sch96] ist eine Entwicklungsumgebung der etwas anderen Art. Diese Applikation wurde selbst in Java geschrieben und ermöglicht das erstellen von Agenten für ein breites Spektrun vom Anwendungen. So können die mit CyberAgent erstellten Makros z.B. zentrale Adminstrative Aufgaben über Systemgrenzen hinweg übernehmen. So lassen sich damit DNS und DHCP-Server synchronisieren, Software angebundener Clients aktualisiernen und ganze Netzwerke nach virentypischen Codefragmenten durchforsten. Auf allen Java-transparenten Ebenen können mit diese Agenten Dateien gelesen, Daten geändert, Files angelegt und Programme zur Ausführung gebracht werden. Der CyberAgent umfasst sowohl Werkzeuge zum Erstellen von Agneten als auch zur Verwaltung derselben.

4.4 Bewertung

Mit der Verwendung von Java werden mit Sicherheit viele der Forderungen aus Kapitel 2.3 erfüllt. So ist die Integration von Datenbankbefehlen u.ä. bei Java vorhanden. Ebenso wird die Forderung nach guter Ausnutzung der Systemressourcen auf Clientseite durchgesetzt. Ebenso wird mit Java die Netzbelastung sicher verringert, da der vom Java-Compiler erzeugte Byte-Code extrem kompakt ist.
Nachteile bei der Verwendung von Java treten beim Laufzeitverhalten auf. Hier ist aber mit den Just-In-Time-Compilern eine mögliche Lösung in Sicht. Anders sieht dies bei der Betrachtung des Sicherheitsaspekts von Java aus. Sicher gibt es schon heute Möglichkeiten einen gewissen Grad an Sicherheit zu gewährleisten und sei dies, indem Java-Programmen durch die Systemrichtlinen nur sehr restriktiv Zugang zu Systemressourcen und Datein gewährt wird. Allerdings fehlt ein Konzept mit dem Java alle Freiheiten gewährt und trotzdem ein Maß an Sicherheit gewährleistet wird, das man von herkömmlichen Programmen gewöhnt ist.
Die oben genannten Werkzeuge verfolgen durchaus die gleiche Zielrichtung wie die in

Abschnitt 2.1 genannten Forderungen. Sie sind aber von den Ideal einer "Entwicklungsumgebung aus einem Guß" weit entfernt. Sie sind bildlich gesprochen Puzzleteile, die noch zu einem Gesamtbild zusammengefügt werden müssen.
Vor allem die Integration aller Teilbereiche von der Erstellung einfacher HTML-Seiten bis hin zu Javaapplikationen scheint noch ein ganzes Stück entfernt zu sein. Allerdings ist hier auch schon einiges in Bewegung, da manche HTML-Editoren mittlerweile über eine Unterstützung der verschiedenen Web-Server-APIs (Appiction Programming Interface) verfügen. In Bezug auf Benutzerführung und Visualisierung der Programmierung sind die entsprechenden Werkzeuge zwar schon ein ganzes Stück vorangeschritten, es bleiben aber noch eine Menge ungenutzter Möglichkeiten.
So fehlen bisher Entwurfswerkzeuge zum rechnergestüzten Design von Web-Anwendungen. Es sind zwar wie in Kapitel 4.1 erwähnt Möglichkeiten vorhanden, bestehende Strukturen zu erfassen, dies entspricht aber in keiner Weise der Forderung, die Implementierung aus dem Entwurfsmodell generieren zu können. In diesem Bereich liegt das Web-Engineering noch weit hinter seinem Vorbild, der Softwaretechnik zurück.
Zusammenfassend läßt sich sagen, daß mit Verwendung des Java-Konzepts die Forderungen nach

- einer speziellen Programmiersprache
- Auslastungsbalance zwischen Client und Server und
- geringem Kommunikationsaufwand

erfüllt sind. Allerdings hat Java momentan noch Nachholbedarf in den Bereichen

- Sicherheit und
- Performance

Im Bereich der Editoren läßt sich zumindest die Tendenz erkennen, HTML-Dokumnete

- leichter wartbar zu machen und
- der Forderung nach stärkerer Visualisierung nachzukommen.

Gänzlich unerfüllt bleiben momentan die Forderungen nach:

- einheitlicher Benutzeroberfläche
- einheitlicher Programmieroberfläche
- Werkzeuge zum rechnergestützten Entwurf
- Werkzeuge zur Versionskontrolle und -verwaltung (nur in Ansätzen vorhanden)
- Sicherheitsmechanismen auf Betriebssystem- bzw. Netzprotokollebene

5 Entwicklungsmöglichkeiten

Entwicklungsmöglichkeiten bestehen für das Web-Engineering sowohl in den Bereichen, in denen es momentan zum Einsatz kommt, als auch in Bereichen, in die derzeit noch nicht von ihm erschlossen wurden. Die Vorteile,die die Applikationen bieten, die mit Java erstellt wurden, sind zu groß um ungenutzt zu bleiben. So ist der wesentlich vereinfachte Installations- und Konfigurationsaufwand für Applikationen, die auf mehreren Rechner zum Einsatz kommen, nicht zu unterschätzen.
Im Bereich der Datenbankanwendungen sind neue Ansätze erkennbar, die vom Modell der relationalen Datenbank abstrahieren und dabei jede Tabelle als eine Menge von Objekten betrachten. Im Hinblick auf die Durchsetzung des objektorientierten Programmierparadigmas ist dies sicher ein wichtiger Schritt, um schon auf Modellebene ein einheitliche Denkweise bei der Entwicklung von Web-Applikationen zu haben und nicht verschiedene Paradigmen kombinieren zu müssen [Lie96].
Ob die Lösung aller Probleme nun wirklich bei der Verwendung von Java liegt ist nicht unbedingt gesagt. Schließlich gibt es auch hier einige Alternativen, die einer Betrachtung durchaus Wert wären, aber keine kozeptionell neuen Aspekte bringen. Falls es in diesem Bereich in nächster Zeit zu Neuerungen kommen würde, könnte diese durchaus einen größeren Einfluß auf die Entwicklung des Web-Engineerings haben. Auch gibt es in diesem Bereich schon einige interessante Denkanstöße, die allerdings noch nicht über das Stadium einer Denkrichtung hinaus sind.
Das größte Entwicklungspotential für die Disziplinen des Web-Engineerings selbst liegen wohl im Entwurfsbereich. Mit der Ausarbeitung neuer Entwurfsmethoden, die auch die Eigenheiten des zugrundeliegenden Mediums - des Internet - besser abbilden, könnte hier ein großer Schritt gemacht werden. Ebenso ist bei der Entwicklung einheitlicher Benutzerschnittstellen noch einiges zu tun.
Nicht zuletzt im Bereich der Sicherheitskonzepte steckt ein großes Potential. Dies ist schon allein mit der leichten Verfügbarkeit des Web zu begründen. Der leichte Zugang zu Daten aus aller Welt macht es nötig, interne und sensible Daten zu schützen und Viren keine Verbreitung zu erlauben. Daß hier bei SHTTP[1] die Entwicklung noch nicht beendet sein kann ist klar. Interessant in diesem Zusammenhang ist auch, ob es weiter dem Entwickler einer Anwendung überlassen bleiben soll, wieviele und welche Sicherheitsüberprüfungen innerhalb seiner Applikation durchgeführt werden sollen, um diese und ihre Anwender zu schützen, oder ob es nicht an der Zeit ist, hier schon von Seiten der tiefer liegenden Schichten einen gewissen Sicherheitskomfort zu bieten.

