Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Aiko Böhm

Einsatz mobiler Datenbanken in einem Ferienclub-Szenario

Aiko Böhm

Einsatz mobiler Datenbanken in einem Ferienclub-Szenario

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.rer.nat. Kai v. Luck
Zweitgutachterin : Prof. Dipl.-Math. Helga Carls

Abgegeben am 29.August 2005

Aiko Böhm

Thema der Diplomarbeit

Einsatz mobiler Datenbanken in einem Ferienclub-Szenario

Stichworte

mobile Datenbanken, PDA, Replikation, Synchronisation

Zusammenfassung

Die Weiterentwicklung mobiler Geräte wie z.B. PDAs und Smartphones erlaubt es, auf ihnen komplexe Anwendungen auszuführen. Mobile Anwendungen erfordern eine eigene Datenhaltung, um ein arbeiten ohne Netzverfügbarkeit zu ermöglichen. Diese Anforderung erfüllen mobile Datenbanken. Für das Arbeiten in einem verteilten System mit einer zentraler Datenbank, werden die Daten auf die mobilen Clients repliziert und nach Bearbeitung, findet eine Synchronisation statt. In einem Anwendungssystem, in welchem mehrere Clients mit den Kopien der selben Daten arbeiten, kann es bei der Synchronisation zu Konflikten kommen. In dieser Diplomarbeit wird gezeigt, welche allgemeinen Strategien es für die Replikation, Synchronisation und Transaktionsverwaltung gibt. Diese werden für verteilte Datenbanken dargestellt, im besonderen wird hierbei auf mobile Datenbanken eingegangen. Anhand eines Beispiels wird exemplarisch gezeigt, wie eine Replikation beim Microsoft SQL Server 2000 CE Edition mit mobilen Geräten funktioniert. Es wird dargestellt, welche Funktionen der SQL Server für die Replikation bietet und wie Konflikte bei der Synchronisation aufgelöst werden können.

Aiko Böhm

Title of the paper

Use of mobile database in a vacations-club scenario

Keywords

mobile databases, PDA, replication, syncronisation

Abstract

The enhancements of mobile devices like e.g. PDAs and smartphones enables them to operate complex applications. Mobile applications need an appropriate data management for beeing able to work without net availability. This need is fulfilled by mobile databases. For working in a distributed system with a centralised database, the data is replicated onto the mobile clients, after the processing, a syncronisation takes place. In an application system in which multiple clients work with copies of the same data, conflicts during the syncronisation process can occur. In this thesis it is shown, which general strategies for the replication, the syncronisation and the administration of the transactions exist. These strategies are displayed for distributed databases, a special focus lays on mobile databases. With the help of an example, it shown exemplarily, how the replication of the Microsoft SQL Server 2000 CE Edition works with mobile devices.It is demonstrated, which functions the SQL Server offers for the replication and how comflicts during the syncronisation can be solved.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Mobile Geräte wie Mobiltelefone und Personal Digital Assistants (PDAs) sind mittlerweile weit verbreitet. Durch Weiterentwicklungen werden sie immer leistungsfähiger, so dass sie über ihre ursprünglichen Verwendungen, wie telefonieren, Kurzmitteilungen versenden oder Termin- und Adressverwaltung, hinaus auch anspruchsvollere mobile Anwendungen ausführen können. Diese komplexeren mobilen Anwendungen sind möglich durch die technischen Leistungssteigerungen der Prozessoren, durch mehr Speicherkapazität und eine verbesserte Darstellung durch neue Displays. Dazu kommt eine bessere Anbindung an bestehende Infrastrukturen durch den Ausbau mobiler Netze mit höheren Übertragungsgeschwindigkeiten. Mit den Anwendungen steigen auch die Menge der zu verwaltenden Daten, dies erfordert eine eigene Datenhaltung und einen Zugriff auf entfernt gespeicherte Daten.

Diese Entwicklung kommt dem Ansatz des *Mobile Computing* einen Schritt näher. Unter Mobile Computing als Oberbegriff versteht man die Benutzung von mobilen Geräten, die über eine drahtlose Netzwerkverbindung verfügen. Die Forderung nach einer allgegenwärtigen Verfügbarkeit von Informationen drückt der von Mark Weiser geprägte Begriff des *Ubiquitous Computing*, kurz *UbiComp*, aus. In seinem von 1991 veröffentlichtem Artikel „The Computer for the 21st Century“ geht er von vielen, für den Benutzer nicht als solche wahrnehmbaren, miteinander vernetzten Computern aus. Diese allgegenwärtigen Computer besitzen auch Informationen über ihren momentanen Einsatzort und Umgebung. Ein anderer Begriff ist der des *Nomadic Computing*. Dieser betont den Menschen, der außerhalb seines festen Arbeitsplatzes Zugriff auf Daten und Dienste haben möchte. Die Benutzung soll unabhängig vom Ort und Gerät sein. Außerdem soll es nicht relevant sein, ob er sich an einem stationären Ort aufhält oder in Bewegung ist.

In Ansätzen sind die Konzepte für ein Mobile Computing möglich. Doch für eine ständige Netzverfügbarkeit sind die Online-Verbindungen noch zu teuer *(UMTS)* oder nicht überall verfügbar (*Wireless LAN, kurz WLAN)*. Zudem sind die Netzverbindungen oft von schwankender Qualität und Geschwindigkeit. Es kommen deshalb mobile Datenbanken zum Einsatz. Sie können auch in einem Offline-Szenario weiterarbeiten. Mobile Datenbanken erlauben die Benutzung mobiler Geräte, die über eine drahtlose Netzanbindung verfügen oder

Daten in regelmäßigen Abständen über eine direkte Verbindung mit stationären Systemen synchronisieren. Dies hat veränderte Anforderungen an Datenbankmanagementsysteme zur Folge.

1.2 Zielsetzung

Ziel dieser Arbeit ist es zu zeigen, wie eine zentrale Terminverwaltung für mehrere Benutzer, die auf Kopien zugreifen und diese bearbeiten können, mit mobilen Datenbanken gelöst werden kann. Dafür wird dem Leser eine Einführung in die grundlegenden Begriffe und Konzepte, die für das Verständnis der Arbeitsweise von mobilen Datenbanken nötig sind, gegeben. An einem Beispiel-Szenario, das einen fiktiven Ferienclub beschreibt, wird dargestellt, wie eine Terminverwaltung, die auf mobile Geräte verteilt ist und ein autonomes arbeiten ohne Netzwerkverbindung ermöglicht, prototypisch umgesetzt werden kann. Die dabei auftretenden Anforderung können mit einer mobilen Datenbank, unter Einsatz von Replikation und Synchronisation der Daten, gelöst werden. Zur Realisierung wird der Microsoft SQL Server CE in Verbund mit einem zentralen Microsoft SQL Server und der Merge-Replikation genutzt. Ziel ist es, anhand der Anforderungen des Beispiel-Szenarios eine exemplarische Lösung mit den Microsoft Datenbanken zu zeigen.

1.3 Inhalt der Arbeit

Nach der in diesem Kapitel erfolgten Einleitung, die die Motivation für diese Arbeit mit anschließender Zielsetzung darlegt, ist die Arbeit wir folgt aufgebaut. In Kapitel 2 werden die nötigen Grundlagen für ein allgemeines Verständnis zu mobilen Datenbanken vermittelt. Hier werden mobile Datenbanken und die dazugehörigen mobilen Geräte vorgestellt. Dann folgt eine kurze Einführung in die verteilten Datenbanken. Dabei werden die Techniken zur Replikation und Synchronisation vorgestellt. Darauf aufbauend werden die Verfahren für mobile Datenbanken , insbesondere mobile Transaktionen, beschrieben. Am Ende werden Konflikte dargestellt, die aus den mobilen Verfahren resultieren.
Die Beschreibung des Beispiel-Szenarios eines Ferienclubs erfolgt in Kapitel 3. Hier wird der „Ferienclub“ allgemein vorgestellt und beschrieben, wie ein Urlaub in diesem Szenario aussehen würde. Daraus ergeben sich die Anforderungen und Probleme für die Realisierung des Beispiels. Aufbauend auf dem vorherigen Kapitel, wird in Kapitel 4 eine Entscheidung für die Architektur einer verteilten Datenbank, die mit Replikation und Synchronisation die Daten auf mobile Datenbenken verteilt, getroffen. Daraufhin wird eine für diese Aufgabe geeignete mobile Datenbank vorgestellt. Anschliessend wird anhand dieser Datenbank ein Realisierungsweg aufgezeigt.

Den Abschluss bildet ein kurzes Resümee, in dem kurz auf die, bei der Umsetzung aufgetretenen, Probleme und mögliche Lösungsansätze eingegangen wird.

2 Grundlagen

2.1 Mobile Datenbanken

Ein mobiles Datenbanksystem wird auch als *Small-Footprint-Datenbanksystem* bezeichnet. Es ist ein Datenbanksystem, dass speziell für den Einsatz auf mobilen Geräten entworfen wurde. Als mobile Geräte kommen in der Regel PDAs, mobile Telefone und Notebooks zum Einsatz (siehe 2.2). Das mobile Datenbanksystem stellt diesen Geräten Datenbankfunktionalitäten zur persistenten Speicherung von Daten zur Verfügung. Es ist an die knappen Ressourcen auf mobilen Geräten angepasst. Aus diesem Grund ist bei mobilen Datenbanken die Größe und Funktionalität reduziert, um unter den eingeschränkten Bedingungen eine akzeptable Leistung zu bringen.
Mobile Datenbanken kommen überall dort zum Einsatz, wo eine ständige Online-Verbindung nicht gewährleistet ist. Hier spricht man von einem *Offline-Szenario*. In einem Online-Szenario wird eine ständige Netzwerkverbindung vorausgesetzt. Hier ist der mobile Client ein Thin-Client, der eine grafische Schnittstelle für den Zugriff auf entfernte Daten darstellt. Eigene Datenbankfunktionalitäten sind in diesem Szenario nicht vorhanden. In einem Offline-Szenario dagegen wird davon ausgegangen, dass keine dauerhafte Netzwerkverbindung besteht. Hier unterscheidet man zwischen zwei Betriebszuständen: Ist der mobile Client mit der Basisstation verbunden, befindet er sich im Zustand „Schwach-Verbunden“ (Connected Mode); wenn keine Verbindung besteht ist er im Betriebszustand „Verbindungslos“ (Disconnected Mode).
Ein mobiles Datenbanksystem ist in der Praxis oft in ein Informationssystem eingebunden. Hier werden Daten einer zentralen Datenbank auf den mobilen Client repliziert. Um die Menge der Daten in der mobilen Datenbank zu reduzieren, werden nur die auf dem mobilen Client benötigten Daten repliziert. Um die Aktualität der Daten zu gewährleisten, werden die replizierten Daten mit der zentralen Datenbank synchronisiert. Mobile Datenbanken werden von verschiedenen Herstellern angeboten. Sie unterscheiden sich in Funktionalität und Umfang. Einige Beispielprodukte sind IBM DB2 Everyplace, Oracle Lite, Microsoft SQL Server CE und Sybase UltraLite.

2.2 Mobile Geräte

Mobile Geräte sind handliche Computer für „unterwegs“. Sie dürfen nicht zu groß und schwer sein, um sie in einem mobilen Umfeld einsetzen zu können. Auch der Kostenfaktor spielt eine Rolle. Um diese Bedürfnisse zu erfüllen, unterliegen sie Einschränkungen bei der Hardware und Bedienung. Bei den Einschränkungen im Hardwarebereich handelt es sich um eine vergleichsweise geringe Leistung der Prozessoren und eine kleinere Speicherkapazität. Bei der Bedienung ist die Darstellung durch kleinere Displays eingeschränkt und die Eingabe erfolgt über kleine Tastaturen oder Stifte. Dadurch werden die Benutzerinteraktion mit den mobilen Geräten erschwert und die Anwendungen müssen an die limitierten Ressourcen angepasst sein. Ein weiterer wichtiger Punkt ist die begrenzte Akkulaufzeit und damit verbundene Verfügbarkeit der Geräte.

2.2.1 Notebook

Notebooks oder *Subnotebooks* (eine etwas kleinere und leichtere Variante eines Notebooks) sind die mobile Variante eines stationären Computers. Sie sind so kompakt, dass sie unterwegs mitgenommen werden können. Notebooks haben inzwischen die Leistungsfähigkeit von stationären Rechnern. Sie verfügen über ein Desktop-Betriebssystem und können dieselben Anwendung ausführen. Der Bildschirm (Display), die Tastatur und Bedienungsgeräte sind in die Geräte integriert und kleiner als bei herkömmlichen PCs. Zusätzliche Peripheriegeräte sind entweder (auch) integriert oder können über Einsteckkarten nachgerüstet werden. Neben der etwas schlechteren Bedienung ist die Haupteinschränkung bei Notebooks die Betriebsdauer. Diese ist durch die Akkulaufzeit vorgegeben und hängt stark von der Leistungsfähigkeit der eingesetzten Hardware ab.

2.2.2 Personal Digital Assistant

Personal Digital Assistants (PDA) sind handliche Universalgeräte, die ohne lange Startphase einsatzbereit sind. Ursprünglich waren sie für die Verwaltung persönlicher Daten bestimmt, wie Adressverwaltung und Terminkalender. Inzwischen sind sie mit eigenem Betriebssystem und einer Vielzahl von Anwendungen ausgestattet.
Zur Bedienung werden sie in einer Hand gehalten und über eine Minitastatur oder mit einem Stift auf einem Touch-Screen bedient. Für Texteingaben mit dem Stift wird eine Tastatur auf dem Display eingeblendet oder die Eingabe erfolgt über Schrifterkennung. PDAs haben durch technische Weiterentwicklung eine hohe Leistungssteigerung erfahren. Im Vergleich

zu Notebooks haben sie aber eine deutlich reduzierte Prozessorleistung, geringere Speicherkapazität und ein kleineres *Display* mit niedriger Auflösung. Die Akkulaufzeiten sind länger als bei Notebooks, dies hängt aber auch mit der für PDAs typischen nur kurzen Benutzungszeiten zusammen. Zur Speicherung besitzen sie einen internen Batterie-gepufferten Speicher, der durch Einsteckkarten erweitert werden kann. Die neusten PDAs haben inzwischen eine Mini-Festplatte, die eine Speicherkapazität von bis zu 4 GByte haben kann (PalmOne) [dal/c´t (2005a)]. Andere Komponenten wie z. B. Bluetooth oder Wireless-LAN sind je nach Ausstattung in die Geräte integriert oder können über Einsteckkarten hinzugefügt werden.
PDAs sind die Hauptanwendungsgeräte für mobile Datenbanken.

2.2.3 Mobiltelefone und Smartphones

Mobiltelefone sind Spezialgeräte, die ausschließlich zum Telefonieren geeignet sind. Sie verfügen zwar über ein Adressbuch und einen Terminkalender, für den universalen Einsatz von Anwendungen wie z. B. eine mobile Datenbank sind sie aber nicht geeignet.
Smartphones sind eine technische Weiterentwicklung hin zu „intelligenten“ Mobiltelefonen. Ein Smartphone ist eine Kombination aus Mobiltelefon und PDA. Sie sind meist nicht viel größer als herkömmliche Mobiltelefone und um zusätzliche Funktionen der PDAs erweitert. Die Leistungsfähigkeit ist zur Zeit noch geringer als bei PDAs, deshalb sind die aktuellen Smartphones noch nicht für den Einsatz von mobilen Datenbanken geeignet. Aber der Trend geht eindeutig in die Richtung von Smartphones und deshalb werden sie voraussichtlich in kurzer Zeit die heutige Leistungsfähigkeit von PDAs erreichen. Der Trend lässt sich an den Absatzzahlen ablesen: Die „reiner“ PDAs sind rückläufig, die von Smartphones legen zu [dal/c´t (2005b)]. Dies liegt auch an den günstigen Preisen für Smartphones, die bei gleichzeitigem Abschluss eines Vertrags von den Mobilfunkanbietern subventioniert werden.

