A survey of academic and commercial approaches to transaction support in mobile computing environments

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Department of Computer Science
Institute of Information Systems

A Survey of Academic and Commercial Approaches to Transaction Support in Mobile Computing Environments

Can Türker and Gabriele Zini

Technical Report #429, 2003

November 2003

Institute of Information Systems
Dr. C. Türker

Address:

C. Türker, ETH Zurich, Institute of Information Systems,
ETH Zentrum, CH-8092 Zurich, Switzerland,
tuerker@inf.ethz.ch

This report is available via http://www.inf.ethz.ch/publications/ or
ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/

A Survey of Academic and Commercial Approaches to Transaction Support in Mobile Computing Environments

Can Türker and Gabriele Zini

Swiss Federal Institute of Technology Zurich
Institute of Information Systems, ETH Zentrum
CH-8092 Zurich, Switzerland
tuerker@inf.ethz.ch

Abstract

The need for information access and processing in mobile computing environments presents great challenges to database researchers. Apart from advances in wireless communications and device technology, new data management techniques are needed to provide a transparent interaction between fixed and mobile devices in global information systems. A key aspect is to provide a flexible and powerful transaction processing infrastructure that takes into account the peculiarities of mobile computing. Mobile transaction processing shall be able to transparently deal with frequent disconnections and occasional movements of mobile devices. This paper surveys the mobile transaction models proposed in academia and outlines the state-of-the-art of transaction processing in commercial mobile databases.

1 Introduction

Mobile devices such as laptops, palmtops, and smart phones have nowadays become more widespread than desktop computers. The possibility they give us to access and process information on the move, everywhere and at any time, has been one of the most prominent reasons for their huge acceptance and success.

Accenture [Acc01] states that by 2005 500 million mobile devices will offer Internet access, a number that outstrips personal computers. Globally, the market for small wireless Internet-capable devices, including handheld computers, basic micro-browser phones, smart phones and next generation multimedia phones, is set to grow from 10 billion USD in 2000 to 73 billion USD in 2005". More than 137 million business users worldwide will be involved in some form of remote work. Every kind of job, in every industry can be affected by remote work. Remote-ready jobs are sales and service, executive travel, banking and brokerage, help desk, marketing, medical, programming, shipping and delivery, customer relations, supply bid and order. These estimates show us the importance and need of a move from a *static* to a *mobile* information processing environment.

It is undisputed that mobility raises several new challenges to the information systems community [Wei93, AK93, IB94, PS99, JHE99, Fra01]. Mobile computing introduces new variables that must be taken into proper consideration: *dynamically changing location* of mobile devices, *varying connection quality* of the link used to access the fixed network, *frequent disconnections* caused by bad connections or device switch-offs, and *limited resources* of mobile devices which tend to have low computing power, little batteries, and small screens. Besides, mobile computing has to cope with all the problems peculiar to heterogeneous information systems [SL90],

such as semantic and syntactic heterogeneity [GSC96], database integration [BLN86], conflict resolution [KCGS95], and global transaction management [BGS92].

In this paper, we focus on *mobile transaction management*. Due to the inherent nature of mobile computing, transaction processing must especially deal with the issue of frequent disconnections. Long-lived transactions started by a mobile host that disconnects for a long time can lead to unacceptable long locking times of data. In such circumstances, applications running on top of disconnected or weakly connected hosts need to be designed to improve data availability and minimize conflict between updates in order to avoid high abort rates. Although a pure flat transaction approach to mobile processing seems to be inappropriate, certain transactional guarantees shall be ensured always and everywhere.

Mobile transaction management must cope with the uncertainties of mobile computing environments while providing a viable way of concurrently accessing and processing data in networks consisting of heterogeneous static and mobile components. As we will see in this paper, there is a significant gap between academic and commercial approaches to transaction support in mobile computing environments. Current commercial approaches focus on rather simple applications like single-user palmtop applications managing address or phone books. There available solutions are based on "primitive" data synchronization techniques while neglecting the transactional semantics of the applications.

A number of papers [Mad98, Bar99, CFHR01, SRA01] have already attempted to survey the transaction models for mobile computing. Unfortunately, these papers consider only a few models and often do not provide a comparative overview. Furthermore, none of these papers considers the current offers of commercial mobile database suppliers. With our contribution we provide a more comprehensive overview of the state-of-the-art of mobile transaction models appeared in literature up to now. We compare and discuss those models with regards to mobility requirements and transaction properties. We additionally examine available approaches of all major commercial mobile databases.

The paper is organized as follows: Section 2 compiles the main characteristics and restrictions that mobile computing environments put on global transaction management, briefly summarizing the basic concepts of transaction models and data replication. Section 3 reviews the advanced transaction models proposed for mobile computing. Section 3.11 provides some related approaches that however do not provide a mobile transaction model per se. Section 4 explores the current commercial approaches to mobile computing undertaken by the major mobile database suppliers. Section 5 concludes the necessity of further research by pointing out the deficiencies of current proposals.

2 Preliminaries

2.1 Characteristics of Mobile Environments

Figure 1 depicts the reference model for mobile computing environments. Terminals, desktop computers, servers are the *fixed hosts* (FH) that are interconnected by means of a fixed network. Large databases run on servers that guarantee efficient processing and reliable storage of data.

Mobile hosts (MH) like palmtops, laptops, notebooks, or cellular phones are, of course, not always connected to the fixed network. They may be disconnected for different reasons. Mobile hosts may differ in their capabilities with respect to computing power and storage space. While notebook computers can run databases as desktop computers, other mobile devices have more limited computational resources. For instance, palmtops have much less memory than a notebook computer. Still, it is possible to use small footprint databases that allow data processing even on mobile devices with very restricted resources. After having uploaded data from the FH to their MHs, mobile users can disconnect from the fixed network. While moving

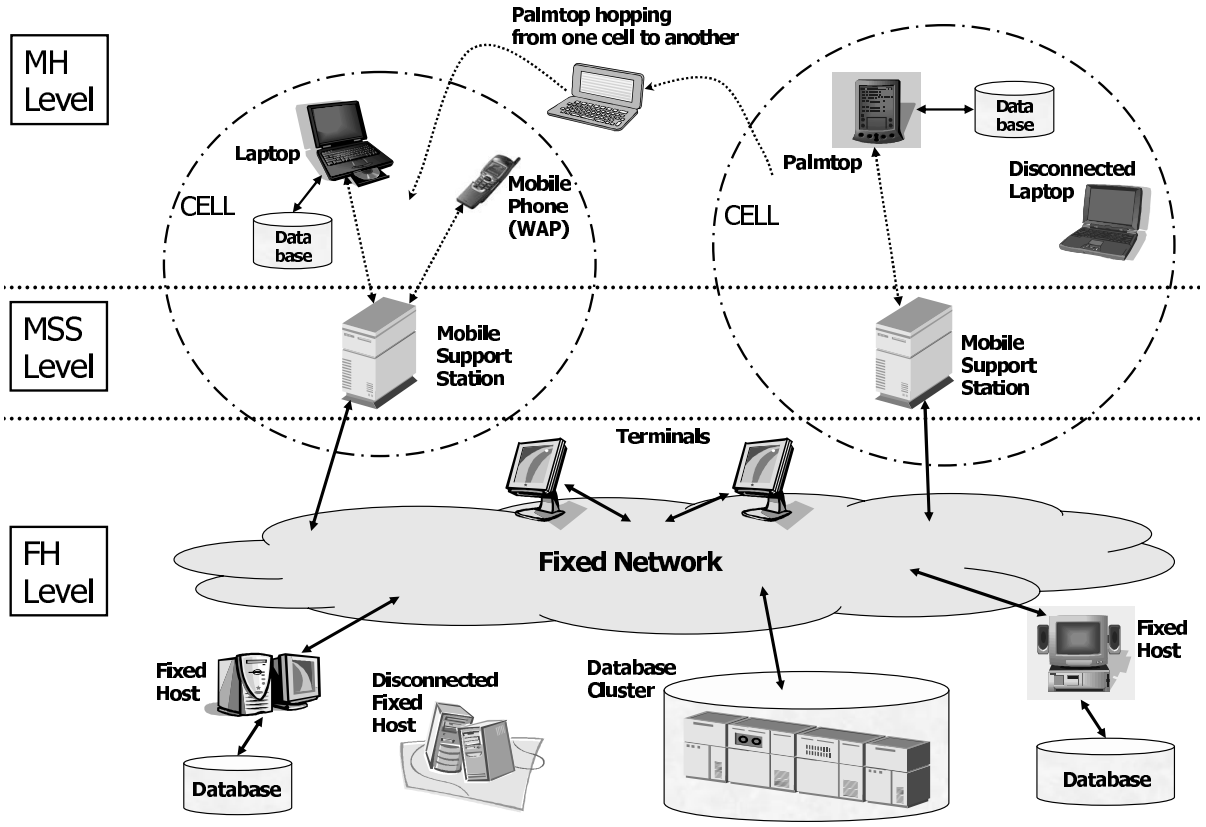

Figure 1: Reference Model of a Mobile Computing Environment

(physically), they might want to run some queries and update their local data copies.

The connection between mobile and fixed hosts is performed by means of *mobile support stations* (MSS). MSSs are dedicated fixed hosts that provide the link between a MH and any FH via wireless LAN connections (using protocols like Bluetooth), cells (like in the GSM cellular phone system), or connections to the network using standard modems.

Wireless communication technology gives users the possibility to reconnect to the fixed network anytime and anywhere to share the results of their work. The possible mobility poses some difficulties to the task:

- Bandwidth variability occurs as the MH changes location, for instance, due to different traffic loads or changing networks that interface with the MH.
- When the information is location-specific, it becomes necessary for the MSS to track the location of the MH.
- Due to a change in the physical location, an MH can switch its supporting MSS when moving to a different cell. This leads to the need for a hand-off procedure to enable the new MSS involved to support and maintain the connection with the MH.
- Unreliability can be introduced by local interruptions in signal strength or by MH's hardware failures or switch-offs that cause a disconnection of the MH from the network.
- Tariff policies can make it uneconomical to be connected at certain times and/or in certain locations. For example, it would be preferable to execute an e-commerce transaction only when I reenter my home country in order to save connection costs.

- Security and authentication must also be taken into proper account since moving to a new MSS can be prevented if local MSS requirements about security are failed to be met.

Mobile devices also introduce issues:

- MHs tend to have less internal computational resources than FHs.
- Small screens limit the man-machine interaction capabilities of some mobile devices.
- Battery life is limited causing devices to limit their connections.
- Due to power limitations it is cheaper to receive than transmit and the transmission bandwidth of the mobile device tends to be lower than the transmission bandwidth of the MSSs. This is often referred to as *communication asymmetry*.
- MHs are inherently less secure and reliable due to the possibility of being damaged or even stolen.

It is worth pointing out that not all of these characteristics are directly affecting mobile transaction computing.

2.2 Basic Notions of Transaction Models

Before we can discuss the proposals for mobile transaction models, we have to introduce some basic notions of transaction models that will be needed in the remainder of the paper. For detailed explanations see [BHG87, GR93, CR90].

A *transaction* is a unit of operations that changes a database from a consistent state into another consistent state. A database is in a *consistent state* if it satisfies all defined semantic integrity constraints and all (previous and current) database states are achieved by performing correct database transitions. A *transaction model* defines the framework for the definition and execution of transactions. A traditional *flat* transaction satisfies the well-known ACID properties (Atomicity, Consistency, Isolation, Durability), meaning that a transaction is an atomic, consistent and recoverable unit that does not interfere with other transactions that are executed concurrently [Gra78].