6 Ausblick

Momentan entdecken die Hersteller klassischer Client-Server Anwendugsentwicklungswerkzeuge das Internet als neuen Markt und Java als neues Programmierkonzept. Deshalb kann man davon ausgehen, daß in naher Zukunft eine Menge neuer Web-Produkte auf den expandierenden Markt gelangen werden. Gerade im Bereich der Web-Datenbanken steckt ein großes Potential, da hier die Daten den direkten Weg vom An-

[1] Secure Hypertext Transfer Protocol

bieter zum Kunden finden und kein Dritter - abgesehen vom Internet-Access-Provider - zwischen den Geschäftspartnern steht (vgl. [Lie96]).
Bei der Anwendung dieser Produkte wird zu Beginn sicher das Intranet eine große Rolle spielen, da die Firmen diesen Weg der innerbetrieblichen Informationsverwaltung gerade entdecken. Anwendungen im Internet werden, bis auf einige Branchen, wohl eher die Ausnahme bleiben und den Status von Prototypen haben. Allerdings dürften nach einem gewissen Reifeprozess der Entwicklungswerkzeuge und der damit erstellten Anwendungen, die Ergebnisse des Web-Engineerings auch global eingesetzt werden.
Aber die Möglichkeiten, die mit den neuen Konzepten - wie z.B. Java - eröffnet werden gehen noch viel weiter. So könnte die ausgiebige Nutzung von Applets das derzeit bestehende Lizensierungssystem revolutionieren, da sich nun die Möglichkeit eröffnet Software nicht komplett kaufen zu müssen, sondern sie für die Dauer ihrer Nutzung zu mieten.
Vielleicht wird sogar das Szenario wahr, das Werner Krauß in seinem Artikel [Kra96] ausmalt: "Pünktlich um fünf Uhr morgens [...] klingelt der Wecker. Natürlich mit einem Java-Chip ausgetattet. Über das wohnungsinterne LAN beauftragt unser Timer-Interrupter den Küchenserver mit der Zubereitung des Frühstücks: Kaffemaschine, Toaster und Eierkocher treten per Java-Threads parallel und wohl synchron in Aktion. Während des Frühstücks kontrollieren wir per Network-Computer den Zustand unserer Server, lesen die neuesten Nachrichten und setzen verschiedene Queries an unser Bankkonto via Internet ab. Schließlich nehmen wir unser Handy mit Java-fähigem Display und machen uns an unser Tagwerk in der weiten Welt."

Nutzung von Broadcast-Medien

Shao-Kang Yang

Kurzfassung

Verteilmedien besitzen attraktive Eigenschaften für die Realisierung von Informationssystemen. Dazu gehören Skalierbarkeit, Verfügbarkeit und konzeptionelle Einfachheit. Je mehr Informationen ein Server über ein Verteilmedium anbieten kann, desto mehr potentielle Clienten sind für ihn erreichbar. Als Problem erweist sich aber, daß die Zugriffszeit mit der Menge der zu verteilenden Daten wächst. Der Zugriff auf eine Teilmenge der verteilten Informationen kann auf Senderseite durch Konstruktion einer entsprechenden Sendefolge optimiert werden.

Caching auf Seiten des Clienten in unidirektionalen Verteilmedien darf nicht nur eine einseitige Optimierung aufgrund von Zugriffswahrscheinlichkeiten durchführen, sondern muß die unterschiedlichen Verzögerungen bis zur Neuübertragung der verschiedene Datenobjekte berücksichtigen. Die Aufgabe besteht darin, die durch eine Sendefolge vorgegebenen Verteilzeiten von Informationen auf Empfängerseite durch den Einsatz des Cache im Clienten auf die erwartete Zugriffsverteilung eines individuellen Nutzers abzubilden. Erschwert wird dies durch die Anforderung, daß Caching möglichst transparent gehalten werden muß. Insbesondere ist der zusätzliche Verabeitungsaufwand für Caching stark begrenzt. Deshalb sind einfache Caching-Strategien gefragt.

Als vielleicht wichtigstes Ergebnis dieser Arbeit ist festzuhalten, daß in unidirektionalen Verteilmedien die Wiederholstruktur von Datenobjekten in einer Caching-Strategie berücksichtgt werden muß.

1 Einführung

Neue Entwicklung bei mobilen Rechnern und drahtlosen Netzwerken ermöglichen die Entstehung des sogenannten „ubiquitous computing“[AFZ95] . Eine Herausforderung ist dabei, das Potential des allgegenwärtigen Zugang zu Daten für mobile Clients zu verwirklichen. In der mobilen Umgebung gibt es Schwierigkeiten bei der Unterstützung von datenintensiven Anwendungen, weil mobile Clients oft von ortsfesten Server abgekoppelt sein können oder sie nur einen Kanal mit geringer Bandbreite für das Senden von Nachrichten zum Server zur Verfügung haben. Dieses Problem wird durch drei weitere Aspekte verschlimmert:

- man kann nicht mit 100-prozentiger Genauigkeit vorhersagen, ob die Daten in Zukunft von vielen Anwendungen benötigt werden oder nicht,
- die Speicherkapazität von mobilen Rechnern (die Clientseite) ist begrenzt,
- die Notwendigkeit,daß mobile Clients mit neuen und aktuellen Daten versorgt werden.

Eine Lösungsweg für diese Probleme ist es, einer ortsfesten Servermaschine einen breitbandigen Broadcast-Kanal zur Verfügung zu stellen, auf der die Daten, die die Clienten

brauchen, durch Rundsendung an alle Clients übertragen werden. Dieses System ist asymmetrisch, weil die Übertragunskapazitäten von beiden Seiten (Server und Client) unterschiedlich sind. Der Zugang zu Informationen kann dabei auf unterschiedliche Art und Weise erfolgen. Eine Alternative, den Zugang zu Informationen zu ermöglichen, besteht darin, daß der Informationsnutzer Zugriff auf den Gesamtbestand an Informationen erhält und sich selbst die benötigten Informationen heraussucht. Der Vorteil dieses Verfahrens besteht in der Einfachheit der Durchführung. Es ist im Gegensatz zu bidirektionaler Kommunikation kein aufwendiger Mechanismus nötig, um Zugang zu Information zu erhalten. Vielmehr werden die Informationen an die Informationsnutzer verteilt. Damit ist es außerdem möglich, über ein Rundsendemedium Informationen einer großen Zahl von Informationsnutzern anzubieten. Im folgenden werden die Einzelheiten der spezifischen asymmetrischer Umgebungen, der Broadcast-Disk-Organisation, des Client-Cache-Management und des Prefetching erläutet.