2.3 Verteilte Datenbanksysteme

In *verteilten Datenbanksystemen* sind die Daten auf mehrere Datenbanksysteme physisch verteilt, wobei die einzelnen Datenbanksysteme in der Regel auf verschiedenen und geographisch verteilten *Knoten* laufen. In verteilten Datenbanksystemen sind mit Knoten Rechner gemeint, auf denen ein eigenes Datenbanksystem läuft. Deshalb ist folgend nur noch von Knoten die Rede. Die auf den Knoten verteilten Daten bilden als ganzes eine logische Einheit. Bei der Verteilung von Daten unterscheidet man zwischen Fragmentierung und Replikation.
Die Knoten kooperieren über Nachrichtenaustausch miteinander, um Datenbankoperationen auf dem verteilten Datenbestand zu bearbeiten. Dies geschieht auf Datenbanksystemebene,

so dass die Verteilung der Daten gegenüber Anwendungen und Benutzern keine Auswirkungen hat. Die Gewährleistung einer solchen Verteilungstransparenz ist die Hauptforderung an verteilte Datenbanksysteme.
Weitere Anforderungen nach [Date (1987)] sind:

- **Lokale Autonomie:** Der Zugriff der einzelnen Knoten auf die lokal gehaltenen Daten kann unabhängig von anderen Knoten (d.h. ohne Kommunikation mit diesen) erfolgen. Jeder Knoten kann über den Zugriff auf „seine“ lokalen Daten bestimmen, er ist verantwortlich für Datensicherheit und Datenintegrität und überwacht alle Operationen auf diesen Daten selbst.
- **Unabhängigkeit:** Jeder Knoten ist unabhängig von zentralen Systemfunktionen.
- **Hohe Verfügbarkeit:** Keine Unterbrechung im Datenbankbetrieb bei Systemfehlern (Ausfall von Knoten) und Konfigurationsänderungen.
- **Ortstransparenz:** Der physische Speicherort der Daten bleibt dem Benutzer verborgen.
- **Fragmentierungstransparenz:** Die verteilte Speicherung von Relationen der Datenbank auf verschiedene Knoten und damit verbundene Fragmentierung bleibt vor dem Benutzer verborgen.
- **Replikationstransparenz:** Die Speicherung von redundanten Daten durch Replikation auf verschiedenen Knoten soll vor dem Benutzer verborgen bleiben. Die Administration der replizierten Daten obliegt ausschließlich dem Datenbankmanagementsystem.
- **Verteilte Anfragebearbeitung:** Datenbank-Operationen können während der Ausführung auf Daten unterschiedlicher Knoten zugreifen. Eine effiziente Ausführung wird durch das verteilte Datenbanksystem ermöglicht.
- **Verteilte Transaktionsverwaltung:** Einhaltung der Transaktionseigenschaften in einer verteilten Datenverarbeitung.
- **Hardware-Unabhängigkeit:** Der Datenbankzugriff muss von verschiedenen Hardware-Plattformen möglich sein.
- **Betriebssystem-Unabhängigkeit:** Die Datenbanknutzung sollte unabhängig vom benutzten Betriebssystem sein.
- **Netzwerk-Unabhängigkeit:** Die verwendeten Netzwerke und Netzwerkprotokolle zwischen den Knoten sollen keinen Einfluss auf die Datenbankverarbeitung haben.
- **Datenbanksystem-Unabhängigkeit:** Die Benutzung von unterschiedlichen (heterogenen) Datenbanksystemen auf den verschiedenen Knoten soll möglich sein.

Die Schemaarchitektur für ein verteiltes Datenbanksystem ist ein erweitertes Modell des Ansi-SPARK-Modells für Datenbanksysteme und wird in Abbildung 2.1 dargestellt und ist in [Dadam (1996)] beschrieben.

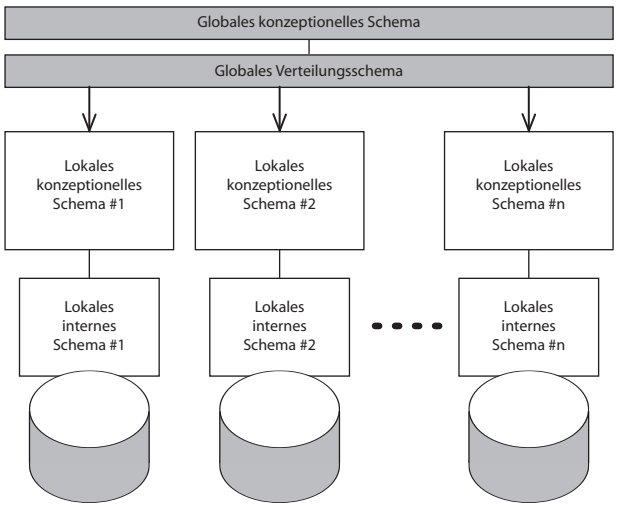

Abbildung 2.1: Schemaarchitektur für verteilte Datenbanksysteme

Das globale konzeptionelle Schema beschreibt die logische Sicht auf die verteilte Datenbank. Dies sichert die Verteilungstransparenz. Das globale Verteilungsschema sorgt für die Datenverteilung innerhalb des Systems. Hier wird zwischen der logischen und der physischen Ebene unterschieden. Auf der logischen Ebene erfolgen die Angaben zur Fragmentierung der Relationen. Die physische Ebene (auch Allokation genannt) beschreibt die genaue Festlegung des Speicherorts. Dies gilt auch für redundante Speicherung von Daten bei der Replikation. Darunter liegen die lokalen Schemata, die die lokalen Datenbanksysteme auf den Knoten darstellen. Dadurch wird ein höherer Grad an Autonomie an den Knoten erreicht.

Performanz: Durch die Verteilung der Daten auf mehrere Knoten können Anfragen gleichzeitig auf mehreren Knoten ausgeführt werden.

Verlässlichkeit und Verfügbarkeit: Durch die mehrfache Speicherung von Daten auf unterschiedliche Knoten ist das Gesamtsystem nicht von einem Knoten abhängig.

Erweiterbarkeit: Verteilte Datenbanksysteme können einfacher erweitert werden, indem neue Knoten hinzugefügt werden. Dies geht in einem verteilten Datenbanksystem einfacher als in einem zentralen.

Wirtschaftlichkeit: Die Daten werden an den Knoten abgespeichert, auf denen sie am meisten abgefragt werden. Dies kann auch mittels Replikation geschehen. Der Kommunikationsaufwand wird hierdurch verringert.

2.4 Architektur mobiler Datenbanken

Bei der Entwicklung von mobilen Datenbanksystemen wird zwischen zwei grundlegenden Ansätzen für die Anwendungsarchitektur unterschieden [Mutschler und Specht (2003)]:

Eine klassische Client-/Serverarchitektur wird durch eine spezielle Middleware erweitert. Diese Architektur besteht aus den mobilen Datenbanken auf den Clients, einem Replikations-/Synchronisationsserver als Middleware und einem Datenbankserver als Backend. Das mobile Datenbanksystem ist eine eigene Komponente und von der Anwendung getrennt. Die Anwendung greift über Schnittstellen auf die Daten zu.

Die mobile Datenbank wird vollständig in die Anwendung integriert. Die Datenbankfunktionalität wird auf die Anwendung abgestimmt. So wird eine auf die Anwendung optimierte Datenbank erzeugt. Die Komponenten der Datenbank wie Tabellen und Zugriffslogik werden in die Anwendung kompiliert. Dadurch gibt man die Unabhängigkeit der Datenbank von der Anwendung auf.

2.5 Replikation und Synchronisation

Replikation ist die gewollte Einführung von Datenredundanz. Synchronisation ist der Abgleich von redundanten Daten, an denen Änderungen durchgeführt wurden. Replikation und Synchronisation sind für die mobile Datenverarbeitung wichtig, weil erst durch die Möglichkeit, redundante Daten zu erzeugen und später zu synchronisieren, ein Arbeiten in einem Offline-Szenario möglich wird. In einer mobilen Umgebung werden den Clients durch Replikation Daten zur Verfügung gestellt. Diese werden lokal auf den Clients gespeichert. So können auch im nicht verbundenen Zustand (Disconnected Mode) die Daten bearbeitet werden. Um die Datenkonsistenz innerhalb des Gesamtsystems zu gewährleisten, müssen die im unverbundenen Zustand geänderten Daten später wieder mit den Originaldaten synchronisiert werden.

Eine Replikation von Daten ohne Synchronisation gibt es ausschließlich bei der ersten Verteilung der redundanten Daten auf einen Client. Danach ist die Replikation immer ein Teilprozess der Synchronisation.

2.5.1 Replikation

Die Methoden und Techniken zum Anlegen redundanter Datenelemente werden mit dem Begriff Replikation zusammengefasst. Der ursprüngliche Einsatz kommt aus den verteilten Systemen/Datenbanken wie in Kapitel 2.3 bereits beschrieben. Dort wird Replikation vor

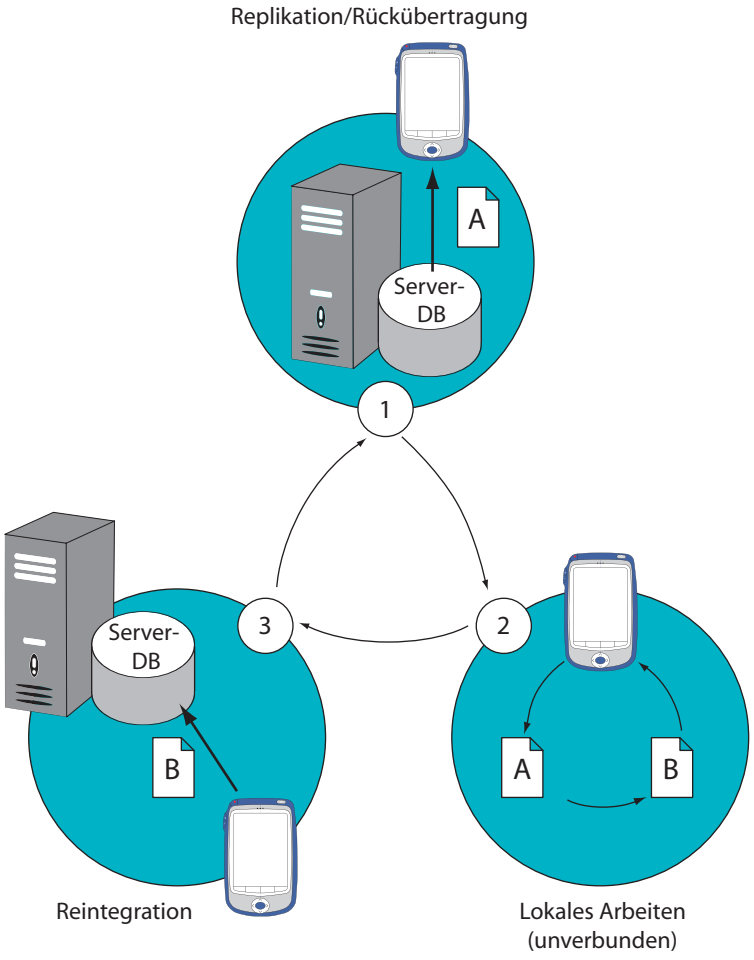

Abbildung 2.2: Replikation und Synchronisation

allem bei kritischen Daten und Daten mit hoher Zugriffsrate angewendet. Alle Knoten, die replizierte Daten vorhalten, bilden zusammen die Replikationsumgebung.

Durch Replikation wird die Verfügbarkeit des Gesamtsystem erhöht, weil auch dann, wenn ein Knoten ausfällt, auf den redundanten Daten auf einem anderen Knoten weitergearbeitet werden kann. Diese redundanten Daten schützen das System auch vor Datenverlust: Sind auf einem Knoten die Daten verloren gegangen, kann der Datenverlust durch die Replikate auf anderen Knoten behoben werden. Durch die mehrfache Speicherung der Daten sinken die Kommunikationskosten, da vielfach auf lokale Daten zugegriffen werden kann. Neben den geringeren Kommunikationskosten wird auch die Performanz des Systems gesteigert, weil es bei lokalen Zugriffen zu weniger Verzögerungen kommt. Außerdem können Abfragen auf redundanten Daten gleichzeitig ausgeführt werden. Für nicht lokal vorhandene Daten kann, wenn die Daten auf mehreren Knoten existieren, derjenige gewählt werden, der die geringste Last hat. Darüber hinaus ist Replikation die Grundvoraussetzung um mit mobilen Clients zu arbeiten, die nicht mit dem System verbunden sind. Allerdings führt Replikation bei Änderungen an den redundanten Daten zu höherem Kommunikations- und Verwaltungsaufwand, da die veränderten Daten auf mehreren Knoten aktualisiert werden müssen.

2.5.2 Synchronisation

Die Synchronisation ist für den Erhalt eines konsistenten Zustandes der Daten innerhalb der Replikationsumgebung zuständig. Synchronisation bedeutet in einer Replikationsumgebung die Bewahrung und Wiederherstellung der Konsistenz durch geeignete Synchronisationsverfahren. Dies wird nötig, wenn auf den redundanten Daten lokale Änderungen stattfinden. Diese lokalen Änderungen müssen mit den Kopien abgeglichen werden um einen konsistenten Datenbankzustand zu gewährleisten. Ein Synchronisationsprozess besteht aus zwei Phasen: der Reintegration und der Rückübertragung.
Bei der Reintegration findet ein Abgleich der replizierten Daten mit der Replikationsquelle statt. Falls nötig muss eine Konfliktbehandlung erfolgen. Dabei ist zu beachten, dass Änderungen auf den Replikaten und auf der Replikationsquelle stattgefunden haben können. Bei der Rückübertragung werden die aktualisierten Daten der Replikationsquelle zurück übertragen. Werden die Reintegration und Rückübertragung aufeinander folgend ausgeführt, spricht man von bidirektionaler Synchronisation, wenn nur eine der beiden Phasen ausgeführt wird von unidirektionaler Synchronisation.

2.5.3 Zielkonflikte

Aus den Kapiteln Replikation und Synchronisation ergeben sich die drei Hauptziele/Anforderungen Verfügbarkeit, Konsistenz und Performanz. Diese lassen sich jedoch

nicht gleichzeitig erreichen. Änderungen zugunsten einer der Kriterien haben eine Minimierung der Anderen zur Folge – deshalb spricht man von einem Zielkonflikt (Abbildung 2.3).