In some application scenarios, ACID properties must be relaxed. This is true not only for transaction models in fixed networked environments but also to a greater extent when mobility is considered. Mobile transaction models has to cope with the characteristics of mobile environments that have been outlined in the previous subsection.

A way to relax the ACID properties is to go beyond the flat transaction approach by extending it with models that allow a transaction to be divided into several other subtransactions that can be nested [Mos85]. A subtransaction can be the child of another subtransaction and be the parent of another one. When transactions are nested, subtransactions can be classified according to the following characteristics [ST97]:

- *Open vs. Closed*: These properties are used to distinguish whether the results of a subtransaction are made visible to all transactions or only to the parent transaction.
- *Vital vs. Non-Vital*: A parent transaction may be dependent on the abortion of the child transaction. If a vital transaction aborts, the parent transaction must abort, too. A non-vital transaction abortion does not lead to the parent transaction's abortion.
- *Dependent vs. Independent*: Analogously, the termination of a child can be dependent on the abortion of the parent transaction. A dependent child transaction will have to abort if the parent transaction aborts.

- *Compensatable vs. Non-Compensatable*: Transactions can further be distinguished whether their effects are compensatable. A transaction is *compensatable* if there exists another transaction, called *compensating* transaction, that semantically undoes its effects.
- *Substitutable vs. Non-substitutable*: Transactions can also be distinguished whether they can be substituted by another transaction. A transaction is called *substitutable* if there exists another transaction, called *contingency* transaction, that can be executed on its behalf.
- *Temporal*: These are transactions that have to observe temporal constraints, such as a given termination deadline.

2.3 Basic Notions of Data Replication

Mobile computing requires the ability to continue information processing on the MH while disconnected from the fixed network. To be able to process information on the MH in the disconnected mode, data must be *replicated* on the MH. *Data replication* [GHOS96, Bur97] is the process of maintaining a defined set of data in more than one location. It involves copying designated changes from one location (*source* or *master*) to another (*target* or *remote*), and synchronizing the data in both locations. An MH has its own complete or partial copy of the master database (that resides on the FH) so that it can process data locally. In general, data replication provides:

- *Data availability*: Data is made available locally to the remote databases, thus avoiding the need to connect to a single central database through less reliable and slow connections, or when a connection to the central server is impractical.
- *Faster response times*: Replication improves response times for data requests since requests are processed on a local server and because local processing decreases workload from the central database server.

The increased flexibility permitted by replication introduces several challenges.

- *Transactional integrity*: A difficulty of any replication system is ensuring that each database retains transactional integrity at all times. Some replication systems replicate portions of the transaction log in such a way that transactions are replicated atomically: either a whole transaction is replicated, or none of it is replicated. This ensures transactional integrity at each database. Other technologies replicate data by consolidating changes made by different transactions that have committed. Then the changes are applied to another database in a single transaction.
- *Data consistency*: Data consistency must be enforced on all databases. In practice, all changes are replicated to each site over time in a consistent manner, but at the same time, different sites may store different copies of data.

Whenever an update is performed on any of the databases, the update must be propagated to the other databases in order to maintain a consistent state in all databases. A strategy to propagate the update can be based on time. In this case, replication can be:

- *Eager*: All replicas are updated synchronously as part of the originating transaction. This approach is often also called *synchronous* replication.
- *Lazy*: One replica is updated by the originating transaction and propagation to other replicas is asynchronous, typically as a separate transaction for each replica. This approach is often also called *asynchronous* replication.

Also, update propagation can be controlled by means of data ownership:

- *Master*: Each data object has a master node. Only the master can update the primary copy of the data object on its local and remote databases.
- *Group*: Any replica with a copy of a data item can update it.

In case of a group update, the chances for conflicting updates are higher than in case of a master update.

3 Overview of Mobile Transaction Models

All subsections on mobile transaction models will follow the same schema. After a *brief description* of the model, we will discuss how it handles *transaction mobility and disconnection*. We will sketch how the model deals with *transaction execution and data management*, describing where the transactions run (on the MH, the MSS and/or the FH) and if the model considers data replication, caching or other techniques. Finally, we will outline the *properties* of transactions supported in the different models. Our overview will include the following models:

- Reporting and Co-Transactions [Chr93],
- Isolation-Only Transactions [LS94],
- Multi Database System Transaction Processing Manager [YZ94b, YZ94a],
- Weak-Strict Transactions [PB94, PB95, Pit96, PB99],
- Two-Tier Replication [GHOS96],
- Kangaroo Transactions [DHB97, DK99],
- Pro-Motion [WC97, WC99],
- Toggle Transactions [DG98],
- Moflex Transactions [KK00].

3.1 Reporting and Co-Transactions

Description. [Chr93] extends the open nested transaction model [WS92] by addressing cell migration issues. The proposed mobile transaction model is devised for MHs constantly connected to the network but moving through different cells, i.e., every MH is always connected to an MSS but shifts from one MSS to another when it moves from one to another cell.

The top-level transaction, which is responsible for the overall computation, has the ability to spawn four types of subtransactions (termed *component transactions*) that are expected to run on both MH and FH:

- *Atomic transactions* are compensatable transactions.
- *Non-compensatable transactions* cannot publish their results at commit time. They give over the responsibility for committing or aborting to the top-level transaction.
- *Reporting Transactions* can share their partial result with the top-level transaction any time during the computation. They can be associated to compensating transactions depending on whether or not they delegate the commit of the results to the top-level transaction.

- *Co-Transactions* are reporting transactions that cannot execute concurrently but behave like *co-routines*. When their results are passed back to top-level transaction they stop their execution. They restart where they were as soon as they are resumed, retaining their state across executions.

Transaction Mobility and Disconnections. Reporting transactions and co-transactions are able to exchange messages one another. They are also able to relocate their execution to different MSS following the path of the MH, aiming at minimizing bandwidth occupation. The details of transaction relocation are not reported in the cited paper. Since MHs are assumed to be always connected to the network, disconnection handling is not considered in the model.

Transaction Execution and Data Management. This model provides the ability to split a mobile computation into a set of transactions that are partly executed on the MH and partly on the FH (see Figure 2). No details are given on data management, although some sort of data replication must be considered in order to cater for some transaction processing capability on the MH.

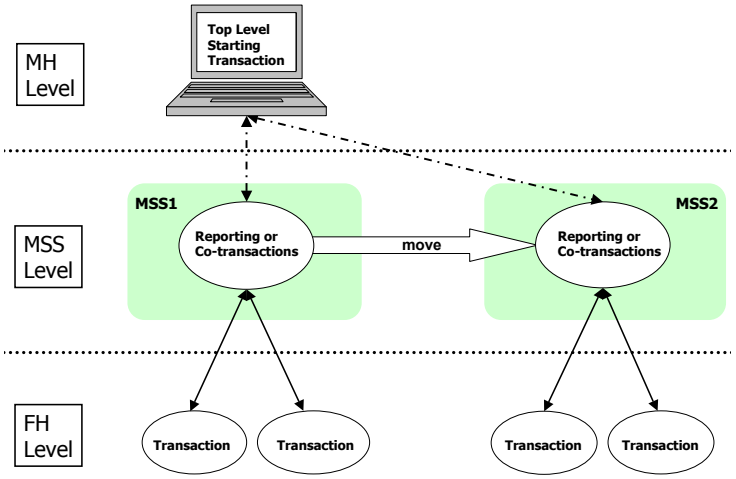

Figure 2: Reporting and Co-Transactions

Transaction Properties. Based on open nested transactions [WS92], this model considers *semantic atomicity* [MRKS92] by using compensating transactions. Non-compensatable, reporting and co-transactions delegate the decision whether or not to commit to the top-level transaction. Isolation is maintained through serialization ordering.

3.2 Isolation-Only Transactions

Description. The *Isolation-Only Transaction Model* (IOT) [LS94] has been developed for reading and writing UNIX files by means of remote MHs. The IOT model, developed within the context of the CODA file system [SKK+90], enables *disconnected operations* through disk caching.

The transaction execution is entirely performed on the MH with a cached copy of the files of the server. The results are not made visible to the server before they are validated. When reconnected, the results are *validated*. If validation is confirmed, the MH commits its results into the FH. Otherwise, if validation is not confirmed, the conflict must be solved automatically or manually by the MH before it can be submitted again for validation.

To guarantee data consistency, two kinds of transactions are introduced:

- *First-Class Transactions* are not executed on partitioned data. They are guaranteed to be serializable with all committed transactions.
- *Second-Class Transactions* are executed on partitioned data. They are locally serializable with the other local second class transactions.

If second-class transactions are serializable with all other committed transactions on the server, their results can be committed. Such transactions are called *global-serializable*. One of the criteria for validation is hence the global-serializability of second-class transactions. If validation fails, then one of the following four actions can take place:

- the transaction re-executes,
- an application-specific resolver is started,
- a user notification is sent, or
- the transaction is aborted.

To guarantee isolation, the *Global Certification Order* criterion may be selected for second-class transactions. This criterion ensures that a second-class transaction is serializable not only with but also after all the committed transactions.

Transaction Mobility and Disconnections. Disconnected operation is supported via disk caching and transaction validation. Transaction mobility, however, is not supported.

Transaction Execution and Data Management. Transactions are performed entirely on the MH using a cached local copy of the server data. Serializability for first-class transactions is achieved by an optimistic replica approach [KS91].

Transaction Properties. The IOT model represents a flat transaction model. Hence all operations are "closed" and "vital-dependent" with respect to the running transaction. As the name of the model indicates, only the isolation criterion is satisfied.

3.3 Multi-Database System Transaction Processing Manager (MDSTPM)

Description. The MDSTPM approach of [YZ94b, YZ94a] sketches an architecture for submitting transactions from MHs and allowing MHs to be disconnected while the transactions are executed on the underlying multi-database system.

An MH has a pre-assigned MSS to which the MH sends a *request* message. Once a transaction request has been submitted, the corresponding MSS schedules and coordinates the execution of the transaction on behalf of the MH. Messages are handled by the MSS in an asynchronous way so that the MH can disconnect. Both MSSs and FHs are expected to be able to perform data processing. When an MH reconnects, it retrieves the results of its computation request from the corresponding MSS. All communication between MH and MSS happens through message passing.

The MSS acts as a *Global Transaction Manager Coordinator* (GTMC). Depending on the data requested, a transaction may run on several MSSs. Such MSSs are denoted as *Global Transaction Manager Participants* (GTMP) (see Figure 3). The global transaction is terminated following the two-phase commit protocol.