2 Asymmetrie der Kommunikation

2.1 Eigenschaften der Kommunikation zwischen Client und Server bei unidirektionalen Verteilmedien

In vielen Anwendungsgebieten ist der „downstream“ Kommunikationsbedarf (d.h. vom Server zum Client) viele größer als der „upstream“ Kommunikationsbedarf (von Client zum Server). Dies spiegelt sich auch in den technischen Möglichkeiten wider. So kann in einer mobilen Umgebung ein Server eine relative große Rundsendebandbreite haben, während der Client nicht senden kann oder nur eine niedrige Bandbreite zur Verfügung hat. Wir nennen solche Systemumgebungen „asymmetrische Kommunikationsumgebung“, bei Verteilsystemen „unidirektionale Verteilkommunikation“. Anwendungsbeispiele sind etwa Verkehrsinformationssysteme, Fernstudiensysteme, Versicherungssysteme usw.[IB94]. Die Kommunikationsasymmetrie kann in zwei Aspekten erscheinen. Der erste ergibt sich aus der asymmetrischen Bandbreite des physikalischen Kommunikationsmediums oder der unterschiedlichen Übertragungsfähigkeit zwischen Server und Client. Der zweite ist, daß die Kommunikationsasymmetrie bereits durch den Informationsfluß zwischen den Partnern in einer Anwendung gegeben ist. So ist beispielweise ein Informations-Retrievalsystem asymmetrisch, da es sehr viel mehr Clients als Server gibt, oder da es Kapazitätsprobleme gibt (im Netz oder beim Server), wenn gleichzeitig Anfragen von vielen Clients gestellt werden. Da die Asymmetrie entweder aus der physikalischen Einrichtung oder aus dem Charakter der Aufgabenlast resultieren kann, spannt sich die Klasse der asymmetrischen Kommunikationsumgebungen über viele der wichtigen System- und Anwendungsgebiete. Speziell ist die Übertragung von Daten in einer unidirektionalen Verteilkommunikationsumgebung nur unidirektional möglich, und zwar vom Sender zum Empfänger. Auch können die Rollen der beiden nicht gewechselt werden, um ein Kommunikation in Gegenrichtung zu ermöglichen.

2.2 Anwendungen

Ein wesentliches Problem in mobilen Umgebungen ist Abkoppelung eines Clients vom Server. Eine Abkoppelung kann sowohl technisch bedingt sein als auch aus prohibitiven Kosten einer ständigen Kommunikation im Mobilen Umfeld resultieren [IB94]. Wenn aber Benutzern die Verbindung zu einem Server fehlt, so müssen die abgekoppelten Benutzer auf lokalen, im Cache befindlichen Daten arbeiten. Jedoch kann es aus mehreren Gründen vorkommen, daß diese Daten nicht ausreichen:

1. In vielen Situationen kann nicht präzise vorhergesagt werden , welche Daten während der Abkoppelung gebraucht werden.

2. Die heruntergeladenen Daten können inzwischen veraltet sein. Außerdem können während der Abkoppelung neue interessante Daten entstanden sein.

Eine Abkoppelung kann als ein extremer Fall einer asymmetrischen Kommunikation betrachtet werden. Viele Aspekte des Datenmanagements, die in einer drahtlosen Umgebung auftreten, können auch in anderen asymmetrischen Umgebungen vorkommen. Das Hauptanwendungsgebiet für asymmetrische bzw. unidirektionale Kommunikationen ist dabei die Realisierung von Informationssystemen. Diese versuchen einer großen Anzahl von Clients Informationen aller Art zugänglich zu machen. Beispielanwendungen sind Wetter- und Finanzberichte. Eine der umfangreichsten unter den vielen potentiellen Anwendungen für die nächste Zukunft ist die Realisierung des „Advanced Traffic Information System (ATIS)“ [AFZ95]. Ein ATIS enthält mehrere zentrale Server, die aktuelle Informationen von Partnern wie dem ADAC oder der Polizei für ein Großstadtgebiet enthalten. Die Fahrer tragen kleine spezialisierte Computer in ihren Fahrzeugen, die ihnen bei Aufgaben wie der Routenplanung helfen. Die Fahrer empfangen von den Server die statische Informationen wie die Straßentopologie und dynamische Informationen wie Staumeldungen. Die meisten Fahrer interessieren nur dynamische Informationen. Da die Datenübertragungskapazität beschränkt ist, braucht man schließlich nur die dynamische Informationen zu versenden, während die statischen Informationen wie Stadtpläne auf CD-Rom vorliegen.

2.3 Unterschiede zu konventionellen Client/Server-System

Durch die Eigenschaften von asymmetrischen unidirektionalen Verteilmedien ergeben sich erhebliche Unterschiede zu konventionellen Client/Server Systemen: Bei konventionellen Client/Server Systemen ist für jeden Client eine individuelle bidirektionale Kommunikation mit einem Server erforderlich. Bei der unidirektionalen Verteilkommunikation kann ein Server (als Sender) mit Clients (als Empfänger) kommunizieren, indem der Server ein Rundsendemedium nutzt. Das Medium überträgt die vom Server spezifizierten Daten an alle angeschlossenen Clients. Die Clients erhalten durch ihre Empfängerzugriff auf alle gesendeten Daten. Die Übertragung von Daten ist nur unidirktional möglich, d.h. vom Server zum Client.

2.4 Problemstellung

Die Nutzung von unidirektionalen Verteilmedien führt zu einigen herausfordernden Problemen. Sie können entlang die Client-Seite und Server-Seite strukturiert werden. Die Hauptforderung aus der Perspektive des Server ist die Datenorganisation in dieser Umgebung. Die serielle Natur von Rundsendemedien schließt den wahlfreien zufälligen Zugriff aus. Deshalb ist die Strukturierung der Daten der entscheidende Faktor bzgl. der Kommunikationsleistung. Darüberhinaus ergibt sich aus der Tatsache, daß die Bandbreite eine beschränkte Ressource ist, das Problem, daß bei der Erhöhung der Sendefrequenz eines Datums die Frequenzen für die andern Daten reduziert werden müssen. Die Probleme, die sich auf der Client-Seite ergeben, ergeben sich aus der Organisation des Cache, konkret der Strategie zur Ersetzung von Seiten im Daten-Cache und dem Prefetching. Die traditionelle Caching-Technik ist in dieser Umgebung unzulänglich, weil manche Daten sehr viel seltener angefordert werden als andere. Das Cache-Management muß die Kosten des Zugriffs entsprechend der lokalen Zugriffswahrscheinlichkeiten optimieren. Die variierenden Kosten zum Erhalt von Daten können zudem auch den Ausschlag zur Nutzung einer Prefetching-Strategie geben. Bisherige Arbeiten zu diesem Problem im unidirektionalen Umfeld gehen von zwei beschreibenden Annahmen aus: Zum einen wird angenommen, daß die Clients nur lesend zugreifen. Zum anderan werden die Zugriffswahrscheinlichkeiten der Clients als statisch vorausgesetzt.

3 Strukturierung der Rundsende-Kommunikation

3.1 Eigenschaften von Rundsendeprogrammen

In einem rundsendungbasierten Informationssystem muß ein Datenserver ein Rundsendeprogramm konstrukturieren, um den gesamten Bedarf aller Clients zu bedienen. Als Strukturierungskonzept hierfür wird im folgenden ein Mechanismus namens „Broadcast Disk“ vorgestellt [AFZ95], der einen Datenbasiszugang in einer asymmetrischen Umgebung liefert. Bei dieser Methode sendet ein Server kontinuierlich und wiederholt seine Daten an die Gemeinschaft der Clients. Die Sendefolge ist dabei so strukturiert, daß die Rundsendung in der Tat als eine „Disk“ angesehen werden kann, von der die Clients die Daten wiedergewinnen können. Im einfachsten Fall sind die zu sendenden Daten nicht nach Zugriffswahrscheinlichkeiten priorisiert. Der Server konstruiert dann aus den Daten eine Resultatfolge und wiederholt diese zyklisch auf dem Medium. Dies ist in Abb. 1.(a) gezeigt.