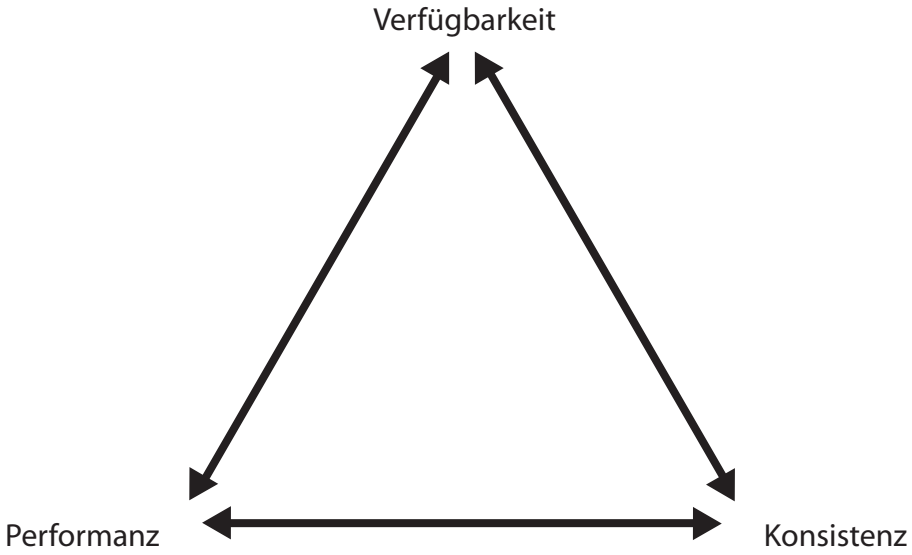

Abbildung 2.3: Zielkonflikt Quelle: [Dadam (1996)]

Je öfter ein Datenelement vorhanden ist, desto größer ist die Verfügbarkeit. Dies erhöht die Fehlertoleranz (etwa bei Ausfall von Knoten) und ist für das Arbeiten mit mobilen Clients Grundvoraussetzung. Bei Sicherung der Konsistenz verhält es sich entgegengesetzt, da hier bei Änderungen alle Replikate an das geänderte Datenelement angepasst werden müssen. Daher gilt hier: Je wenige Kopien eines Datenelements bestehen, desto einfacher ist die Datenkonsistenz zu bewahren. Um die Performanz zu steigern kann man einerseits die Anzahl der Kopien erhöhen und so die Lesezugriffe beschleunigen, andererseits steigt mit der Anzahl der Kopien der Aufwand bei Änderungen, da alle Replikate geändert werden müssen.

2.5.4 Klassische Replikationsverfahren

Zur Lösung des Zielkonflikts gibt es verschiedene Replikations-/ Synchronisationsverfahren. Die Verfahren haben unterschiedliche Primärziele, anhand derer sie sich in Gruppen einteilen lassen (siehe Abbildung 2.4).

Auf der linken Seite sind die konsistenzerhaltenden Verfahren (strenge Konsistenz) zu sehen, sie haben die strikte Durchsetzung der Konsistenz als Primärziel. Auf der rechten

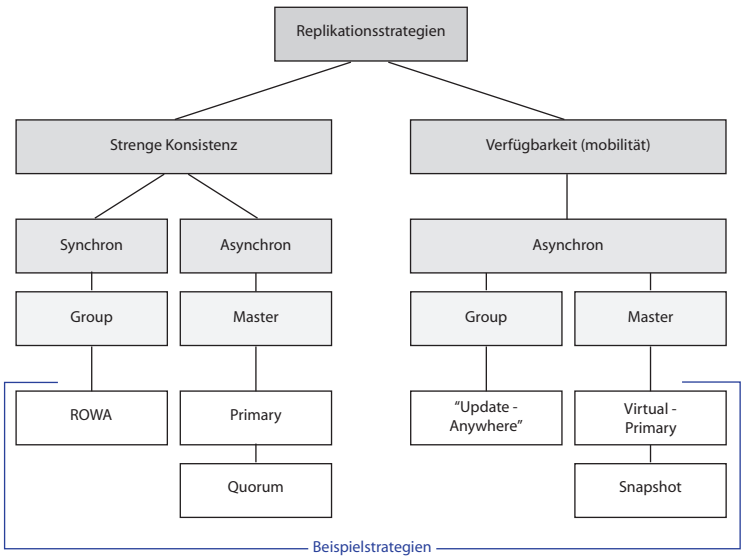

Abbildung 2.4: Replikations- und Synchronisationsverfahren

Seite stehen die Verfahren, die eine größte mögliche Verfügbarkeit erreichen wollen (schwache Konsistenz). Bei den konsistenzerhaltenden Verfahren ist jede Kopie immer auf dem aktuellen Stand. Um dies zu erreichen wird eine Änderung entweder auf allen oder auf keinem Replikat durchgeführt. Dadurch entsteht ein erheblicher Aufwand, der zu Lasten der Performanz geht. Auch die Verfügbarkeit und Fehlertoleranz ist dann nicht mehr gegeben, da bei Ausfall eines Knotens das System nicht verfügbar ist. Um ein Weiterarbeiten bei vorübergehendem Ausfall eines Knoten zu ermöglichen, wird die strenge Konsistenz aufgeweicht. Bei dieser Variante müssen nicht alle Kopien synchron aktualisiert werden, sondern können nach einer bestimmten Zeit aktualisiert werden.
Bei verfügbarkeitserhaltenden Verfahren werden Inkonsistenzen in bestimmten Situationen toleriert. Dies geschieht aber innhalb gewisser Grenzen. Auch hier muss ein konsistenter Zustand nach einer bestimmten Zeit wieder hergestellt sein.

Weiterhin kann zwischen *synchroner (eager)* und `asynchroner (lazy)` Replikation unterschieden werden. Synchrone Replikation arbeitet mit Sperren und aktualisiert sofort alle Kopien bei Änderungen. Dadurch wird eine hohe Datenkonsistenz erreicht und es treten keine Replikationskonflikte auf. Um Änderungen auf redundanten Daten zu machen, müssen alle Knoten, die eine Kopie dieser Daten besitzen, verfügbar sein. Für den Einsatz mobiler Endgeräte ist die synchrone Replikation deshalb nicht geeignet, weil hier nicht immer alle Knoten verfügbar sind.

Bei asynchroner Replikation können die Daten zu einem späteren Zeitpunkt aktualisiert wer-

den. Dadurch kann auf Sperren verzichtet werden und die Verfügbarkeit der Daten wird erhöht. Dies hat aber zur Folge, dass es zu temporären Inkonsistenzen kommen kann. Bei einem späteren Abgleich der Daten müssen auftretende Konflikte aufgelöst werden.

Die Unterscheidung zwischen *Group* und *Master* legt fest, wer der Besitzer des Datenbestands ist. Bei „Master“ hat jedes Objekt eine Primärkopie (primary copy) auf einem Masterknoten. Nur dieser kann die Primärdaten aktualisieren. Auf alle anderen Replikate kann nur lesend zugegriffen werden. Wenn ein Knoten Änderungen an den Daten durchführen möchte, wird dieser Änderungswunsch an den Master weitergegeben. Der Master führt die Änderung an der Primärkopie durch und übermittelt die Änderung an alle Knoten, die eine Kopie besitzen. Bei „Group“ können auf jedem Knoten, der eine Kopie besitzt, Änderungen durchgeführt werden. Die Änderungen werden dann an alle anderen Kopien weitergegeben.

Im Folgenden werden die gebräuchlichsten Replikations- und Synchronisationsverfahren vorgestellt. Vergleiche [Mutschler und Specht (2004)] und [Dadam (1996)].

Konsistenzerhaltende Verfahren

Read-One-Write-All (ROWA)

ROWA ist ein synchrones Verfahren. Bei Änderungen werden alle Kopien parallel aktualisiert. ROWA garantiert, dass stets alle Kopien konsistent sind. Alle zu aktualisierenden Kopien sind bis zum Abschluss der Änderungstransaktion gesperrt. Dadurch gibt es keine veralteten oder inkonsistente Daten. Für das ROWA-Verfahren müssen alle Kopien erreichbar sein.
Eine Variante des ROWA-Verfahrens ist **Write All Available**. Bei dieser Variante existiert ein Verzeichnis über alle verfügbaren Kopien. Vor jedem Aktualisierungsvorgang wird dieses Verzeichnis zu Rate gezogen. Die nicht erreichbaren Replikate (ein mobiler Client im Offline-Modus) müssen asynchron aktualisiert werden. Ein zweite Variante ist das **Missing Update**-Verfahren. Hier wird im fehlerfreien Zustand (alle Replikate sind erreichbar) die klassische ROWA-Variante benutzt. Bei auftretenden Netzpartitionierungen wechselt es in den Fehlermodus und ein anderes Verfahren kommt zum Einsatz. In einer mobilen Umgebung, in der der Fehlerfall die Regel ist, würde das ROWA-Verfahren nicht zum Einsatz kommen.

Primary-Copy

Beim Primary-Copy Verfahren gibt es eine zentrale Master-/Primärkopie. Die Aktualisierung bei Änderungstransaktionen der anderen Kopien erfolgt in zwei Schritten. Im ersten Schritt wird die Primärkopie gesperrt und auf ihr die Änderung durchgeführt und abgeschlossen. Diese Primärkopie ist nun im zweiten Schritt dafür zuständig die anderen Kopien asynchron zu aktualisieren. Im Vergleich zum ROWA-Verfahren wird nur die Primärkopie gesperrt, damit die Änderung an die anderen Kopien weitergegeben werden kann. Das Primary-Copy

Verfahren kann mit Einschränkungen in mobilen Umgebungen eingesetzt werden. Voraussetzung ist, dass ein erreichbarer stationärer Knoten als Primärkopie gewählt wird. Jedoch muss für jede Änderungstransaktion eine Netzwerkverbindung hergestellt werden. Änderungstransaktionen sind so nicht im unverbundenen Zustand möglich. Für den Einsatz in einer mobilen Umgebung gibt es eine spezielle Variante, das Virtual-Primary-Copy Verfahren (siehe 2.5.4).

Quorum
Das Quorum Verfahren basiert auf Abstimmung. Jede Kopie besitzt eine Stimme. Um Änderungen durchzuführen, muss eine Transaktion eine vorher festgelegte Anzahl von Stimmen auf sich vereinen. Sobald eine Kopie erreichbar ist wird sie gesperrt und gilt als Stimme. Quorum an sich bedeutet eine entscheidungsfähige Anzahl an Stimmen zu bekommen. Hier spricht man von der Mehrheit der Stimmen, auch Majority consensus genannt. Welche Stimmenanzahl der Mehrheit entspricht kann definiert werden. Es kann ein Quorum für Lese- und für Schreibzugriffe geben. Ein Beispiel für eine Definition der Mehrheit bei einer Schreibtransaktion mit n Knoten wäre: $Q_W = (n/2) + 1$
In einer Variante des Quorum Verfahrens können die einzelnen Kopien eine unterschiedliche Stimmgewichtung bekommen. So ist es möglich, dass einige Kopien eine höhere Stimmzahl haben als andere.
Dabei kann zwischen statischem Quorum, bei dem die Mehrheiten fest vorgegeben sind, und dynamischem Quorum, bei dem die Stimmzahl zur Laufzeit angepasst wird, unterschieden werden. Beim dynamischem Quorum wird die Mehrheit auf Grundlage der zum Abstimmungszeitunkt verfügbaren Gesamtstimmen ermittelt. Das Quorum-Verfahren ist nur bedingt zum Einsatz in mobile Umgebungen geeignet, da für jede Änderungstransaktion und für das Einholen der Stimmen eine Netzwerkverbindung nötig ist. Mobile Knoten, die nicht online sind, können nicht an der Abstimmung teilnehmen.

Verfügbarkeitserhaltende (mobile) Verfahren

Virtual Primary Copy
Das *Virtual Primary Copy* Verfahren ist eine Erweiterung des Primary Copy Verfahrens. Der mobile Client erhält die Primärkopie und kann auf ihr uneingeschränkt arbeiten. Da mobile Clients nicht immer erreichbar sind und alle anderen Kopien dann blockiert wären, wird zu jeder realen Primärkopie eine virtuelle Primärkopie erstellt. Ist die reale Primärkopie nicht erreichbar, übernimmt die virtuelle Primärkopie gegenüber den anderen Replikaten die Funktion der Primärkopie. Später findet ein Abgleich zwischen der regulären und der virtuellen Primärkopie statt.

Snapshot
Die *Snapshot-Replikation* basiert auf *Snapshots*. *Snapshots* sind materialisierte Sichten auf

eine Original/Master-Tabelle. Ein Snapshot kann einzelne Tupel oder komplette Relationen enthalten. Replikationsquelle ist eine zentrale Datenbank. Normalerweise wird die Snapshot-Replikation nur für lesenden Zugriff verwendet [Schneebauer (2003)]. In kommerziellen Anwendungen werden Snapshots auch als Teil anderer Replikationsverfahren eingesetzt. So nutzt der *Microsoft SQL Server* die Snapshot-Replikation zum erstmaligen Ausbringen der Replikate. Bei der *Oracle Database 10g lite* Datenbank wird unter *einfachen* und *komplexen Snapshots* unterschieden. Einfache Snapshots basieren auf einer einzelnen Tabelle der Original-Tabelle. Auf diesen Snapshots können beliebige Änderungen durchgeführt werden, die später mit der Original-Tabelle abgeglichen werden. Komplexe Snapshots können auf mehreren Tabellen basieren und schon vorhandene Snapshots mit einbeziehen. Auf komplexe Snapshots kann nur lesend zugegriffen werden. Snapshots unterstützen das Arbeiten von mobilen Clients im nicht verbundenen Zustand. Wenn im unverbunden Zustand Änderungen gemacht werden, müssen diese Änderungen bei der Synchronisation mit der zentralen Datenbank abgeglichen werden. Dabei können Synchronisationskonflikte auftreten, die erkannt und aufgelöst werden müssen. Eine snapshotbasierte Replikation im mobilen Umfeld basiert oft auf dem *Publish-Subscribe-Modell*. Dabei werden Publikationen erstellt, die aus mehreren Snapshots bestehen. Die Publikationen werden den mobilen Clients anhand von *Subskriptionen* zugeordnet [Mutschler und Specht (2004) und [Mutschler und Specht (2003)].

2.6 Mobile Transaktionen

Mobile Transaktionen haben, wie der Name schon sagt, ihren Ursprung in einer mobilen Umgebung, in der sie ausgeführt werden. Dies bedeutet, dass die Transaktionen zumindest mit dem Schwach-Verbunden-Modus (2.1) bei mobilen Clients funktionieren müssen.
Die verschiedenen Transaktionsmodelle haben ein unterschiedliches Verständnis von einer mobilen Transaktion. Einige Transaktionsmodelle gehen von einer langlebigen Transaktion aus. Diese langlebige Transaktion wird in ihrer Ausführung nicht unterbrochen, egal in welchem Betriebszustand der mobile Client sich befindet (siehe Abbildung 2.5). Andere Transaktionsmodelle sehen für mobile Transaktionen ein zweistufiges Verarbeitungsmodell vor. Bei Transaktionen in Datenbanksystemen wird allgemein davon ausgegangen, dass die Transaktionen *ACID* konform sind. Unter dem ACID-Prinzip versteht man die vier Anforderungen Atomicity, Consistency, Isolation und Durability. Sie schließen Inkonsistenzen nach Ausführung von Transaktionen aus. Bei mobilen Transaktionen werden bei einigen Transaktionsmodellen die ACID-Eigenschaften nur eingeschränkt oder gar nicht erfüllt.
Nach [Mutschler und Specht (2004)] kann man die mobilen Transaktionsmodelle in zwei unterschiedliche Klassen einteilen. Das Kriterium zu Unterscheidung ist, ob ein Transaktionsmodell den *Disconnected-Mode* bei mobilen Geräten unterstützt. Die vorgestellten Transakti-

onsmodelle gehen von einer Umgebung aus, in der ein zentraler Datenbankserver mehreren mobilen Clients gegenübersteht.