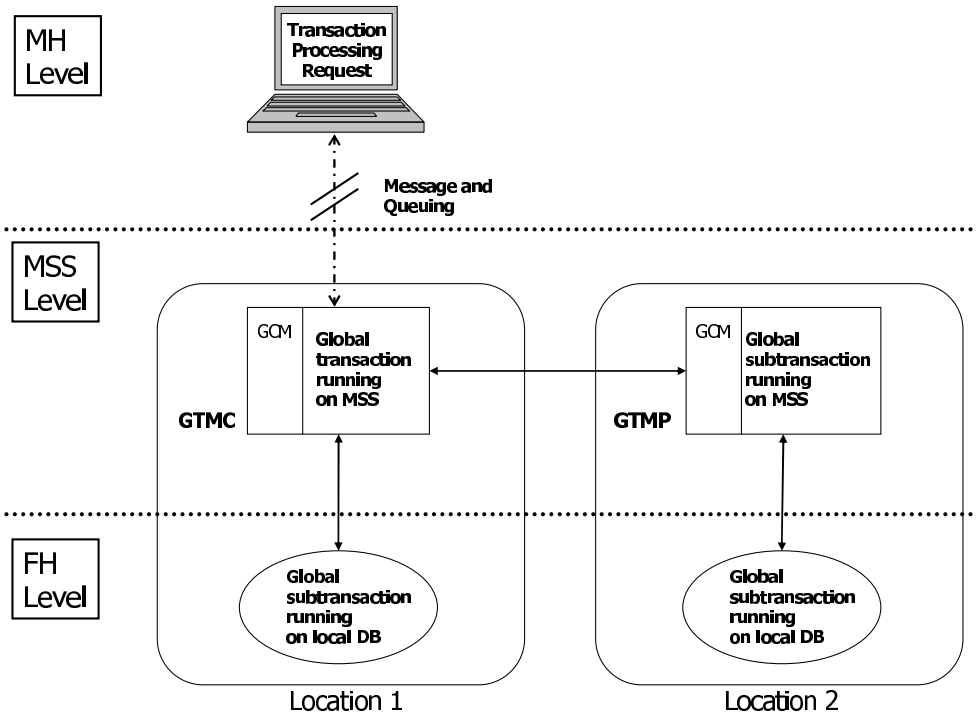

Figure 3: MDSTPM Architecture [YZ94b]

Transaction Mobility and Disconnections. This model does not explicitly considers transaction mobility. There is no description on whether the MH can perform a hand-off between MSS. The MH submits a computation to an assigned MSS and then disconnects. To retrieve the results, the MH must reconnect to the assigned MSS and upload its relevant data.

Transaction Execution and Data Management. Transactions are performed entirely on the MSS and FHs and it is up to the underlying multi-database system to guarantee the correctness of invoked operations. No data management techniques like data replication on the MH are discussed.

Transaction Properties. Since transactions are running completely on the underlying multi-database system, the properties of the mobile transaction completely relies on the transaction model that is provided by the multi-database system.

3.4 Weak-Strict Transactions (The "Clustering" Model)

Description. [PB94, PB95, Pit96, PB99] propose a mobile transaction model that enables transaction processing on an MH in the disconnected mode. To provide the MH with the capability to operate on data while disconnected, data is replicated locally on the MH via *clusters*. A cluster groups semantically related data together into units of consistency.

An MH can be considered as a cluster in its own right. After having performed disconnected operations on locally cached data, the MH must reconnect to reconcile its changes. When reconnected, the MH performs the usual operations that an always-connected device would be able to carry out. To be able to achieve such distinct modes of operation, two kinds of transactions are introduced by the clustering model:

- *Weak (or loose) transactions* read and write on a local copy during disconnections. They are committed in their associated cluster only (through a *local-commit* procedure) and are accepted by the rest of the clusters only after reconciliation (*global-commit*).
- *Strict transactions* read and write data on any cluster while the MH is connected.

Transaction Mobility and Disconnections. The model introduces the notion of *transaction migration* to support mobility aiming at improving performance by moving computation close to the MH. No details about the implementation are given in the paper.

The concept of weak transactions is the cornerstone to support transaction processing in the disconnected mode. As mentioned above, weak transaction may commit locally. Their results need to be reconciled later within the global clusters since they can conflict with the results of a strict transaction.

Transaction Execution and Data Management. Due to data replication, a transaction can take place on the MH when disconnected, or both on MH and FHs when connected. To deal with the frequent disconnections or even with fatal damages that can happen to an MH, the model makes use of a *transaction proxy*, namely a subtransaction that runs on the MSS and backups the transaction running on the MH.

Transaction Properties. To maintain atomicity in all clusters, weak transactions are committed globally if and only if they do not conflict with strict transaction updates. If weak transactions conflict, they must be compensated. Weak transactions are hence useful when handling private or seldom accessed data, or data that can be processed with an acceptable degree of imprecision such as in statistical or data-mining applications.

Isolation is ensured by a locking protocol especially introduced for weak transactions. Read and write operations can be weak or strict. Lock compatibility between strict and weak operations is checked using conflict tables which depend on the semantics of the application requirements.

3.5 Two-Tier Replication

Description. [GHOS96] proposes a data replication and transaction processing method devised for occasionally disconnected MHs. Every MH replicates two versions of the accessed data items:

- *Master Version*: The most recent value received from the FH. These are values that have not been yet processed by local transaction.
- *Tentative Version*: The most recent value due to local updates made by local transactions. These values are considered tentative and will be subject to validation upon reconnection.

After every reconnection, data on the MH are updated so they are consistent with the base version.

Two transaction types can run on MSS:

- *Base Transactions*: Run only on master data and can produce only master data. They are performed only by connected MHs.
- *Tentative Transactions*: Process data when the MH is disconnected. These transactions are local to the MH and create new tentative versions of the replicated data.

The two-tier replication scheme works as follows. MHs store a replica of the database on the FH. When disconnected, an MH accumulates tentative transactions that run against the tentative database stored at the MH. Tentative transactions are then re-run as base transactions when the MH reconnects to the fixed network. The base transactions generated by tentative transactions are checked for consistency by means of application-specific acceptance criteria and may fail when reprocessed. In case a tentative transaction fails, the originating MH and the person who generated it are informed about the failures and the reason why it failed. The tentative transactions are aborted and the tentative data on the MH are replaced by the original data of the master version.

Transaction Mobility and Disconnections. Tentative transactions can be executed and committed locally on disconnected MHs. Upon reconnection, those transactions are re-processed and validated on the FH. Transaction mobility is not considered in this approach.

Transaction Execution and Data Management. As described before, each MH replicates data to execute tentative transactions in the disconnected mode. Furthermore, each MH must be equipped with transaction management capabilities in order to correctly coordinate tentative transactions.

Transaction Properties. Base transactions are atomic because they process master data and use commit algorithms during connection. Tentative transactions commit locally. They can be aborted afterwards during the re-processing on the FH if they fail to meet their acceptance criteria. Isolation is enforced locally on the MH by means of locking mechanisms. Results of tentative transactions are made visible locally on the MH. Durability of tentative transactions is provided only after successful validation on the FH.

3.6 Kangaroo Transactions

Description. The *Kangaroo Transaction Model* [DHB97, DK99] incorporates the ability of transactions to migrate from one MSS to another as the MH moves through cells. This model builds on the concepts of the open nested transaction model [WS92] and the split transaction model [PKH88].

Each MSS is equipped with a *Data Access Agent* (DAA) to which the MHs connect to through a wireless link. A DAA serves as a mobile transaction manager. It keeps track of the transaction execution flow and coordinates the movement of transaction from one MSS to another.

When an MH requests a transaction, the corresponding DAA starts a *Kangaroo Transaction* (KT) and instantiates a subtransaction, called *Joey Transaction* (JT). A JT is local to the corresponding MSS (see Figure 4). It can nest one or more subtransactions. These subtransactions can be local or global transactions which are managed by the underlying multi-database system. When the MH moves to another cell, the control of the KT moves over to the DAA at the MSS controlling the new cell. During the hand-off procedure, the DAA at the new MSS starts a new JT. The previous JT is then committed independently of the new JT.

There are two different modes for processing a KT:

- *Compensating Mode*: The abort of a JT causes the entire KT to be undone by performing compensating transactions.
- *Split Mode*: The abort of a JT requires to the termination of the KT but not the undo of previously committed JTs. Currently running subtransactions are committed or aborted upon decision of the local DBMSs.

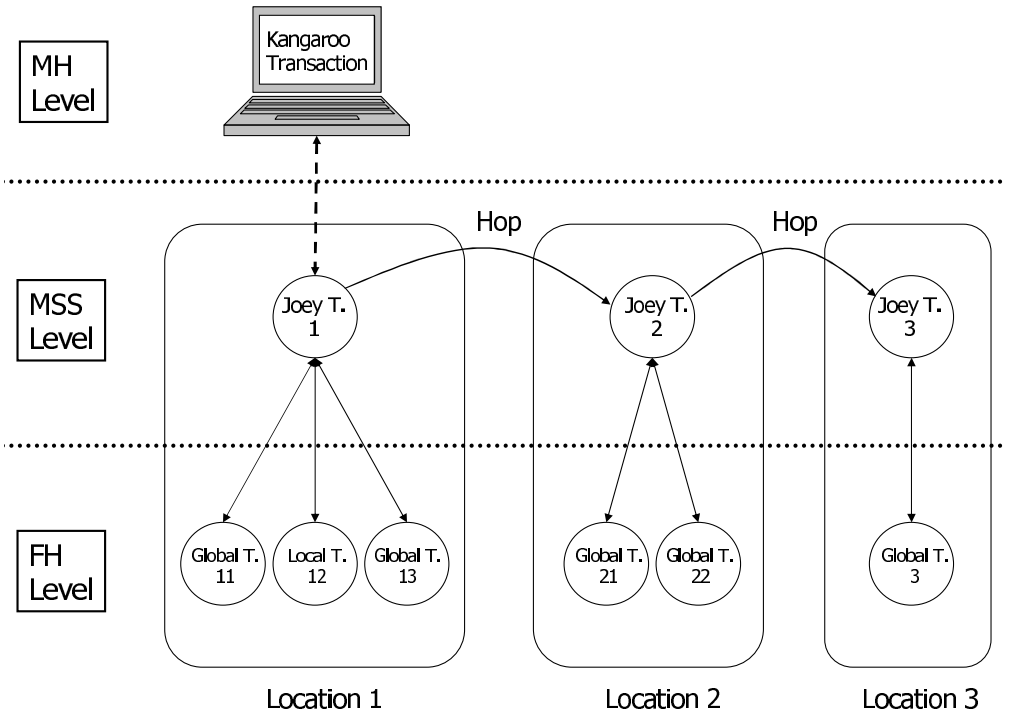

Figure 4: Kangaroo Transactions [DHB97]

Transaction Mobility and Disconnections. Transaction mobility is managed by splitting JTs and executing the new JT at the MSS level. Disconnections and replication techniques are disregarded.

Transaction Execution and Data Management. The MH just initiates the KT. The remaining executions are completely performed on the MSS and the FHs running the local DBMSs. Hence, this model does not except much computational resources on the MH.

Transaction Properties. The split operation mode guarantees neither atomicity nor isolation. The compensating mode at least ensure semantic atomicity by executing compensating transactions in case a JT was committed before the abortion of the KT.

3.7 Pro-Motion

Description. The Pro-Motion approach [WC97, WC99] focuses on ensuring data consistency under disconnections and mobility. It is centered on the concept of a *compact*. A compact is an abstraction that encapsulates data, methods (to operate on data), consistency rules (guaranteeing data consistency), obligations (i.e. deadlines), information on the current state of the compact, and interface methods to allow interaction between compacts and the MH. It serves as a replication and consistency unit that represents an agreement between the MH and the local DBMSs.

Figure 5 depicts the architecture of Pro-Motion. The *compact agent* is responsible for processing transactions on the MH. It handles disconnections and manages storage on the MH. The compact agent communicates with the *mobility manager* running at the MSS of the corresponding cell to retrieve the data needed for elaboration. The mobility manager interacts between the MH and the FH, in order to implement mobility transparency by performing automatic hand-offs when the MH moves from one cell to another. The mobility manager works together

with a *compact manager* on the FH to coordinate the flow of data among the components of the system. The compact manager creates a compact and transmits it back to the compact agent, while recording a copy of the compact in its *compact store*. When the MH receives the compact, the compact agent saves it in the *compact registry*, which is used to manage location and status of all compacts accessed by the MH.

