# Erweiterte Transaktionskonzepte in mobilen, verteilten Umgebungen

Studienarbeit

Universität Rostock, Fachbereich Informatik
Lehrstuhl für Datenbank- und Informationssysteme


vorgelegt von Michael Drews
geboren am 04.07.1973 in Rostock

Betreuer: Prof. Dr. A.Heuer
Dr. H. Meyer
Dipl.-Inf. G. Flach
Dipl.-Inf. A. Lubinski

# Inhaltsverzeichnis

# 1. Einleitung

Durch die Entwicklung drahtloser Kommunikation und mobiler Geräte ist mobiles Arbeiten möglich geworden und heute weit verbreitet. Damit sind neue Möglichkeiten für das rechnergestützte Arbeiten vorhanden. Das  Arbeiten am Rechner ist nicht mehr nur von bestimmten Punkten innerhalb des festen Netzwerkes möglich, sondern praktisch überall innerhalb des durch das Funknetz abgedeckten Gebietes. Dabei wird auch auf das feste Netzwerk zugegriffen, sollen Anwendungen die in festen Netzwerken vorhanden sind, weitergenutzt werden. Dies gilt auch für Datenbankanwendungen. Es soll möglich sein, dass wie im festen Netzwerk, Lese- *und* Updateoperationen auf dem Datenbestand ausgeführt werden können. Hierfür stehen im festen Netzwerk verschiedene Konzepte der Transaktionsverarbeitung zur Verfügung. Diese sind jedoch in mobilen Umgebungen aufgrund der dort vorhandenen Einschränkungen, wie z.B. langsamer, unzuverlässiger Datenübertragung, nicht ohne weiteres anwendbar. Es müssen an die Bedingungen mobiler Umgebungen angepasste Konzepte angewand werden.

Ziel dieser Studienarbeit ist es, einen Überblick über solche erweiterte Transaktionskonzepte in mobilen Umgebungen zu geben. Dabei wird zunächst auf den Aufbau und die Charakteristika mobiler Umgebungen, die Grundlagen der Transaktionsverarbeitung und auf die Anforderungen an Transaktionen in mobilen Umgebungen eingegangen. Versciedenen Konzepte zuer Transaktionsverarbeitung in mobilen Umgebungen werden im Kapitel 4 vorgestellt, bevor dargestellt wird, wie die Konzepte mit neu entwickelten kommerziellen Systemen umgesetzt werden und wie eine Umsetzung auf Standarddatenbankensystemen möglich ist. Eine Zusammenfassung in Kapitel 6 schließt die Arbeit ab.

# 2. Mobile Umgebungen

## 2.1. Aufbau mobiler Umgebungen

Mobile Umgebungen bestehen grundsätzlich aus zwei Arten von Geräten, den fest verbundenen (stationären) und den mobilen Einheiten. Mit mobilen Geräten ist ortsunabhängiges Arbeiten und Arbeiten während Ortsveränderungen möglich.
Die stationären Geräte sind über feste Leitungen miteinander verbunden und bilden so das fest verbundene Netzwerk (Festnetz). Zu den stationären Geräten gehören die Datenbankserver, auf denen DBMS (Datenbankmanagementsysteme) laufen. Sie können logisch miteinander verknüpft sein und so eine verteilte Datenbank bilden.
Einige der stationären Gerate, so genannte Mobile Suport Stationen (MSS), besitzen eine Funkschnittstelle, mit deren Hilfe eine Kommunikation mit den mobilen Geräten möglich ist
Mobile Support Stationen decken ein bestimmtes geographisches oder logisches Gebiet, sogenannte Zellen, ab. Ein mobiles Gerät kommuniziert mit der MSS, in deren Zelle es sich befindet.

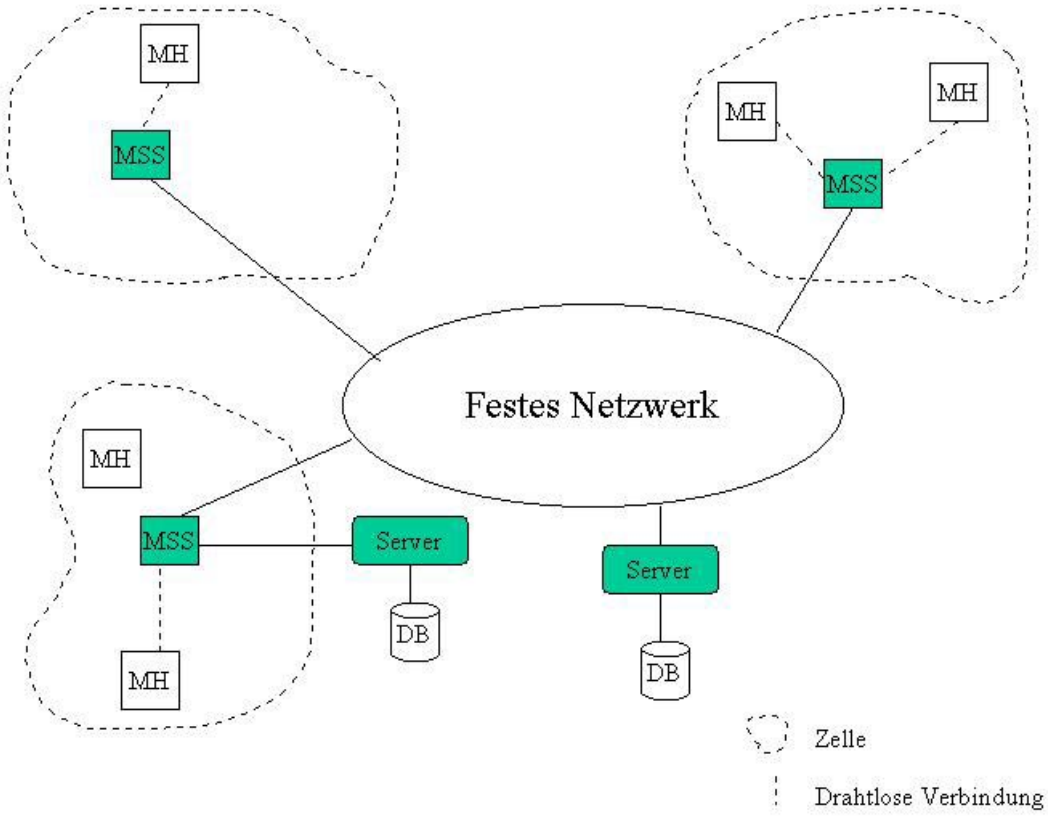



**Abbildung 1 : Mobile Umgebung**

Mobile Support Stationen und mobile Geräte bilden das mobile Netzwerk. Mobile Geräte sind tragbare Computer verschiedenster Art und Ausstattung, von leistungsstarken Laptops mit großem Farbdisplay über Handhelds oder Pocket-PCs mit kleinerem Display und geringerer Rechenleistung bis hin zu Palmtops mit sehr geringer Rechenleistung und Speicherkapazität und kleinem zum Teil monochromem Display. Die Funktion letzterer geht kaum über die der Ein- und Ausgabe von Daten hinaus. Durch die Entwicklung von WAP können auch Mobiltelefone zu den mobilen Endgeräten gezählt werden, wobei man hier zur Zeit noch von reinen Ein-/Ausgabegeräten sprechen muss. Allen mobilen Geräten gemeinsam ist die Stromversorgung über Batterie und damit eine eingeschränkte Nutzungsdauer.
Die häufigste Art der mobilen Kommunikation ist das Modem. Hierbei wird auf das Funktelefonnetz zurückgegriffen. Die Aufrechterhaltung der Verbindung von einer Zelle in eine andere wird hierbei vom Funknetzprotokoll geleistet. Bei der mobilen Kommunikation mittels Funktelefonmodem sind die Übertragungsraten wesentliche geringer als im festen Netzwerk. Eine Kommunikationsart mit höheren Übertragungsraten stellt das Wireless Lokal Area Network (WLAN) dar. Die Reichweite dieses Systems ist aber sehr beschränkt, so dass es nur für Hausnetzanwendungen in Frage kommt.

## 2.2. Charakteristika mobiler Umgebungen

Aus den oben genannten Gegebenheiten mobiler Umgebungen lassen sich nun Charakteristika solcher Umgebungen ableiten.

Eine Charakteristik mobiler Umgebungen ist das Vorhandensein unterschiedlichster mobiler Geräte mit unter Umständen geringer Leistungsfähigkeit, sowohl in bezug auf die Rechenleistung und Speicherkapazität als auch auf die Batterielebensdauer. Des weiteren sind mobile Geräte oft mit nur kleinen, zum Teil monochromen Displays ausgestattet, was die Darstellungsmöglichkeiten sehr einschränkt. Ein gemeinsames Merkmal aller mobilen Geräte, und damit ein Hauptmerkmal mobiler Umgebungen, ist die durch den Batteriebetrieb hervorgerufene sehr eingeschränkte Stromversorgung. Der Vorteil der Mobilität wird durch die eingeschränkte Batterielebensdauer, die zu häufigem Aufladen, Mitnahme zusätzlicher Akkus oder Batteriesparmaßnahmen, wie minimierter Nutzung, führen, eingeschränkt. Stromsparmaßnahmen sind nicht nur Aufgabe der Gerätesoftware, die Teilkomponenten wie Display oder Festplatte in Sparbetrieb versetzt. Auch Anwendungen für mobile Geräte sollten die eingeschränkte Batterielebensdauer berücksichtigen und versuchen, die Kommunikations-, Verarbeitungs- und Speicherkosten zu minimieren.
Energiesparmaßnahmen können auch auf der Kommunikationsebene durchgeführt werden. Da das Senden über Radiomodems ungefähr 10 mal mehr Strom verbraucht als das Empfangen können Kosten dadurch gesenkt werden, dass Sende- durch Empfangsleistungen ausgetauscht werden. So könnten die MSS periodisch Informationen versenden (Broadcasting), die sonst durch die mobilen Geräte wiederholt angefragt werden würden [ZZ 1998].

Eine weitere Eigenschaft mobiler Umgebungen ist das Auftreten von Verbindungsunterbrechungen. Sie sind zum einen auf die unzuverlässige drahtlose Verbindung oder das Verlassen des durch das Funknetz abgedeckten Gebietes zurückzuführen, aber auch darauf, dass mobile Geräte aufgrund der Stromversorgung über Batterie eine zeitlich begrenzte Laufleistung haben

und das Gerät so zu Zwecken der Stromeinsparung in einen Sparmodus versetzt oder ganz ausgeschaltet wird. Diese Verbindungsunterbrechung ist im Gegensatz zu der durch instabile Übertragung hervorgerufenen eine gewollte Unterbrechung und sollte nicht als Fehler angesehen werden. Es können Maßnahmen getroffen werden, die eine Wiederaufnahme der Verbindung erleichtern oder zur Vorbereitung des Arbeitens ohne Netzwerkverbindung dienen. Gewollte Verbindungsabbrüche können, außer zur Batterieeinsparung, auch zur Reduzierung der Verbindungskosten herbeigeführt werden. Dies ist besonders dann der Fall, wenn wie im Mobilfunknetz GSM für Verbindungsdauer und nicht für die übertragene Datenmenge gezahlt werden muss.

Unerwartete Verbindungsabbrüche und das Vorhandensein leistungsschwacher Geräte sowie eine durch den Batteriebetrieb hervorgerufene eingeschränkte Laufleistung sind die Hauptmerkmale mobiler Umgebungen.

# 3. Transaktionen

## 3.1. Grundlagen

Eine Aufgabe von Datenbanksystemen ist es, das gleichzeitige Arbeiten mehrerer Nutzer auf dem Datenbestand zu ermöglichen. Dabei soll der Mehrbenutzerbetrieb für den einzelnen transparent sein. Das bedeutet vor allem, dass Aufträge anderer Nutzer keinen Einfluss auf die Ausführung der eigenen Aufträge haben. Um dies zu erreichen, wurde das Konzept der Transaktionen eingeführt.