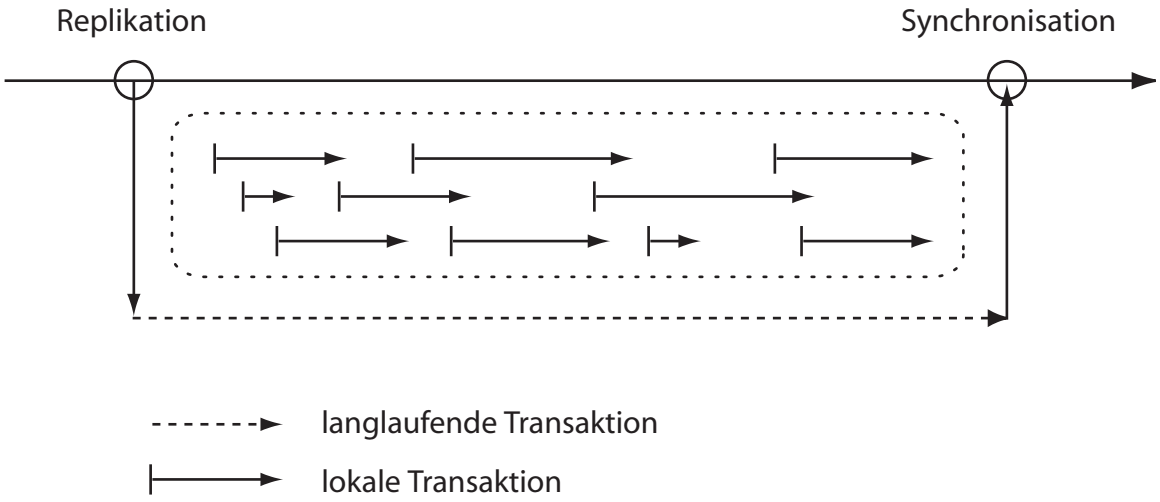

Abbildung 2.5: Langlebige (oder langlaufende) Transaktion, Quelle: [Silberschatz u. a. (2005)]

2.6.1 Transaktionsmodelle der Klasse 1

Die Transaktionsmodelle dieser Klasse können lediglich in Online-Szenarien eingesetzt werden. Bei Transaktionen wird keine ständig bestehende Netzwerkverbindung vorausgesetzt, es wird aber davon ausgegangen, dass bei Bedarf eine Verbindung hergestellt werden kann. Die Transaktionen arbeiten entweder direkt auf dem Backendserver oder bei lokaler Datenverarbeitung wird für den Transaktionsabschluss (Commit) eine Verbindung zum Server hergestellt.

Dazu gehören:

- **Kangaroo-Transaktionen**
- **Preserialization Transaction Management Technique**
- **Offen-geschachtelte Transaktionen**
- **Prewrite Transaktionen**

Kangaroo-Transaktionen [Dunham u. a. (1997)] können in verschiedenen Architekturen eingesetzt werden. Es ist das einzige Transaktionsmodell, dass die Bewegung eines mobilen Clients zwischen zwei Funkzellen während einer Transaktion berücksichtigt. „Bewegt“

sich der mobile Client von einer Funkzelle zur anderen und werden die Funkzellen von verschiedenen Basisstationen versorgt, so wird die Transaktion zwischen den Basisstationen aufgeteilt. So wird ein Teil der Transaktion auf der alten Basisstation ausgeführt und die neue Teiltransaktion auf der Basisstation in dessen Funkzelle sich der Client aktuell befindet.

Preserialization Transaction Management Technique (PSTMT) [Dirckze und Gruenwald (2000)] zeichnet sich durch ein effizientes Verhalten bei Abbruch der Verbindung für einen kurzen Zeitraum aus. Bei Verbindungsabbruch während einer Transaktion ist es das Ziel dieses Verfahrens, durch Subtransaktionen belegte Ressourcen auf dem lokalen Client schnell freizugeben. Dadurch soll verhindert werden, dass Ressourcen auf dem Client unnötig blockiert werden.

Offen-geschachtelte Transaktionen [Chrysanthis (1993), Serrano-Alvarado u. a. (2001)] unterstützen wie Kangaroo-Transaktionen nur den *Connected Mode* mobiler Clients. Bei diesem Transaktionsmodell sollen so viele Datenbankoperationen einer Transaktion wie möglich direkt auf den Datenbankserver verlagert werden. Es wird versucht Systemressourcen, die durch Transaktionen belegt sind, frühzeitig wieder freizugeben. Dies wird erreicht, indem Teiltransaktionen möglichst früh abgeschlossen werden.

Prewrite Transaktionen [Madria und Bhargava (2001), Serrano-Alvarado u. a. (2001)] teilen die Ausführung einer Transaktion zwischen mobilem Client und Datenbankserver auf. Die Transaktion wird auf den replizierten Daten und auf dem mobilen Client durchgeführt. Nach Beendigung der lokalen Transaktion sorgt ein Transaktionsmanager für die sofortige Weitergabe an den zentralen Datenbankserver.
Die Ausführung einer Prewrite-Transaktion geschieht in zwei Schritten. Zuerst wird auf den lokalen Daten die Transaktion ausgeführt und vorläufig beendet. Unmittelbar darauf werden auf dem zentralen Datenbankserver *Sperren* angefordert. Sind diese verfügbar, wird die Transaktion auf den Server übertragen und endgültig abgeschlossen. Der Nachteil bei diesem Verfahren ist, dass, wenn die Sperren auf dem Server in einem festgelegten Zeitrahmen nicht verfügbar sind, die clientseitige Transaktionsverarbeitung unterbrochen werden muss.

2.6.2 Transaktionsmodelle der Klasse 2

Die Transaktionen der Klasse 2 können auch bei unverbundenen Clients (Diconnected Mode) durchgeführt werden. In dieser Klasse werden die Transaktionen *lokal* auf den replizierten Daten ausgeführt, ohne dass nach Abschluss der Transaktion (Commit) ein Zugriff auf den zentralen Datenbankserver notwendig ist. Um die *globale* Datenkonsistenz wieder herzustellen, werden die lokal ausgeführten Transaktionen später mit den anderen Kopien der geänderten Daten abgeglichen. Bei der Synchronisation müssen auftretende Konflikte erkannt und, wenn möglich, aufgelöst werden. Gelingt die Konfliktauflösung im ersten Versuch

nicht, wird die Transaktion temporär zurückgesetzt. Kann auch bei einem erneuten Versuch der Konflikt nicht aufgelöst werden, wird die Transaktion zurückgesetzt. Dies kann dann zu einer kaskadierenden Rücksetzung auf dem Client führen.

Transaktionsmodelle dieser Klasse sind:

- **Provisorische Transaktionen**
- **Semantische Transaktionen**
- **Schwache/Strikte Transaktionen**
- **Isolation-Only-Transaktionen**
- **Promotion-Transaktionen**

Provisorische Transaktionen [Gray u. a. (1996), Höpfner, Serrano-Alvarado u. a. (2001)] erlauben mobilen Clients während der *Offline*-Phase Transaktionen auf den lokalen Daten durchzuführen. Provisorische Transaktionen sind dem Prewrite Transaktionsmodell sehr ähnlich, es muss aber nicht nach jeder Transaktion eine Verbindung zum zentralen Datenbankserver hergestellt werden. Auf den lokalen Daten können beliebig viele Transaktionen ausgeführt werden und zu einem späteren Zeitpunkt auf den zentralen Datenbankserver eingebracht werden.
Das provisorische Transaktionsmodell wird auch als *2-Phasen-Replikation* (Two-Tier-Replication) bezeichnet. In diesem Modell wird zwischen einem Datenbankserver im Festnetz und den mobilen Clients unterschieden. Ein Datenbankserver im Festnetz besitzt die Primärkopie der Replikate. Die Transaktionen unter den Rechnern im Festnetz erfolgt synchron (Primary-Kopie-Verfahren). Auf den mobilen Clients können während der *Offline-Phase* beliebige Transaktionen ausgeführt werden. Diese Transaktionen haben einen vorläufigen Charakter und sind deshalb provisorische Transaktionen (Tentative Transactions). Um die provisorisch lokal geänderten Daten auf dem primären Datenbankserver einzubringen, werden die Transaktionen auf diesem noch einmal ausgeführt. Hierbei kann es zu Konflikten (inkonsistentem Datenbestand) kommen – diese müssen erkannt und aufgelöst werden. Die provisorischen Transaktionen kommen, auch in Varianten, in vielen kommerziellen mobilen Datenbankprodukten zum Einsatz.

Semantische Transaktionen [Walborn und Chrysanthis (1995), Serrano-Alvarado u. a. (2001)] ermöglichen Transaktionen auf lokalen Daten wie bei der provisorischen Transaktion. Die Zielsetzung hier ist die Reduzierung der Kommunikation zwischen Client und zentralem Datenbankserver. Beim Arbeiten auf lokalen Daten der mobilen Clients im *Offline-Modus* kann es beim Einbringen der geänderten Daten auf den Server zu Konflikten kommen. Je mehr Transaktionen im *Offline-Modus* gemacht wurden, desto häufiger entstehen Konflikte beim Einbringen. Diese Konflikte sollen durch semantische Verfahren frühzeitig vermieden oder reduziert werden.

Schwache/Strikte Transaktionen [Pitoura und Bhargava (1999), Serrano-Alvarado u. a. (2001)] beruhen auf dem Konzept der verteilten Datenbanken ohne primären Datenbankserver. Das Transaktionsmodell bezieht sich auf mobile und stationäre Clients. Die Knoten einer verteilten Datenbankumgebung werden in Cluster eingeteilt. Ein Cluster kann aus stationären und verbundenen mobilen Clients bestehen. Alle Kopien eines Replikats innerhalb eines Clusters müssen strikt konsistent sein. Kopien eines Replikats, die sich auf verschiedenen Clustern befinden, dürfen Inkonsistenzen aufweisen. Wechselt ein mobiler Knoten vom verbundenen Modus in den unverbunden Modus, so wird er vom Cluster getrennt und bildet einen eigenen Cluster. Auf dem mobilen Client können so beliebige Transaktionen im Offline-Modus durchgeführt werden. Ist der Client wieder mit dem Netzwerk verbunden (online), so gehört er wieder seinem vorherigen Cluster an und gleicht seine Daten mit den anderen Knoten, die sich im Cluster befinden, ab.

Isolation-Only-Transaktionen [Lu und Satyanaranyanan (1994)] basieren auf einer zweistufigen Transaktionsverarbeitung. Bei den Isolation-Only-Transaktionen unterscheidet man zwischen *Firstclass*- und *Secondclass*-Transaktionen. *Firstclass*- greifen nicht auf lokal replizierte Daten zu, hier wird für jeden Zugriff auf Datenobjekte eine Verbindung zum Datenbankserver aufgebaut. Die *Secondclass-Transaktionen* werden im Disconnected Mode mobiler Clients auf lokal replizierten Daten ausgeführt. Die Secondclass-Transaktionen werden auf den mobilen Clients lokal abgeschlossen und gehen in einen schwebenden Zustand über. Bevor sie auf den zentralen Datenbankserver eingebracht werden, findet eine Validierung auf Konsistenzkriterien auf dem Datenbankserver statt. Sind die Konsistenzkriterien erfolgreich überprüft worden, werden die Transaktionen auf den Datenbankserver nachgeführt und beendet. Dadurch wird eine Reduzierung / Vermeidung von Synchronisationskonflikten erreicht. Schlägt die Validierung fehl, wird durch automatische oder manuelle Konfliktlösung versucht, den Konflikte aufzulösen. Kann der Konflikt nicht aufgelöst werden, muss die Transaktion zurückgesetzt werden.

Promotion-Transaktionen [Walborn und Chrysanthis (1997), Walborn und Chrysanthis (1999), Serrano-Alvarado u. a. (2001)] sind eine Variante der geschachtelten Transaktionen, die den Disconnected Mode mobiler Clients unterstützen. Die auf einen mobilen Client zu replizierenden Daten werden auf dem Datenbankserver zu so genannten *Compacts* zusammengefasst. Auf dem mobilen Client sorgt ein *Compact Agent* für die Ausführung der lokalen Transaktionen im unverbundenen Zustand. Auf dem Server werden Integritätsbedingungen definiert, die auf dem mobilen Client bei Transaktionen eingehalten werden müssen. Bei der Wiedereinführung der replizierten Daten auf den Server kümmert sich der serverseitige *Compact Manager* um die Konfliktauflösung.

2.7 Synchronisationskonflikte

Da verschiedene mobile Clients unabhängig voneinander auf mehreren Kopien eines Datums arbeiten können, kann es bei der Reintegration der Daten auf dem Server zu Konflikten kommen. D.h. bei der Reintegration werden die lokalen Änderungen an Replikaten beim Server eingebracht und der nachgelagerten Integritätsüberprüfung sowie einer Konfliktbehandlung (primärschlüsselorientierte Konflikterkennung) unterzogen. Dabei werden drei Arten von Konfliktsituationen unterschieden: Einfügekonflikte, Löschkonflikte und Änderungskonflikte.

2.7.1 Einfügekonflikte (INSERT-INSERT)

Einfügekonflikte entstehen, wenn beispielsweise zwei Clients den gleichen Datenbestand repliziert haben und einer der beiden Clients (A) bereits erfolgreich ein neues Tupel X eingefügt hat, während der andere Client (B) ein anderes Tupel X´ mit einem identischen Primärschlüssel hierfür reintegrieren will. Bei den Einfügekonflikten unterscheidet man drei verschiedene Fälle:

- Beide Tupel sind vollkommen identisch, sie besitzen den gleichen Primärschlüssel und den gleichen Inhalt.
- Beide Tupel besitzen den gleichen Primärschlüssel, aber der Inhalt ist unterschiedlich.
- Die Tupel haben verschiedene Primärschlüssel, aber der Inhalt ist gleich.

2.7.2 Löschkonflikte

Löschkonflikte entstehen immer dann, wenn ein Tupel gelöscht wurde und ein anderer Client auf das bereits gelöschte Tupel zugreifen will. Man unterscheidet zwischen:

- Löschen-Löschen-Konflikt (DELETE-DELETE)
  Ein bereits gelöschtes Tupel wird von einem anderen Client ein zweites Mal gelöscht.
- Löschen-Ändern-Konflikt (DELETE-UPDATE)
  Auf einem bereits gelöschtem Tupel will ein anderer Client eine Änderung einbringen.

2.7.3 Änderungskonflikte

Änderungskonflikte werden immer dann ausgelöst, wenn eine Änderung oder Löschung von einem zweiten Client auf einem bereits geänderten Tupel vorgenommen wird. Hierbei wird ebenfalls zwischen zwei Arten unterschieden:

- Ändern-Löschen-Konflikt (UPDATE-DELETE)
  Ein zuvor geändertes Tupel wird von einem anderen Client gelöscht, der den Löschvorgang auf Basis von alten Daten ausgeführt hat.
- Ändern-Ändern-Konflikt (UPDATE-UPDATE)
  Zwei Clients ändern unabhängig von einander den Inhalt eines Tupel.

3 Anforderungsanalyse

3.1 Ferienclub Szenario

Das hier vorgestellte Szenario eines Ferienclubs beruht auf dem UbiComp-Projekt der Hochschule für Angewandte Wissenschaften Hamburg [UbiComp@HAW-Hamburg (2002)] und wurde unter anderem in den Diplomarbeiten von [Bresch (2004)] und [Lüpke (2004)] behandelt.
Der Ferienclub bietet seinen Gästen ein vielfältiges Angebot von Veranstaltungen zur Freizeitgestaltung an. Die Angebote sind abhängig von regionalen Faktoren und Ausrichtung des Anbieters. Ein Anbieter, der sich auf Familien konzentriert, hat andere Angebote als einer, der sich auf Single-Urlauber spezialisiert hat. Beispiele für Freizeitaktivitäten könnten aus dem Bereich Sport sein und aus einem Kursangebot oder einem Turnier bestehen. Für den Bereich Kultur könnten es Besichtigungstouren zu einer Sehenswürdigkeit sein.
Der Ferienclub selbst kann ein Hotel oder eine "Clubanlageßein, die Freizeitaktivitäten anbieten. Denkbar wären aber auch kleinere Hotels und Apartments, die sich regional zusammenschließen und Aktivitäten der örtlichen Tourismus-Information anbieten. So können sie ihren Gästen einen Mehrwert bieten, den sonst nur große Hotelanlagen haben.