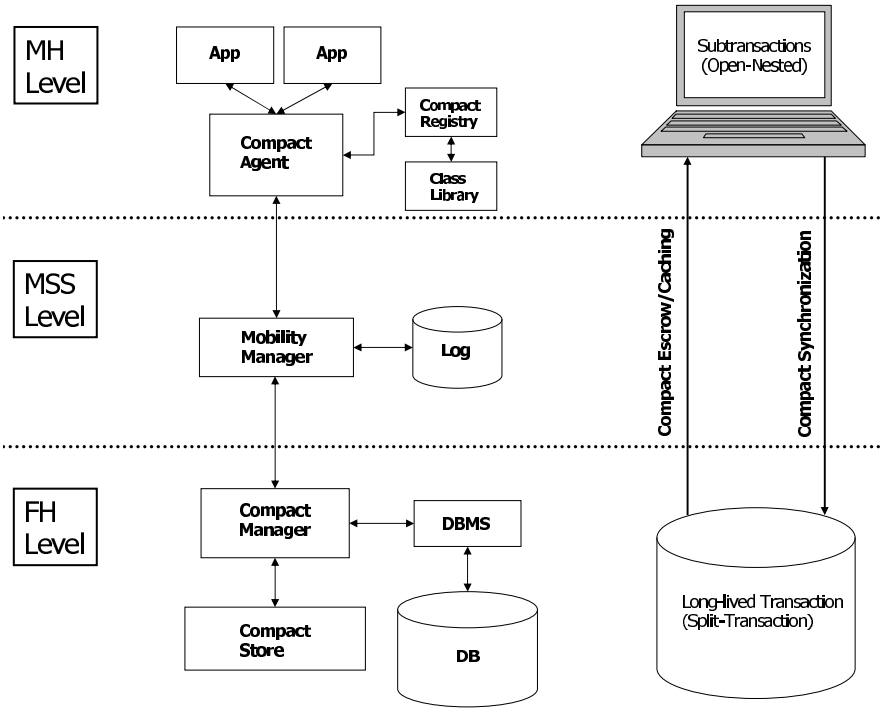

Figure 5: The Pro-Motion Architecture [WC97]

The compact manager is not aware of what the MH will do with the compact data. It only receives back updates of those data when the computation on the MH is ready to be reported. The MH must comply with the methods and obligations specified in the compact.

The transaction model of Pro-Motion relies on a combination of the open nested and the split transaction model. The local DBMS sees the compact manager as a client executing a single long-lived transaction. The subtransactions are executed on the MH. Every subtransaction on the MH can be a root transaction nesting other subtransactions running on the same MH. The compact agent is responsible for transaction management on the MH guaranteeing concurrency and recovery control.

Upon reconnection, the MH recognizes which compacts are ready to be submitted to the FH. On the FH, the compact manager then creates a split-transaction to handle the compact updates. This split-transaction is eventually committed by the compact manager and the results are made visible to all other transactions.

Transaction Mobility and Disconnections. When disconnection occurs, transactions can proceed on the MH if the compacts on which they operate have been correctly obtained and replicated in the local compact registry. A transaction operating in the local mode can locally commit, and thus make its results visible to other local transactions on the MH. The commit of a transactions that is not operating in the local mode is delayed until the MH reconnects.

Transaction mobility is not investigated in the paper although it is mentioned that the mobility manager can perform hand-offs between MSSs.

Transaction Execution and Data Management. Pro-Motion requires relative powerful MHs to be able to executed nested transaction computations on the MH. Transactions are executed locally on the MH even in the connected mode. Beside transaction processing capabilities, the MH must provide mechanisms for caching compacts. The flexibility of the notion of compact allows a wide range of semantics in sharing data. For instance, depending on information needs of the MH, which may change during lifetime, the compact agent can renegotiate the compact with the compact manager. Furthermore, a *lease* semantics can be adopted [GC89], and compacts can be sent to several MHs with an expiration date. Read is then allowed to all MHs, update is conditional upon the agreement of every other compact agent. If the expiration deadline is reached, the compact is sent back to the compact manager that regains full control over the data. Also other semantics-based caching techniques [WC95] can be adopted, (e.g. *escrow* [O'N86] and *fragmentable* items) to achieve an adaptable granularity of data, which can prove to be especially helpful in mobile computing. The hoarding of data can be tailored based on user-profile enabling the possibility to participate in broadcasting protocols.

Transaction Properties. Local commit on an MH is performed using a two-phase commit protocol. During reconnection of the MH, the locally committed compacts are validated and synchronized with the master copies by the compact manager. Semantic atomicity is achieved by compensating transactions which are attached to each local commit procedure. A compensating transaction is started if a locally committed transaction will not be able to commit on the FH. An abort will lead the compact to reload a logged consistent state.

To guarantee the correctness of a transaction execution, Pro-Motion uses a ten-level scale loosely based on the degree of isolation defined in ANSI SQL standard as in [BBG$^+$95]. Each method in the compact is assigned an *isolation level* and a *mode* that categorizes it as a READ or WRITE operation. Isolation is thus enforced by checking the level of every operation that participates in the transaction.

3.8 Toggle Transactions

Description. The *Toggle Transaction Management Technique* [DG98] relies on the Multi-level Transaction Model [Wei91]. It provides extensions to deal with predicted disconnections as well as with transaction migrations. Similar to the MDSTPM approach, a multi-database system environment is assumed where the local databases are independent from each other.

As depicted in Figure 6, a service interface layer is provided on top of each local DBMS. This layer provides the services that can be invoked by transactions. A *Site Transaction Manager* (STM) supervises the transactions executed on a node. A *global transaction* is started by the MH. It may include service calls that have to be executed on different nodes. A *Global Transaction Coordinator* (GTC) resides on each MSS. Its task is to handle disconnections and migrations of the MH as well as to submit the service request to the corresponding STM.

Disconnections are dealt with by introducing two new transaction states in addition to the usual ones (*active*, *committed*, and *aborted*):

- *Disconnected*: If a predicted disconnection occurs, then the transaction is disconnected. In this state, the transaction is allowed to continue later when reconnected.
- *Suspended*: If the disconnection stems from a failure, the transaction is suspended. However, suspended transactions will not be aborted until they hinder the execution of other transactions. In this way, needless abortions are minimized.

When a global transaction manifests its willingness to commit, the respective GTC initiates a commit algorithm based on a *Partial Global Serialization Graph*. If a cycle is detected, the algorithm attempts at breaking those cycles by aborting its non-vital subtransactions and also

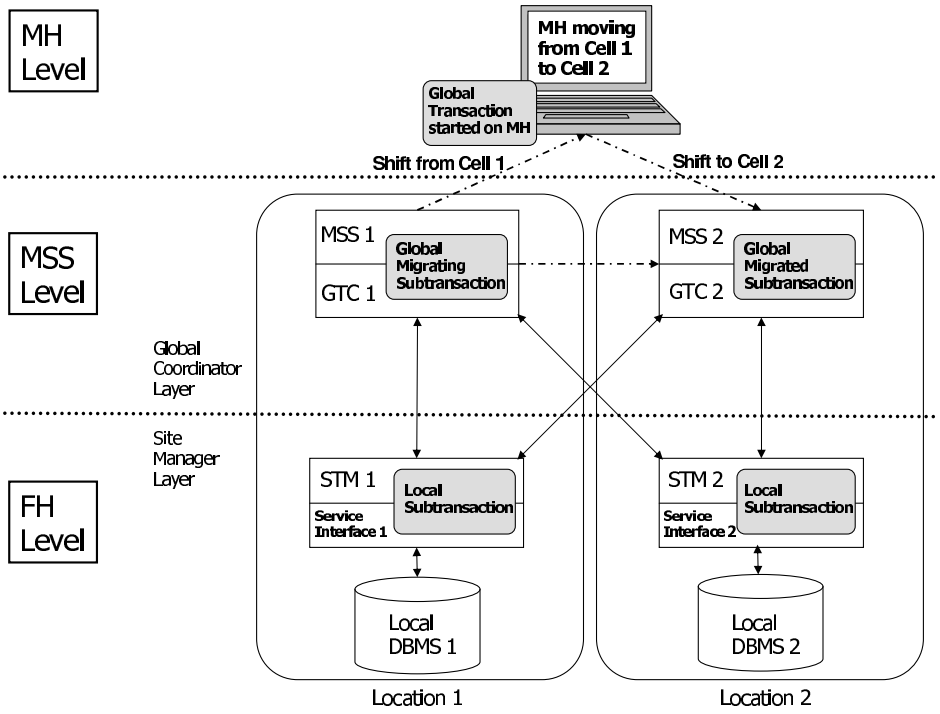

Figure 6: The Toggle Transaction Model Architecture [DG98]

suspended subtransactions of other global transactions. If a cycle cannot be resolved, then the global transaction is aborted. Otherwise, it is committed or *toggled*. *Toggling* means pre-committing the results of a transaction. A toggled transaction can start only non-vital subtransactions, and it is only aborted in case of a non-predicted disconnection.

Transaction Mobility and Disconnections. The GTM provides the ability to move transactions between different MSS by maintaining information on suitable entry tables.

The disconnected and suspended transaction states allows to treat disconnection as a possible legal state rather a failure.

Transaction Execution and Data Management. Most of the computation runs on the FHs and the MSSs, so the model is suitable for use in resource-limited devices. No data replication techniques are discussed in the paper.

Transaction Properties. Atomicity is based on the concept of semantic atomicity allowing non-vital subtransactions. Isolation is maintained on each node using the ticket method [GRS91].

3.9 Moflex Transactions

Description. The *Moflex* Transaction Model [KK00] generalizes the Flex Transaction Model [ELLR90] to support location-dependent subtransactions in heterogeneous multi-database environments. It supports hand-offs between MSS when the MH moves from one cell to another.

The MH creates a Moflex transaction and submits it to the *Mobile Transaction Manager* (MTM) residing on the corresponding MSS. The MTM sends the steps of the Moflex transaction to the respective local database systems and coordinates their global commitment (see Figure 7). The

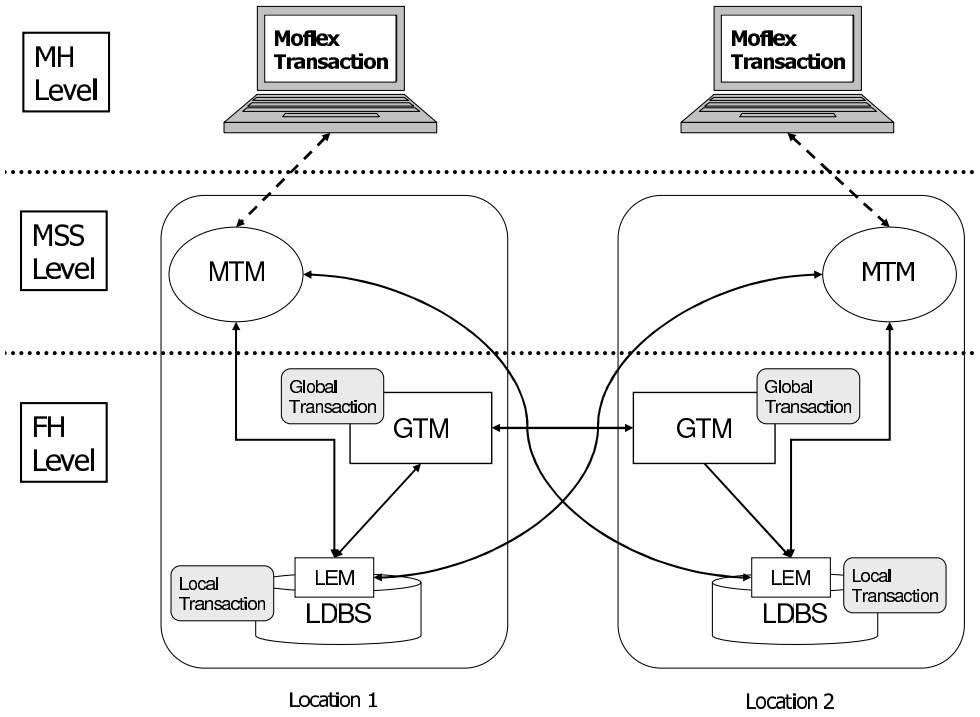

Figure 7: The Moflex Architecture [KK00]

Local Execution Monitor (LEM) acts as an interface between MTMs and LDBSs in order to detect global conflicts.