Transaktionen können als spezielle Applikationsprogramme, mit denen die User mit der Datenbank interagieren, angesehen werden [AE92]. Sie bestehen aus einer Menge von Schreib- und Leseoperationen auf Datenbankobjekten. Eine Datenbank besteht in dieser Sicht aus einer Menge von Objekten auf denen in einem atomaren Schritt gelesen oder geschrieben wird. Das Modell der Transaktionen als Folge von Lese- und Schreibaktionen auf Datenbankobjekten wird Read-Write-Modell genannt. Um einen transparenten und zuverlässigen Mehrbenutzerbetrieb zu ermöglichen, müssen Transaktionen traditionell vier Eigenschaften, den sogenannten ACID-Eigenschaften, genügen. Dies sind im Einzelnen:

- Atomarität (atomicity): die Transaktion mit ihren Einzeloperationen muss als eine Einheit gesehen werden, d.h. entweder alle Operationen der Transaktion werden ausgeführt oder keine

- Konsistenz (consistency): eine Transaktion überführt die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand, dh. nach der Ausführung der Transaktion müssen die Konsistenzbedingungen der Datenbank erfüllt sein

- Isolation (isolation): jede Transaktion läuft unabhängig von gleichzeitig im System ablaufenden Transaktionen ab, d.h. im Besonderen, dass sie keine Teilergebnisse anderer Transaktionen liest, sondern nur konsistente Ergebnisse anderer (abgeschlossener) Transaktionen

- Dauerhaftigkeit (durability): die Ergebnisse einer einmal abgeschlossenen Transaktion bleiben dauerhaft in der Datenbank unabhängig von eventuell auftretenden Fehlern

Um diese Eigenschaften bei der gleichzeitigen Ausführung mehrerer Transaktionen zu gewährleisten, werden zwei Protokollarten eingesetzt: Concurrency Control-Protokolle für die korrekte Ausführung der konkurrierenden Transaktionen und Recovery-Verfahren zur Gewährleistung der Fehlerfreiheit.

### 3.1.1. Concurrency Control

Um eine korrekte Abarbeitung der Transaktionen zu gewährleisten, werden Scheduler eingesetzt. Diese DBMS-Komponenten ordnen die eintreffenden Operationen verschiedener Transaktionen (Input-Schedule) so um, dass mit der neu entstandenen Verarbeitungsfolge (Output-Schedule) eine fehlerfreie Verarbeitung gesichert ist. Der Output-Schedule kann dann in der Datenbank bearbeitet werden. Das Schema des Scheduling Prozesses ist in Abbildung 2 dargestellt.

Zur Erzeugung eines korrekten Output-Schedules folgen Scheduler unterschiedlichen Strategien. Korrektheitskriterium ist aber immer Serialisierbarkeit. Ein Schedule ist serialisierbar, wenn es eine serielle Abarbeitung (Hintereinanderausführung) der beteiligten Transaktionen gibt, in der von den beteiligten Transaktionen jeweils die gleichen Werte wie bei der Abarbeitung des Schedules gelesen werden und die das gleiche Ergebnis liefert. Diese Eigenschaft von Schedules wird auch Sichtenserialisierbarkeit (View Serializability) genannt, da die Transaktionen die bei beiden Ausführungen die gleiche Sicht auf den Datenbankzustand haben. Die Überprüfung der Sichtenserialisierbarkeit ist NP-vollständig [Pap79]. Aus diesem Grund werden in Schedulern konfliktserialisierbare Schedules, eine Untermenge der sichten-serialisierbaren Schedules erzeugt. Ein Schedule ist konfliktserialisierbar, wenn es einen seriellen Schedule gibt, und in beiden die Reihenfolge von in Konflikt stehenden Operationen die gleiche ist. Dabei stehen zwei Operationen konkurrierender Transaktionen in Konflikt, wenn sie auf das gleiche Objekt zugreifen und wenigstens eine von ihnen eine Schreiboperation ist. Zur Einhaltung der (Konflikt-) Serialisierbarkeit kommen in Schedulern verschiedene Verfahren zur Anwendung. Eine allgemeine Klassifizierung dieser Verfahren ist schwierig, da von unterschiedlichen

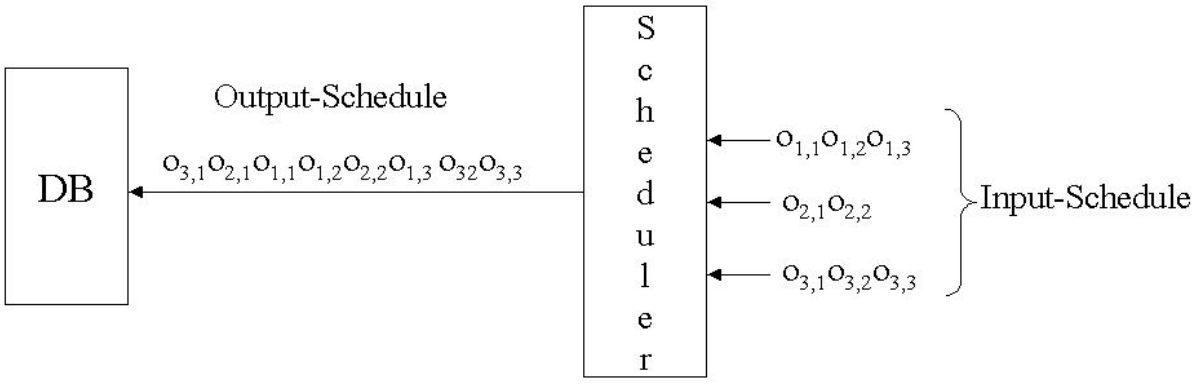



**Abbildung 2: Scheduling-Prozess**

Sichtweisen ausgegangen werden kann. So wird zwischen optimistischer und pessimistischer, aggressiver und konservativer sowie sperrender und nicht-sperrender Strategie unterschieden.

Bei **sperrenden Verfahren** müssen Transaktionen, um auf Objekten zu arbeiten, Sperren anfordern. Diese geben ihr dann das Recht auf den Objekten zu arbeiten, d.h. die Lese- bzw. Schreiboperation auf dem Objekt durchzuführen. Die Objekte sind für die Transaktion gesperrt. Es werden zwei Arten von Sperren unterschieden: exklusive Sperren (exklusive locks) für den schreibenden Zugriff und nicht-exklusive Sperren (shared locks) für den lesenden Zugriff. Ist auf einem Datenbankobjekt x eine nicht-exklusive Sperre, also eine Sperre für den lesenden Zugriff auf x gesetzt, können andere Transaktionen auch eine nicht-exklusive Sperre setzen und lesend auf x zugreifen. Ist ein Objekt exklusive gesperrt, können andere Transaktionen weder Sperren für den lesenden noch für den schreibenden Zugriff auf diesem Objekt setzen. Sie werden verzögert bis die Sperre freigegeben ist.

Das Standardsperrverfahren ist das **Zwei-Phasen-Sperr-Protokoll** (2PL 2-Phase-Locking). Beim 2PL darf eine Transaktion nach der Freigabe einer ihrer Sperren keine weitere Sperre anfordern. Durch diese Restriktion wird Konfliktserialiserbarkeit gewährleistet. Die zwei Phasen des 2PL sind die Wachstumsphase, in der die Sperren angefordert werden und die Schrumpfphase, in der die Operationen ausgeführt und die Sperren wieder freigegeben werden. Werden die Sperren sofort nach der Bearbeitung freigegeben, kann es zu kaskadierenden Abbrüchen kommen. Bricht die Transaktion in der Schrumpfphase ab, müssen auch die Transaktionen, die auf den freigegebenen Objekten Sperren gesetzt haben abgebrochen werden, da sie ungültige Werte gelesen haben. Um kaskadierende Abbrüche zu verhindern, wird das sogenannte **strikte** Zwei-Phasen-Sperr-Protokoll (S2PL) eingesetzt. Dabei werden alle Sperren erst nach Beendigung der gesamten Transaktion freigegeben. Ein weiteres Problem des 2PL ist die Möglichkeit des Auftretens von Dead-Locks. Zwei Transaktionen, die nach einer Lesesperren auf einem Objekt eine Schreibsperren auf dem Objekt setzen wollen warten gegenseitig darauf, dass die andere Transaktion die Lesesperre freigibt. Eine Lösung für dieses Problem bietet das **konservative** 2PL (C2PL), bei dem zu Beginn der Transaktion alle

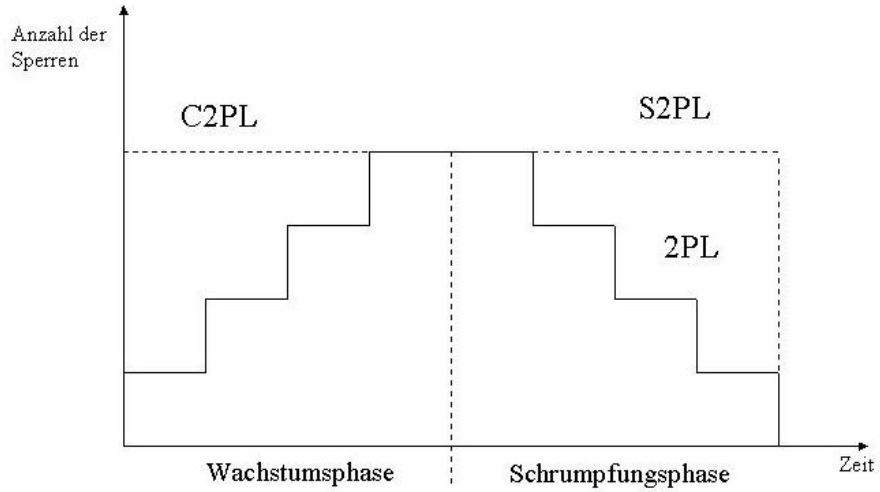



**Abbildung 3: Zwei-Phasen-Sperr-Protokolle**

benötigten Sperren zusammen angefordert werden. Wartende Transaktionen können so keine Sperren halten. Abbildung 3 zeigt die drei Varianten des 2PL im Vergleich.

**Time-stamp-ordering**-Verfahren arbeiten mit Zeitstempeln für Transaktionen und Datenbankobjekte. Jeder Transaktion (und ihren Operationen) wird ein Zeitstempel zugewiesen. Datenbankobjekte erhalten den Zeitstempel, der Operation, die als letzte auf ihnen gearbeitet hat. Stehen zwei Operationen in Konflikt, so muss die Operation mit dem kleineren Zeitstempel, d.h. die Operation deren Transaktion früher begonnen hat, vor der anderen ausgeführt werden. Ist eine Operation „zu spät“ [VG93] , d.h. ist eine mit der Operation in Konflikt stehende Operation mit größerem Zeitstempel schon ausgeführt worden, so kann die Operation nicht durchgeführt werden. Die zugehörige Transaktion muss zurückgesetzt, und mit einem neuen (größeren) Zeitstempel neu gestartet werden.

Beim **Serialisierungsgraphtest** wird ein so genannter Serialisierungsgraph aufgebaut, der als Knoten die zu serialisierenden Transaktionen enthält und die Kante $T_i \rightarrow T_j$, wenn $T_j$ eine Operation ausführt, die mit einer Operation von $T_j$ in Konflikt steht und vor dieser ausgeführt wird. Der Graph wird online aufgebaut und solange er azyklisch ist, werden die Operationen der Transaktionen ausgeführt. Anderenfalls wird die Transaktion, die den Zyklus verursacht, abgebrochen [AE92]. Serialisierungsgraphentest und Zeitstempelverfahren sind nicht-sperrende Verfahren.

Die oben genannten Verfahren zählen zum pessimistischen Concurrency Control. Bei diesem Ansatz wird davon ausgegangen, dass Konflikte auftreten können. Es wird für jede Operation des Input-Schedules geprüft, ob sie konfliktfrei ausgeführt werden kann. Einem anderen Ansatz folgt das **optimistische Concurrency Control**. Hier wird davon ausgegangen, dass Konflikte selten sind. Die Operationen werden in der Reihenfolge, in der sie beim Scheduler ankommen, weitergegeben und ausgeführt. Optimistisches Concurrency Control erfolgt in drei Phasen: der Read-Phase, der Validation-Phase und der Write-Phase (siehe Abbildung 4) In der Read Phase werden die Operationen der Transaktion ausgeführt. Dabei werden die Write-Operationen nur im jeweiligen Pufferbereich ausgeführt. Ist eine Transaktion beendet, so muß sie validiert werden (Validierungs-Phase). Es wird überprüft, ob die Ausführung

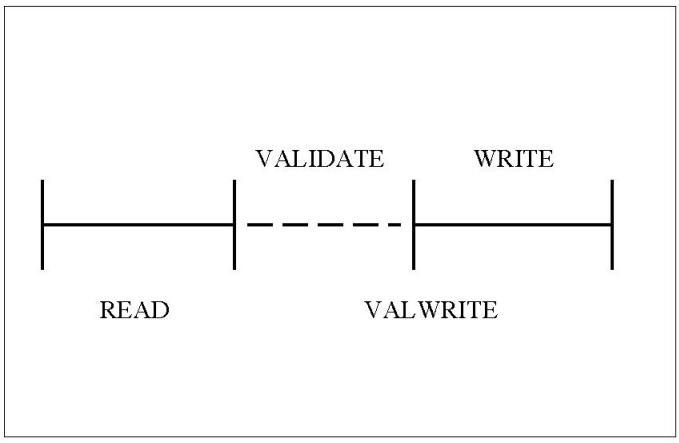



**Abbildung 4: Phasen des optimistischen Concurrency Control**

konfliktserialisierbar war. Erst wenn dies der Fall war, wird das Ergebnis vom Puffer in die Datenbank übertragen (Write-Phase). Traten Konflikte auf, so wird die Transaktion zurückgesetzt und muss wiederholt werden. Validation- und Write-Phase werden oft als Einheit betrachtet. Man spricht dann von der Valwrite-Phase. Bei der Validierung von Transaktionen unterscheidet man zwischen Vorwärtsvalidierung (forward-oriented cerification, FOC) und Rückwärstvalidierung (backward-oriented certification ,BOC).