Die Informationen der angebotenen Aktivitäten werden in einem allgemeinen, jedem Gast zugänglichen Veranstaltungskalender veröffentlicht. Jeder Gast hat einen persönlichen Kalender, in dem er seine Termine eintragen kann. Damit der Gast die Informationen jederzeit und ortsungebunden abrufen kann, bekommt er einen PDA zur Verfügung gestellt. Auf dem PDA befindet sich der jeweilige persönliche Terminkalender und Teile oder der gesamte öffentliche Veranstaltungskalender. Auf dem PDA kann der Gast seine privaten Termine eingeben, sich die angebotenen Veranstaltungen zu den Freizeitaktivitäten ansehen und sich zu diesen anmelden. Um die Daten auf seinem PDA zu aktualisieren oder mit dem Hauptsystem des Ferienclubs abzugleichen, muss der Gast den PDA mit dem Hauptsystem verbinden.

3.2 Komponenten des Szenarios

3.2.1 Freizeitangebote

Das Freizeitangebot kann die unterschiedlichsten Ausprägungen haben. Es wird sich je nach Urlaubsort und Anbieter unterscheiden. Um die Angebote zu strukturieren werden sie in Kategorien eingeteilt, wie z. B. Sport, Kultur oder Kinderveranstaltungen. Die Angebote können vom Veranstalter selbst, von externen Anbietern oder von Gästen sein. Zu den festen Bestandteilen eines Angebots gehört eine Beschreibung, evtl. mit Bild. Weitere Informationen könnten die Mindest- oder maximale Teilnehmerzahl und die Dauer sein, aber auch bestimmte Voraussetzungen wie körperliche Fitness oder ein Mindestalter der Teilnehmer. Bei kostenpflichtigen Veranstaltungen ist der Preis mit angegeben.

Beispiel aus der Kategorie Sport:

Bogenschießen

Pfeil und Bogen waren für den Menschen Jahrtausende lang eine unverzichtbare Jagd- und Kriegswaffe. Heute wird Bogenschießen als eine Sportart geschätzt, die Körper und Geist gleichermaßen beansprucht.
Erproben Sie unter der fachkundigen Anleitung unserer Guides Ihre Konzentrationsfähigkeit und Ihre Körperbeherrschung. Mit ein wenig Übung, Ruhe und Gelassenheit wird auch Ihr Pfeil ins Ziel treffen.
Ob sich im Kreise Ihrer Familie, Freunde oder Kollegen ein Meisterschütze wie Robin Hood befindet, wird sich herausstellen.

Abbildung 3.1: Bogenschiessen

Leistungen:	Ausrüstung, Einweisung und Betreuung durch ausgebildete Guides
Mindestteilnehmerzahl:	6 Vollzahler
Dauer:	2-3 Std.
Preis:	15.- € Person im Schnupperkurs,18.- € Person am Wochenende. Für Schulen, Jugendgruppen und Vereine Sonderpreise auf Anfrage.
Termin:	Im Sommer immer freitags als Schnupperkurs, Sondertermine für Ihre Gruppe auf Anfrage
Mindestalter:	12 Jahre

Das Beispiel mit Bild ist aus [Aktiv-Reisen (2005)] entnommen.

3.2.2 Veranstaltungskalender

Der Veranstaltungskalender enthält alle angebotenen Veranstaltungen zu den Freizeitangeboten. Es werden die Veranstaltungen sortiert nach Datum und Uhrzeit angezeigt. Die Einträge beinhalten alle benötigten Informationen zu den Veranstaltungen wie zum Beispiel den Ort an dem die Veranstaltung stattfindet, die Dauer, wieviel Plätze noch frei sind und einen kurzen Info-Text. Ausserdem kann über einen Verweis das Freizeitangebot für ausführliche Informationen angewählt werden. Durch die Auswahl der Kategorien nach Interessen in der Profileingabe und die Aufenthaltsdauer der Gäste findet eine Selektion der Angebote statt.
Wenn ein Gast mehr Informationen zu einem Angebot haben möchte, wählt er die Veranstaltung an und erhält eine Detailbeschreibung, wie unter 3.2.1 beschrieben. Um an Veranstaltungen teilzunehmen, kann er diese buchen. Einträge in den öffentlichen Kalender macht ausschließlich der Veranstalter.

3.2.3 Benutzer-Profil

Für jeden Gast wird ein Profil erstellt. Das Profil setzt sich aus den persönlichen Stammdaten und den individuellen Interessen des Gastes zusammen. In den Stammdaten sind enthalten die Namen des einzelnen Gastes oder der Familie, bei einzelnen Gästen das Geschlecht, der Familienstand, Ankunfts- und Abreisedatum sowie die Zimmernummer. Diese Daten werden zu Beginn des Urlaubs vom Veranstalter eingegeben.
Zusätzlich kann der Gast zu seinem Profil Kategorien aus dem Freizeitangebot nach persönlichem Interesse auswählen. Dadurch erhält er nur Informationen über Freizeitaktivitäten, die seinem Profil entsprechen. Dies könnte auch teilweise automatisiert erfolgen, indem z. B. für einen ledigen männlichen Gast spezielle Frauenkurse und Kinderangebote abgewählt werden. Wenn die Stammdaten geändert werden müssen, wie beispielsweise bei Ankunft eines

weiteren Familienmitglieds oder bei einem Zimmerwechsel, muss eine Person des Ferienclubmanagements diese Änderung vornehmen. Die persönlichen Interessen können von den Gästen selbst geändert werden. So können die Gäste durch Hinzufügen von Kategorien die Auswahl von Veranstaltungen erweitern oder durch Abwählen von Kategorien die Auswahl einschränken.

3.2.4 Persönlicher Kalender

Der persönliche Kalender wird für die Dauer des Aufenthalts des Gastes angelegt. Er enthält die Termine des Gastes und wird auch von ihm gepflegt. Wenn der Gast sich an einer Veranstaltung anmeldet, wird dieser Termin in seinem persönlichen Kalender angezeigt. Außerdem kann der Gast private Termine eintragen. Der Gast hat vollen Zugriff auf seinen persönlichen Kalender, er kann private Termine löschen, neu anlegen und ändern. Zu den Veranstaltungen kann er sich an- und abmelden, so dass sie in seinen Terminkalender ein- und ausgetragen werden. Außer ihm hat noch der Veranstalter einen eingeschränkten Zugriff auf seinen persönlichen Kalender. Der Veranstalter kann nur die Termine im Klartext sehen, zu denen der Gast sich aus dem öffentlichen Kalender angemeldet hat. Bei den privaten Terminen sieht er nur das Datum und die Uhrzeit. Für wichtige Informationen, wie z. B. die Absage einer Veranstaltung kann der Veranstalter einen Termin in den Persönlichen Kalender eintragen.

3.3 Typischer Urlaubsverlauf

3.3.1 Ankunft

Bei der Ankunft der Gäste im Ferienclub werden die Stammdaten, soweit noch nicht vorhanden, an der Rezeption in das Ferienclub-Informationssystem eingegeben. Anhand dieser Daten wird der persönliche Terminkalender des Gastes angelegt. Daraufhin werden die Daten vom persönlichen Kalender, dem Veranstaltungskalender und der Veranstaltungen auf den PDA geladen. Der Gast bekommt einen Benutzernamen und Passwort um sich im System anzumelden. Ausserdem bekommt er seinen PDA ausgehändigt und eine Einführung in die Bedienung des Gerätes und der Software.
Zu Beginn wählt der Gast aus dem Freizeitangebot die Kategorien aus, für die er sich interessiert. Dadurch erhält er ein seinen Interessen angepasstes Freizeitangebot.

3.3.2 Aufenthalt

Nachdem der Gast die aktuellen Daten auf seinem PDA geladen hat, kann er mit diesen Daten offline arbeiten. Er sollte sich aber jeden Tag einmal mit dem Ferienclub-Informationssystem verbinden, um die Daten abzugleichen und aktuelle Informationen nachzuladen. Beim täglichen Betrieb des PDAs bieten sich den Gästen folgende Anwendungsmöglichkeiten:

Informationen zu den Veranstaltungen abrufen

Der Gast ruft den öffentlichen Kalender auf und erhält eine Liste der Veranstaltungen die seinen Interessen entsprechen und während seines Urlaubsaufenthalts stattfinden. So erhält er einen Überblick über die Angebote. Über die Detailansicht bekommt er genauere Informationen zu der gewählten Veranstaltung.

Veranstaltung buchen / stornieren

Möchte sich ein Gast zu einer Veranstaltung anmelden, wählt er diese aus dem öffentlichen Kalender aus. Wenn noch Plätze frei sind und er keinen Termin in seinem persönlichen Kalender hat, der sich mit dem der Veranstaltung überschneidet, kann er sich für die Veranstaltung anmelden. Daraufhin wird der Termin in seinen persönlichen Kalender eingetragen und im öffentlichen Veranstaltungskalender die Zahl der Plätze um einen reduziert. Um eine Freizeitaktivität zu stornieren, muss der Gast in seinen persönlichen Terminkalender gehen und den Termin anwählen und die Funktion löschen angeben. Daraufhin wird der Termin aus dem persönlichen Kalender gelöscht und die Anzahl der freien Plätze in dem Freizeitangebot wieder erhöht. Für einen Wechsel zu einer äquivalenten Veranstaltung an einem anderen Datum muss man erst die erste Anmeldung löschen und sich dann zur anderen Veranstaltung anmelden.

Private Termine verwalten

In seinen persönlichen Kalender kann ein Gast jederzeit private Termine eintragen, ändern oder löschen. Um einen neuen Termin einzugeben, gibt er zuerst die Anfangs- und Endzeit des Termins und dann einen Text ein. Beim Ändern und Löschen eines Termins wählt er den betreffenden Termin aus und ändert oder löscht ihn.

3.3.3 Abreise

Bei der Abreise werden das Profil und der persönliche Kalender gelöscht und der Gast gibt seinen PDA zurück. Der Gast muss für die Teilnahme an kostenpflichtigen Freizeitveranstaltungen bezahlen.

3.4 Akteure im Ferienclub

In dem beschriebenen Ferienclub-Szenario ergeben sich drei Hauptakteure: der Gast, die Rezeption und der Redakteur.

Der Gast, in diesem Szenario ein Urlauber, hat einen persönlichen Terminkalender in dem er seine Termine verwaltet. Er kann auf den Veranstaltungskalender zugreifen und sich die Freizeitangebot ansehen und buchen.

Die Rezeption ist für die Eingabe der Stammdaten des Gastes zuständig. Dies geschieht bei der Ankunft des Gastes. Sind die Stammdaten eingegeben, stellt die Rezeption dem Gast seinen persönlichen Kalender und den aktuellen Veranstaltungskalender mit den Freizeitangeboten zur Verfügung.

Der Redakteur ist für die Verwaltung des Veranstaltungskalenders und der Freizeitangebote zuständig.

3.5 Use Cases

AF-Nr. 1	**Gastprofil eingeben**
Akteur	Rezeption
Vorbedingung	Ein PDA mit installierter Anwendung ist vorhanden
Nachbedingung	Der Gast ist im System registriert und ein persönlicher Kalender wurde erstellt.
Ablauf	Für das Gastprofil werden die Stammdaten der Gäste eingegeben. Ein Benutzername und ein Passwort werden angelegt. Anhand der persönlichen Daten findet eine Selektion der Kategorien statt und der persönliche Kalender wird für die Dauer des Aufenthalts erstellt. Die Daten werden auf den PDA repliziert.

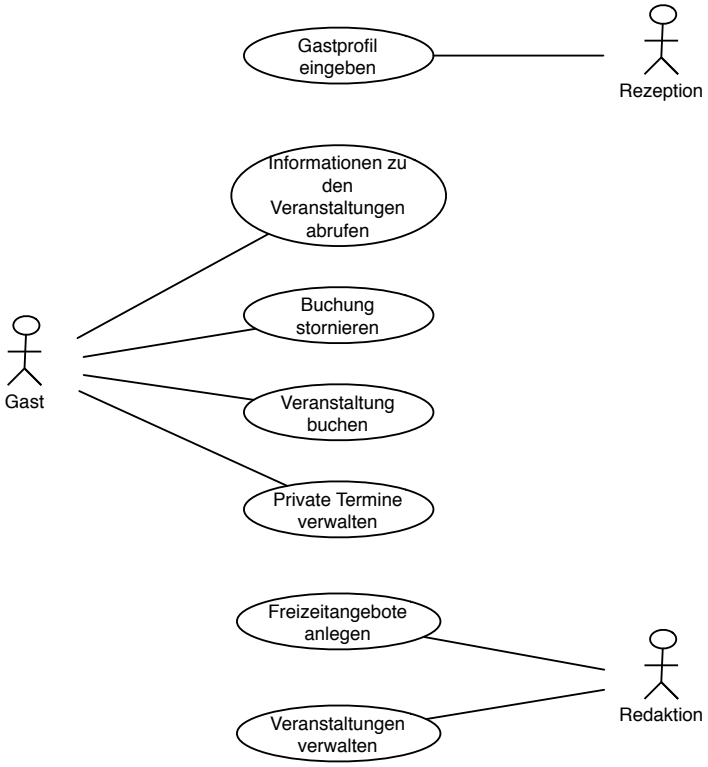

Abbildung 3.2: Use Cases

AF-Nr. 2	**Informationen zu den Veranstaltungen abrufen**
Akteur	Gast
Vorbedingung	Eine erstmalige Replikation der Daten muss stattgefunden haben.
Ablauf	Der Gast wählt den Veranstaltungskalender aus und bekommt eine Auswahl der Kategorien zu sehen. Er wählt eine oder mehrere Kategorien aus und bekommt daraufhin eine Liste der Veranstaltungen, nach Datum und Uhrzeit sortiert, angezeigt. Um nähere Informationen zu bekommen, wählt er das zu der Veranstaltung gehörende Freizeitangebot aus.

AF-Nr. 3	**Veranstaltung buchen**
Akteur	Gast
Vorbedingung Der Gast befindet sich in der Veranstaltungsübersicht.	Das Freizeitangebot muss noch mindestens einen Platz frei haben.
Nachbedingung	Die Anzahl der freien Plätze in der Veranstaltung wurde reduziert. Es darf sich kein Termin einer gebuchten Veranstaltung oder ein privater Termin mit dem Termin der Veranstaltung überschneiden.
Ablauf	Der Gast wählt eine Veranstaltung aus und betätigt den „Buchen-Button“

AF-Nr. 4	**Buchung stornieren**
Akteur	Gast
Vorbedingung	Eine Veranstaltung ist gebucht.
Nachbedingung	Die Anzahl der freien Plätze der Veranstaltung wurde erhöht.
Ablauf	Der Gast ruft seine Terminübersicht auf. Der Termin der gebuchten Veranstaltung wird ausgewählt und gelöscht.

AF-Nr. 5	**Private Termine verwalten**
Akteur	Gast
Nachbedingung	Es dürfen sich keine Termin einer gebuchten Veranstaltung oder eines privaten Termin überschneiden.
Ablauf	Nach der Auswahl der Terminübersicht werden die privaten Termine und die Termine der gebuchten Veranstaltungen angezeigt. Um einen privaten Termin zu bearbeiten, wird der betreffende Termin ausgewählt und kann bearbeitet oder gelöscht werden. Um einen neuen Eintrag zu machen wird das Datum, Anfangs- und Endzeit sowie der Text eingegeben.