Eine Rundsendung kann als eine zusätzliche Schicht in der Speicherhierarchie der Clients angesehen werden. Wenn eine Anwendung auf einem Client ein Datum benötigt, so sucht sie es zunächst im lokalen Speicher oder auf lokalen Platten. Wenn das Datum dort nicht gefunden wird, dann hört der Client die Rundsendung ab, bis das gesuchte Datum empfangen werden kann. Die Struktur der Rundsendung berücksichtigt aber die relative Wichtigkeit der Daten für die Clients nicht. Damit stellt sich die Frage, wie die Anforderungshäufigkeiten der Clients auf die gesendeten Daten für eine Optimierung genutzt werden können.

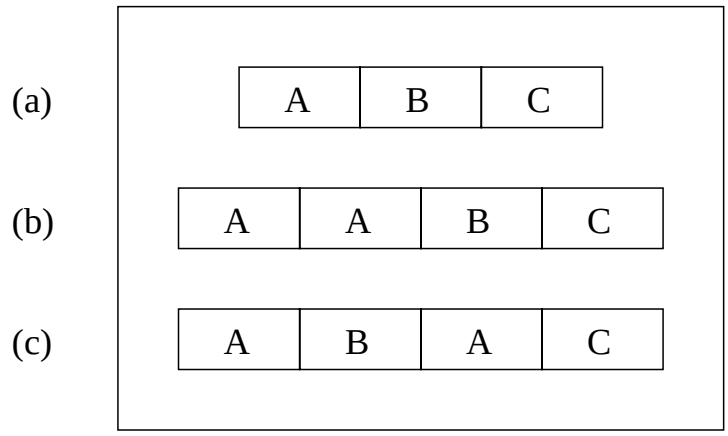

Abbildung37. Rundsendeprogramme.

In der „Broadcast Disk“-Methode werden alternative Rundsendestrukturen untersucht. Die Rundsendung wird als Analogon zu einer „Multidisk“, d.h. einer Menge von Platten unterschiedlicher Größen und Rotationsfrequenzen konstruiert. Das Prinzip ist, daß mit kleinen und schnellen Platten begonnen wird, die mit zunehmender Größe immer langsamer werden. Deshalb werden Daten, die am häufigsten gebraucht werden, auf den schnellsten Platten abgelegt. Daten, die seltener gebraucht werden, werden auf langsameren Platten gesendet. Ein einfaches Beispiel demonstriert Abb.1. Die Abbildung zeigt drei verschiedene Broadcast-Programme für eine Sendefolge, die drei gleichlange Datenobjekte (z.B Seiten) umfaßt. Die einfachste Möglichkeit, eine Sendefolge zu konstruieren, besteht darin, jedes Datum einmal zu senden und dies dann zyklisch zu wiederholen. Dies wird in der Broadcast-Disk Methode als eine „flat disk“ [AFZ95] bezeichnet. Es resultiert das Programm aus Abb.1(a). Diese Struktur kommt zum Einsatz, wenn für alle Daten die gleichen Zugriffswahrscheinlichkeiten bestehen. Wenn die Zugriffswahrscheinlichkeiten ungleich verteilt sind, dann läßt sich dies durch eine Anpassung der Sendefrequenz der Daten berücksichtigen. Soll die Zugriffszeit auf A verkürzt werden, dann wird die Sendefrequenz von A erhöht. Dies kann durch zwei verschieden strukturierte Sendefolgen erfolgen. Die Struktur aus Abb.1.(b) wird als „skewed disk“ und die aus Abb.1.(c) als „multidisk“ bezeichnet. Im Programm (b) sind die Vorkommnisse von A in eine Gruppe zusammengefaßt, die geschlossen gesendet wird. Dadurch sind die Zwischenankunftszeiten variabel. Im Gegensatz hierzu ist Programm (c) regulär, d.h. es gibt keine Variationen in den Zwischenankunftszeiten für die Daten. Die Zugriffsverzögerung eines Empfängers für ein häufiger angefordertes Datum ist folglich relativ kürzer als bei einer flachen Disk. Andererseits ist der Zugriff auf ein seltener angefordertes Datum langsamer als bei einer flachen Disk. Die Abb.2. zeigt die Zugriffswahrscheinlichkeiten und die Zugriffszeiten bei den drei verschiedenen Rundsendsprogrammen.

Insgesamt ergeben sich die folgenden drei Feststellungen.

1. Die flat-disk ist optimal , wenn die Zugriffswahrscheinlichkeiten gleich verteilt sind. Der Grund hierfür ist, daß bei einer festen Bandbreite zur Übertragung die Sendefrequenz eines Datums nur auf Kosten der Sendefrequenz eines anderen Datums erhöht werden kann.

Zugriffswahrscheilichkeiten			Zugriffszeiten		
A	B	C	Flat(a)	Skewed(b)	multidisk(c)
0.333	0.333	0.333	1.50	1.75	1.67
0.50	0.25	0.25	1.50	1.63	1.50
0.75	0.125	0.125	1.50	1.44	1.25
0.90	0.05	0.05	1.50	1.33	1.10
1.0	0.0	0.0	1.50	1.25	1.00

Abbildung38. Zugriffszeiten bei verschiedenen Zugriffswahrscheinlichkeiten.

2. Die anderen beiden Folgen (b) und (c) erreichen bei Zugriffswahrscheinlichkeiten, die Seite A bevorzugen, bessere Zugriffszeiten als die flache Folge.

3. Das Multidisk-Programm ist immer besser als das "Skewed Programm". Wenn nämlich die Zwischenankunftszeit einer Seite fest ist,dann beträgt die erwartete Zugriffszeit für eine Anforderung, die zu einem zufälligen Zeitpunkt gemacht wird, die Hälfte der Zeit zwischen zwei aufeinanderfolgenden Übertragungen einer Seite.

Variieren hingegen die Zwischenankunftszeiten, dann sind die Abstände zwischen aufeinanderfolgenden Übertragungen einer Seite unterschiedlich groß. In dieser Fall ist die Wahrscheinlichkeit eines Zugriffs während einer größeren Lücke auch größer, weshalb sich die erwartete Zugriffszeit mit der Varianz in den Zwischenankunfszeiten erhöht. Die Broadcast-Disk ist in vielen Situationen eine sehr effektive Methode, um Daten zum Client zu übertragen. Der Hauptvorteil des Broadcast-Disk-Paradigmas ist seine „Skalierbarkeit". Die Leistung des Systems ist nicht davon abhängig, wieviele Benutzern gerade die „Broadcast Disk" nutzen, weshalb sie das ideale Medium zur Verbreitung öffentlich genutzter Informationen ist. Durch die obige Diskussion ergibt sich, daß jedes Rundsendeprogramm die folgenden Eigenschaften haben sollte:

- Die Zwischenankunftszeiten von aufeinanderfolgenden Kopien eines Datums sollten fest sein.

- Es sollte eine wohldefinierte Rundsendeeinheit geben, nach der die Sendefolge wiederholt wird.

- Es sollte eine kleine Anzahl von verschiedenen Rundsendefrequenzen genutzt werden.

- Im Rahmen der oben erwähnten Einschränkungen sollte so viel der verfügbaren Bandbreite wie möglich genutzt werden.