Bei der Rückwärtsvalidierung wird in der Valwrite-Phase einer Transaktion $t_i$ überprüft, ob sich das Read-Set von $t_i$ (alle Obkekte, die von $t_i$ gelesen wurden) mit dem Write-Set (Menge von Objekten auf denen geschrieben wurde) von Transaktionen, die parallel aktiv sind und die Read-Phase vor $t_i$ abgeschlossen haben, überschneidet. Ist dies der Fall muss die Transaktion $t_i$ zurückgesetz werden.

Vorwärtsvalidierende Verfahren überprüfen, ob sich das Write-Set der zu validierenden Transaktion $t_i$ mit dem Readset von Transaktionen, die die Read-Phase noch nicht beendet haben überschneidet. Ist die Schnittmenge leer, so können die Ergebnisse von $t_i$ in die Datenbank geschrieben werden. Ansonsten kann entweder $t_i$ oder die in konflikt stehende Transaktion abgebrochen werden. Das ist z.B. sinnvoll, wenn $t_i$ eine lange Transaktion ist und viele Schritte wiederholt werden müssten. Vörwärstvalidierung ist damit in Konfliktsituationen flexibler einzusetzen als Rückwärtsvalidierung.

### 3.1.2. Recovery

Recovery-Protokolle werden genutzt, um die Fehlersicherheit zu garantieren. Dabei sind Transaktionsfehler und Systemfehler zu unterscheiden.
Wenn eine Transaktion aufgrund eines Fehlers (Abbruch durch Nutzer) abgebrochen wird, so müssen alle Effekte der Operationen der Transaktion auf die Datenbank rückgängig gemacht werden. Dies geschieht durch das Rücksetzen aller von der Transaktion geschriebenen Objekte auf die Werte vor der Ausführung (before images), welche in einen Logfile im nicht flüchtigen Speicherbereich abgelegt sind. Außerdem müssen die Effekte auf Transaktionen rückgängig gemacht werden, die Werte von Objekten gelesen haben, die durch die abgebrochene Transaktion geschrieben wurden. Auch diese Transaktionen werden zurückgesetzt. Es kann zu kaskadierenden Abbrüchen kommen, d.h. eine Transaktion bricht ab und darauf hin werden Transaktionen, die von ihr gelesen haben und auch die Transaktionen, die von den lesenden Transaktionen gelesen haben, abgebrochen. Um dies verhindern werden strikte Protokollen eingesetzt, die Sperren erst dann freigeben, wenn die Transaktion beendet ist.
Tritt ein Systemfehler auf, so müssen Transaktionen die erfolgreich beendet wurden, deren Ergebnisse aber noch nicht in den stabilen Speicherbereich der Datenbank übertragen wurden erneut ausgeführt werden (redo). Außerdem müssen Transaktionen, die noch nicht beendet wurden, von denen aber schon Teilergebnisse in die stabile Datenbank geschrieben wurden , rückgägig gemacht werden (undo), da sonst nicht freigegebene Werte in der Datenbank vorhanden wären.

## 3.2. Forderungen an Transaktionsverarbeitung in Mobilen Umgebungen

Die vorgestellten Transaktionskonzepte wurden für den Einsatz im festen Netzwerk entwickelt. Dabei wird die Zuverlässigkeit des Netzes und eine permanent vorhandene, ausreichende Leistung der Client-Rechner vorausgesetzt. Wie in Abschnitt 2.2. gezeigt, sind diese Voraussetzungen nicht ohne weiteres in mobile Umgebungen übertragbar. Abgeleitet aus den Eigenschaften mobiler Umgebungen ergibt sich die Notwendigkeit neuer Transaktionskonzepte. Dabei zeigt sich, dass die Notwendigkeit besteht die traditionellen ACID-Eigenschaften aufzuweichen, um Konzepte zu entwickeln [WC99], die den Anforderungen mobiler Umgebungen genügen.

Um leistungsschwache Geräte zu unterstützen, muss es die Möglichkeit geben, die Verarbeitung zu teilen und so sowohl den Aufwand zur Verarbeitung als auch zur Kommunikation zu begrenzen[Chr93]. Dazu müssen Transaktionen geteilt und die einzelnen Teiltransaktionen getrennt verarbeitet werden können. Die all-or-nothing-Eigenschaft würde dadurch verletzt werden.

Mobile Geräte sind nicht immer mit dem festen Netzwerk verbunden. Um die Verarbeitung auch im verbindungslosen Zustand zu unterstützen sollten Teilergebnisse sichtbar gemacht werden. Die gilt sowohl für Ergebnisse auf dem mobilen Gerät, das so mit diesen Teilergebnissen weiterarbeiten kann, als auch für Teilergebnisse im festen Netzwerk. Hier müssen dann andere Transaktionen nicht darauf warten, dass sich das mobile Gerät wieder verbindet. Diese Zusicherung lokaler Autonomie in der Transaktionsverarbeitung ermöglicht kurze Antwortzeiten im verbindungslosen Zustand. Die Isolationseigenschaft sowie möglicherweise auch die Konsistenzeigenschaft werden dadurch jedoch verletzt. Es ist also zusätzlich notwendig weichere Konsistenzbedingungen zuzulassen und die Behandlung partieller Fehler zu ermöglichen.
Die zeitweiligen asynchronen Verbindungsunterbrechungen in mobilen Umgebungen wirken sich negativ auf Verarbeitungstechniken mit sperrender Strategie aus. Verbindungslose Rechner können Sperren auf Objekten nicht freigeben und blockieren so Operationen anderer Transaktionen, die auf die gesperrten Objekte zugreifen wollen.

Die in der traditionellen Transaktionsverarbeitung angewandten Konzepte gehen von Transaktionsdauern im Sekundenbereich aus. Transaktionen in mobilen Umgebungen sind durch Verbindungsunterbrechungen und geringere Übertragungsraten den langen Transaktionen zuzuordnen. Diese müssen unterstützt werden, etwa durch Übernahme der Konzepte aus Entwurfsumgebungen. Im Falle einer gewollten Verbindungsunterbrechung können Vorbereitungen getroffen werden, die die Ausführung der langen Transaktion unterstützen.

Das mobile Netzwerk arbeitet unzuverlässig. Das gilt sowohl für die Datenübertragung, die langsam und störanfällig ist, als auch für die Geräte selbst, die auf Batterieleistung angewiesen sind und daher nicht immer die volle Leistungskraft haben. Transaktionsverarbeitung in mobilen Umgebungen muss Fehler, die sich aus diesen Einschränkungen ergeben, tolerieren oder behandeln können. Dazu zählt auch, dass neue Recoverystrategien eingesetzt werden, die die Besonderheiten mobiler Umgebungen berücksichtigen. So muss aufgrund der hohen Kosten mobiler Kommunikation, eine Strategie eingesetzt werden, die mit geringem Kommunikationsaufwand auskommt. Andererseits steigt der Kommunikationsbedarf, je autonomer die mobilen

Geräte agieren. Hier muss zwischen Fehlertoleranz und Kommunikationskosten abgewogen werden.

Insgesamt sind die speziellen Forderungen an Transaktionskonzepte in mobilen Umgebungen von Gewichtung der Einschränkungen in mobilen Umgebungen abhängig.

- Wanderung des mobilen Geräts während er Verarbeitung?
  Bei Transaktionsverarbeitung während einer Ortsveränderung kann es vorkommen, dass das mobile Gerät die Zelle verlässt, in der die Transaktion begonnen hat. Es gelangt in eine neue Zelle, für die eine andere Supportstation zuständig ist. Für den Übergang von einer Zelle/Supportstation zur nächsten müssen Handoff-Protokolle/Proxies eingesetzt werden, damit die neue Supportstation die Transaktion weiter unterstützen kann. In der verbreitetsten Art der mobilen Kommunikation, dem Mobilfunknetz, wird der Übergang von einer Zelle zur nächsten durch das Netzprotokoll geleistet und ist für den Nutzer dieser Dienste transparent.

- Rechenleistung der mobilen Geräte bestimmt Ort der Verarbeitung
  Um Transaktionen auch im verbindungslosen Zustand zu bearbeiten, müssen Daten aufdem mobilen Rechner gehalten und Datenbankoperationen lokal verwaltet werden. Hierfür muss sowohl ausreichend Speicherplatz als auch genügend Rechenleistung vorhanden sein. Beides ist auf Kleinstgeräten nicht der Fall. Die Verarbeitung der Transaktionen muss hier ganz (transaction shipping) oder teilweise(transaction splitting) auf einem Server/Supporthost im festen Netzwerk stattfinden.

- Autonomie des mobilen Gerätes vs. Aktualität/ Verbindungskosten vs. Konsistenz
  Hat das mobile Gerät ausreichende Rechenleistung und Speicherkapazität, um Daten lokal zu halten und zu verwalten stellt sich eine weitere Frage. Ist es immer notwendig Transaktionen mit Verbindung zum Server und damit auf aktuellen Daten durchzuführen, oder soll das mobile Gerät in der Lage sein, Transaktionen autonom ohne Verbindung zum Server auf den lokalen (replizierten) Daten zu verarbeiten. Vorteil ersterer Variante ist, dass die Daten in konsistentem Zustand gehalten werden. Im zweiten Fall werden Kommunikationskosten eingespart, was auf Kosten der Konsistenz geschieht. Die Art Entscheidung richtet sich hier nach der Art der Anwendung. In bestimmten Anwendungsszenarien ist völlige Konsistenz der Daten nicht notwendig oder Konflikte die zu Konsistenzverletzungen führen gering, etwa bei selten oder nur privat genutzten Daten. In anderen Fällen, wie Geschäftsanwendungen, ist Konsistenz absolut erforderlich, so dass hier der erste Ansatz zu wählen ist.

# 4. Transaktionskonzepte für mobile Umgebungen

Wie im vorhergehenden Abschnitt gezeigt, erfordern es die Charakteristika mobiler Umgebungen, in diesen Umgebungen erweiterte Transaktionskonzepte einzusetzen. Da die verschiedenen Beschränkungen mobiler Umgebungen unterschiedlich stark gewichtet werden können, gibt es auch eine größere Zahl von Konzepten zur Verarbeitung von Transaktionen im mobilen Umfeld ([Mad98],[WC95]). Eine Klassifikation der Konzepte erscheint, wie auch schon bei den klassischen Verfahren als schwierig, da verschiedene Sichtweisen vorhanden sind. Ich habe mich für eine grobe Zweiteilung entschieden. Zur ersten Gruppe zählen Transaktionskonzepte, die den Zugriff auf die Daten für die Transaktionen auf zeitliche oder semantische Art begrenzen, die Daten aber in konsistentem Zustand halten. Autonomie verbindungsloser Rechner wird hierbei nicht durch Konsistenzverlust „erkauft“, sondern durch beschränkten Zugriff auf die Daten. Im Abschnitt 4.2. werden dann Konzepte vorgestellt, die die Validität der Daten einschränken und so versuchen, Verbindungsabbrüche zu tolerieren und autonomes Arbeiten zu ermöglichen.

## 4.1. Beschränkung des Zugriffs auf Daten

### 4.1.1. Semantische Modelle

Eine Möglichkeit, die autonome Verarbeitung von Transaktionen durch mobile Geräte zu ermöglichen, ist die Ausnutzung von Semantikeigenschaften von Operationen auf Datenbankobjekten. Die allgemein am häufigsten genutzte Semantikeigenschaft von Operationen ist die **Kommutativität**. Wenn zwei Operationen kommutativ sind, sind ihre Auswirkungen auf den Zustand eines Objektes und die zurückgelieferten Ergebnisse unabhängig von der Reihenfolge ihrer Ausführung. Walborn und Chrysanthis [WC95] unterscheiden Operationen die unabhängig vom Wert der Objekte, auf denen sie arbeiten, kommutativ sind (z.B. zwei Inkrementoperationen) und solchen mit zustandsbasierter Kommuntativität (state based comutativity), die nur kommutativ sind wenn die Operanden bestimmte Zustände haben. Ein Beispiel hierfür ist die push-Operation auf einem Stack, die mit identischen Eingabewerten kommutativ ist mit unterschiedlichen jedoch nicht. Auch Ausgabewerte können zur Bestimmung von zustandsbasierter Kommutativität genutzt werden. So sind zwei push-Operationen, die „stack overflow“ zurückliefern, kommutativ.