AF-Nr. 6	**Freizeitangebote anlegen**
Akteur	Redakteur
Ablauf	Der Redakteur legt ein neues Freizeitangebot mit den erforderlichen Daten an. Er ordnet das Freizeitangebot einer Kategorie zu.
Varianten	Das neu angelegte Freizeitangebot kann keiner bestehenden Kategorie zugeordnet werden. Der Redakteur legt eine neue Kategorie an und ordnet dem Freizeitangebot die neu erstellte Kategorie zu.

AF-Nr. 7	**Veranstaltungen verwalten**
Akteur	Redakteur
Vorbedingung	Das Freizeitangebot, für das die Veranstaltung angelegt werden soll, muss vorhanden sein.
Nachbedingung	Falls eine Veranstaltung gelöscht wurde, kann diese Veranstaltung nicht mehr gebucht werden. Die Gäste, die von der gelöschten Veranstaltung müssen benachrichtigt werden.
Ablauf neu anlegen	Der Redakteur gibt eine neue Veranstaltung für ein bestehendes Freizeitangebot ein. Die Veranstaltung ist über das Freizeitangebot einer Kategorie zugeordnet.
Ablauf löschen	Die Veranstaltung und die dazughörigen Buchungen werden gelöscht.
Anmerkung	Die abgesagte Veranstaltung wird im Terminkalender der Gäste, die diese Veranstaltung gebucht haben, besonders hervorgehoben.

AF-Nr. 8	**Privaten Termin bei Gästen eintragen**
Akteur	Redakteur
Vorbedingung	Es dürfen sich keine Termine mit dem neuen Eintrag überschneiden
Ablauf	Der Redakteur wählt einen Gast und öffnet dessen Terminkalender. Um einen neuen Eintrag zu machen gibt er das Datum, die Anfangs- und Endzeit sowie den Text ein.

4 Design und Realisierung

4.1 Client-Server Architekturen

In dem beschriebenen Anwendungssystem für den Ferienclub arbeiten die Benutzer (Gäste) auf einem Datenbestand, den sie sich mit anderen Benutzern teilen müssen. Das Anwendungssystem hat einen gemeinsamen Datenbestand, der von mehreren Anwendungen auf den mobilen Clients gemeinsam genutzt wird. Solch ein Anwendungssystem, dass auf mindestens einem zentralen Datenbankserver zugreift, wird als Client/Server-System realisiert. Ein Client/Server-System lässt sich nach [(CTR)] in drei logische Einheiten aufteilen: Die Präsentationsschicht, die Anwendungslogik und die Datenhaltung. Diese logischen Einheiten können unterschiedlich auf die Client-Rechner oder auf dem Server verteilt sein. Dies bietet für die Anwendungsentwicklung unterschiedliche Schnittmöglichkeiten für die Aufteilung der logischen Einheiten. Ein Schnitt kann auch durch eine logische Einheit erfolgen. Der Schnitt durch die logischen Einheiten wird in Abbildung 4.1 dargestellt.

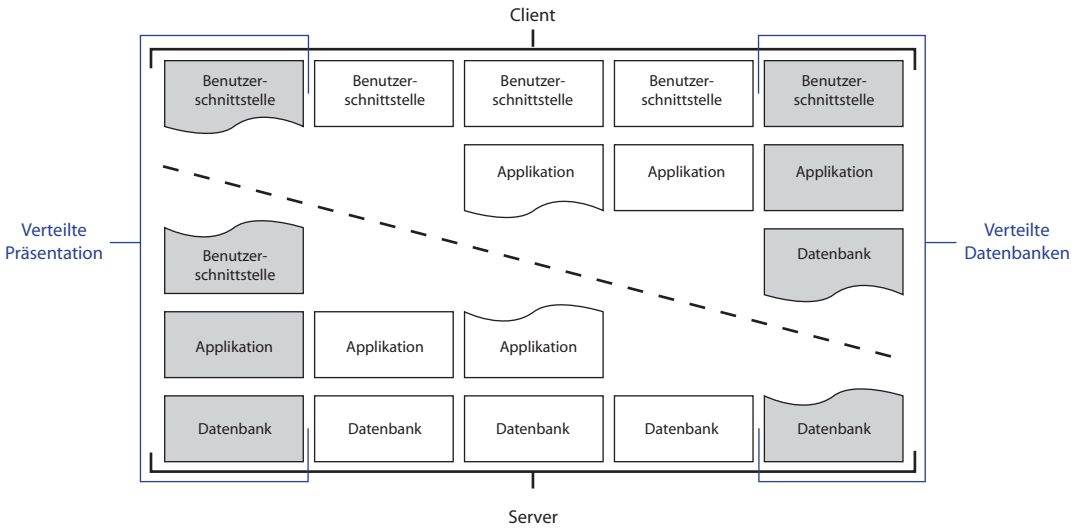

Abbildung 4.1: Ausprägung von Client/Server-Systemen. Quelle [(CTR)]

Jeder Schnitt ist unterschiedlich motiviert und hat verschiedene Vor- und Nachteile. Für das Ferienclub-Szenario sollen die beiden Schnitte, die in der Abbildung markiert sind, näher

betrachtet werden. Die *verteilte Präsentation* wurde von Marco Bresch in [Bresch (2004)] verwendet um das Ferienclub-Szenario zu realisieren. Bei diesem Schnitt befindet sich nur die Präsentation auf den mobilen Clients. Auf den mobilen Clients ist kein anwendungsspezifischer Code installiert, er ist nur für die Darstellung zuständig. Hier spricht man auch von einem Thin-Client.
Bei dieser Ausprägung der Client/Server-Architektur entfällt eine Installation und die damit verbundene Verteilung der Anwendung. Neue Versionen der Anwendung müssen nur auf dem Server installiert werden. Dadurch ergibt sich ein geringerer Administrationsaufwand. Nachteil diese Architektur ist die Abhängigkeit von der Netzqualität und der Netzverfügbarkeit.
Alle Schnitte, bis auf die der verteilten Datenbanken, sind auf eine Netzverfügbarkeit angewiesen, ohne sie kann auf den Clients nicht gearbeitet werden. Durch die Replikation der Daten ist es möglich, autonom auf den mobilen Clients zu arbeiten. Daher kommt für eine Umsetzung des Ferienclub-Szenarios, in dem keine ständige Netzwerkverfügbarkeit vorherrscht, nur eine Umsetzung der Client/Server-Architektur mit verteilten Datenbanken in Frage. Diese Architektur stellt besondere Ansprüche an den Client in Punkto Sicherheit. Durch die Datenhaltung auf einem Client, ausserhalb des geschützten Intranets, ist dieser Gefahren ausgesetzt, welche nicht mehr durch das interne Sicherheitskonzept abgefangen werden. Von daher sind besondere Maßnahmen bezüglich des Zugriffs auf die lokal gespeicherten Daten zu treffen. Dies kann nur über ein umfassendes Sicherheitskonzept erreicht werden, welches mindestens folgende Aspekte behandelt:

- Sichere Anmeldung an der Domäne
- Firewall
- Viren-Scanner
- Security Patches
- Verschlüsselung des Datenträgers (mobile Datenbank)

4.2 Entwurfsentscheidung

Wie aus der Analyse des Ferienclub-Szenarios und aus der Auswahl der Client/Server-Architektur hervorgeht, besteht das System aus einem zentralen Datenbankserver und mobilen Clients. Auf den mobilen Clients soll ein Arbeiten ohne bestehende Netzverbindung zum zentralen Datenbankserver möglich sein. Dies soll mit verteilten Datenbanken geschehen. Die Verteilung der Daten und der spätere Abgleich der geänderten Daten soll mittels Replikation und Synchronisation stattfinden (siehe Abbildung 4.2).

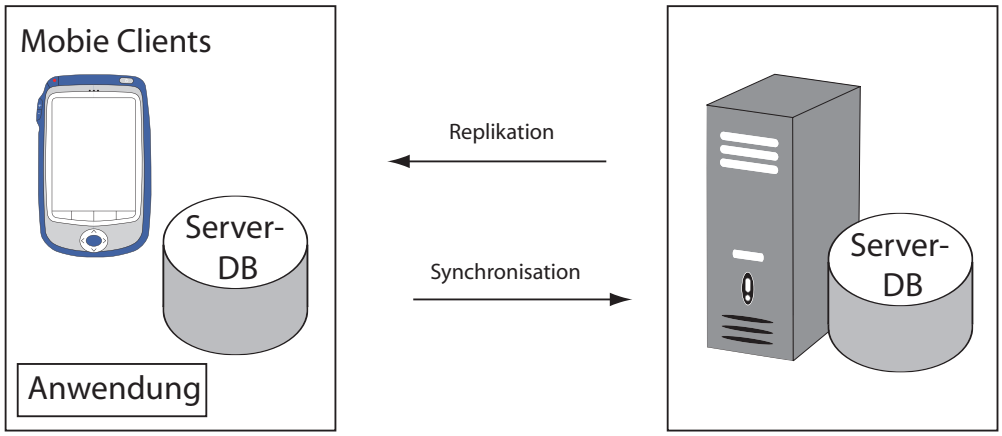

Abbildung 4.2: Grobarchitektur

Auf dem PDA befindet sich eine Anwendung mit grafischer Oberfläche und eine mobile Datenbank, die eine Kopie von einem Teil der zentralen Datenbank enthält. Diese Kopie soll nur die Daten enthalten, die für den einzelnen Gast relevant sind. Es ist erforderlich die zentrale Datenbank in horizontale Fragmente zu zerlegen. Auf den lokalen Daten können die Gäste arbeiten ohne eine Netzwerkverbindung zu haben. Sie können neue Daten eingegeben und bestehende Daten ändern und löschen. Ausserdem soll es dem Redakteur möglich sein, auf den zentralen Daten Änderungen auszuführen. Um die Daten wieder in einen konsistenten Zustand zu bringen, findet ein Abgleich in zwei Phasen statt. Die Reintegration überträgt die auf dem mobilen Client geänderten Daten auf den Datenbankserver. Bei der anschließenden Rückübertragung werden zwischenzeitlich getätigte Änderungen auf dem Server an den Client übertragen. Da sowohl Daten auf dem Client als auch auf dem Server geändert werden dürfen, spricht man von einer bidirektionale Synchronisation. Bei der Synchronisation der Daten kann es zu Konflikten kommen, die aufgelöst werden müssen. Die Auflösung der Konflikte sollte nach Möglichkeit automatisierbar sein.
Für die Realisierung ist eine mobile Datenbank nötig, die mit einem zentralen Datenbanksystem kommuniziert. Die Replikation/Synchronisation muss asynchron erfolgen, da nur so ein Arbeiten im Offline-Modus auf den mobilen Clients möglich ist, ohne nach jeder Transaktion eine Verbindung zum Replikationsserver aufbauen zu müssen. Der zentrale Datenbankserver enthält die Primärkopie der Daten, die auf die mobilen Clients repliziert wird.
Die Umsetzung des Ferienclub-Szenarios soll mit dem *SQL Servers 2000 CE* auf den mobilen Geräten und dem SQL Server 2000 als zentraler Datenbankserver stattfinden. Die Kombination der beiden Komponenten erfüllt alle aufgeführten Anforderungen. Im folgenden Absatz wird der SQL Server CE näher beschrieben.

4.3 Microsoft SQL Server 2000 CE Edition

Die *Microsoft SQL Server 2000 CE Edition (SQL Server CE)* ist eine *kompakte* Version des Microsoft SQL Server 2000. Im Folgenden wird die Version *Microsoft SQL Server 2000 CE Edition 2.0* beschrieben [SQLServerCE (2002)]. Der SQL Server CE stellt Funktionen für die Datenverwaltung auf mobilen Geräten zur Verfügung. Als Betriebssystem auf den mobilen Geräten wird *Windows CE* oder einer seiner Nachfolger vorausgesetzt (Pocket PC, Windows Mobile 2003). Die Installation des SQL Servers CE nimmt zwischen 1 und 3 MB Speicherplatz auf dem mobilen Gerät in Anspruch. Der benötigte Speicherplatz ist abhängig von den installierten Komponenten und der Version des Betriebssystems. Als *Backend-Server* muss ein Microsoft SQL Server eingesetzt werden.

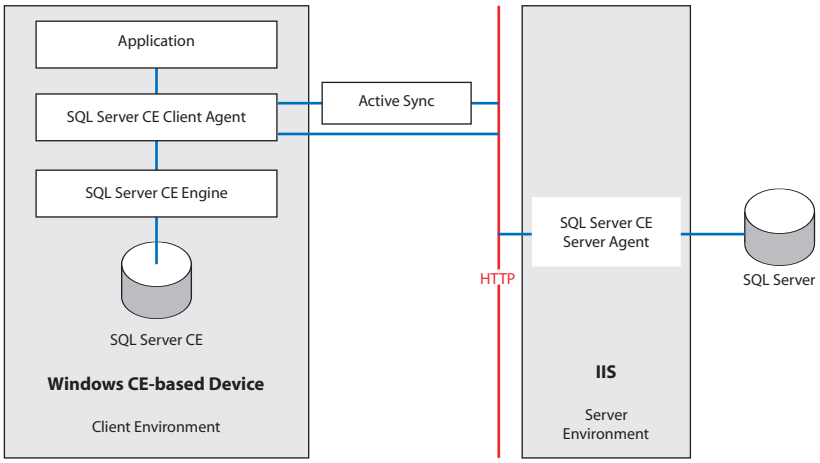

Abbildung 4.3: SQL Server CE Übersicht. Quelle [SQLServerCE (2002)]

4.3.1 Komponenten

Um eine mobile Anwendung mit replizierten Daten auf dem SQL Server Ce einzusetzen, werden folgende Komponenten benötigt:

- **SQL Server** Der SQL Server ist der zentrale Datenbankserver im Backend, dessen Daten auf die mobilen Clients repliziert werden.
- **SQL Server CE** ist das mobile Datenbanksystem, das auf den mobilen Clients installiert ist.
- **ActiveSync** ActiveSync wird in der Entwicklung von Anwendungen auf den mobilen Clients eingesetzt. Es wird benutzt, um die Anwendungen zu installieren und zu debuggen. Außerdem kann ActiveSync von mobilen Geräten für die Kommunikation mit

dem Backendserver genutzt werden, die über keine eigene Netzwerkkarte verfügen. Sie verbinden sich über ActiveSync mit einen Computer, der sich im festen Netzwerk befindet, und stellen so eine Verbindung zum Backendserver für die Synchronisation her.

- **Internet Information Services (IIS)** Der *Internet Information Services* ist der Web-Server von Microsoft. Über den IIS läuft die Kommunikation und der Datenaustausch zwischen einer SQL Server CE Datenbank auf einem mobilen Gerät und der zentralen SQL Server Datenbank.
- **Entwicklungsumgebung** Für die Entwicklungsumgebung steht das Microsoft Visual .NET Framework, das .NET Compact Framework und ein PDA-Emulator zur Verfügung. Zur Anwendungsentwicklung kann man zwischen verschiedenen Programmiersprachen wählen.

4.3.2 SQL Server CE Architektur

Die clientseitige Umgebung besteht aus den folgenden Komponenten:

- **Database-Engine** Die SQL Server CE *Database-Engine* ist für die Verwaltung und Speicherung der Daten auf dem mobilen Client zuständig. Für replizierte Daten protokolliert die Database Engine jede Datenoperation und speichert sie ab.
- **Client Agent** Der *Client Agent* ist auf dem mobilen Client für das Herstellen von Verbindungen zur zentralen Datenbank und das Versenden von Abfragen verantwortlich. Er besteht aus drei Objekten: dem *Replikations-Objekt*, dem *Remote Data Access-Objekt* und dem *Engine Objekt*. Diese drei Objekte werden benutzt, um aus Anwendungen heraus eine Verbindung zum zentralen Server zu steuern.
- **Server Agent** Der *Server Agent* ist die serverseitige Komponente, die für den Empfang von Anfragen mobiler Clients zuständig ist. Er ist ein Modul des IIS. Anfragen von mobilen Datenbankanwendungen werden vom Client Agent in einen HTTP-Request gekapselt und an den Server Agent übermittelt. Der Server Agent empfängt diese Nachricht und leitet sie an den SQL-Server weiter. Anschließend wartet er auf eine Antwort des SQL-Servers, wenn er diese erhält leitet er sie an den mobilen Client weiter.