A Moflex transaction has a built-in set of *dependencies*, *acceptable goals* and *rules*. The set of dependencies considers:

- *Success*: A subtransaction is executed only if a related subtransaction successfully completes.
- *Failure*: A subtransaction is executed only if a related subtransaction fails.
- *External conditions*: A subtransaction is executed only if an external condition is verified. An external condition can concern time, cost, and/or location.

The set of acceptable goals is the set of the final subtransaction execution states that the user considers proper for his application. There are two set of rules closely related to each other. *Hand-Over Control* rules determine the subtransaction execution policy when the MH changes from a cell to another. Depending on whether or not the subtransaction is compensatable and location-dependent, four different execution policies can take place, as dictated by Table 1:

	Compensatable	**Non-Compensatable**
Location-Independent	SplitResume	Continue
Location-Dependent	Restart, SplitRestart	Restart

Table 1: Hand-Over Control Rules [KK00]

- *SplitResume*: The subtransaction is split. The already executed portion of the subtransaction is committed. The remaining execution is continued as a new subtransaction on the new MSS.

- *Restart*: This is the default operation. The subtransaction is restarted on the new MSS.
- *SplitRestart*: The executed portion of the subtransaction is committed on the old MSS and the subtransaction is restarted as a whole on the new MSS.
- *Continue*: The execution continues on the new MSS without interruption.

To be able to combine the executions of the split subtransactions, a set of acceptable join rules is defined. It is up to the user to define whether the concatenated partial results of two split subtransactions are valid and computationally equivalent to the original subtransaction if it had executed without a split. If the users approves, then the results are committed.

Transaction Mobility and Disconnections. The possibility to split a transaction from one MSS to another gives the model the possibility to let an MH switch between different locations without having to interrupt its computations.

Transaction Execution and Data Management. Transactions take place on MH as well as on FHs but tailored to run mostly on FHs so to give the model suitability to be implemented on resource-limited devices like palm-tops. No data replication technique is considered.

Transaction Properties. Semantic atomicity is enforced by the use of compensating transactions. Isolation of Moflex transactions is based on that of the flex transaction model.

3.10 Summary of Transaction Model Characteristics

Table 2 summarizes the characteristics of the (sub)transactions supported in the various approaches types. Since all transaction models support subtractions that are vital, dependent, non-compensatable, and non-substitutable, these characteristics are not denoted in this table.

We see that only two models, Pro-Motion and Moflex, provide a broader range of transaction properties that grant greater flexibility and power to transaction processing than the other models.[1]

Transaction Models	Open	Closed	Non-Vital	Indep.	Subst.	Comp.	Temporal
Reporting/Co†	✓	✓	—	—	✓	✓	—
Isolation-Only	*	*	*	*	—	—	—
MDSTPM	**	**	**	**	—	—	—
Weak/Strict	**	**	**	**	—	✓	—
Two-Tier Replication	*	*	*	*	—	—	—
Kangaroo	✓	—	—	—	—	✓	—
Pro-Motion	✓	✓	—	—	✓	✓	✓
Toggle	✓	—	✓	—	—	—	—
Moflex	✓	✓	✓	—	✓	✓	✓

Table 2: Overview of Mobile Transaction Characteristics

Table 3 [2] outlines the compliance with the requirements for mobile computing for the same transaction models.

[1] †Non-compensatable, Reporting and Co-transactions are closed. * says that the approach relies on the flat transaction model; ** stands for "it depends on the transaction model used in the underlying multi-database system".

[2] *L* stands for "low resource usage", meaning that the model is suitable for low-resource devices like palm-tops, while *H* stands for "high resource usage", meaning that the model would only be usable on laptop computers with substantial computational capabilities (including transaction processing).

Transaction Models	Mobility	Disconnections	MH Load	MT Execution	Profiling
Reporting/Co	✓	—	H	MH/MSS	—
Isolation-Only	—	✓	H	MH	—
MDSTPM	—	✓	L	MSS	—
Weak/Strict	—	✓	H	MH	✓
Two-Tier Replication	—	✓	H	MH	—
Kangaroo	✓	—	L	MSS	—
Pro-Motion	(✓)	✓	H	MH	✓
Toggle	✓	(✓)	L	MSS	—
Moflex	✓	—	L	MSS	—

Table 3: Overview of Mobility Characteristics

By joining the results from both Tables, it appears that Pro-Motion is, in principle, one of the more complete frameworks for mobile transaction processing between all the models considered. Apart from mobility and disconnection support, it indeed provides user-profiling capabilities that other models fail to even consider. Pro-Motion and Moflex, if the latter was extended with some user-profiling support functions, could therefore be good candidates for further research and development. Pro-Motion needs MHs with good processing capabilities: compacts are collected and processed locally by the MH so that a large part of the computation runs on the mobile device. Conversely, Moflex is suitable to be implemented on MHs with low computing resources: most of the computation indeed runs on the FHs.

3.11 Further Approaches

In the recent years, further methods and protocols have been developed for mobile transactional processing. In the following, we briefly sketch some of these approaches which contain some interesting issues which were not considered so far.

Timeout-based. [Kum00] present a non-blocking commit protocol based on timeouts as a way of minimizing wireless message costs and grant decision-making capabilities to MHs and FHs. MHs are assumed to be always connected. Hence, no disconnection handling takes place. Hand-off procedures are supported.

A timeout is a duration within which a transaction is declared or assumed to be completed by the *transaction coordinator*. The coordinator is the MSS. A transaction admits subtransactions that can be processed on MHs or FHs. Two types of timeout are defined:

- *Execution timeout* (*Et*): The time value within which a subtransaction is completed. The coordinator will assume that the MH or FH will complete the subtransaction before the timeout.
- *Shipping timeout* (*St*): The time value within which the data resulting from a subtransaction execution are shipped and logged on the FH database.

A subtransaction can be aborted only before *Et* expires. After *Et* expires, the subtransaction can only be compensated. After the duration *Et+St*, data is available to the other subtransactions.

The coordinator decides to commit a transaction only if it has received the updates before the expiration of the timeout. If the coordinator does not receive updates from all members (MHs and FHs), it aborts the transaction. Since the procedure is limited by a timeout, the coordinator will never enter into an infinite wait.

Finding a correct value for a timeout is difficult although a good estimate can be reached if an appropriate number of system characteristics are considered.

Alternating Client Host. [BZSH00] sketches a model that allows transactions to be started on one host and then later resumed on another host. The model assumes that MHs are always reachable by the communication system. Therefore, hand-offs are not taken into consideration.

The architecture is based on a classical *Client-Agent-Server* approach, where the agent is a middleware component residing on the FH. The agent routes all queries from a client (the MH) to the data resource (on the FH) and manages transactions.

Transaction processing is done by the MH. The agent keeps a replica of the transaction execution state. A failure of the MH does not lead to a rollback since the execution can be resumed on another MH using the information stored by the agent.

To reduce the communication workload, the copy saved on the agent is not updated at any change on the MH but only during checkpoints. After a failure a new client will resume the transaction from the last checkpoint.

Planned Disconnections. [HAE00] proposes a method to support operations of mobile teams using databases residing on team members' MHs without the support of any FH.

Transaction processing may run locally to every MH if disconnected. Locally, a concurrency control mechanism like 2PL is employed. A *pre-commit* record is propagated to other MHs through an epidemic-like exchange of messages. All MHs then perform the needed synchronization and communicate their actions again through epidemic messages.

Two or more MHs can also process transactions while connected to one another. It can happen that one MH needs to disconnect while the other MHs continue to process transactions. In this case, a *sign-off procedure*, which takes advantage of such *planned disconnection*, is used.

If a MH decides to disconnect, it grants to another MH the rights to act as his *proxy*. The proxy-MH represents that MH in the transaction processing that continues between the remaining MHs. The MH that has disconnected can now solely run read-only transactions. When this MH reconnects, it will directly call its proxy MH in order to retrieve the updates and get back into the processing network of MHs.

Pre-write. [MB01] aims at reducing the computational burden placed on the MH and the wireless network traffic between the MH and the FH. The model also increases data availability on the MH during its disconnected operations.

A *pre-write* operation is introduced. Differently from standard write operations, the pre-write does not perform any update, rather it makes visible the future state that the data will hold after the transaction will have committed on the FH. After a transaction has declared all its pre-writes, it pre-commits on the MH and makes the results visible to both MH and FH.

Transaction execution is divided between the MH and the FH. Part of the execution is sent to the FH where eventually all the time-consuming writes on the database will be performed. Delegating the database chores to the FH ensures less message passing and reduces computation on the MH.

4 Overview of Commercial Approaches

This section gives an overview of the current commercial mobile database solutions:

- IBM DB2 Everyplace,
- Informix Cloudscape,
- Microsoft SQL Server CE,
- Oracle Lite,

- Sybase Anywhere.

We will briefly describe the above products and we will discuss how transactions are performed on the database running on the MH. A description of data management, synchronization and conflict resolution will follow.

Commercially available mobile databases employ nearly the same data replication architecture: one or more *remote* databases on the MHs have replicas of the *master* database stored in the FH; flat transactions can be performed on both remote and master databases. Changes are captured locally to each database. When reconnected to the fixed network, the MH may initiate a synchronization that reconciles the copies of the databases by creating a unique consistent database state on remote and master databases.

Synchronization can be performed basically in two ways:

- *Session-based*: An MH database directly connects to the FH database. During connection, the synchronization takes place. Only when synchronization is complete, the MH and the FH can disconnect. When disconnecting, both MH and FH have a consistent database state. This approach is also referred to as *connection-based* synchronization.

- *Message-based*: A messaging system (such as e-mail) transfers the information needed to perform the reconciliation. Synchronization is performed when the MH and the FH are disconnected. This strategy is also known as *store-and-forward* or *file-based* synchronization.

4.1 An Excursion: SyncML and HotSync

Due to the importance synchronization in mobile computing, and especially on palm-like devices, an industry consortium of several major software firms has defined an industry-standard synchronization protocol called *SyncML* [Syn00]. The goal of this protocol is to provide cross-format and cross-platform synchronization capabilities between MHs and FHs. SyncML works with a standard set of messages represented as XML documents. SyncML defines seven different synchronization modes:

- *Two-way Sync*: MH and the FH exchange information about modified data in these devices. The MH sends the modifications first.

- *Slow sync*: All items are compared with each other on a field-by-field basis. In practice, this means that the MH sends all its data to the FH and the FH does the sync analysis (field-by-field).

- *One-way sync from MH only*: MH sends its modifications to the FH but the FH does not send its modifications back to the MH.

- *Refresh sync from MH only*: MH sends all its data from a database to the FH (i.e., exports). The FH is expected to replace all data in the target database with the data sent by the MH.

- *One-way sync from FH only*: MH gets all modifications from the FH but the MH does not send its modifications to the FH.

- *Refresh sync from FH only*: FH sends all its data from a database to the MH. The MH is expected to replace all data in the target database with the data sent by the FH.