Kommutativität erleichtert den gleichzeitigen Zugriff auf gemeinsam genutzte Daten. Wenn in einer Datenbank alle Operationen untereinander kommutativ sind, so ist für das mobile Gerät autonome Verarbeitung auf den lokal gehaltenen Daten ohne Serververbindung möglich. In realen Anwendungen sind jedoch nur wenige Operationen kommutativ. Kommutativität kann aber ausgenutzt werden um bei Transaktionsverarbeitung mit optimistischen concurrency control die Anzahl der validierten Transaktionen zu erhöhen [WC95].

### 4.1.2. Escrow Methode

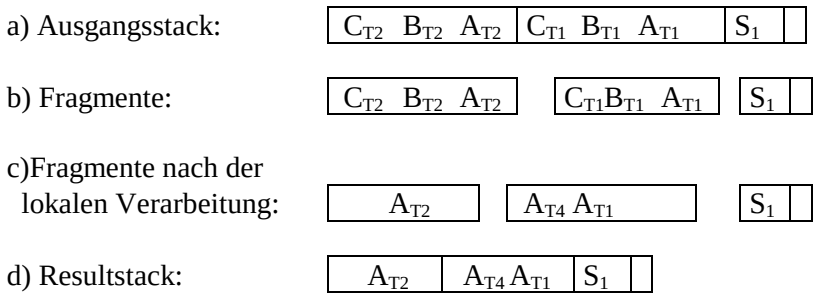
Mobile Geräte müssen, damit sie Transaktionen auch im verbindungslosen Zustand bearbeiten können, Datenbankobjekte lokal speichern und verwalten. Traditionell werden Replikate der Datenbankobjekte im Cache des mobilen Gerätes gehalten. Diese können sehr groß sein. Das Senden solch großer Objekte ist in mobilen Umgebungen durch die langsame Verbindung und die hohen Kosten problematisch. Durch die geringe Speicherkapazität mobiler Geräte kann außerdem nur eine geringe Anzahl großer Datenbankobjekte lokal gehalten werden. In mobilen Umgebungen ist es daher besonders wichtig, dass die Granularität der Daten so fein wie möglich ist. Mit der **Escrow Transaction Methode** [ONe86] werden unter Ausnutzung des Wissens über Objektorganisation große Datenbankobjekte in kleinere fragmentiert. Die Escrow Methode arbeitet mit Aggregatwerten. Ausgenutzt wird dabei, dass Aggregatwerte numerische Werte sind, die eine Menge austauschbarer Objekte repräsentieren, wie z.B. Eintrittskarten. Diese Menge austauschbarer Objekte kann in Teilmengen zerlegt werden, die auf einzelne mobile Geräte je nach Bedarf aufgeteilt werden können. Führen die mobilen Geräte nur die kommutativen Operationen decrement und increment aus und werden Rahmenbedingungen, die von Integritätsbedingungen, wie z.B. keine negativen Werte, eingehalten, so kann die Konsistenz der Daten gesichert werden. Die Operationen können lokal bearbeitet werden und bei der nächsten Verbindung mit dem Server unter Einhaltung globaler Serialisierbarkeit in die Datenbank eingefügt werden.

Walborn und Chrysanthis führen in [WC95] die Klasse der **fragmentierbaren Objekte** (fragmentable objects) ein, um die Anwendung der Escrow Methode auf diese zu erweitern. Fragmentierbare Objekte können auf verschiedene Seiten (mobile Geräte) aufgeteilt, auf jeder Seite autonom verarbeitet und dann auf semantisch konsistente Weise wieder zusammengefügt werden. Zu den fragmentierbaren Objekten zählen, neben den eben genannten Aggregatwerten, Mengen, Stacks und Warteschlangen (queues).

**Mengen**: Mengen sind eine Sammlung von Objekten, deren relative Ordnung keine Rolle spielt. Sie können in Teilmengen zerlegt werden, die dann in beliebiger Reihenfolge wieder zusammengefügt werden. Um eine Teilmenge zur lokalen Verarbeitung anzufordern müssen mobile Geräte einen Wertebereich angeben, in dem sich die Elemente der Teilmenge befinden. Die erlaubten Operationen, Test auf Vorhandensein und Einfügen neuer Elemente, beschränken sich auf den gewählten Wertebereich. Dadurch wird die Konsistenz beim Zusammenfügen der Teilmengen gewährleistet. Da der Speicherbedarf von der Anzahl der Elemente in der Teilmenge abhängt, unterstütz dieser Ansatz außer der autonomen Verarbeitung auch die Forderung nach minimalem Speicherbedarf in mobiler Geräte.

**Stacks**: Stacks sind Listen von Elementen die in LIFO (last-in, first-out) Ordnung organisiert sind. Operationen auf Stacks sind push (Ablegen eines Elements an der Spitze des Stacks) und pop (Entnahme eines Elements von der Spitze des Stacks). Ein Stackfragment ist ein Teil eines Stacks in dem nur eine Transaktion Elemente abgelegt hat. Stacks sind physikalisch fragmentiert. Jeder mobile Host kann ein oder mehrere Stackfragmente lokal verwalten. Wenn die Stackfragmente zum Server zurück transferiert werden, können sie dort wieder in den Ausgangsstack eingeordnet werden.

In Beispiel 1 ist ein mögliches Szenario für die Fragmentierung von Stacks angegeben. Der Ausgangsstack entstand durch jeweils drei push-operationen der Transaktionen T1 und T2 auf einem Stack mit dem Inhalt S1. Der Stack kann danach in Fragmente zerlegt werden, die auf die mobilen Geräte MH1 und MH2 zur lokalen Verarbeitung verteilt werden können. Wenn auf dem mobilen Gerät dass die Transaktion T2 ausführte zwei Pop-Operationen (als Transaktion T3) und auf dem anderen Gerät zwei Pop- Operationen und eine Pussh-Operation (als Transaktion T4) ausgeführt werden, so erhalten wir die unter c) angegebenen Teilfragmente. Diese können dann wieder zu einem einzelnen Stack zusammengefaßt werden (d). Der so entstandene Stack entspricht dem durch die serielle Abarbeitung T1→T4→T2→T3 entstehenden . Die Operationen von T3 und T4 können aber unabhängig voneinander und somit auch im verbindungslosen Zustand ablaufen. Stacks sind umordbar, d.h. Fragmente können umgeordnet werden, um längere zusammenhängende Segmente zu bilden und so Datenanforderungen der mobilen Geräte zu erfüllen.




**Beispiel 1:Fragmentierung von Stacks**

**Queues:** Queues sind Listen von Objekten, die in FIFO (first-in, first-out) Ordnung organisiert sind. Die Operationen auf Queues sind Einfügen eines Elementes am Ende der Queue und Entnehmen eines Elementes vom Anfang der Queue. Ein Queuefragment ist wie bei Stacks ein Teil der interdependente Daten enthält. Alle Elemente, die von einer einzelnen Transaktion eingefügt wurden, sind interdependent. Queuefragmente in denen die Originalelemente vollständig aufgebraucht wurden können auf konsistente Weise in die Ausgangsqueue eingefügt werden.

Mit der Fragmentierung können zwei Anforderungen, die mobile Umgebungen stellen, efüllt werden. Durch die Fragmentierung entstehen kleinere Teile potentiell grosser Objekte, was im Hinblick auf den geringen Speicherplatz mobiler Geräte interresant ist. Außerdem können die einzelnen Fragmente unabhängig voneinander bearbeitet werden, also speziell auch im verbindungslosem Zustand.

### 4.1.3. Erweiterung pessimistischer Concurrency Control

Pessimistische Verfahren arbeiten wie schon in Abschnitt 2.1. gezeigt mit Sperren auf Objekten. Verbindungslose Geräte können die Sperren nicht wieder freigeben. Damit werden Transaktionen, die auf diese Objekte warten blockiert. Eine Lösung des Problems der Blockierung durch nicht freigegebene Sperren ist das Einführen von Zeitintervallen. Transaktionen besitzen die Sperren nur einen bestimmten Zeitraum lang, in dem die Bearbeitung des Objektes abgeschlossen und die Änderungen zum Server propagiert werden müssen. Droht das Intervall abzulaufen, so kann beim Server/Transaktionsmanager eine Verlängerung beantragt werden. Ist die Zeitspanne abgelaufen, so wandelt sich das pessimistische concurrency control in ein optimistisches. Die Ergebnisse der Transaktion müssen dann zum Server propagiert und dort validiert werden.

Momin und Vidyasankar führen in [MV99] ein Transaktionsverarbeitungsmodell ein, dass neben Zeitintervallen verschiedene Pessimistikgrade (degrees of pessimism) benutzt, um die Einschränkungen oder Verzögerungen, die sperrende Ansätze auf die Ausführung konkurrierender Transaktionen haben, zu minimieren. Die Pessimistikgrade stellen eine Integration von optimistischer und pessimistischer concurrency control auf der Konfliktebene dar. Zwei Transaktionen stehen in Konflikt, wenn der Durchschnitt des write-sets der einen Transaktion mit dem read-set (RW-Konflikt) oder mit dem write-set (WW-Konflikt) der anderen Transaktion nicht leer ist. Um auf einem Objekt zu schreiben (lesen) muss eine Transaktion vorher eine Zugriffserlaubnis anfordern. Die Zugriffsberechtigungen variieren in ihrem Pessimistikgrad. Es gibt die **P**riority- und die **N**o-**P**riority-Berechtigung, was einer Beachtung oder Nichtbeachtung der Konflikte und damit einem pessimistischen bzw. optimistischen Zugriff auf die Daten entspricht. Damit gibt es dann zwei Berechtigungen für den lesenden Zugriff und vier Berechtigungen für den schreibenden Zugriff.

Für den lesenden Zugriff sind die Berechtigungen P_RW und NP_RW. Dabei bedeutet P_RW auf einem Objekt, dass der diese Berechtigung haltenden Transaktion zugesichert wird, dass während des Zugriffs keine RW-Konflikte auftreten. Es darf also keine andere Transaktion schreibend auf das Objekt zugreifen.

Die Berechtigung für den schreibenden Zugriff sind (P_RW P_WW), (P_RW NP_WW), (NP_RW P_WW) und (NP_RW NP_WW). Eine (P_RW NP_WW) Berechtigung garantiert, dass keine andere Transaktion liest, aber nicht, dass keine andere Transaktion schreibt. Das bedeutet, dass eine Transaktion, die diese Berechtigung auf einem Objekt x hat, warten muss, bis Transaktionen, die auf x geschrieben haben beendet sind.
Die Berechtigungen werden auf Grundlage der Kompatibilitätsmatrix (Tabelle 1) vergeben. Dabei bedeutet „Y“, „$Y^1$“, dass gleichzeitige Vergabe von Berechtigungen mit Priorities erlaubt ist.

Nach dieser Matrix kann einer Transaktion immer eine optimistische Leseberechtigung (NP_RW) für ein x erteilt werden, unabhängig davon, ob andere Transaktionen irgendwelche Berechtigungen für den Zugriff auf x haben. Auf ein Objekt x kann also in beiden Modi, optimistischen und pessimistischem, gleichzeitig zugegriffen werden.

| | | Lesend | | Schreibend | | | |
|---|---|---|---|---|---|---|---|
| | | P_RW | NP_RW | P_RW<br>P_WW | P_RW<br>NP_WW | NP_RW<br>P_WW | NP_RW<br>NP_WW |
| Lesend | P_RW | Y | Y | N | Y | N | Y |
| | NP_RW | Y | Y | Y | Y | Y | Y |
| Schreibend | P_RW<br>P_WW | N | Y | Y[1] | Y[1] | Y | Y |
| | P_RW<br>NP_WW | Y | Y | Y[1] | Y[1] | Y | Y |
| | NP_RW<br>P_WW | N | Y | Y | Y | Y | Y |
| | NP_RW<br>NP_WW | Y | Y | Y | Y | Y | Y |

**Tabelle 1 Kompatibilitätsmatrix nach [MV99]**

Mit diesem Ansatz werden Transaktionen nicht mehr blockiert. Kann kein konfliktfreier (pessimistischer) Zugriff garantiert werden, so arbeitet die Transaktion im optimistischen Verfahren. Kombiniert wurde dieses Verfahren des flexiblen Einsatzes optimistischer und pessimistischer Concurrency Control mit dem Konzept der Probetransaktionen für den optimistischen Zugriff. Damit wird versucht das Problem des wiederholten Ausführens von im Server nicht validierter Transaktionen auf dem mobilen Gerät zu lösen (siehe auch Abschnitt 4.2.3.).