4.3.3 Sicherheit

Der SQL-Server bietet verschiedene Stufen der Sicherheit einmal für den IIS und für den SQL Server CE an. Für den Zugriff auf den IIS werden Anonymous Access, Basic Authenticati-

on und Integrated Windows Authentication unterstützt. Zusätzlich unterstützt der IIS Secure Sockets Layer (SSL) zur Verschlüsselung der zu übertragenden Daten an. Der SQL Server CE unterstützt Integrated Windows Authentication und SQL Server Authentication. Als Sicherheitsmechanismen ist hierbei eine Kombination aus SSL geschützter Datenübertragung und integrierter Windows-Anmeldung auf Basis des Kerboros-Protokolls (das seit der Pocket PC 2003 unterstützt wird) vorzuziehen.

4.3.4 Replikation

Der SQL Server CE bietet zwei Methoden für den Datenaustausch zwischen dem SQL Server im Backend und dem SQL Server CE auf dem Client: die Merge-Replikation und den Remote Data Access (RDA).

Remote Data Access (RDA)

RDA stellt ein Framework für die Kommunikation mittels HTTP und HTTPS bereit, das die Verbindung sowohl auf der Client- als auch der Serverseite zwischen den Komponenten des SQL Servers CE und SQL-Server 6.5/7.0/2000 ermöglicht. RDA ist die einzige Möglichkeit um zwischen dem SQL-Server CE und den älteren Versionen des SQL Servers 6.5/7.0 zu kommunizieren (die Merge-Replikation funktioniert nur mit dem SQL-Server 2000). Der Internet Informationsserver (IIS) fungiert als Vermittler bei der Übertragung der Daten zwischen den mobilen Geräten und dem Server. D.h. aufgrund der Verwendung des IIS ist es RDA möglich, die Fähigkeiten des Proxy-Servers (Vermittlers) für verschiedene Sicherheitsmechanismen und die Kommunikation durch Firewalls hindurch auszunutzen. Die drei Methoden, die in der Klasse SqlCeRemoteDataAccess enthalten sind und die die ganze Arbeit rund um die RDA Anwendung übernehmen sind Pull, Push und SubmitSQL. Die Pull- Methode wurde entwickelt, um Tabellen, Indizes und Daten vom SQL Server zu holen, damit diese anschließend in der lokalen SQL Server CE Datenbank gespeichert werden können. Um alle lokal veränderten Daten aufzuspüren und diese anschließend an den Server-Agenten, der die passenden Einfüge-, Aktualisierungs- und Löschoperationen auf dem SQL-Server vornimmt, zu übertragen, wurde die Push-Methode entworfen. Durch die Methode SubmitSQL wird eine direkte Ausführung von SQL-Statements auf dem SQL Server ermöglicht, wobei nur Einfüge- Aktualisierungs- und Löschoperationen sowie der Aufruf von „stored-procedures“ erlaubt sind.

Merge-Replikation

Für die Replikation nutzt der SQL Server CE das Publish/Subscribe Modell des SQL Servers 2000. Von den drei Varianten auf dem SQL-Server (Snapshot-, Transaktions- und Merge-Replikation) wird auf dem SQL Server CE nur die Merge-Replikation unterstützt. Um die Daten auf den mobilen Client zu replizieren, werden auf dem zentralen SQL Server (Publisher) Publikationen erstellt. Diese können aus einer beliebigen Anzahl von Artikeln (Tabellen) bestehen. Auf den Tabellen können horizontale und vertikale Filter angewendet werden. So ist es möglich, die zu replizierenden Tabellen zeilen- oder spaltenweise einzuschränken. Die mobilen Clients können diese Publikationen abonnieren (subscribe).
Auf den replizierten Daten kann lokal gearbeitet werden. Datensatzänderungen können auf den mobilen Clients sowie auf dem Server stattfinden. Die Änderungen an den Datensätzen werden über Trigger protokolliert. Jede replizierte Tabelle verfügt über drei Systemtrigger für die Änderungsoperationen *INSERT*, *UPDATE* und *DELETE*. Für die Synchronisation werden die geänderten Daten an den SQL Server weitergeleitet. Die Kommunikation übernehmen die oben beschriebenen SQL CE Client/Server Agents. Für die Ausführung der Synchronisation ist der SQL Server Reconciler zuständig. Dieser führt die notwendigen Änderungen an der zentralen Datenbank durch. Bei auftretenden Konflikten wird ein Konfliktlöser vom SQL Server aufgerufen. Im selben Vorgang werden bei der Synchronisation die Änderungen, die auf dem Publisher gemacht wurden, an den Subscriber übertragen. Zur Lösung von Konflikten stellt der SQL Server eine Reihe von Standardkonfliktlösern bereit. Sollten diese nicht ausreichen, können benutzerdefinierte Konfliktlöser erstellt werden.

Durch die Verwendung eigener Konfliktlöser kann dieser Mechanismus ideal auf die besonderen Anforderungen des Gesamtsystems angepasst werden.

Für das Ferienclub-Szenario wird die Merge-Replikation eingesetzt, da sie für diese Aufgabe die besseren Optionen bietet. Dies ist zum einen die Möglichkeit Publikationen zu erstellen. Durch die Auswahl von Tabellen und den Einsatz von Filtern ist es möglich, dass die einzelnen Clients nur die Daten bekommen, die sie benötigen. Dadurch wird die Datenmenge auf den Clients reduziert und es können schon im Vorfeld Konflikte vermieden werden. Außerdem stellt die Merge-Replikation Standardoptionen und benutzerdefinierte Optionen zur Konfliktauflösung bereit. Weil im Ferienclub-Szenario mehrere Benutzer auf denselben Tabellen arbeiten, ist von Konflikten auszugehen. Daher eignet sich die Merge-Replikation für dieses Szenario, um konsistente Datenbestände zu garantieren.

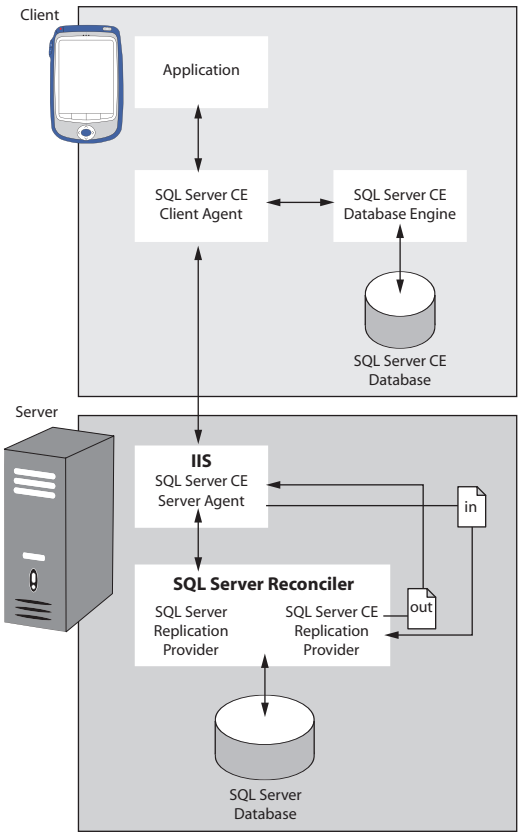

Abbildung 4.4: SQL Server CE Merge-Replikation. Quelle [SQLServerCE (2002)]

4.3.5 Konfliktlösung bei der Merge-Replikation

Beim SQL-Server CE werden Konflikte nur auf Zeilenebene erkannt. Wenn ein Konflikt beim Einfügen, Ändern oder Löschen auftritt, bietet der SQL Server Standardoptionen an um den Konflikt aufzulösen:

- Mittelwert: Aus dem numerischen Wert wird ein Mittelwert gebildet.
- früher gewinnt(DATETIME):Die Spalte mit dem früheren DATETIME-Wert wird übernommen.
- später gewinnt (DATETIME):Die Spalte mit dem späteren DATETIME-Wert wird übernommen.
- Maximum: Der Datensatz mit der Spalte mit dem höheren numerischen Wert wird übernommen.
- Textspalten zusammenführen: Der Text wird aus beiden Werten zusammengeführt.
- Minimum: Der Wert mit dem niedrigeren numerischen Wert wird übernommen.
- Subscriber gewinnt immer: Der Wert des Subscribers wird übernommen.

Zusätzlich können noch Benutzerdefinierte Konfliktlöser implementiert werden. In diesen können spezifische Regeln zur Lösung von Konflikten entworfen werden. Diese können z. B. als Stored-Procedures erstellt werden.

4.3.6 Einschränkungen

Um die SQL Server CE Datenbank an die beschränkten Ressourcen auf mobilen Geräten anzupassen wurden einige Elemente des SQL Server nicht implementiert und können so auch nicht repliziert werden. Es werden keine *Check constraints*, *Stored Procedures*, *Views*, Funktionen und *Trigger* unterstützt. Diese Funktionalität muss in der Anwendung programmiert werden.

4.4 Realisierung

Für die Entwicklung der Anwendung wird Microsoft Visual Studio .Net mit dem Compact Framework verwendet. Die Entwicklung erfolgt auf einem Emulator auf dem die Anwendung getestet werden kann.

4.4.1 Relationales Datenbankmodell

Für die Umsetzung des Szenario wird zunächst ein relationales Datenbankkonzept erstellt.

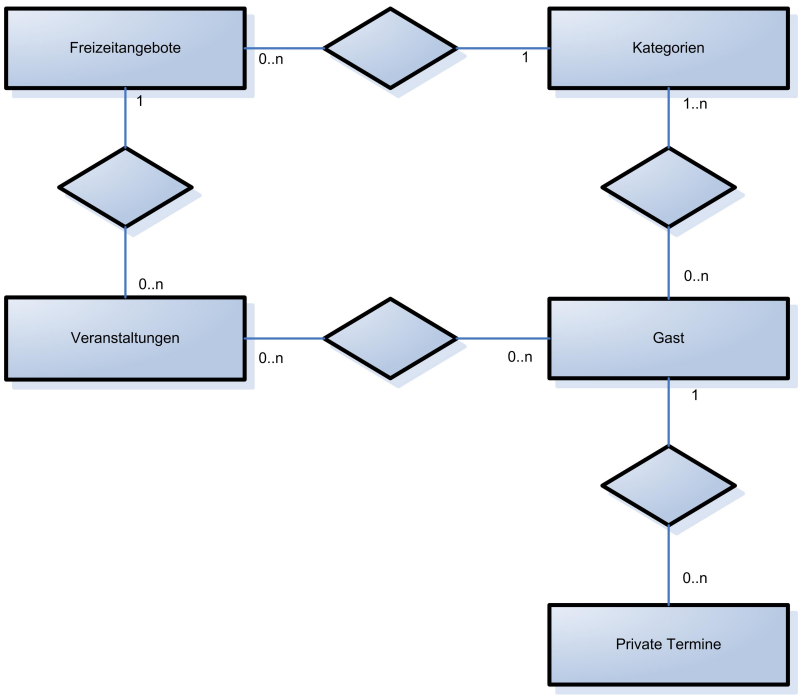

Abbildung 4.5: E/R Modell

Freizeitangebot	ID	KategorieID	Beschreibung
	01	01	Bogenschießen
	02	01	Fußballtunier
	03	02	Fahrt zur Sehenswürdigkeit

Kategorie	ID	Beschreibung
	01	Sport
	02	Kultur

Veranstaltungen	ID	FreizeitID	Datum	Beginn	Dauer	Ort	Plätze
	100	01	25.08.2005	10:00	2	große Wiese	10
	1001	02	26.08.2005	09:00	2	Fussballplatz	24
	1002	03	26.08.2005	07:00	8	Eingangshalle	20

Gast	ID	Name	ZimmerNr	Ankunft	Abreise
	01	Müller	105	24.08.2005	07.08.2005
	02	Meier	200	24.08.2005	01.09.2005

GastKatWahl	GastID	KategorieID
	01	01
	02	01
	02	02

Buchungen	GastID	VeranstaltungenID
	01	100
	02	100
	02	103

PrivateTermine	ID	GastID	Datum	Begin	Dauer	Beschreibung
	01	01	24.08.2005	09:00	1	joggen
	02	01	26.08.2005	16:00	2	Verabredung mit Frau Meier
	02	02	26.08.2005	16:00	3	Verabredung mit Herrn Müller

Dieser relationale Datenbankentwurf soll einen Überblick über die Tabellenstruktur geben. Außerdem sollte noch mindestens ein „View“ für die Umsetzung des persönlichen Kalenders der Gäste erstellt werden. Zusätzlich muss eine Funktion für das Buchen einer Veranstaltung und Eintragen eines privaten Termins geschrieben werden, die Überwacht, ob es bei neuen Einträgen zu Terminüberschneidungen kommt. Bei einer Buchung muss mit einem Trigger dafür gesorgt werden, dass die Spalte *Plätze* in der Tabelle Veranstaltungen reduziert wird.

4.4.2 Erstellen der Publikation

Für die Replikation von Daten auf einem mobilen Client muss zuerst eine Publikation der zu replizierenden Daten auf dem SQL Server erstellt werden. Diese kann der mobile Client dann abonnieren. Dazu wird auf dem SQL Server eine neue Publikation mit Hilfe des Enterprise Managers angelegt. Hierzu wird der Publikationstyp als Merge-Publikation und als Subscriber der SQL Server CE festgelegt. Dann können die Tabellen, die repliziert werden sollen, ausgewählt werden. Der SQL Server fügt den Tabellen automatisch eine „unique identifier“-Spalte hinzu. Mit dieser können die Zeilen einer Tabelle global identifiziert werden. Es ist es möglich die zu replizierenden Tabellen zu filtern. Hierbei können horizontale und vertikale Filter, die statisch oder dynamisch sein können, hinzugefügt werden. Für die Beispieldatenbank wird ein dynamischer Filter auf die Tabelle *Gast* gelegt.

Dynamische Filter können System- oder benutzerdefinierte Funktionen verwenden. Die Funktion wird dann anhand der Verbindungseigenschaften vom Merge-Agent für die Abonnenten ausgewertet. Dafür steht die Systemfunktionen *SUSER_SNAME()* zur Verfügung.

Um den Filter auf die Tabellen auszuweiten, die die Tabelle *Gast* mit einem Fremdschlüssel referenzieren, wird ein Verknüpfungsfilter eingesetzt. So wird die referentielle Integrität zwischen den Tabellen hergestellt.

4.4.3 Konflikte

Um entstehende Konflikte auflösen zu können, wird für die einzelnen Tabellen jeweils ein Konfliktlöser in der Publikation eingestellt. Im Folgenden werden einige Anwendungsfälle beschrieben, die zu Konflikten führen und die entsprechende Lösung mit angeführt.