- *FH Alerted Sync*: FH alerts the MH to perform sync. That is, the FH informs the MH to starts a specific type of sync with the FH.

Palm's *HotSync* [CFZ01a] is an example of a widely used synchronization protocol (that shares a lot of ideas with SyncML). HotSync supports a two-way synchronization that enables updates to be performed on the MH and the FH. Every record of data on the MH has a set of status bits that indicate if the record has been created, modified or deleted after last synchronization. The FH maintains a copy of each MH that connected to that FH to perform synchronization. The synchronization process is initiated by the MH. Two synchronization modes are distinguished:

- *Fast*, if the MH was last synchronized with the FH. The MH sends to the FH only the records that have changed from last synchronization depending on the value of the status bits. The FH updates its copy and sends updates to the MH.
- *Slow*, if the last synchronization time of the MH and FH does not match. In this case, the synchronization software performs a *field by field* comparison to capture the changes.

Depending on the status bit values, HotSync determines which actions to perform (see Table 4).

Status on MH	**Status on FH**	**Action**
Created	Not Present	Send to FH
Not Present	Created	Send to MH
Deleted	No Change	Delete from FH
Deleted	Updated	Send to MH
No Change	Updated	Send to MH
Updated	Updated	Conflict Resolution

Table 4: *HotSync* Logic

When updates are in conflict, *conflict resolution* is performed. For instance, a row in a database table, present in the master and some remote databases, could be modified by two different users, one working on its MH copy, another working on the master copy residing on the FH. The two users would request a modification to the master database, hence causing a dilemma: which of the two updates is the one that should be saved to the FH and propagated to the other database replicas?

Several conflict resolution techniques can be devised. For instance, *Intellisync for PalmPilot* by Pumatech (www.pumatech.com) implements five options:

- Add all conflicting items. This option creates whole new records on both the MH and the FH for the conflicting fields.
- Ignore all conflicting items. This option ignores any difference between fields in the MH and the FH.
- A user notification is sent.
- MH wins and overrides data on the FH.
- FH wins and overrides data on the MH.

Further data management techniques have been proposed. *Data Dissemination* [Fra96] is the distribution of data from a set of producers to a large number of users. Dissemination can be performed through *publish and subscribe*, *notification* systems or by *broadcasting* information in a way that a number of devices can connect and download the portion of those data that are relevant to them. For MHs, this option is convenient because they could tune up to a cellular/radio transmission and download the data they need to refresh. The convenience stems from the fact that receiving is less energy-consuming than transmitting and transmission

bandwidth from FHs to MHs is generally higher than from MHs to FHs (the aforementioned communication asymmetry). This way of sending information from a small number of FHs to a large number of client MHs is often referred to as *push-based*. *Broadcast Disks* [AAFZ95] have been proposed as a way of efficiently pushing information to MHs.

Alongside with the push approach, also a *pull* approach can be considered. For instance, a push could be used for data that are not needed urgently on the MH. In case of more important ("hot") data, the MH could transmit a request and be satisfied by the FH responsible for data broadcasting, hence "pulling" the data it needs. This is a hybrid solution called *Interleaved Push and Pull* [AAFZ95].

Another advancement in the field of data management is *Data Recharging* [CFZ01b]. The idea is to combine synchronization with data dissemination. The process is similar to battery power recharging when notebook plugs are connected to the electrical power outlet. Likewise, by connecting the MH to a network plug where data is streaming, information can be uploaded to the MH. The more the MH maintains its connection to the network, the better the data gets, until all information is uploaded. In data recharging, user profiling is an important issue since only the information that is relevant to the user should be "charged". Different mobile users might have different data needs. For instance, a businessman may need contact information updates, restaurant and hotel pricing guides for his travel destinations while students would need information about lecture notes and exams date. In contrast to simple database queries, data recharging techniques must cope with partial data recharges that can occur if the MH has not been connected to the data source long enough.

4.2 IBM DB2 Everyplace

Description. The IBM DB2 solution [IBM00, IBM01] for mobile computing consists of *DB2 Everyplace*, a relational database residing on the MH and *DB2 Everyplace Sync Server*, a middle-tier component that handles a two-way data synchronization between the MH and FH database under a message-based protocol. Synchronization is based on SyncML.

Transaction Mobility and Disconnections. Transactions are executed only on the MH during disconnections.

Transaction Execution and Data Management. *DataPropagator* is the software component that manages data replication. Data is replicated on the MH and updates are propagated through the *publish and subscribe* technique, so MHs can only synchronize the subset of data and files which they are subscribed to.

Replication is asynchronous. It is possible to control how frequently the changes are applied to the target by specifying time intervals, events, or both. But for mobile environments that have occasionally connected clients, data is replicated *on demand*.

It is possible to replicate entire tables, subsets of columns, subsets of rows, views, joins and unions. Replication can be achieved through *refresh propagation*, when all data is sent to subscribers, or *incremental propagation*, when only changes are propagated to the subscribing MHs.

The upper half of Figure 8 depicts the data synchronization flow when the MH database submit changes to the FH database. After a change has been done on the MH database, the user selects the synchronization from the device display. Synchronization is indeed decoupled from the transactions on the MH because it is explicitly initiated by the user. The synchronization request, holding the data to be synchronized, is saved in an *input queue* on the FH, waiting for a reply from the FH database. After the request is acknowledged, the request data is written in a *Staging Table*, where data waits for other updates to be completed. Then, data is copied in

the *Mirror Table*, where possible conflicts are resolved. Changes to the mirror table are recorded in the *DB2 Log* files. The *DataPropagator* capture program records all the changes in the mirror table and sends them to a *Change Data Table* where the FH database collects its updates. The *Administration Control* database performs checks about authentication and subscriptions.

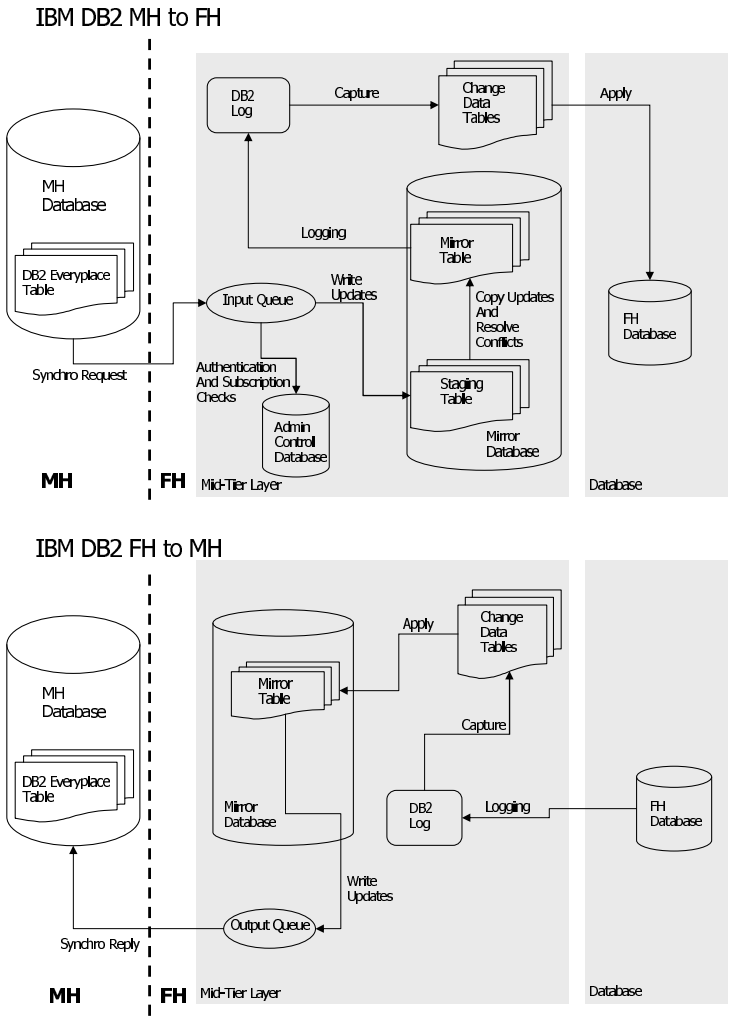

Figure 8: IBM DB2 Everyplace Synchronization [IBM01]

The reverse process is represented in the lower half Figure 8, where we see the data synchronization flow when the FH database submits its changes to the MH databases. We do not describe again all the steps that are straightforward to understand when looking at the figure. It is worth pointing out though that the synchronization server sends to a MH database only the data to which the MH is subscribed.

Conflicts are handled by the *Sync Server* running on the FH. This component logs and checks the version of each record in each table in the replication subscription for the FH and MH databases. The Sync Server can therefore determine whether or not a MH is trying to update a row based on an obsolete version of the data for that row. In such cases, the update is rejected. The conflict resolution is performed when data is staged to the mirror table.

Transaction Properties. DB2 Everyplace implements a subset of the standard SQL:1999, but the SAVEPOINT statement is not included. Also transaction nesting is not possible.

4.3 Informix Cloudscape

Description. The Informix solution [Inf99] for mobile computing consists of *Cloudscape*, a lightweight object-relational database residing on the MH and the FH, *Cloudconnector*, a server framework supporting connection residing on the FH, and *Cloudsync*, a component handling the synchronization between the MH and FH.

Transaction Mobility and Disconnections. Transactions are executed only on the MH during disconnections.

Transaction Execution and Data Management. Cloudscape uses the common architecture where the FH holds the master (also called *source*) database, and the MHs keep replicas (called *target*) of the master database. A source database can have many targets, while a target database has only one source database.

Replication can manage database objects such as complete tables or only selected columns and rows, views and indexes, schemata and Java code (JAR) files, stored prepared statements foreign keys, triggers, and user-defined aggregates.

Cloudscape uses a synchronization technique (see Figure 9) that sends data out from the FH database, and moves *business events* back from the MH databases. This technique is called by Informix "LUCID", meaning *Logic Up*, *Consistent Information Down*.

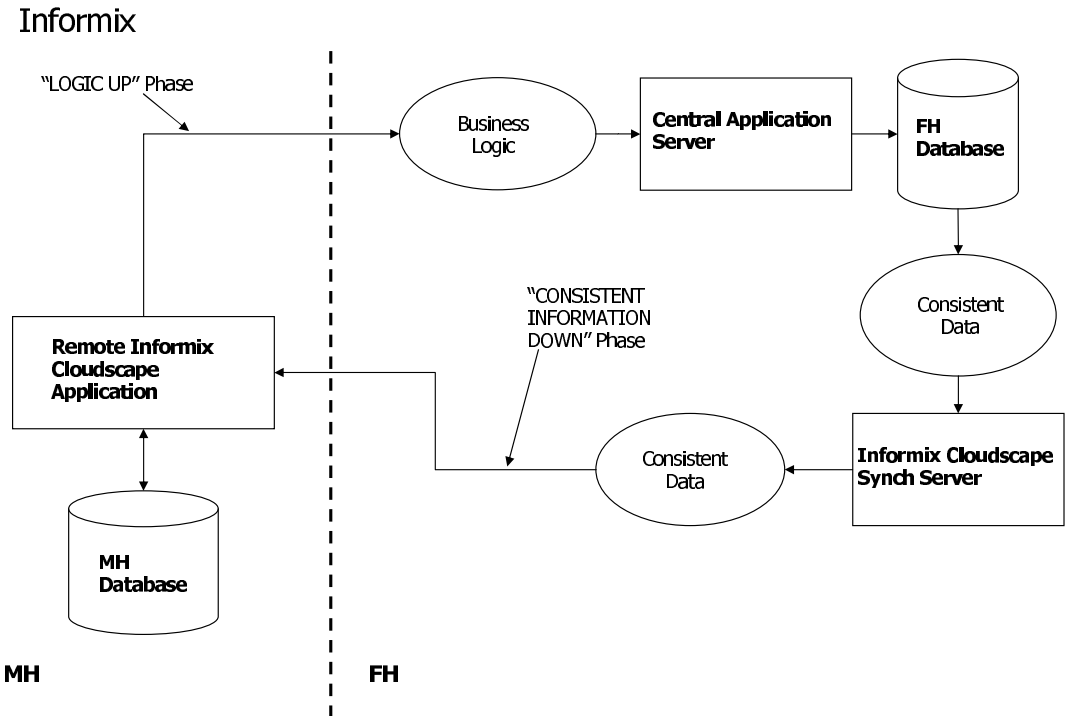

Figure 9: Informix Cloudscape Synchronization [Inf99]

The FH database lists the objects to synchronize in the *publication* using statements very similar to a view definition. The MH processes the data over its database using calls to database-hosted Java methods. Such methods are each called a *Work Unit*.