Insgesamt unterscheiden sich diese Arbeiten und damit auch die weitere Diskussion in zwei Aspekten von den konventionellen Arbeiten. Zum einen können Daten, die verschieden häufig von Clienten genutzt werden, mit mehreren verschiedenen Frequenzen rundgesendet werden. Die „Broadcast Disk"-Methode erzeugt damit in Wirklichkeit eine beliebige feine Hierarchie auf dem Broadcst-Medium. Diese Hierarchie kombiniert mit der Umkehr der traditionellen Beziehung zwischen Client und Server in einem Rundsendebasierenden System, wirft fundamentale neue Fragen für das Management des Cache

auf Seiten des Clients und des Daten-Prefetchings auf. Zum zweiten ist zu untersuchen, wie die Ressourcenspeicher auf Seiten des Clients beim Caching möglichst optimal ausgenutzt werden.

4 Client Cache Management

4.1 Unterschied zum klassischen Cache-Management

Im folgenden werden die Aspekte der Speicherung von rundgesendeten Daten im Cachespeicher von Clients beleuchtet. Ein zentrales Ergebnis wird dabei sein, daß sich die Rolle des Caching bei Rundsendeverfahren vom Caching in konventionellen Client-Server-Informationssystem grundlegend unterscheidet. In traditionellen Caching-basierten System halten die Clients die häufig angeforderten Daten im Cache, d.h. die Daten, die sehr wahrscheinlich in der nächsten Zukunft gebraucht werden, werden in den lokalen Speicher geholt. Wird diese Caching-Strategie in der Broadcast-Disk-Umgebung unverändert übernommen, kann dies zu unnötig langen Zugriffszeiten führen, womit die Leistung des Systems nicht mehr gewährleistet werden kann. Der Grund ist, daß in einer Multidisk-Rundsendung die Daten, die die Clients anfordern, nicht gleichmäßig auf dem Rundsendemedium verteilt sind. D.h. daß ein Zugriff auf die über ein Rundsendemedium gesendeten Daten zu variierenden Kosten führt, da die Zugriffszeiten auf unterschiedliche Daten variieren. Zudem kann die Zugriffsverteilung eines individuellen Clienten von der in einer Sendefolge berücksichtigten Zugriffsverteilung abweichen. Gründe hierfür sind:

- Die angenommene Zugriffsverteilung des Client kann nur ungenau geschätzt werden.
- Eine Client-Zugriffsverteilung kann sich mit der Zeit verändern.
- Die Server kann den abweichenden Zugriffsverteilungen anderer Clients eine höhere Anforderungspriorität geben.
- Die Mittelung der Anforderung an dem Server führt zu Abweichungen von den meisten Clients.

Wegen dieser Gründe sollten die Clients in einem Rundsendesystem nicht einfach die am häufigsten verwendeten Daten holen und speichern, sondern die Daten, die eine höhere lokale Zugriffswahrscheinlichkeit aufweisen als die in der Datensendefolgefrequenz der Rundsendung berücksichtigte. Das ist der fundamentale Unterschied zum Caching in traditionellen Client-Server Systemen. Gibt es z.B. nur ein Datum, eine Seite P, auf die nur von einem Client C zugegriffen wird, so wird sie auf einer langsameren Rundsendedisk gesendet. Wenn Client C das lange Warten vermeiden möchte, dann muß Client C die Seite P in den lokalen Cache aufnehmen. Eine Seite Q andererseits, auf die von den meisten Clienten (inklusive Client C) zugegriffen wird, wird auf einer sehr schnellen Disk gesendet. Damit reduziert sich die Wirkung der Aufnahme von P in den lokalen Cache. Die obigen Argumente führen zur Notwendigkeit, eine kostenbasierte Ersetzungsstrategie von angeforderten Daten bei Clients zu fahren. D.h. daß in einem unidirektionalen Verteilmedium eine kostenbasierte Ersetzungsstrategie zur Verwaltung des Caches zur

Anwendung kommen muß. Wenn ein Datum ersetzt werden soll, gilt es die variierenden Kosten zu betrachten, die entstehen, um das Datum anzufordern, sofern es sich zum Anforderungspunkt nicht im Cache befindet (Cache-Miss).

4.2 Betrieb

Eine Standard Seiten-Ersetzungsstrategie versucht im Cache-Speicher die Seite, die aktuell die niedrigste Zugriffswahrscheinlichkeit hat (LRU–approximates), durch eine neu geholte Seite zu ersetzen, wenn im Cache-Speicher freier Platz benötigt wird. Eine einfache kostenbasierte Ersetzungsstrategie ist, daß die Seite mit dem kleinsten Verhältnis zwischen ihren Zugriffswahrscheinlichkeit(P) und ihre Übertragungsfrequenz(X) ersetzt wird. Dieses Verfahren wird als PIX (P Inverse X) bezeichnet. Als Beispiel betrachten wir zwei Seiten A und B. A werde von einem Client zu 1% der Zeit gebraucht und werde auch zu 1% der Rundsendeszeit übertragen. B werde nur zu 0.5% der Zeit vom Client gebraucht und werde zu 0.1% der Rundsendeszeit übertragen. Der PIX-Wert von A ist 1 und damit kleiner als 5, dem PIX-Wert von B. Die PIX-Ersetzungsstrategie wird folglich die Seite A ersetzen, obwohl die Seite A eine doppelt so hohe Zugriffswahrscheinlichkeit wie die Seite B hat. Das PIX-Verfahren ist eine optimale Ersetzungsstrategie, basiert aber auf Informationen, die bei praktischen Anwendungen kaum bekannt sind, da man genau wissen muß, wie die Verteilung der Zugriffswahrscheinlichkeiten aussieht. Zudem müssen zum Ersetzungszeitpunkt die PIX-Werte aller im Cache befindlichen Seiten verglichen werden, um die zu ersetzende Seite zu bestimmen. Dies ist in der Implementierung zu teuer. Deshalb hat man im Rahmen des Broadcast-Disk Modells eine implementierbare Approximation von PIX entwickelt, die als LIX bezeichnet wird. LIX ist eine Modifikation der LRU-Strategie, die zusätzlich zu einer Schätzung der Zugriffswahrscheinlichkeiten die Sendefrequenz von Seiten berücksichtigt. Das LIX-Verfahren hat ein ähnliches Verhalten wie PIX, ist aber als Approximation in den zu erwartenden Zugriffszeiten etwas langsamer als PIX. LIX baut separat für jede logische Disk der Multidisk eine Liste der im Cache Seiten auf. Wenn eine Seite in den Cache geholt wird, dann wird sie in die Liste ihrer Disk eingeordnet. Die Bewertung basiert auf der durchschnittlichen „Zwischenzugriffszeit“ Zeit der Seite. Wenn eine Seite im Cache ersetzt werden soll, dann wird die Seite mit dem kleinsten L-Wert jeder Liste geprüft. Für jede Seite wird der LIX-Wert berechnet, indem L durch die Frequenz X der gesendeten Seite dividiert wird (X ist für jede Disk konstant). Die Seite mit dem kleinsten LIX-Wert wird als Ersetzungskandidat ausgewählt und ersetzt.