Zwei Gäste melden sich für dieselbe Veranstaltung an. Betroffen von dieser Operation sind die Tabellen *Buchungen* und die Tabelle *Veranstaltungen*. In der Tabelle *Buchungen* wird jeweils ein neuer Datensatz eingefügt. Auf der Tabelle *Veranstaltungen* entsteht ein Konflikt, weil zwei Benutzer denselben Datensatz ändern möchten (UPDATE-UPDATE). Die UPDATE-Anweisung sieht wie folgt aus:
UPDATE Veranstaltungen SET Veranstaltungen.Plätze = Veranstaltungen.Plätze - 1 WHERE VeranstaltungID = 1;
Dieser Konflikt kann mit dem Konfliktlöser *Minimum* gelöst werden. Der Datensatz mit dem kleineren numerischen Wert wird übernommen. Wenn aber nur noch ein Platz in der Veranstaltung frei ist, wird der Konflikt über *früher gewinnt* gelöst. Zusätzlich muss dafür gesorgt werden, dass für den Gast, dessen Änderung verworfen wurde, der eventuell schon eingefügte Datensatz in der Tabelle *Buchungen* wieder rückgängig gemacht wird. Um zwischen den beiden Konfliktsituationen zu unterscheiden, muss man einen benutzerdefinierten Konfliktlöser erstellen.

Eine Veranstaltung findet nicht statt und wird vom Redakteur gelöscht. Hier wird zwischen zwei *DELETE* Konflikten unterschieden: 1. Ein Gast, der seine Daten noch nicht synchronisiert hat, meldet sich für den gelöschten Kurs an. Hier tritt ein *DELETE-UPDATE* Konflikt auf. Der Konflikt braucht nicht aufgelöst zu werden. Die Änderung kann nicht eingebracht werden und wird verworfen. 2. Ein Gast mit nicht synchronisierten Daten, der für die gelöschte Veranstaltung angemeldet war, möchte sich abmelden. Dazu storniert er die Buchung und ein *DELETE-DELETE* Konflikt entsteht, da der Datensatz schon gelöscht wurde. Die zweite Löschung wird verworfen.

Wenn eine Veranstaltung vom Redakteur gelöscht wird, benachrichtigt er die Gäste, die diese Veranstaltung gebucht hatten. Dazu macht er einen Eintrag in der Tabelle *PrivateTermine* zu der Zeit, an dem die Veranstaltung stattgefunden hätte. Wenn nun ein Gast, der zu dieser Veranstaltung angemeldet war, diese storniert und einen neuen Termin zu diesem Zeitpunkt eingetragen hat, gibt es einen *INSERT-INSERT* Konflikt. Durch die „unique identifier“-Spalte, die bei der Erstellung der Publikation hinzugefügt wurde, kann es nicht zu einem identischen

Primärschlüssel kommen, aber die Termine können sich überschneiden. Hier muss ein benutzerdefinierter Konfliktlöser erstellt werden, der auf dem Standardkonfliktlöser *Subscriber gewinnt immer* basiert. Damit wird sichergestellt, dass der Gast die Verfügungshoheit über seine Termine hat.

4.4.4 Realisierung der Replikation und Synchronisation

Für die Merge-Replikation stellt der Client-Agent Methoden zur Datenbankbearbeitung und Kommunikation mit dem SQL-Server CE Server Agent zur Verfügung. Die folgenden Programmausschnitte sind nach einer Beispielanwendung aus [Tiffany (2003)] entstanden.

Abonnement erstellen
Die *AddSubscription* Methode wird benutzt, um ein Abonnement an den Verleger (Publisher) zu erstellen. Die gesamte Publikationsdefinition, einschließlich der Daten und des Schemas der replizierten Tabellen, wird in der SQL Server CE Datenbank nach der Synchronisierungsmethode erstellt.

```
private void btnCreateSubscription_Click(object sender,System.EventArgs e)
{
 SqlCeReplication rep =null;
 try
 {
  btnCreateSubscription.Enabled =false;
  rep =new SqlCeReplication();
  //This is the connection string to your local SQL Server CE database file
  rep.SubscriberConnectionString ="Data Source=\\My Documents \\"+
          "FerienclubDatenbank.sdf;"+
          "SSCE:Database Password=Muellerpass";
  //If database doesn't exist then...
  if (!File.Exists("\\My Documents \\FerienclubDatenbank.sdf"))
  {
   //Add a subscription and create the local database
   rep.AddSubscription(AddOption.CreateDatabase);
  }
 }
 ...
}
```

Abbildung 4.6: Ausschnitt aus AddSubscription

Synchronisieren Die *Synchronize* Methode startet die Merge-Replikation zwischen dem Abonnement auf dem mobilen Client und dem Verleger der zentralen Datenbank auf dem Server. Wenn auf dem Client Daten geändert wurden, werden diese Änderungen an den Server geschickt und abgeglichen. Dies geschieht auch in umgekehrter Richtung, wenn Daten auf dem Server geändert worden sind.

Abonnement löschen Die *DropSubscription* Methode löscht das Abonnement einer Publikation. Es wird auch dafür genutzt, um die lokale Datenbank auf dem mobilen Client zu löschen.

```
private void btnSynchronize_Click(object sender,System.EventArgs e)
{
 SqlCeReplication rep =null;
 try
 {
  btnSynchronize.Enabled =false;
  rep =new SqlCeReplication();
  //This is the URL that points to sscesa20.dll running on IIS
  rep.InternetUrl ="http://BARNEY2/sqlceWeb/sscesa20.dll";
  //This is your Basic Authentication username
  rep.InternetLogin ="test";
  //This is your Basic Authentication password
  rep.InternetPassword ="test";
  //This is the name of the server running SQL Server
  rep.Publisher ="BARNEY2";
  //This is the name of the actual database that you're publishing
  rep.PublisherDatabase ="Ferienclub";
  //This is your SQL Server username
  rep.PublisherLogin ="PocketPC";
  //This is your SQL Server password
  rep.PublisherPassword ="muellerpass";
  //This is the name that you give to your publication
  //during the wizard setup
  rep.Publication ="FerienclubPub";
  //This anonymous subscriber name can be anything you want it to be
  rep.Subscriber ="Subscriber";
  //This is the connection string to your local SQL Server CE database file
  rep.SubscriberConnectionString ="Data Source=\\My Documents \\"+
          "FerienclubDatenbank.sdf;"+
          "SSCE:Database Password=muellerpass";//If database already exists then...
  if (File.Exists("\\My Documents \\FerienclubDatenbank.sdf"))
  {
   //Synchronize with SQL Server
   rep.Synchronize();
  }
  else
  {
   MessageBox.Show("You must first create a database","Error");
  }
  ...
 }
```

Abbildung 4.7: Ausschnitt aus Synchronize

```
private void btnDropSubscription_Click(object sender,System.EventArgs e)
{
 SqlCeReplication rep =null;
 try
 {
  btnDropSubscription.Enabled =false;
  rep =new SqlCeReplication();
  //This is the connection string to your local SQL Server CE database file
  rep.SubscriberConnectionString ="Data Source=\\My Documents \\"+
          "FerienclubDatenbank.sdf;"+
          "SSCE:Database Password=muellerpass";
  //If database already exists then...
  if (File.Exists("\\My Documents \\FerienclubDatenbank.sdf"))
  {
   //This drops the subscription and deletes the database
   rep.DropSubscription(DropOption.LeaveDatabase);
  }
  else
  {
   MessageBox.Show("You must first create a database","Error");
  }
 }
 ...
}
```

Abbildung 4.8: Ausschnitt aus DropSubscripion

5 Fazit und Ausblick

Für diese Arbeit habe ich die grundlegenden Funktionen für die Replikation und Synchronisation mit mobilen Datenbanken ausgeführt. Die Entwicklung einer „einfachen“ Testanwendung für die SQL Server CE Datenbank mit Replikation und Synchronisation von Daten eines zentralen SQL Servers ist einfach zu erstellen. Mit Hilfe der Entwicklungsumgebung Visual Studio .NET und dem Compact Framework auf einem Emulator sind schnell Erfolge zu erzielen. Der SQL Server CE bietet einfache Methoden an, die einem einen Großteil der Replikations- und Synchronisationsproblematik abnehmen. Auf Seiten des SQL Server können die grundlegenden Aktionen, wie z. B. die Erstellung der Publikation oder der Konfliktlöser, über Wizards (Menügesteuert) erstellt werden.
Will man jedoch eine komplexere Anwendung erstellen, die nicht triviale Anforderungen an die Datenbank stellt, muss man viele Zeilen programmieren, um die gewünschte Funktionalität zu erzielen. Dabei muss auch viel SQL-Code von Hand geschrieben werden.
Auch auf dem SQL Server kommt man mit den Standardfunktionen dann nicht mehr weiter. Die vorgegebenen Konfliktlöser sind beschränkt und es müssen häufig benutzerspezifische Konfliktlöser eingesetzt werden. Dies verleitet dazu, das Datenbankdesign an die vorgegebenen Lösungsstrategien anzupassen.

Da der SQL Server CE im Gegensatz zur Servervariante keine Trigger zulässt, ist kein einheitliches Design der Anwendung möglich. Wenn serverseitig Trigger eingesetzt werden, müssen Änderungsoperationen doppelt implementiert werden: Einmal mit einem Trigger oder mit einer „stored procedure“ und einmal innerhalb der Anwendungslogik auf dem Client. Diese Implementierung auf unterschiedliche Weise ist anfällig für inkonsistente Prüfungen und damit für Fehler. Um das Ferienclub-Szenario vollständig Umzusetzen, müsste ich mich noch intensiv mit in dem SQL-Server auseinandersetzen und für den Client eine eine Benutzerschnittstelle programmieren.

Das vorgestellte Ferienclub-Szenario könnte verbessert werden, indem eine eine Strategie zur Auswahl alternativer Veranstaltungen entworfen wird. Denn in dem vorliegendem Entwurf kann man sich nur für eine Veranstaltung anmelden und hoffen, dass kein Konflikt auftritt und die Buchung nicht abgewiesen wird. So könnten etwa mit der Auswahl von „alternativen Veranstaltungen“ parallel laufende Termine bestimmt werden, die, im Fall von Konflikten beim Buchen der primären Veranstaltung, ersatzweise gebucht werden.

Literaturverzeichnis

[Aktiv-Reisen 2005] AKTIV-REISEN: *Incentives und Events*. 2005. – URL http://www.aktiv-reisen.com/bogenschiessen.htm

[Bresch 2004] BRESCH, Marco: *Erweiterung des JavaServer Faces Frameworks für den Einsatz in mobilen Anwendungen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2004. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/bresch.pdf

[Chrysanthis 1993] CHRYSANTHIS, Panos K.: Transaction Processing in Mobile Computing Environment. In: *Proceeding of IEEE Workshop on Advances in Parallel and Distributed Systems* (1993), S. 77–83

[(CTR) 1995] (CTR), Computer Technology Research C.: *Client/Server Application Development Tools and Techniques*. 1995

[Dadam 1996] DADAM, Peter: *Verteilte Datenbanken und Client/Server-Systeme: Grundlagen, Konzepte, Realsierungsformen*. Springer-Verlag, 1996. – ISBN 3-540-61399-4

[dal/c´t 2005a] DAL/C´T: PDA mit Festplättchen. In: *c´t magazin für computer technik* (2005), Juni, Nr. 12, S. 27

[dal/c´t 2005b] DAL/C´T: *Studie: PDA-Verkäufe erneut rückläufig*. 2005. – URL http://www.heise.de/newsticker/meldung/55929

[Date 1987] DATE, C. J.: Twelve Rules for a Distributed Database. In: *Computer World* 23 (1987), Juni, Nr. 2, S. 77–81

[Dirckze und Gruenwald 2000] DIRCKZE, R. A. ; GRUENWALD, L.: A Pre-Serialization Transaction Management Technique for Mobile Multidatabases. In: *Mobile Networks and Applications* 5 (2000), Nr. 4, S. 311–321

[Dunham u. a. 1997] DUNHAM, Margaret H. ; HELAL, Abdelsalam ; BALAKRISHNAN, Santosh: A mobile transaction model that captures both the data and movement behavior. In: *Mobile Networks and Applications* 2 (1997), Nr. 2, S. 149–162

[Gray u. a. 1996] GRAY, Jim ; HELLAND, Pat ; ONEIL, Patrick ; SASHA, Dennis: The Dangers of Replication and a Solution. In: *Proceeding of ACM SIGMOD Conference* (1996), S. 173–182

[Höpfner] HÖPFNER, Hagen: Grundlagen von Replikationstechniken für mobile Datenbanken. In: *Tagungsband 13. GI-Workshop*

[Lu und Satyanaranyanan 1994] LU, Qi ; SATYANARANYANAN, M.: Isolation-only transaction for mobile computing. In: *Operating Systems Review* 28 (1994), April, Nr. 2, S. 81–87

[Lüpke 2004] LÜPKE, André: *Entwurf einer Sicherheitsarchitektur für den Einsatz mobiler Endgeräte*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2004. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/luepke.pdf

[Madria und Bhargava 2001] MADRIA, S. K. ; BHARGAVA, B.: A Transaction Model for Improving Data Availability in Mobile Computing. In: *Distributed and Parallel Databases* 10 (2001), Nr. 2, S. 127–160

[Mutschler und Specht 2003] MUTSCHLER, Bela ; SPECHT, Günther: Implementierungskonzepte und Anwendungsentwicklung kommerzieller mobiler Datenbanksysteme - Ein Vergleich. In: *Workshop GI-Arbeitskreis: Mobile Datenbanken- und Informationssysteme* (2003)

[Mutschler und Specht 2004] MUTSCHLER, Bela ; SPECHT, Günther: *Mobile Datenbanksysteme*. Springer, 2004. – ISBN 3-540-20886-0

[Pitoura und Bhargava 1999] PITOURA, Evaggelia ; BHARGAVA, Bharat: Data Consistency in Intermittently Connected Distributed Systems. In: *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,* 11 (1999), Nr. 6, S. 896–915

[Schneebauer 2003] SCHNEEBAUER, Christina: *Automatisches Testen von heterogenen Datenbank-Replikationstools mit GTS*, FH-Hagenberg, Diplomarbeit, 2003. – URL http://webster.fh-hagenberg.at/staff/kurschl/pubs/advisedDT/Schneebauer.2003.pdf

[Serrano-Alvarado u. a. 2001] SERRANO-ALVARADO, Patricia ; RONCANCIO, Claudia ; ADIBA, Michel: Analyzing Mobile Transactions Support for DBMS. In: *LSR-IMAG Laboratory* (2001)

[Silberschatz u. a. 2005] SILBERSCHATZ, Avi ; KORTH, Henry F. ; SUDARSHAN, S.: *Database System Concepts*. McGraw-Hill, 2005. – ISBN ISBN 0-07-295886-3

[SQLServerCE 2002] SQLSERVERCE: *SQL Server CE Books Online*. Microsoft Corporation, 2002

[Tiffany 2003] TIFFANY, Rob: *SQL Server CE Database Development with the .NET Compact Framework*. Springer-Verlag, 2003. – ISBN 1-59059-119-4

[UbiComp@HAW-Hamburg 2002] UBICOMP@HAW-HAMBURG: *The UbiComp-Lab at the University of Applied Sciences Hamburg concerning ubiquitous computing, esp. software architectures for distributed systems*. 2002. – URL http://users.informatik.haw-hamburg.de/~ubicomp/

[Walborn und Chrysanthis 1995] WALBORN, Gary D. ; CHRYSANTHIS, Panos K.: Supporting Semantics-Based Transaction Processing in Mobile Database Applications. In: *Symposium on Reliable Distributed Systems* (1995)

[Walborn und Chrysanthis 1997] WALBORN, Gary D. ; CHRYSANTHIS, Panos K.: ProMotion: Management of Mobile Transactions. In: *In Proceedings of 11th ACM Annual Symposium on Applied Computing* (1997), S. 101–108

[Walborn und Chrysanthis 1999] WALBORN, Gary D. ; CHRYSANTHIS, Panos K.: ProMotion: Management of Mobile Transactions. In: *In Proceedings of 14th ACM Annual Symposium on Applied Computing* (1999), S. 389–398

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 29. August 2005	
Ort, Datum	Unterschrift