## 4.2. Einschränkung der Validität der Daten/Aufhebung der Isolationseigenschaft der TA

Die unter 3.1. vorgestellten Konzepte suchten nach Wegen, wie konfliktfreies Nebeneinanderlaufen von Transaktionen in mobilen Umgebungen verbessert werden kann, ohne dass Serialisierbarkeit als Korrektheitskriterium verletzt wird. Unter Ausnutzung des Wissens über die Art der Nutzung von Datenbankobjekten, können weichere applikationsspezifische Konsistenzbedingungen entwickelt werden, die es erlauben, noch mehr Transaktionen gleichzeitig auszufüh-

ren [WC95]. Konzepte die solches Wissen ausnutzen betrachten im Gegensatz zu Serialisierbarkeit, Datenkonsistenz und Transaktionskorrektheit unabhängig voneinander.

Der Grad der Anforderungen an Datenkonsistenz kann sich zwischen strikter Konsistenz und eventueller Datenkonsistenz bewegen. Eventuelle Datenkonsistenz bezeichnet eine zeitliche oder räumliche Abweichung von strikter Datenkonsistenz und kann im Grad der Inkonsistenz angegeben werden. So kann Konsistenz (nur) zu einem bestimmten Zeitpunkt, innerhalb eines bestimmten Zeitraumes, oder nach einer bestimmten Menge von Datenänderungen gefordert/sichergestellt werden. Mit Hilfe spezieller Divergence Control Protkolle wie denen für Quasi-Copies und Epsilon-Serialisierbarkeit können Applikationen die Nebenläufigkeitsanforderungen und den Grad der Inkonistenz, den sie handhaben können, spezifizieren [WC95]. Gleichzeitig erlauben es solche Protokolle Feinabstimmungen im Cachemanagement vorzunehmen und so auf die vorhanden Bandbreite und Kosten der drahtlosen Kommunikation einzugehen.

Transaktionskorrektheitsanforderungen betrachten Korrektheit aus Sicht des Verhaltens und der Struktur von Transaktionen. Dabei könne Transaktionen verschieden **Grade der Isolation** haben, so dass sich gegenseitiges Überschneiden, Delegieren von Operationen oder frühzeitiges Beenden von Teiltransaktionen erlaubt ist. Daraus ergeben sich verschiedene **Grade der Transaktionsautonomie**, so dass gegenseitige Abhängigkeiten, wie Commit- oder Abort-Abhängigkeit erlaubt sind.

Die Verwaltung und das Management solch gelockerter Ansätze sind schwieriger, da nicht mehr davon ausgegangen werden kann, dass einzelne Transaktionen die Datenbank in einem konsistenten Zustand belassen. Es ist aber auf der anderen Seite eine höhere Autonomie der mobilen Geräte gegeben, so dass solche Ansätze in mobilen Umgebungen nützlich sein können.

### 4.2.1. Clustering

Um den Datentransfer über die teure drahtlose Verbindung zu reduzieren und autonomes Arbeiten der mobilen Geräte zu ermöglichen, führen Pitoura und Bhargava in [PB94] das Clusteringverfahren mit dem Konzept der **loose transaction** ein. Sie unterteilen die replizierten Datensätze einer mobilen Umgebung in Cluster. Die Replikate in einem Cluster sind untereinander synchronisiert. Die Werte der Replikate anderer Cluster können in geeignet definiertem Maß abweichen.

Maße für diese Abweichung können auf verschiedene Arten definiert werden. So kann der Unterschied zur Hauptkopie (primary copy) angegeben werden, indem eine Höchstzahl unterschiedlicher Versionen eines Objektes, eine größte Abweichung vom Originalwert oder eine Höchstzahl von Transaktionen, die auf inkonsistenten Objekten arbeiten dürfen, gesetzt wird. Außerdem könne Abweichungsgrade als Anzahl der abweichenden Datenobjekte oder als Anzahl der abweichenden Replikate pro Datenobjekt definiert werden.
Die Anordnung der Cluster ist nicht statisch. Cluster könne dynamisch gebildet, und auch wieder zusammengefügt werden. Beim Zusammenfügen (merging) von Clustern werden die Werte aller Replikate eines Objekts untereinander abgeglichen.

Auch zur Definition von Clustern gibt es verschiedene Möglichkeiten. Cluster können gebildet werden, indem Daten, die sich auf demselben oder benachbarten Hosts befinden, zusammengefasst werden. So können Replikate, die sich auf verbindungslosen Geräten befinden, ein Cluster bilden. Bei auftretender Verbindungsunterbrechung werden diese Cluster gebildet und bei Wiederverbindung mit den Daten im festen Netzwerk abgeglichen. Andere Möglichkeiten der Clusterbildung sind Clustering über die Art der Daten, Daten- oder Applikationssemantik oder die Ausnutzung von Daten in Nutzerprofilen, wobei Daten die oft oder nur privat genutzt werden, in einem Cluster zusammengefasst werden, unabhängig davon, wo sie abgelegt sind.

Um auf den sich innerhalb eines Clusters befindlichen Daten zu arbeiten, wurden neue Operationen eingeführt: loose read und loose write. Diese erlauben es, auf Daten mit abgeschwächter Konsistenz zuzugreifen. Die Standard Schreib- und Leseoperationen werden mit strikt read und strikt write bezeichnet. Eine loose read Operation liest den Wert, der von einer loose write Operation oder einem strikt write auf einem lokalen, also in einem Cluster befindlichen, Replikat geschrieben wurde. Sie liest also unter Umständen[1] einen inkonsistenten Wert. Eine loose write Operation schreibt auf einer lokalen Kopie und ist erst nach dem Merging vollständig abgeschlossen und permanent. Eine strikt read Opertation liest den Wert den eine strikt write Operation in einem lokalen oder globalen Cluster geschrieben hat. Eine strikt write Operation ist sofort nach dem Commit permanent.

Abgeleitet von den zwei Schreib- und Leseoperationen werden zwei Arten von Transaktionen unterschieden: **strikt transactions** und **loose transactions**. Loose transactions bestehen nur aus loose read und loose write Operationen, strikt transactions nur aus strikt read und strikt write Operationen. Loose transactions werden lokal in dem zugehörigen Cluster abgeschlossen (lokal Commit). Die Ergebnisse lokaler Transaktionen sind nur für andere lokale Transaktionen sichtbar und werden für globale Transaktionen (strikt transactions) erst nach dem Merging und dem globalen Commit sichtbar. Lokale Transaktionen können vor dem globalen Commit rückgängig gemacht werden, obwohl sie schon lokal abgeschlossen wurden. Updates aus loose Transaktions werden nur dauerhaft, wenn sie nicht mit Operationen aus strikt transactions in Konflikt stehen. Operationen zweier Transaktionen stehen in Konflikt, wenn sie auf das selbe Objekt zugreifen und eine Operation eine write Operation ist. Operationen die im Modell mit loose Transaktion vor eine Merging ausgeführt wurden, stehen wie in Tabelle 3 abgebildet in Konflikt.

| | $LR_J(a_k)$ | $SR_J(a_k)$ | $LW_J(a_k)$ | $SW_J(a_k)$ |
|---|---|---|---|---|
| $LR_i(a_k)$ | | | x | x |
| $SR_i(a_k)$ | | | | x |
| $LW_i(a_k)$ | x | | x | x |
| $SW_i(a_k)$ | x | x | x | x |

Tabelle 2 Konfliktmatrix

Um nach dem Merging volle Konsistenz zu erreichen, werden alle loose transactions, deren loose write Operationen mit einer strikt transaction in Konflikt stehen, zurückgesetzt. Updates können also beim Merging wieder rückgängig gemacht werden. Dieser Ansatz ist damit nur für

[1] wenn die letzte Operation ein loose write war

selten oder privat genutzte Daten einsetzbar oder falls kompensierende Transaktionen, also solche, die die Ergebnisse einer zurückgesetzten Transaktion semantisch rückgängig machen können, möglich sind. In Applikationen kann festgelegt werden ob strikte Konsistenz erforderlich ist oder eine lockere Konsistenz erlaubt ist, so dass die Operationen der Transaktionen im strikt- oder loose- Modus ausgeführt werden, d.h. im verbindungslosen Zustand auf eine Verbindung warten müssen oder lokal durchgeführt werden können. Mit dem Einsatz dieses Verfahrens wird die Autonomie mobiler Geräte in verbindungslosen Zuständen erhöht und die Kosten für die drahtlose Kommunikation gesenkt.

### 4.2.2. Geschachtelte Transaktionen

Das Konzept der geschachtelten Transaktionen betrachtet Transaktionen nicht als Menge von einzelnen Schreib- und Leseoperationen, sondern als Menge von Transaktionen, sogenannten Subtransaktionen, die weiter geschachtelt sein können. Geschachtelte Transaktionen lassen sich also als Baumstruktur darstellen wobei die Blätter eines solchen Baumes dann die einzelnen Schreib- und Leseoperationen auf den physischen Datenbankobjekten darstellen. Bei der Bildung solcher geschachtelter Transaktionen fließen semantische Informationen ein, so dass als eigenständig betrachtete Operationen eine Subtransaktion bilden. Die Subtransaktionen stellen aus der Sicht der Vatertransaktion atomare Operationen dar und können unabhängig voneinander bearbeitet und beendet werden. Dadurch muss eine Transaktion nicht notwendigerweise zurückgesetzt werden, wenn eine Subtransaktion abbricht. Ein Beispiel hierfür ist eine Reisereservierung mit Buchung von Flügen, Hotelzimmer und Mietwagen. Werden die einzelnen Aufträge als Teiltransaktionen angesehen und unabhängig voneinander bearbeitet, so kann etwa bei fehlgeschlagener Reservierung eines Mietwagens nach Alternativen gefragt werden, ohne die gesamte Transaktion, und damit die vielleicht schon erfolgreichen Teiltransaktionen Flug- und Hotelbuchung, zurückzusetzen. Die Subtransaktionen einer geschachtelten Transaktion haben nur die Eigenschaften A und I des ACID-Prinzips. Sie laufen atomar und untereinander isoliert ab. Dauerhaft sind sie jedoch nicht, denn wenn die Vatertransaktion zurückgesetzt wird, müssen auch alle schon beendeten Subtransaktionen zurückgesetzt werden.

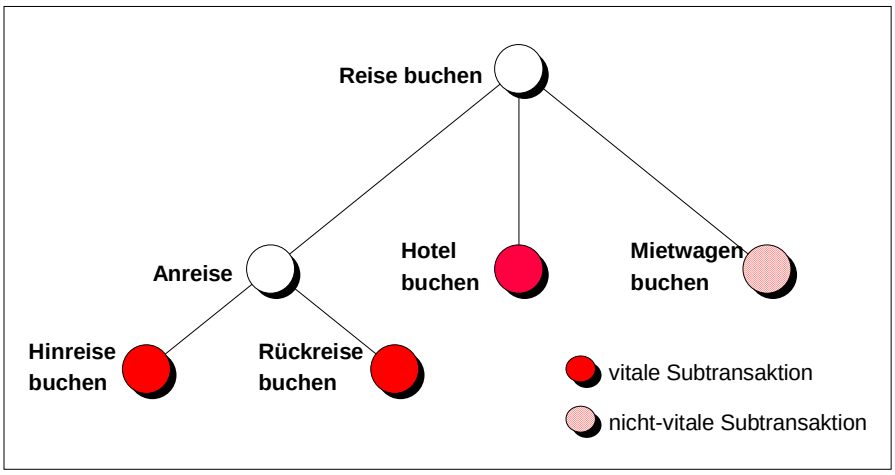



**Abbildung 5: geschachtelte Transaktion**

Man unterscheidet offen und geschlossen geschachtelte Transaktionen, je nachdem, ob (die noch nicht dauerhaften) Ergebnisse der Teiltransaktionen für andere Transaktionen sichtbar sind oder nicht. Bei **geschlossen** geschachtelten Transaktionen werden Sperren von Subtransaktionen gehalten, nach erfolgreichem Ende an die Vatertransaktion vererbt und erst am Ende der gesamten Transaktion freigegeben. Falls eine Subtransaktion abgebrochen werden muss, folgt daraus nicht unbedingt der Abbruch der gesamten Transaktion. Es müssen aber alle Subtransaktionen abgeschlossen sein, bevor die Vatertransaktion abgeschlossen wird, und erst nach Abschluss der Wurzeltransaktion werden die Ergebnisse der Subtransaktion nach außen sichtbar. Die Transaktionen sind in diesem Ansatz also, wie in traditionellen Konzepten voneinander isoliert. Vorteil ist lediglich eine Erhöhung des Grades an Intratransaktions-Parallelismus [VG93]. Für mobile Umgebungen ist auch der Aspekt der Teilung von Transaktionen interessant. Es können leistungsschwache mobile Geräte unterstützt werden, indem Teile der Transaktionen im festen Netzwerk ausgeführt werden.