5 Prefetching

Im vorausgegangenen Kapitel wurden die Aspekte des Managements des Client-Cache in einer Broadcast-Disk-Umgebung diskutiert, sofern Seiten nur durch einer anforderungsgesteuerte Strategie in den Cache geholt werden. Eine alternative Methode ist es, daß Clients die Seiten vor dem Bedarf in den Cache holen (prefetch). Die Broadcast-Disk-Umgebung ist für prefetching besonders geeignet, da die Seiten kontinuierlich durch die

Rundsendungen den Clients angeboten werden. Damit ist es möglich, Seiten beim Prefetching in Vorwegnahme des erwarteten Zugriffsverhaltens in den Cache aufzunehmen. Im Gegensatz hierzu ist in traditionellen Systemen das Prefetching ein riskantes Vorgehen, da durch Prefetching eine Last auf gemeinsamen Ressourcen (z.B. Platten oder Netzwerken) erzeugt wird. Damit sind auch negative Auswirkungen auf die Leistung des Gesamtsystems möglich. Beim Prefetching in einer Broadcast-Disk-Umgebung werden nur lokale Client-Ressourcen benötigt, so daß das Risiko eines Leistungseinbruchs wesentlich kleiner als bei den meisten traditionellen Systemen ist.

5.1 Tag Team Caching

Prefetching bedeutet, daß spekulativ Seiten vom Verteilmedium in den Cache geholt werden. Das folgende als Tag-Team-Caching bezeichnete Beispiel motiviert den Einsatz von Prefetching in Verteilmedien.(siehe Abb.3.)

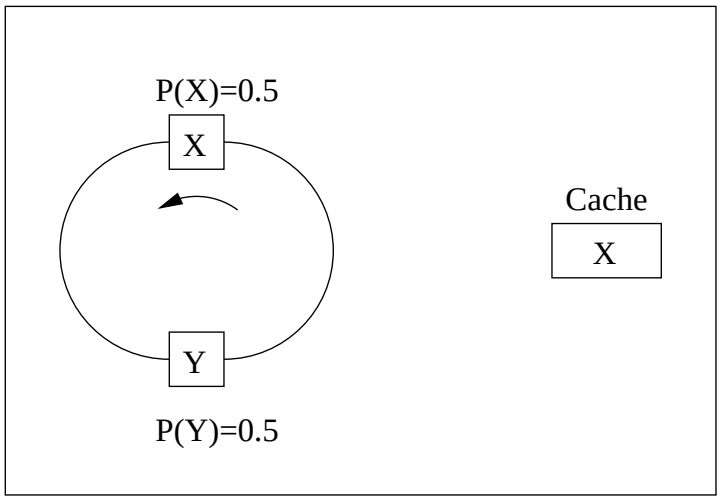

Abbildung39. Tag-Team-Caching.

Ein Client habe gleiche Zugriffswahrscheinlichkeiten für zwei Seiten A und B,
z.B. $P(A) = P(B) = 50\%$
und verfüge über Platz für eine Seite im Cache. Ferner seien die beiden Seiten mit gleichem Wiederholabstand in der Rundsendefolge plaziert. Bei einer anforderungsgesteuerten Ersetzungsstrategie wird Seite A in den Cache aufgenommen, nachdem Seite A angefordert und empfangen wurde. Nachfolgende Zugriffe auf Seite A können aus dem Cache heraus bedient werden. Wenn jedoch Seite B benötigt wird, so muß gewartet werden, bis Seite B empfangen wird und dann Seite A aus dem Cache zugunsten von Seite B verdrängt werden. Seite B bleibt im Cache, bis Seite A benötigt wird. Dann wird Seite B verdrängt. Die erwartete Verzögerung für den Zugriff auf eine beliebige Seite ergibt sich aus der Summe der erwarteten Verzögerungen für jede einzelen Seite gewichtet mit der Zugriffswahrscheinlichkeit. Die Zugriffswahrscheinlichkeit war mit je 50% vorgegeben und die erwartete Verzögerung beträgt die Hälfte der Periode um beide Seiten zu übertragen. Also

0,5*(P(A)C(A)+P(B)C(B))=0,5*(0,5*0,5+0,5*0,5)=0,25

Die Zugriffszeit für die anforderungsgesteuerte Strategie beträgt folglich ein Viertel der Periodenzeit zur Übertragung beider Seiten. Betrachtet sei demgegenüber eine einfache Prefetching-Strategie: Die Seite A wird in den Cache aufgenommen, wenn sie auf dem Medium erscheint und verbleibt dort bis die Seite B erscheint, die dann Seite A verdrängt, usw. Diese Strategie wird als Tag-Team-Caching bezeichnet, da sich die beide Seiten kontinuierlich ersetzen.

sum=0,5(0,5*0+0,5*0,25)+0,5(0,5*0+0,5*0,25)=0,125 [AFZ95]

Die Zugriffszeit für die Prefetching-Strategie beträgt ein Achtel der Periodezeit zur Übertragung beider Seiten und liegt damit bei der Hälfte der anforderungsgesteuerten Strategie. Diese Verbesserung resultiert aus den reduzierten Kosten im Falle eines Cache-Miss von 0,5 auf 0,25 der Periodelänge. Mit der anforderungsgesteuerten Strategie kann ein Cache-Miss während der gesamten Periodelänge auftreten.Die Kosten liegen deshalb bei 0,5 der Periodelänge. Bei der Prefetching-Strategie kann ein Cache-Miss nur während der Hälfte der Periode auftreten und ergibt sich deshalb zu 0,25 der Periode. Also reduziert Tag-Team die Kosten eines Cache-Miss, indem es sicherstellt, daß die Seite im Cache gehalten wird, für die die Verzögerungszeit bis zum Wiederempfang am größten ist.

5.2 Prefetching-Heuristik: PT

Im Rahmen des Broadcast-Disk-Modells wurde folgende erweiterte Ersetzungsstrategie für die Realisierung des Prefetching vorgeschlagen. Eine einfache Prefetching-Heuristik, die als PT bezeichnet wird, berechnet für jedes Datenobjekt, welches auf dem Medium erscheint, ob es sinnvoller ist, dieses Objekt zu speichern anstelle eines anderen Datenobjektes, das sich derzeit im Cache befindet. Der zu berechnende Wert wird als PT-Wert bezeichnet und ist das Produkt der Zugriffswahrscheinlichkeit für ein Datenobjekt und der verbleibenden Zeit, die vergeht, bis das Datenobjekt erneut empfangen werden kann.Wenn der PT-Wert des aktuell empfangenen Datenobjekts größer ist als der minimale Wert eines Datenobjekts im Cache, so wird das letzte Datenobjekt ersetzt und das empfangene Datenobjekt in den Cache gebracht.

Seite	Zugriffs-wahrscheinlichkeit	Sende-frequenz	PIX-Wert
A	P	2	P/2
B	P/2	2	P/4
C	P/2	1	P/2

Abbildung40. Vergleich von PT mit PIX.

PT berücksichtigt ähnlich wie PIX die Zugriffswahrscheinlichkeiten und eine Größe, die die Kosten für den Zugriff im Falle eines Cache-Miss beschreibt. Jedoch sind PT-Werte dynamisch d.h. mit jedem Zeitschritt ändert sich der Wert der Objekte. Wenn ein statisches Maß wie PIX für eine prefetch-gesteuerte Ersetzungsstrategie verwendet würde,

würde sich ein Cache sehr schnell mit den Datenobjekten füllen, die die größten PIX-Werte haben und sich dann stabilisieren. Im Vergleich dazu werden die Datenobjekte eines nach der PT-Strategie verwalteten Caches zyklisch durchgetauscht wie bei der Tag-Team Strategie.