The *Logic Up* phase happens any time the MH application is connected to the network. At this time, the MH performs the synchronization with the FH by forwarding its Work Units to the FH. The FH database re-executes the Work Units performed at the MHs, although the code that executes at the FH may now take advantage of interfacing with other systems, such as ERP

packages, or perform additional integrity checks that were not feasible when the application was running on the disconnected MH database. After the Work Units have successfully re-run in the FH database, the FH publishes its results to the MH databases, thus initiating the *Consistent Information Down* phase. This phase updates all the MH databases. A single consistent database state is thus achieved.

Re-processing the logical operations of the Work Units on the FH database allows to run business rules. This permits to resolve possible conflicts between different Work Units submitted by several MHs. All conflicts are resolved at the source. All transactions running on the targets are conditional upon successful application at the source during the connection. If the transaction is accepted at the source, it becomes durable. If the transaction is rejected on the source database, it is rolled back on the target.

Transaction Properties. Its native interface is JDBC. It implements the SQL-92 standard plus some extensions [WC01]. No savepoints are available and neither transaction nesting is possible.

4.4 Microsoft SQL Server CE

Description. The *Microsoft SQL Server CE* (SSCE) [Mic01] is the database running on the MH; it is the mobile version of the *SQL Server* database running on the FH. Figure 10 shows its architecture. The *Client Agent* on the MH connects (through HTTP) to the *Internet Information Server* (IIS) residing on the FH. The *Server Agent* acts as the interface on the FH for the client running on the MH.

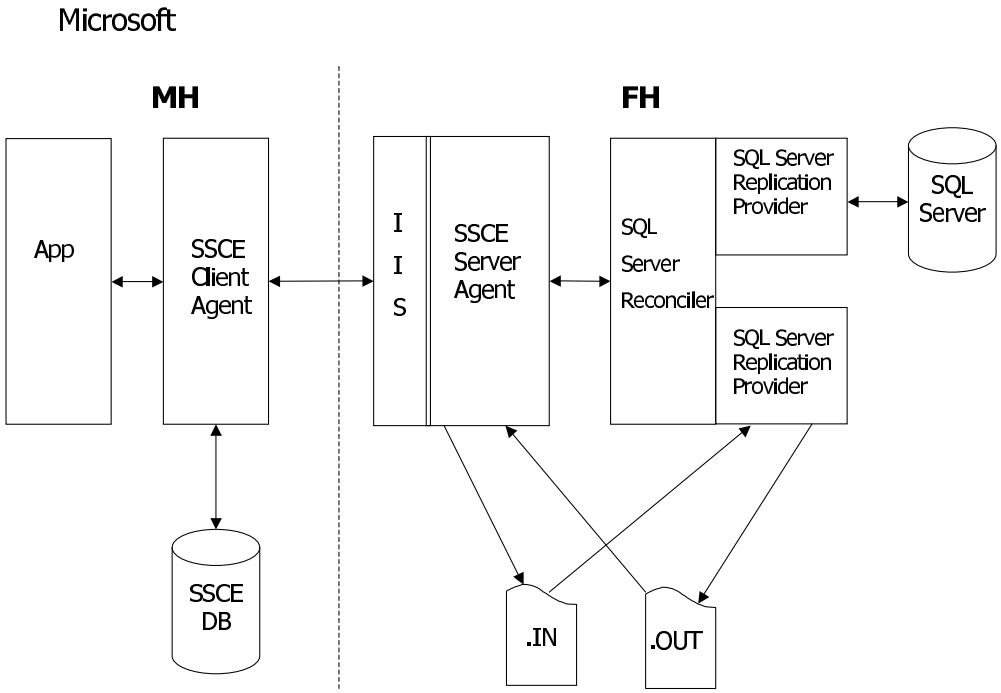

Figure 10: Microsoft SQL Server CE Synchronization [Mic01]

Transaction Mobility and Disconnections. Transactions are executed only on the MH during disconnections.

Transaction Execution and Data Management. Replication in SQL Server CE is based on Microsoft SQL Server 2000 merge replication.

SQL Server CE can replicate entire tables, subset of rows and columns, but not views and user-defined functions. The replication is always initiated and controlled by the Windows CE application.

Data replication and synchronization are achieved through a *publish and subscribe* model with a message-based communication protocol. The *publication* (namely a collection of articles where an article is a table enabled for replication), is created at the FH. An application on a MH subscribes to the publication through the *SQL Server CE Replication Object*. Updates from the subscribers are sent to the publisher that merges these updates with the updates from all the other subscribers. Eventually, the publisher propagates the updates back to the subscribers to achieve consistency in the MH replicated databases. Publications are tailored to specific groups of users, so a MH will receive only the updates it is subscribed to.

The *SQL Server CE Client Agent* is the interface that applications call to control replication. When synchronization occurs, the Client Agent extracts all modifications to data records and sends them to the Server Agent through HTTP. Conversely, when the Server Agent propagates data changes at the publishing FH database back to the subscribing MH, it is the Client Agent that applies these changes to the subscribing database.

When synchronization occurs, the *SQL Server CE Server Agent* creates a new input message file on the IIS server and writes the data modification requests sent by the Client Agent. When the input message file has been written, the Server Agent initiates the *SQL Server Reconciler* process and waits for it to complete.

The SQL Server Reconciler detects and resolves conflicts. Conflict resolution can be devised to behave with simple strategies. The default conflict resolver is priority-based. A subscriber can be set to have priorities. The reconciler then checks update priorities and choose the update with the highest priority. Other strategies can consider that the first change to propagate to the publisher software always wins the conflict. More complex conflict-resolution rules can be customized by the application programmer. In addition, an interactive resolver is also supplied.

SQL Server CE Replication Provider is invoked by the Server Reconciler when synchronization is performed. It reads the input message file to inform the Server Reconciler about changes made to the MH subscriber database that must be applied to the publication. Vice versa, the Server Reconciler informs the Server CE Replication Provider about changes made at the FH publishing database that must be applied to the subscription database on the MH. The SQL Server CE Replication Provider writes these changes to the output message file.

When the Server Reconciler process is completed, the Server Agent reads the output message file and transmits it back to the Client Agent on the MH.

Transaction Properties. SSCE allows the use of savepoints and nested transactions. Transactions can be nested up to a depth of five levels. Updates made within the nested transaction are not visible to the top-level transaction. Results become visible to the parent subtransaction after the nested subtransaction has committed. Changes are not visible outside the top-level transaction until that transaction commits. Transactions are closed and are executed sequentially.

4.5 Oracle Lite

Description *Oracle Lite* [Ora00] uses a client-server architecture, depicted in Figure 11, for both session and message based replication.

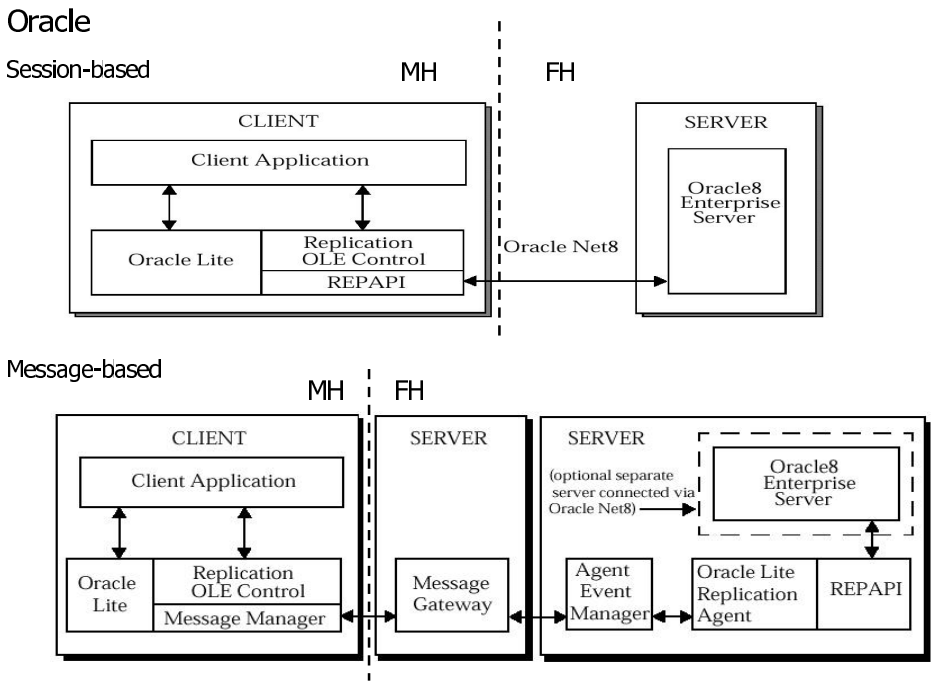

Figure 11: Oracle Lite Synchronization [Ora00]

Transaction Mobility and Disconnections. Transactions are executed only on the MH during disconnections.

Transaction Execution and Data Management. Data replicated on a MH database are copied from the master site database on the FH. The MH databases are called *snapshot sites*. Oracle Lite database can only work as a snapshot site, supporting two types of snapshots:

- *Read-only*: Used only for local queries so no changes can be done by local processing. Only changes on the FH database can be written to the snapshot. This is a *uni-directional* replication.
- *Updatable*: Used for both queries and updates. Changes to this kind of snapshot are propagated back to the FH database. Changes on the FH database can be written to the snapshot. This is a *two-way or bi-directional* replication.

A snapshot is *simple* if it does not have distinct or aggregate functions, GROUP BY or CONNECT BY clauses, sub-queries, joins, or set operations. Otherwise it is called *complex*. Updatable snapshots can only be simple.

The *snapshot refresh* process propagates updates to all the databases. It involves three steps:

1. Changes to updatable snapshots are sent to the master site.
2. The valid transactions are applied (in the same sequence as they were performed at the snapshot).
3. Snapshots are synchronized with the data pulled from the FH site.

Refreshes can be:

- *Fast*: When only the changes are transmitted (this option is only viable for simple snapshots).
- *Complete*: When the original snapshot is completely replaced by a new snapshot.
- *Optimum*: The refresh attempts to perform the fast option. If this is not possible (due for instance to a snapshot being complex), the refresh executes as complete.

During a refresh operation, the snapshot site enters the *Refresh-In-Progress* mode. When in this mode, attempts to update any updatable snapshots at that snapshot site are rejected. When the refresh operation completes, the snapshot site returns to the normal mode.

It is possible to replicate entire tables, subsets of columns, subsets of rows. For a more advanced data setting, it is possible to specify a WHERE clause to qualify the snapshots defining query, or a WHERE EXISTS clause to define a subquery.