Da geschlossen geschachtelte Transaktionen Sperren erst freigeben, wenn die gesamte Transaktion beendet ist, sind sie für den Einsatz in mobilen Umgebungen nicht sehr geeignet. Auch wenn die Transaktion verteilt ausgeführt werden kann, so können die Sperren erst dann freigegeben freigegeben werden, wenn alle Subtransaktionen erfolgreich geendet haben, was andere Transaktionen blockieren könnte. Die Isolation der einzelnen Transaktionen wird erst mit dem Konzept der **offen** geschachtelten Transaktionen abgeschwächt. Hier geben Subtransaktionen die Sperren, die sie halten, sofort nach erfolgreichem Abschluss wieder frei. Die Ergebnisse der Subtransaktionen werden so sofort für andere Transaktionen sichtbar. Natürlich kannn die Vatertransaktion auch nach erfolgreichem Beenden und daraus resultierender Sperrenfreigabe noch abbrechen. In diesem Fall muss die schon beendete Subtransaktion zurückgesetzt werden. Da andere Transaktionen die Ergebnisse der zurückzusetzenden Subtransktion aber schon gelesen und damit gearbeitet haben können, können die von der Subtransaktion geschriebenen Objekte nicht einfach auf den before image Wert gesetzt werden. Es muss eine sogenannte Kompensationstransaktion durchgeführt werden, die die Ergebnisse der Subtransaktion semantisch rückgängig macht. Allerdings lassen sich nicht für alle Transaktionen solche Kompensationstransaktionen finden, so dass der Einsatz von offen geschachtelten Transaktionen applikationsabhängig ist.

Geschachtelte Transaktionen können um weitere Eigenschaften erweitert werden. So können Subtransaktionen mit Kontingenttransaktionen verbunden werden. Diese können alternativ ausgeführt werden, falls die betreffende Subtransaktion abbricht. Zusätzlich können Subtransaktionen in vitale und nicht-vitale Transaktinen eingeteilt werden. Vitale Transaktionen müssen erfolgreich enden, damit die Vatertransaktion abgeschlossen werden kann. Nicht-vitale Subtransaktionen sind für eine Transaktion nicht „lebenswichtig“, dh. sie kann auch beendet werden, wenn eine ihrer nicht-vitalen Subtransaktionen abbricht. Eine Transaktion muss jedoch darauf warten, ob ihre nicht-vitalen Subtransaktionen mit COMMIT oder ABORT enden[Chr93]. Für das Beispiel oben könnte eine nicht-vitale Transaktion die Buchung des Mietwagens sein (siehe Abbildung 5). Denn dies könnte auch noch am Urlaubsort geschehen. Für die Flugbuchung wären Kontingenttransaktionen möglich. So könnten alternative Abflugorte angegeben werden.

### 4.2.3. Erweiterung optimistischer Verfahren

Optimistische Verfahren sind für den Einsatz in mobilen Umgebungen in denen mobile Geräte benötigte Daten lokal speichern gut geeignet. Die Transaktionen werden ohne Kommunikation mit dem festen Netzwerk abgearbeitet. Kommunikationskosten, die zur Anforderung von Sperren notwendig sind, entfallen, und Transaktionen werden nicht durch andere, die Sperren nicht freigebende, Transaktionen blockiert. Nach Abarbeitung der Transaktionsoperationen muß das Ergebnis im festen Netzwerk validiert werden. Die Validierung schlägt fehl, wenn sich die Werte der lokal im mobilen Gerät gehaltenen Daten im festen Netzwerk geändert haben. In diesem Fall muß die Transaktion abgebrochen werden. Gray et al. haben in [GHOS96] für diesen Fall ein Wiederausführen (re-execution) der Transaktion im festen Netzwerk vorgeschlagen, anstatt sie abzubrechen. Die Transaktionen auf den lokal gehaltenen Daten werden als Probetransaktion (tentative transactions) bezeichnet. Sie werden im festen Netzwerk als Basistransaktionen (base transaction) wiederholt. Erfüllt die Wiederholung der Transaktion im festen Netzwerk bestimmte Akzeptanzkriterien, so wird die Transaktion als Erfolgreich beendet angesehen. Es wird also versucht, ein ähnliches Ergebnis wie auf dem mobilen Gerät zu erreichen. Die Transaktionsverarbeitung im mobilen Gerät wird also an den Datenbankzustand im festen Netzwerk angepasst. Aber auch trotz dieser Anpassung gibt es keine Garantie für ein erfolgreiches Ende der Transaktion. In [MV99] schlagen Momin und Vidyasankar daher eine Integration diesen Ansatzes mit dem erweiterten pessimistischen concurrency control, wie in Abschnitt 4.1.3 beschrieben, vor.

### 4.2.4. Sperrverfahren mit speculativ lock

Da Transaktionen in mobilen Umgebungen durch geringe Übertragungsraten und auftretende Verbindungsunterbrechungen langandauernd sind, werden wartende Transaktionen blockiert oder müssen abgebrochen werden. Um die Wartezeit zu verkürzen und somit eine Blockierung zu verhindern und die Nebenläufigkeit der Transaktionen zu erhöhen, führen Reddy und Kitsuregawa in [RK99] das Konzept des mobile speculative locking (MSL) ein.
In diesem Ansatz werden die Änderungen auf Datenbankobjekten sichtbar gemacht, sobald die Transaktion die Arbeit auf diesem Objekt beendet hat, und nicht erst nach erfolgreichem Beenden der Transaktion. Die Sperren werden also freigegeben, sobald die Transaktion auf das der Sperre zugeordnete Objekt geschrieben hat. Die wartende Transaktion liest dann sowohl den Wert des Datenbankobjekts vor der Änderung (before image) als auch den Wert nach der Änderung (after image) und führt sogenannte spekulative Verarbeitungen (speculative execution) durch. Die Operationen werden mit dem Wert des before image und mit dem des after image durchgeführt. Die Transaktion, die die spekulative Bearbeitung durchgeführt hat, kann erst nach Beendigung der vorhergegangenen Transaktion beendet werden. Sie sucht dann die passende Ausführung heraus: das Ergebnis der Bearbeitung mit dem before image, falls die vorhergegangene Transaktion abgebrochen wurde, das Ergebnis aus der Bearbeitung mit dem after image, wenn die Transaktion erfolgreich abgeschlossen wurde. Je nach Leistungsfähigkeit des mobilen Gerätes können verschiedenen Schachtelungstiefen der spekulativen Verarbeitung genutzt werden. In diesem Ansatz wird verbindungsloses Arbeiten nicht unterstützt. Es wird versucht, die Verbindungskosten durch eine Veringerung der Transaktionsdauer zu reduzieren.

# 5. Umsetzung der Transaktionskonzepte mit kommerziellen DBMS

Kommerzielle Datenbankmanagementsysteme unterstützen meist nur die Standardtransaktionskonzepte. Die oben vorgestellten erweiterten Transaktionskonzepte sind daher nicht ohne weiteres mit diesen Systemen umzusetzten. Einige Hersteller bieten spezielle Datenbanklösungen für mobile Umgebungen an. In den folgenden Abschnitten werden zwei dieser Lösungen, Oracles „Oracle Lite“ und Informix’ „Cloudscape“, sowie ein Ansatz für die Umsetzung der erweiterten Konzepte mit Standarddatenbanken vorgestellt.

## 5.1. Kommerzielle Lösungen

### 5.1.1. Cloudscape

Für den Einsatz in mobilen Umgebungen bietet Informix das Produkt Cloudscape [Clou00] an. Cloudscape ermöglicht das Arbeiten im verbindungslosen Zustand. Es gibt eine zentrale Datenbank (Source-Datenbank) und verschiedene sogenannte Targetdatenbanken. Auf der Source-Datenbank befinden sich sämtliche Datenobjekte als primary copy. Auf den Targetdatenbanken werden Subsets der Daten der Source-Datenbank gehalten. Die Targetdatenbanken sind, wie die Sourcedatenbank, in Java geschrieben und können so auf verschiedenen mobilen Plattformen eingesetzt werden. Sie verfügen über eine eingeschränkte Datenbankfunktionalität, die das Verarbeiten von Datenbankoperationen ermöglicht. So ist eine Verarbeitung im verbindungslosen Zustand möglich.

Daten auf der Target-Datenbank (auf dem mobilen Gerät) können ohne Verbindung zum Server bearbeitet werden. Zur Synchronisation mit der Source-Datenbank wird, wenn eine Verbindung vorhanden, ist eine refresh-Operation durchgeführt. Dabei werden zur Source-Datenbank alle die Aktionen geschickt, die seit der letzten Verbindung durchgeführt wurden. Umgekehrt wird von der Source-Datenbank eine konsistente Menge von Datenänderungen zum Target geschickt. Diese Art der Synchronisation wird in Cloudscape LUCID, Logic Up Consistent Information Down, genannt. Zur Source-Datenbank werden nur ausgeführte Aktionen geschickt (Logic Up). Dort werden diese Aktionen auf dem konsistenten Datenbestand ausgeführt. Dann werden die geänderten und damit konsistente Daten zur Target-Datenbank geschickt (Consistent Information Down), wo die lokalen Daten ergänzt oder ausgetauscht werden. Zur Optimierung des Download der geänderten Daten wird die so genannte Delta-Optimization eingesetzt. Dabei werden nur die Daten, die wirklich geändert wurden zur Target-Datenbank geschickt und weder ganze Tabelleninhalte noch ganze Tupel.

Durch die verbindungslosen Operationen kann keine ständige Konsistenz der Targetdatenbank gewährleistet werden. Nach einem Refresh müssen jedoch die Daten der Target- und der Sourcedatenbank übereinstimmen. Zur Konsistenzsicherung wird der Begriff des agreement eingeführt. Eine Target-datenbank ist mit ihrer Zieldatenbank in agreement, wenn alle Unterschiede

zwischen ihnen nur auf noch nicht synchronisierten Operationen beruhen, und beim nächsten Refresh ausgeglichen werden.

Cloudscape nutzt zur Transaktionskontrolle das Konzept der „provisional transactions“. Transaktionen, die auf der Targetdatenbank ausgeführt werden, sind so lange nur provisorisch (provisional), bis sie von der Sourcedatenbank bestätigt sind. Wenn die Sourcedatenbank die Transaktion bestätigt, werden die Änderungen dauerhaft. Treten während der Verarbeitung der Transaktion in der Source-Datenbank Konflikte auf, so wird ein Log mit den Operationen, die den Konflikt auslösten und nicht durchgeführt werden können geschrieben und zur Targetdatenbank gesendet, so dass dort der Nutzer informiert werden kann.

Zur Konfliktlösung können über das statische Abbrechen und Benachrichtigen hinaus, auch applikationsspezifische Strategien eingesetzt werden. Dabei werden so genannte Work-Units eingesetzt. Work-Units sind als Javamethoden implementiert, in denen SQL-Befehle eingebettet sind. Durch sie findet die Verarbeitung der Daten sowohl auf der Targetdatenbank als auch der Sourcedatenbank statt. Damit kann für den Fall eines Fehlers bei der Verarbeitung auf der Source-Datenbank eine applikationsspezifische Ausnahmebehandlung implementiert werden, die neben SQL-Befehlen (Kompensationstransaktion) auch Javamethoden, wie etwa das Senden einer e-mail zur Benachrichtigung, enthalten können.

Cloudscape arbeitet bei Updates und Konfliktbewältigung also nicht auf den Datenlevel sondern auf der logischen Ebene. Die Daten werden nicht auf einen festen Wert gesetzt. Es werden die Operationen der Targetdatenbank übertragen und auf der Source ausgeführt. Wenn die Operation nicht ausgeführt werden kann, wird eine Kompensationsoperation ausgeführt und diese auch zum Target geschickt, wo die Applikation nach Änderungen scant und Werte gegebennenfalls zurücksetzt. Ergebnisse von provisorischen Transaktionen können als noch nicht feststehende Werte markiert werden. Der Nutzer kann so darauf hingewiesen werden, dass sich der Wert unter Umständen noch ändern kann.

Damit wird das rein optimistische Concurrency Control um Ansätze aus dem Clusteringprinzip erweitert. Dabei sind die Cluster die einzelnen Target-Datenbanken. Die Konsistenz innerhalb des Clusters (Target) ist immer gewährt, und nach der Refresh-Operation (Merging) stimmen die Daten wieder mit denen im festen Netzwerk (Source-Datenbank) überein. Die Transaktionen sind bis zum Ende des Refreshzyklus nur provisional (loose).