Der PT-Wert eines Datenobjektes ändert sich ständig, da sich der Zeitparameter ständig ändert. Wenn ein Datenobjekt empfangen wird, ist sein PT-Wert am größten, denn zu diesem Zeitpunkt dauert es am längsten bis das Datenobjet erneut empfangen werden kann. Von da an verringert sich der PT-Wert ständig mit jedem Zeitschritt und erzeugt dadurch eine Sägezahnfunktion.

Zum Vergleich von PIX und PT betrachten wir folgendes Beispiel. Es sind drei Seiten A,B,C, wie in Abb.4. angegeben, zu übertragen. Die Seite A wird doppelt so häufig benötigt wie B und C. Außerdem werden A und B doppelt so häufig gesendet wie C. PIX liefert dann eine gleiche Wertung für A und C, die doppelt so hoch ist wie die für B. Ein Cache, der nach der PIX-Strategie verwaltet wird, bevorzugt also A und C. Demgegenüber verändert sich die Bewertung durch PT mit der Zeit (siehe Abb.5.). Wir nehmen an, daß A zu den Zeitpunkten 5, 15, 25, ...; B zu den Zeitpunkten 2, 12, 22, .., und C zu Zeitpunkten 4, 24, 44 gesendet wird. Wie in der Abb.5. zu sehen, steigt der PT-Wert direkt nachdem die Seite gesendet wurde und fällt dann stetig bis zum Nullpunkt kurz vor der erneuten Sendung. Die untere Linie gibt die potentiellen Ersetzungsopfer der PT-Strategie an. Im Vergleich zu PIX gibt es bei PT Zeiträume, in denen jede Seite zum Ersetzungsopfer wird.

Aufgrund dieses Verhaltens neigt PT dazu, Seiten immer dann im Cache zu halten, wenn ein Warten auf die Übertragung der Seite am teuersten kommen würde. Die Simulation im Rahmen der Broadcast Disks ergab, daß PT eine deutliche Verbesserung der Zugriffszeit gegenüber PIX erzielt.

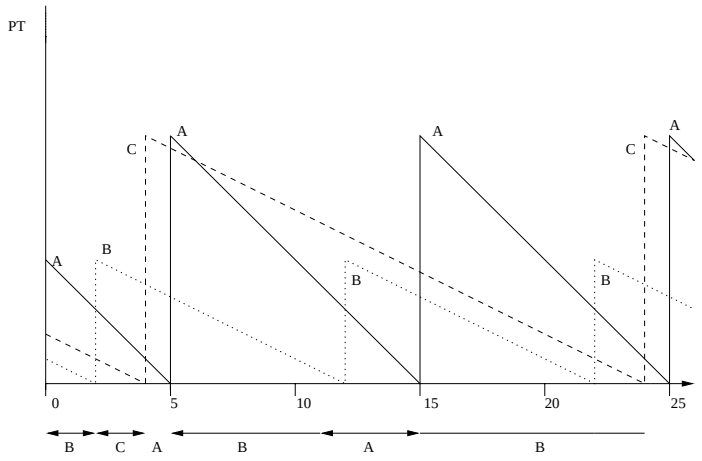

Abbildung41. Beispiel für das Sägezahn-Verhalten von PT.

Das Problem mit der PT-Strategie ist, daß sich der Zeitparameter ständig ändert und die Zugriffswahrscheinlichkeit für jedes Datenobjekt verschieden sein kann. Dadurch muß zu jedem Ersetzungszeitpunkt der PT-Wert für alle Seiten im Cache neu berechnet werden. Zudem ist die Zugriffswahrscheinlichkeit nie genau bekannt, sondern immer nur in Form

einer mehr oder weniger guten Schätzung gegeben. Im Rahmen des Broadcast-Disks-Projekts wird deshalb eine implementierbare Approximation von PT mit der Beziehung APT (Approximate PT) angegeben. Es hat sich gezeigt, daß APT das Verhalten von PT approximieren kann und in der Leistung nur geringfügig schlechter ist.

6 Fazit

In den vorhergenden Kapiteln werden die Verfahren „Broadcast-Disk“, „Client Cache Management“ und „Prefetching“ untersucht. Daraus ergeben sich die folgenden wichtigen Ergebnisse.

In den Rundsendeprogrammen ist die „flat-disk“ optimal, wenn die Zugriffswahrscheinlichkeiten gleich verteilt sind. Der „Skewed-“ und „Multidisk“-Ansatz sind besser als „flat-disk“, wenn die Zugriffswahrscheinlichkeiten ungleich verteilt werden. Der „Multidisk“-Ansatz ist immer besser als „Skewed“. Variieren die Zwischenankunftszeiten, dann erhöht sich die erwartete Zugriffszeit.

Eine Standard-Ersetzungsstrategie im Cache-Management ist das PIX. Das PIX Verfahren ist zwar optimal, benötigt aber bei der Implementierung großen Aufwand. Deshalb wird das LIX Verfahren empfohlen, das zusätzlich zu einer Schätzung der Zugriffswahrscheinlichkeiten die Sendefrequenz von Seiten berücksichtigt. Das LIX-Verfahren hat ein ähnliches Verhalten wie PIX, ist aber als Approximation in den zu erwartenden Zugriffszeiten etwas langsamer als PIX.

Eine erweiterte Ersetztungsstrategie ist Prefetching-Heuristik, als „PT“ bezeichnet. PT berücksichtigt ähnlich wie PIX die Zugriffswahrscheinlichkeiten und eine Größe, die die Kosten für den Zugriff im Falle eines Cache-Miss beschreibt. Jedoch sind PT-Werte dynamisch, d.h. mit jedem Zeitschritt ändert sich der Wert der Objekte. Das Problem mit der PT-Strategie ist, daß sich der Zeitparameter ständig ändert und die Zugriffswahrscheinlichkeit nie genau bekannt, sondern immer nur in Form einer mehr oder weniger guten Schätzung gegeben sind.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[ACFW93] H. Attiya, S. Chauduri, R. Friedman und J. L. Welch. Shared Memory Consistency Conditions for Non-Sequential Execution: Definitions and Programming Strategies. June/July 1993.

[AFZ95] S. Acharya, M. Franklin und S. Zdonik. Dissemination-Based Data Delivery Using Broadcast Disks. *IEEE Personal Communications* **2**(6), Dec 1995, Seite 50.

[Bac96] Svend Back. Bunte Bohnen: Einführung in die Programmierung mit Java. *c't* **7**(7), Juli 1996, Seite 258–262.

[Bag96] Jo Bager. Angebot und Abfrage: Datenbank/Web-Gateways machen Web-Server zu Informationsbrennpunkten. *c't* **6**(6), Juni 1996, Seite 268–272.

[BCK95] R. Bragodia, W. W. Chu und L. Kleinrock. Vision, Issues, and Architecture for Nomadic Computing. *IEEE Personal Communications* **2**(6), Dezember 1995.

[Beh96] H. Behme. Publishing für alle: Marktübersicht zur Bearbeitung von Web-Seiten. *iX* **9**(9), September 1996, Seite 84–90.

[BG95] Hendrik Berndt und Peter Graubmann. Service Architecture, Sevice Session and Service Federation in TINA-C. In *ISS 95 Proceedings*, 1995.

[BI94] D. Barbara und T. Imielinski. *Sleepers and Workaholics: Caching Strategies in Mobile Environments.* May 1994.

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

[CGH+95] D. Chess, B. Grossof, C. Harrison, D. Levine, C. Parris und G. Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications* **2**(5), Oktober 1995.