### 5.1.2. Oracle Lite

Oracle hat für den Einsatz in mobilen Umgebungen sein Produkt „Oracle Lite“ entwickelt [Ora99]. Oracle Lite Datenbanken auf mobilen Geräten verwalten ebenso wie in Cloudscape Subsets der Daten. Diese werden hier Snapshots genannt. Snapshots sind Tabellen der Masterdatenbank oder nach bestimmten Selektionskriterien ausgewählte Tupel aus Tabellen. Einfache (simple) Snapshots bestehen nur aus Tupel einer Mastertabelle, komplexe (complex) Snapshots können auch aus mehreren Tabellen zusammengesetzt sein. Es gibt zwei Arten von Snapshots: read-only und updateable Snapshots. Read-only Snapshots können zum lesenden Zugriff eingesetzt werden. Hier ist also keine Transaktionsverarbeitung notwendig, da Daten nicht geändert werden können. Auf updateable Snapshots können auch Schreiboperationen ausgeführt werden. Zum Abgleich mit der Masterdatenbank werden Before- und After-Image des Datenobjektes

zum Server geschickt. Ein Konflikt tritt auf, wenn das Before-Image nicht mit dem aktuellen Wert der Masterdatenbank übereinstimmt, der Wert in der Masterdatenbank also schon von einer anderen Transaktion nach dem letzten Refresh des Snapshots geändert wurde.

Das Transaktionsverfahren hinter Oracel Lite ist also ein rein optimistisches. Transaktionen werden lokal verarbeitet und die Ergebnisse zur Validierung zum Server geschickt. Tritt ein Konflikt auf, muss die Transaktion wiederholt werden. Die Art der Konflikterkennung ist zu restriktiv. Operationensemantische Eigenschaften werden nicht berücksichtigt. So wird in diesem Ansatz ein Konflikt erkannt, wenn eine Transaktion $t_i$ den Wert eines Warenbestandes von 100 Stück um zehn Stück verringert und eine Transaktion $t_j$ zuvor (aber nach dem letzten Refresh des Snapshots auf dem $t_i$ arbeitet) den Wert um 20 verringert hat. Before-Image von $t_i$ (100) und Wert der Masterdatenbank (80) stimmen nicht überein. Der Snapshot muss aktualisiert und die Transaktion erneut ausgeführt werden. Im von Cloudscape verfolgten Ansatz wird diese Transaktion nicht abgebrochen. Der neue Wert 70 wird in Source- und Targetdatenbank geschrieben. Bei Oracle Lite wird muss zur Konfliktbewältigung ein undo ausgeführt werden, wärend in Cloudscape durch die Übertragung der Änderungsoperationen ein redo der Transaktion auf dem Datenbankserver ausgeführt werden kann.

## 5.2. Umsetzung mit Standard-DBMS

In Standard-DBMS werden zur Transaktionsverarbeitung meist nur einfache Konzepte für flache Transaktionen unterstützt. Um den Gegebenheiten in mobilen Umgebungen gerecht zu werden, müssen die erweiterten Konzepte auf die Standarddatenbanken und deren (eingeschränkten) Möglichkeiten übertragen werden. Um auch im verbindungslosen Zustand arbeiten zu können, müssen Daten auf dem mobilen Gerät gehalten und verarbeitet werden können. Es muss des weiteren eine Methode gefunden werden, um die auf dem mobilen Gerät verarbeiteten Transaktionen in Übereinstimmung mit dem Datenbankserver(n) zu bringen.

Preguica, Baquero u.a [PBM+00] nutzen dazu in ihrem Projekt Mobisnap so genannte „mobile Transactions“, die durch eine erweiterte Untermenge der PL/SQL Sprache von Oracle [Ora97] spezifiziert werden. Diese mobilen Transaktionen werden vom mobilen Gerät zur zentralen Datenbank übermittelt, wo sie ausgeführt werden und so den Datenbankinhalt ändern. Durch den Einsatz von PL/SQL können Vor- und Nachbedingungen

```
--------------------------------NEW ORDER -----------------------------
BEGIN
  SELECT price, stock into prd_price, prd_cnt FROM products
                                                WHERE name =’BLUE THING“;
  IF prd_price <= 10.00 ND prd_cnt >=50 THEN
  -- update orders, current stock,…
  NOTIFY ( ‘SMTP’, ‘sal-07@thingco.t’, ‘Order completed …’);
  COMMIT;
  ENDIF;
  ROLLBACK;
ON ROLLBACK NOTIFY ( ‘SMS’, ‘351927435456’, ‘Impossible order ..’);
END;
```

**Beispiel 2 : Definitionn einer mobilen Transaktion nach [PBM+00]**

sowie Alternativen für die Ausführung der Transaktion definiert werden. Das Ergebnis einer mobilen Transaktion ist erst dauerhaft, wenn die Transaktion auf der zentralen Datenbank ausgeführt wurde. Die mobilen Transaktionen beinhalten eine Möglichkeit der Benachrichtigung der Nutzer über das endgültige Resultat der Verarbeitung auf dem Server. Sie werden unterrichtet, auch wenn sie nicht mehr mit der Datenbank verbunden sind. Je nach Wichtigkeit der Nachricht sind unterschiedliche Benachrichtigungsformen möglich. So genügt es nach erfolgreicher Verarbeitung der Transaktion eine e-mail zu senden. Schlägt die Verarbeitung der Transaktion jedoch fehl, so ist eine sofortige Benachrichtigung durch SMS notwendig. Das Konzept der mobilen Transaktionen in Mobisnap ähnelt dem der Work-Units in Cloudsacape. Dort können durch den Einsatz von Java Alternativen zur Verarbeitung und Benachrichtigungsarten im Fehlerfall angegeben werden. Auch werden in beiden Realisierungen nur die Datenbankoperationen und nicht die geänderten Werte übertragen, was eine höhere Nebenläufiigkeit ermöglicht.

Die Transaktionen werden auf dem mobilen Gerät nur probeweise (tentatively) ausgeführt, um einen Hinweis für das zu erwartende Resultat zu erhalten. Um eine größere Sicherheit über das zu erwartende Ergebnis zu erhalten arbeitet Mobisnap zusätzlich mit einem Reservationsmechanismus. Dieser kombiniert die Ansätze aus der Escrowmethode (siehe Abschnitt 4.1.2) mit denen des Leasings [GC89]. Reserviert werden kann: eine Teilmenge eines partitionierbaren Objektes zur Verarbeitung auf dem mobilen Gerät (Escrow), das Recht, Daten mit vordefinierten Werten einzufügen (um zum Beispiel einen Raum zu einem bestimmten Zeitpunkt zu mieten), das Recht, bestimmte Werte in der Datenbank zu ändern (etwa Beschreibungen von Produkten) sowie das Recht, einen festen Wert für ein Objekt zu nutzen (etwa einen festen Preis für ein Produkt auch wenn dieser geändert wird). Für jedes Datenobjekt der Datenbank muss in einem speziellen Script spezifiziert werden, welche Reservierungen auf ihm möglich sind. Die Reservierungen sind nur für einen bestimmten Zeitraum gültig (leasing), so dass sie auch für den Fall, dass das mobile Gerät, welches die Reservation hält, nicht mehr verbunden ist, die Reservation freigegeben werden kann. In Mobisnap wird also versucht die Validitätseinschränkungen, die der optimistische Zugriff mit sich bringt, durch eine (logische) Begrenzung des Zugriffs auf die Daten zu minimieren.

Ein Nachteil der Lösung mit Mobisnap ist Beschränkung auf einen Datenbankserver und die Festlegung auf ein System. Universell einsetzbare Lösungen müssen in Betracht ziehen, dass in mobilen Umgebungen oft mehrere Datenbanken zu einer verteilten Datenbank miteinander verbunden sind. Ein System zur Transaktionsverarbeitung auf mobilen Geräten sollte also neben unterschiedlichen mobilen Geräten auch verschiedenen Datenbankserver einbeziehen können. Als Architektur für solche Umgebungen ist eine Mehr-Schichten-Architektur mit (Datenbank-)Server, Mittelschicht und (mobilem) Client zu bevorzugen. Sie bietet Vorteile gegenüber der Client-Server-Architektur. Als Mittelschicht wird ein Transaktionsmanager im festen Netzwerk eingesetzt. Dieser kann die mobilen Geräte bei der Verarbeitung unterstützen, die Koordinierung in verteilten Umgebungen leisten, die Transaktionskonzepte auf die Möglichkeiten der Standard-DBMS umsetzen, Loginformationen für das Recovery halten sowie das Recovery durchführen.

Ein System, das auch in vertreilten Umgebungen zum Einsatz kommen kann, stellen Walborn und Chrysanthis mit PRO-MOTION [WC98] vor. Das System ist auf einer Mehr-Schichten-Architektur aufgebaut, mit einem so genannten compact agent auf dem mobilen Gerät, dem compact manager als stationäres Server-front-end und einer Mittelschicht aus mobility managern, die den Datenflusses zwischen den Komponenten unterstützen. Grundbaustein des Systems ist der Compact, der als Basiseinheit der Datenreplikation fungiert.

Applikationen auf mobilen Geräten greifen auf die Daten über diese Compacts, die als Java-Objekte, die die Datenwerte enthalten, implementiert sind, zu. In den Compacts befinden sich außer den Daten Methoden, um auf die Daten zugreifen zu können, Informationen über den aktuellen Zustand des Compacts, Konsistenzregeln zur Wahrung der globalen Konsistenz, Verpflichtungen, wie ein Zeitraum in dem die Daten genutzt werden können, und schließlich einheitliche Basis-Methoden, mit denen das mobile Gerät (genauer der Compact Agent) den Compact managen kann. Die Basismethoden stellen ein einheitliches Interface zur Verfügung, welches es ermöglicht Statusinformationen des Compacts zu erhalten (inquire()), Operationen auf dem Compact auszuführen (dispatch()), Operationen einer Transaktion in der Datenbank permanent zu machen (commit()), und die Änderungen auf Compactdaten rückgängig zu machen(abort()). Der compact repräsentiert eine Übereinkunft zwischen Datenbankserver und mobilem Gerät. Darin wird die Kontrolle der Daten vom Datenbankserver zum mobilen Gerät delegiert, um dort lokal Transaktionen verarbeiten zu können. Im Gegenzug erkennt das mobile Gerät die spezifizierten Bedingungen an, um einen konsistenten Zustand nach der Wiedereingliederung der Daten auf dem Server sicherzustellen.

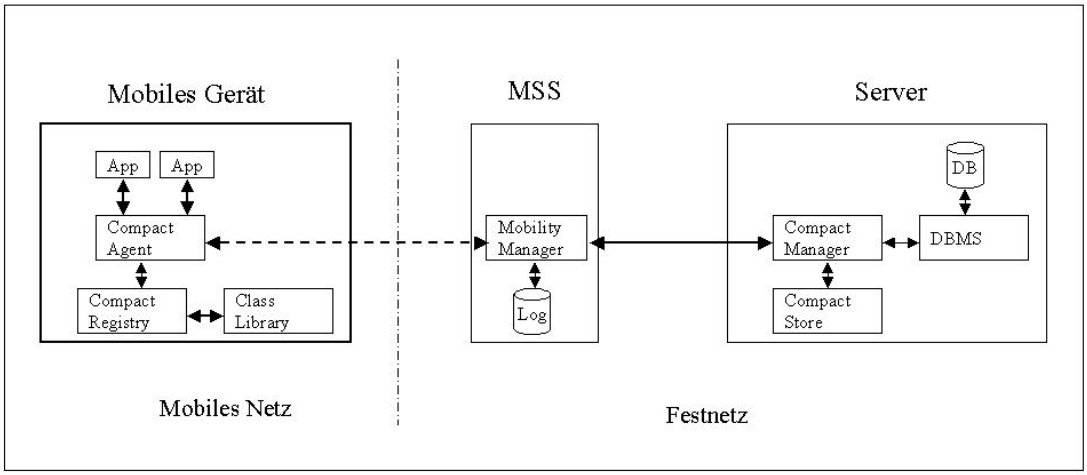



**Abbildung 6: PRO-MOTION Architektur**

Compacts werden durch das mobile Gerät beim compact manager des Datenbankservers angefragt, wenn es Daten zur Verarbeitung benötigt. Werden die Daten schon von einem anderen mobilen Gerät gehalten, überprüft der compact manager, ob eine gleichzeitige Verarbeitung möglich ist. Ist die nicht der Fall wird ein Null-Compact zum Client gesendet und der Request muss zu einem späteren Zeitpunkt wiederholt werden.