[CM95] Martin Chapman und Stefano Montesi. Overall Concepts and Principles of TINA. Tina baseline, TINA Consortium, 1995.

[CR94] Rong N. Chang und Chinya V. Ravishankar. A Service Acquisiton Mechanism for Server-based Heterogeous Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems* **5**(2), February 1994, Seite 154–169.

[Dav96] N. Davies. The Impact of Mobility on Distributed Systems Platforms. In *Proc. Int. Conf. on Distributed Platforms*, Dresden, Februar 1996.

[dlFW94] L.A. de la Fuente und Tony Walles. Management Architecture. Tina baseline, TINA Consortium, 1994.

[DMB+94] Fabrice Dupuy, Roberto Minerva, Jack Bloem, Heikki Hammainen und Juan Carlos Moreno. The TINA-C Architecture as Related to IN and TMN. In *Proceedings af the 3rd International Conference on Intelligence in Networks*, 1994.

[DNI95] Fabrice Dupuy, Gunnar C. Nilson und Yuji Inoue. The TINA Consortium: Towards Networking Telecommunications Information Services. In *ISS 95 Proceedings*, 1995.

[DPS+94] A. Demers, K. Peterson, M. Spreitzner, D. Terry, M. Theimer und B. Welch. The Bayou Architecture for Data Sharing among Mobile Users. In *1st IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.

[Fri94] R. Friedman. Consistency Conditions For Distributed Shared Memories. Israel Institute Of Technologie June 1994.

[Gel96] H.-W. Gellersen. WebComposition: Softwaretechnik für Entwicklung ud Wartung von Web-Anwendungen. In *Forschungs- und Arbeitsgebiete des Instituts für Telematik Interner Bericht Fakultät für Informatik, Universität Karlsruhe*, Nr. 36, Oktober 1996.

[Gil93] W. K. Giloi. *Rechnerarchitektur*. Springer-Verlag. 2. Auflage 1993.

[GKLS94] R. Gruber, F. Kaashoek, B. Liskov und L. Shrira. Disconnected Operations in the Thor Object-Oriented Database System. In *1st IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.

[HH95] P. Honeyman und L. Huston. Communications and Consistency in Mobile File Systems. *IEEE Personal Communications* **2**(6), December 1995, Seite 44.

[Hil95] S. Hild. Disconnected Operations for Wireless Nodes. In *ECOOP 95 Workshop on Replication and Mobility*, August 1995.

[HSW94] Y. Huang, P. Sistla und O. Wolfson. Data Replication for Mobile Computing. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, May 1994.

[IB93] T. Imielinski und B. Badrinath. Data Management for Mobile Computing. *Sgimod Record 22. Jahrgang* Band 1, March 1993.

[IB94] Tomasz Imielinski und B.R. Badrinath. *Communications of the ACM* **37**(10), Oct 1994, Seite 18.

[ISO95a] ISO/IEC. 10746-2 Part 2: Foundations. Technischer Bericht, ISO/IEC JTC1/SC21/WG7, 1995.

[ISO95b] ISO/IEC. 10746-3 Part 3: Architecture. Technischer Bericht, ISO/IEC JTC1/SC21/WG7, 1995.

[JB96] K. Obermayer J. Bager, S. Ehrmann. Seitenspinner: HTML-Werkzeuge für Anfänger und Profis. *c't* **7**(7), Juli 1996, Seite 190–196.

[Kel93] Ludwig Keller. Vom Nameserver zum Trader - Ein Überblick über Trading in verteilten Systemen. *Praxis der Information und Kommunikation* **16**(3), 1993, Seite 122–133.

[Kir96] C. Kirsch. Turbo Pascal der 90er: Suns Entwicklungsumgebung Java WorkShop. *iX* **11**(11), November 1996, Seite 58–62.

[KK94] Lea Kutvonen und Petri Kutvonen. *Broadening the User Environment with Implicit Trading*. Open Distributed Processing, II, IFIP, Elsevier Science B.V. 1994.

[Kot95] D. Kottmann. Datenmanagement im Mobile Computing. *HMD - Theorie und Praxis der Wirtschaftsinformatik* (184), Juli 1995.

[Kra96] Werner Krauß. Das Konzept im Spiegel der Praxis: C/S-Anwendungsprogrammierung mit Java (1). *Datenbank Fokus* **11**(11), November 1996, Seite 44–51.

[KS95] D. Kottmann und J. Sievert. Trading. In *Vorlesungsscript – Einsatz verteilter Systeme*. Universität Karlsruhe, Fakultät für Informatik, 1995.

[Kun96] M. Kunz. Neuzeit: Entwicklungsumgebung für Java: Symantec Café. *iX* **8**(8), August 1996, Seite ?

[Kut96] Lea Kutvonen. *Overview of the DRYAD trading system implementation.* Chapmann & Hall. 1996.

[Lie96] B. Liechti. Zweikomponentenkleber: Datenbanken ans Web mit dem Composite Objects System. *iX* **8**(8), August 1996, Seite 52–55.

[MM96] A. Schimpf M. Matzer. Produkte: Vom Modewort zur Schlüsseltechnologie. *ORACLE-Magazin* **4**(4), Oktober 1996, Seite 36–38.

[Mos93] David Mosberger. Memory Consistency Models – Operating Systems Review. **No. 1**(Vol. 27), January 1993.

[NDSC95] N. Natarajan, F. Dupuy, N. Singer und H. Christensen. Computational Modelling Concepts. Tina baseline, TINA Consortium, 1995.

[NG94] Y. Ni und A. Goscinski. Trader cooperation to enable object sharing among users of homogeneous distributed systems. *Computer Communications* **17**(3), March 1994, Seite 218–.

[Rud93] S. Rudkin. Templates, types and classes in open distributed processing. *BT Technology Journal*, Juli 1993.

[Sat96] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications* **3**(1), Februar 1996.

[Sch96] U. Schmitz. Goldfinger: CyberAgent als Entwicklungsumgebung für Web-Appilkationen. *iX* **9**(9), September 1996, Seite 66f.

[SKM+93] M. Satyanarananan, J. Kistler, L. Mummert, M. Ebling, P. Kumar und Q. Lu. Experience with Disconnected Operations in a Mobile Computing Environment. In *Usenix Symposium on Mobile and Location Independent Computing*, August 1993.

[SNKP95] M. Satyanarananan, B. Noble, P. Kumar und M. Price. Application-Aware Adaption for Mobile Computing. *ACM Operating Systems Review* **29**(1), January 1995, Seite 52.

[Tan95] A. S. Tanenbaum. *Distributed Operating Systems – Chapter.* Prentice Hall International Edition. Englewood Cliffs 1995.

[TB95] H.-J. Knobloch Th. Beth, H. Danisch. In *Seminar Netzwerksicherheit, Kryptographie und der ganze Rest Europäisches Institut für Systemsicherheit, Universität Karlsruhe*, Sommersemester 1995.

[TDP+91] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer und B. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings International Conference on Parallel and Distributed Information Systems (PDIS)*, December 1991.

[Win96] W. Winkhardt. Persönliche Note - Aktueller als Papier: Interaktiver Produktkatalog auf dem Web. *iX* **3**(3), Mrz 1996, Seite 116–122.

[WSD+95] O. Wolfson, P. Sistla, S. Dao, K. Narayanan und R. Raj. View Maintenance in Mobile Computing. *SIGMOD Record* **24**(4), Dezember 1995.

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