Im verbindungslosen Zustand können Transaktionen lokal mit Hilfe des compact-agent durchgeführt werden. Applikationen, die auf dem mobilen Gerät laufen, greifen auf Daten über Events zu, die zum Compact Agent geschickt werden. Eine Applikation initiiert eine Transaktion mit dem Senden des BEGIN-Events. Dabei sind einige zusätzliche Optionen, die das Verhalten der Transaktion bestimmen, angebbar. So kann angegeben werden ob lokales COMMIT erlaubt ist, dh. ob die Ergebnisse der Transaktion schon vor der Validierung auf dem Datenbankserver für andere Transaktionen sichtbar sein sollen. Eine weitere Konsistenzabschwächung ist beim Start einer Transaktion durch Angabe einer Gruppen-ID möglich. Transaktionen mit gleicher ID können untereinander Ergebnisse sehen, die noch nicht global validiert sind. Diese Transak-

tionen befinden sich damit in Commit-Abhängigkeit. Es ist außerdem möglich beim Start einer Transaktion den Grad ihrer Isolation und damit der Korrektheit der Abarbeitung festzulegen. So können weichere Konsistenzkriterien als Serialisierbarkeit, und somit loose Operationen (vgl. Abschnitt 4.2.1.), unterstützt werden.

Um einen Datenzugriff auszuführen, sendet die Applikation ein OP („perform operation“) – Event zum Compact Agent. Darin sind die ID des Compacts, die Operation die ausgeführt werden soll , Parameter der Operation und der Transaktions-ID, die beim Start der Transaktion vergeben wurde, enthalten. Der Compact Agent nutzt die Compact-ID um, die dispatch()-Methode des Compacts mit dem OP-Event als Parameter aufzurufen. Der Compact überprüft, ob bei der Verarbeitung der Operation Konflikte mit anderen (noch nicht abgeschlossenen) Transaktionen, die auf den Compact zugegriffen haben bestehen. Ist die der Fall, so wird ein Konfliktcode zum Agent übertragen, wo, die Operation in eine Warteschlange eingereiht wird, bis sich der Zustand des Compacts geändert hat. Traten keine Konflikte auf, führt der Compact die Operation auf dem Objekt aus und liefert ein Event mit neuem Wert des Objektes, Bestätigung des korrekten Ablaufs der Operation, und Informationen zum Redo oder Undo der Operation an den Compact Agent, der das Event zu Zwecken des Recovery in ein Eventlog schreibt und das Resultat der Operation an die Applikation zurückliefert.

Bei der Wiederherstellung der Verbindung mit dem festen Netzwerk wird durch den Compact Agent überprüft, ob die geänderten Compacts valid sind, d.h. ob die Zeitfrist ihrer Nutzung nicht abgelaufen ist. Ist dies nicht der Fall, so wird versucht die Frist zu verlängern. Schlägt dies fehl, muss ein weiterer, aufwendiger Schritt durchgeführt werden. Die Einträge des Eventlogs werden für alle (lokal) abgeschlossen Transaktionen durchgegangen. Wenn eine Transaktion einen nicht validen Compact gelesen oder geschrieben hat wird sie abgebrochen und alle Compacts, die sie verändert hat als nicht verfügbar markiert. Gleichfalls werden Transaktionen die nicht verfügbare Compacts gelesen haben abgebrochen und von ihnen veränderte Compacts als nicht verfügbar markiert. Wenn das Eventlog durchgegangen wurde, können alle validen und die nicht verfügbaren Compacts, nachdem für die nachträglich zurückgesetzten Transaktion Kontingenttransaktionen ausgeführt wurden, zur Eingliederung zum Server geschickt werden. Dabei generiert jeder geänderte Compact ein OP-Event, dass zum server-seitigem Compact geschickt wird, um die Daten auf dem Server in Übereinstimmung zu bringen. Wenn alle Operationen gesendet wurden, schickt das mobile Gerät ein Commit-Event an den Compact Manager (Server). Dabei werden die einzelnen Updates des Resynchronisationsprozesses zu einer Transaktion zusammengefasst, die auf dem Server abgeschlossen wird.

Durch den Einsatz der Compacts ist das System sehr flexibel einsetzbar. Es könnnen verschiedene Konzepte zur Transaktionsverarbeitung in mobilen Umgebungen eingesetzt werden. Neben den schon genannten verschiedenen Konsistenzgraden und den Zeitfristen zur Verhinderung von durch nicht freigegebenen Sperren heforgerufenen Blockierungen, kann auch die Escrow-Methode für fragmentierbare Objekte (vgl. 4.1.2) unterstützt werden.
Allerdings hat der Compact Agent keinerlei Datenbankfunktionaliät. Er verwaltet lediglich den Zugriff auf die Compacts. Methoden für den Zugriff und die Manipulation der Datenbankinhalte werden nicht einheitlich durch das System zur Verfügung gestellt, sondern müssen für die unterschiedlichen Datentypen innerhalb des jeweiligen Compacts spezifiziert werden. Es gibt also keine einheitliche Anfrageschnittstelle. Die Methoden, um auf die Daten zuzugreifen, sind an die Daten direkt gebunden. So sind auch insbesondere Anfragen über mehrere Compacts nicht möglich. Sollen im verbindungslosen Zustand eine Anfrageoperationen unterstützt werden, so muss ein Compact der den relevanten Datenbankausschnitt enthält angefragt werden.

Die Komplexität verlagert sich in Richtung festes Netzwerk, wo Compacts zusammengestellt werden müssen. Die geringe Komplexität auf Clientseite ermöglicht aber den Einsatz auch auf leistungsschwächeren Geräten.
Die mobilen Geräte erhalten die Compacts vom Compactmanager, der aus der Sicht der Datenbank als Client auftritt, welcher die zur Compactbereitstellung notwendigen Daten sperrt, und erst wieder freigibt, wenn die mobilen Geräte die Verarbeitung beendet haben. Andere Clients (aus dem festen Netzwerk) können während dieser Zeit nicht auf die Daten zugreifen, da sie nicht in die Vergabestrategie des Compact Managers einbezogen werden. Dazu müssten sie auch über Compacts auf Daten zugreifen, was zu einem „Flaschenhals“ führen und die Funktionen der Standarddatenbank völlig ungenutzt lassen würde. Durch das Sperren von Daten für mobile Transaktionen wirken sich die Verzögerungen aus dem mobilen Netzwerk negativ auf die Arbeit im festen Netzwerk aus.
Die Lösungen von Pro-Motion sind zwar gut für den Einsatz in moblen Umgebungen geeignet, aber nur schwer mit den Standarddatenbanken vereinbar. Dass die Autoren mit der Entwicklung eines Datenbankservers beschäftigt sind, der Compacts direkt unterstützt, unterstreicht dies.

# 6. Zusammenfassung

Die Eigenschaften mobiler Umgebungen, insbesondere das Vorhandensein von Verbindungsunterbrechungen und leistungsschwacher Geräte, erfordern neue Strategien der Transaktionsverarbeitung. Die unterschiedliche Gewichtung der Einschränkungen führte zu der Entwicklung einer Anzahl verschiedener Konzepte zur Transaktionsverarbeitung in mobilen Umgebungen. Die Konzepte wurden vor allem mit dem Ziel entwickelt, eine autonome Verarbeitung zu ermöglichen, die Verbindungskosten/ Transaktionsdauer zu reduzieren und die Blockierung bei sperrenden Verfahren zu vermeiden. Auf die wichtigsten dieser Konzepte wurde genauer eingegangen. Sie ließen sich einteilen in Konzepte, die den Zugriff auf die Daten beschränken und solche, die die Validität der Daten einschränken, um den Einschränkungen in mobilen Umgebungen gerecht zu werden.

Eine Analyse kommerzieller Lösungen zur Transaktionsverarbeitung in mobilen Umgebungen zeigte, dass dort auf Replikation und auf das klassische optimistische Verfahren gesetzt wird. Unterschiede zeigten sich in der Art der Konfliktbewältigungsstrategie. Der Ansatz auf logischer Ebene, bei dem zur Validierung der Transaktionen die ausgeführten Schritte wiederholt werden, zeigt sich dabei dem Ansatz, bei dem lediglich auf der Datenebene ein Vergleich der before images stattfindet, überlegen. Es können operationssemantische Eigenschaften ausgenutzt oder alternative Verarbeitungsschritte ausgeführt werden.

Die Umsetzung der erweiterten Transaktionskonzepte mit Standarddatenbanken erweist sich als schwierig. Sie unterstützen nur die klassischen Konzepte und stellen kaum Möglichkeiten für deren Erweiterungen und Anpassung an mobile Umgebungen zur Verfügung. Zur Anpassung optimistischer Verfahren können prozedurale Anfragesprachen eingesetzt werden, mit deren Hilfe Ausnahmebedingungen für den Fall eines Scheiterns der Validierung der Transaktion im festen Netzwerk spezifiziert werden können. Spezifische Elemente des pessimistischen Concurrency Controls können jedoch nur auf Anwendungsebene einfließen. Die Lösungen zur Transaktionsverarbeitung in mobilen Umgebungen sind damit sehr applikationsspezifisch. Lösungen, die flexiblere Ansätze verfolgen und weitere Konzepte unterstützen, sind nur schwer mit Standarddatenbanken vereinbar. Für Anwendungen in mobile Umgebungen müssen also noch eigenständige Lösungen zur Umsetzungen der vorhandenen erweiterten Transaktionskonzepte auf Applikationsebene entwickelt werden.

# Literatur:


[ABG90] Alonso R., D. Barbara, H. Garcia-Molina (1990): Data Caching Issues in an Information Retrieval Systems. in ACM Transaction on Database Systems, 15(3) Sep.1990 zitiert in [WC95]

[AE92] Agrawal D., A. El Abbadi (1992): Transaction Management in Database Systems in: Elmagarmid A.K.(Hsg.): Database Transaktion Models for Advanced Applications. Morgan Kaufmann Publishers

[Chr93] Chrysanthis, P.K. (1993): Transaction Processing in Mobile Computing Enviroment in Procceedings of the IEEE Workshop on Advances in Parallel and Distributed Sysstems October 1993

[Clou00] Cloudscape(2000): Cloudscape Version 3.6 Cloudscape Synchronisations Guide www.cloudscape.com

[GC89] Gray, C., D. Cheriton (1989):Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. in: Proceedings of the 12th ACM Symposium on Operating System Principles zitiert in [PBM+00]

[GHOS96] Gray, J., P. Helland, P. O'Neil, D. Sasha (1996) The Dangers of Replication and a Solution. in. Proceedings of the ACM SIGMOD '96

[Mad98] Madria S. K.(1998) :Transaction Models for Mobile Computing in: Proceedings of the 6th IEEE Singapore International Conference on Network, World Scientific, Singapore, July 1998

[MV99] Momin K.A., K. Vidyasankar (1999): Flexible Integration of Optimistic and Pessimistic Concurrency Control. in: Current Issues in Database and Information Systems, Proc. of the ADBIS-DASFAA 2000, Prague, Czech Republic, September 2000

[ONe86] O'Neil P.E. (1986): The Escrow Transactional Method. ACM Transactions on Database Systems , 11(4) zitiert in [WC95]

[Ora97] Oracle (1997): PL/SQL Users guide and reference – release 8.0 June 1997

[Ora99] Oracle (1999): Oracle8i Lite Oracle Replication Guide Release 4.0 unter: www.oracle.com

[Pap79] Papadimitriou C. H.(1979) :The Theory of Database Concurrency Control. Computer Science Press, 1979 zitiert in [AE92]

[PBM+00] Preguica N, C. Baquero, F. Moura, J. L .Martins, R. Oliveira, H. Domingos, J. O. Pereira and S. Duarte (2000): Mobile Transaction Management in Mobisnap. in: Current Issues in Database and Information Systems, Proc. of the ADBIS-DASFAA 2000, Prague, Czech Republic, September 2000

[PB94] Pitoura E., B.Bhargava (1994): Builing Information Systems for mobile Enviroments in Proc. of the 3rd International Conf. on Information and Knowledge Management

[PL91] Pu C., A. Leff.(1991): Replica Control in Distributed Systems : An Asynchronous Approach. Proc. of the ACM Sigmod Conf , May 1991 zitiert in [WC95]

[RK99] Reddy P.K., M. Kitsuregawa (1999): Speculativ Lock Management to Increase Concurrency in Mobile Enviroments in Mobile Data Access, Proceedings of the First International Conference, MDA'99, Honkong, China, December 1999

[VG93] Vossen G., M. Groß-Hardt(1993): Grundlagen der Transaktionsverarbeitung. Addison Wesley

[WC95] Walborn G.D., P.K. Chrysanthis (1995): Supporting Semantic-Based Transaction Processing in Mobile Database Applikations. in: Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems, Sep. 1995

[WC98] Walborn G.D., P.K. Chrysanthis (1998): Transaction Processing in PROMOTION. in: Proceedings of the 1999 ACM Symposium on Applied Computing 1999, February 28 - March 2, 1999, San Antonio, Texas, USA. ACM, 1999

[ZT98] Zaslavsky A., Z. Tari (1998) Mobile Computing:Overview and Current Status. in: Australien Computer Journal 30(2)