The architecture that performs session-based replication consists of a component called *Oracle Net* to connect to the *Oracle Enterprise server*. Oracle Net is a remote data access software product that enables bi-directional data replication over LAN and dial-up networks. *REPAPI* is the replication engine. Message-based replication architecture requires a file transfer method between the client and the server. Apart from HTTP connection and disk file exchange, another method is used, called *Oracle Mobile Agents Replication*. Such component is a networking communications middleware software that enables bi-directional data replication over LAN, dial-up, and wireless networks. Client components include a *Message Manager*. The server components include a *Message Gateway* and *Agent Event Manager*.

Conflicts are detected by the master site during synchronization. The following types of conflicts can be handled:

- *Update*: When the snapshot old row values do not match the current FH row values.
- *Uniqueness*: Detected if a unique constraint is violated during an INSERT or UPDATE of the replicated row.
- *Delete*: When a row is deleted from an updatable snapshot, and the old values of the deleted row do not match the current values at the FH.

Transaction Properties. Oracle Lite has built-in Java components that enable support of savepoints but not nested transactions. Transactions are equivalent to closed, one-level, nested transactions.

4.6 Sybase Anywhere

Description. *Adaptive Server Anywhere* (ASA) [Syb00a] is a relational database suitable to be used on small devices. ASA is capable of both session and message-based synchronization.

Transaction Mobility and Disconnections. Transactions are executed only on the MH during disconnections.

Transaction Execution and Data Management. Sybase Anywhere provides three replication technologies:

- *SQL Remote*: Intended for a *two-way message-based* replication of transactions. It is designed for two-way replication involving a consolidated data server and large number of remote databases. Lag times for replication can be on the order of seconds, minutes, or hours depending on the implementation.

- *MobiLink*: Intended for a *two-way session-based* replication of data. It also supports non-Sybase databases. It is designed for replication of data between a central consolidated database and a large number of remote databases. At the end of each synchronization session, the databases are consistent.

- *Replication Server*: Intended for a *two-way connection-based* replication of transactions. It is suited to replicate data between a small number of enterprise databases connected by a high-speed network. Lag times can be as low as a few seconds, providing near real-time replication system.

The replication server is not suited for occasionally connected devices. On the contrary, MobiLink and SQL provide more flexible solutions in environments where the remote machines are mobile or are only occasionally connected. For these reasons, we will discuss only SQL Remote and MobiLink because they are more suited to mobile computing needs.

In both replication technologies, replicated data can be tables, subsets of rows and subsets of columns, joins, SQL queries.

SQL Remote (see Figure 12) uses a *publish and subscribe* process: synchronization data is saved in *publications* at the FH database, where the administrator subscribes the MH databases to the publications they need. The publications are sent to the relevant MHs, and the MHs, on their turn, send their synchronization data to the FH to be incorporated in the FH database.

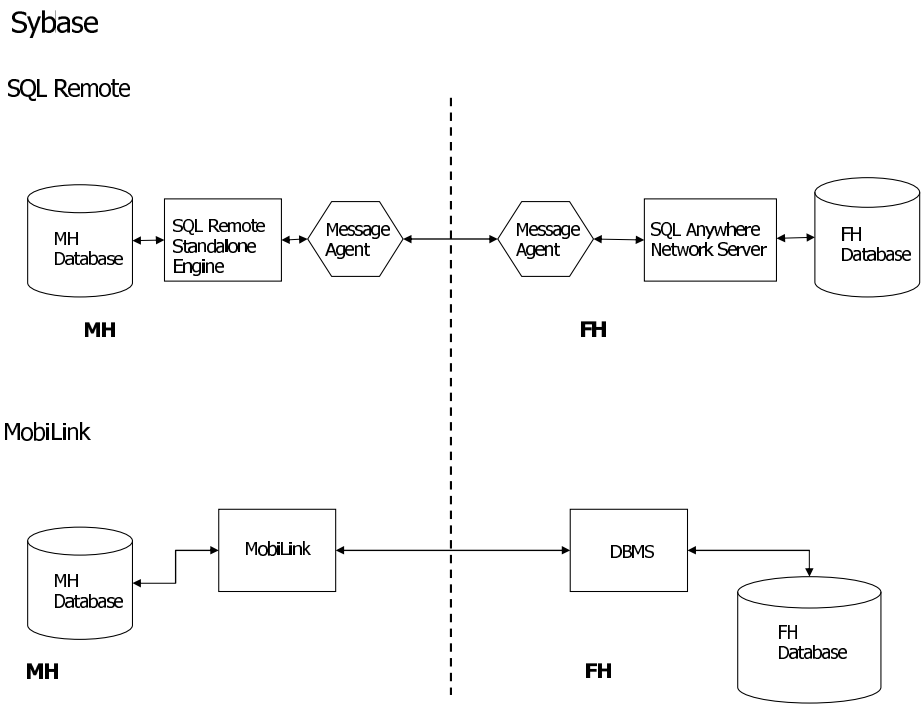

Figure 12: Sybase Anywhere Synchronization [Syb00a]

The SQL Remote synchronization process follows the next steps.

1. At MH and FH database, a *Message Agent* and a *transaction log* manage synchronization. The transaction log tracks all committed changes to data.

2. Periodically, the message agent on the FH database scans the transaction log and sends, by means of a messaging system, the relevant changes to MH databases that are subscribed to those publications.

3. The message agent at the MH accepts the messages sent from the FH and applies the transactions to its own database.

4. At any time, the MH can run the message agent to collect and send to the FH database the transactions made locally.

5. The message agent at the FH processes the messages received from the MH and applies the transactions to its own database.

Conflict resolution is performed at the FH through the use of conflict triggers.

Session-based synchronization is performed by the software component called MobiLink, which synchronizes data through four main steps (see Figure 12).

1. The MH application prepares and transmits to the FH a list of all rows that have been modified since the last synchronization.

2. The synchronization server, running on the FH, prepares and transmits data changes from the FH database to the MH application.

3. The MH application then sends a confirmation message to the synchronization server confirming that the changes have been made to the data. The transaction is recorded in the FH database.

4. The synchronization server sends the MH application a message confirming that synchronization is complete. The connection between the MH application and the FH is shut down.

In situations where data rarely change, MobiLink uses a *timestamp* technique to synchronize only data that have changed since the last synchronization. It uses a *snapshot* technique when a lot of changes are frequently carried out on data. In that case, the software synchronizes all the data in the database.

To resolve conflicts, business rules must be implemented to determine which data is correct. Conflict resolution is performed on the FH database, and then conflict resolution actions are replicated to the remote databases.

Transaction Properties. Transactions can use savepoints. All savepoints are released when a transaction ends. Transactions can be nested. Changes to the database are permanent only after the uppermost transaction is committed.

Transactions can run on remote servers. In such case, savepoints are not considered on the remote servers. Also in the case of nested transactions that involve remote servers, only the statements belonging to the top-level transaction are processed.

The software UltraLite [Syb00b] is a deployment component of the Adaptive Server Anywhere database for very small, mobile, and embedded devices like cell phones and personal organizers. Its footprint can be as small as 50 Kb, but the drawback is that it does not support savepoints and nested transactions.

4.7 Brief Summary

Table 5 briefly summarizes the commercial mobile databases from the point of view of the transaction models implemented. Only Microsoft SQL Server CE, Oracle Lite and Sybase Anywhere use savepoints, and only Microsoft SQL Server CE and Sybase Anywhere can run nested transactions. The trade-off is that their memory footprint tends to be large (e.g. around 1 Mb for Microsoft SQL Server CE).

Commercial Mobile DB	Savepoints	Transaction Nesting
IBM DB2 Everyplace	–	–
Informix Cloudscape	–	–
Microsoft SQL Server CE	✓	✓
Oracle Lite	✓	–
Sybase Anywhere	✓	✓

Table 5: Comparison of Commercial Approaches

5 Conclusions and Outlook

In this paper, we have surveyed both transaction models for mobile computing and current solutions provided by major vendors in the field of mobile databases. Regarding mobile transaction models, we gave an overview of the approaches that deal with transaction movements and disconnections. The presented mobile transaction models exploits a lot of the results of previous research in the field of transaction models and processing, while adapting them to mobile computing environments. The study of the approaches of current commercial mobile databases shows that little of the theoretical advances developed in the "academic" models have been established in the products. Advanced nested transaction models are still missing. Compensating and/or alternative transactions are also not available. Commercial products still rely on basic data replication and synchronization techniques. Only few of them provide the capability to use savepoints and restricted closed nested transactions. In some cases, even basic transactional properties are not respected.

We state that more study in the field is still needed. We know that the use of mobile devices will increase in the near future and that a lot of the work now carried out on fixed computers will shift to portable devices. Some guarantees for improved transaction correctness and efficient data management techniques will be needed when users will ask for more complex task to be performed on their mobile devices. Therefore, advances in mobile transaction processing are needed. For instance, a combination of concepts developed in the theoretical models outlined in Section 3 together with the ideas described in the Section 3.11, can form the basis upon which to build new proposals and produce prototypes to be tested in real world applications.

References

[AAFZ95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communication Environments. In M. J. Carey and D. A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA*, ACM SIGMOD Record, Vol. 24, No. 2, pages 199–210, ACM Press, June 1995.

[Acc01] Accenture. The Future of Wireless: Different than You Think, Bolder than You Imagine. Working Paper, June 2001.

[AK93] R. Alonso and H. F. Korth. Database Issues in Nomadic Computing. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C.*, ACM SIGMOD Record, Vol. 22, No. 2, pages 388–392, ACM Press, June 1993.

[Bar99] D. Barbará. Mobile Computing and Databases – A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, January/February 1999.

[BBG+95] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A Critique of ANSI SQL Isolation Levels. In M. J. Carey and D. A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA*, ACM SIGMOD Record, Vol. 24, No. 2, pages 1–10, ACM Press, June 1995.

[BGS92] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, October 1992.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[Bur97] M. Buretta. *Data Replication*. John Wiley & Sons, Inc., 1997.

[BZSH00] S. Buchholz, T. Ziegert, A. Schill, and A. Held. Transaction Processing in a Mobile Computing Environment with Alternating Client Hosts. In *RIDE 2000, Proc. of the 10th Int. Workshop on Research Issues in Data Engineering: Middleware for Mobile Business Applications and E-Commerce, February 27–28, 2000, San Diego, CA, USA*, pages 1–8, IEEE Computer Society Press, 2000.

[CFHR01] A. Coratella, M. Felder, R. Hirsch, and E. Rodriguez. A Framework for Analyzing Mobile Transaction Models. *Journal of Database Management*, 12(3), 2001.

[CFZ01a] M. Cherniack, M. Franklin, and S. Zdonik. Data Management for Pervasive Computing. In G. Mecca and P. Valduriez, editors, *Tutorials of the 27th Int. Conf. on Very Large Data Bases, VLDB'01, September 11–14, 2001, Rome, Italy*, pages 71–140, 2001.

[CFZ01b] M. Cherniack, M. J. Franklin, and S. Zdonik. Expressing User Profiles for Data Recharging. In *IEEE Personal Communications: Special Issue on Pervasive Computing*, pages 6–13, Aug. 2001.

[Chr93] P. K. Chrysanthis. Transaction Processing in a Mobile Computing Environment. In *Proc. of the IEEE Worskhop on Advances in Parallel and Distributed Systems*, pages 77–82, 1993.

[CR90] P. K. Chrysanthis and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ*, ACM SIGMOD Record, Vol. 19, No. 2, pages 194–203. ACM Press, New York, June 1990.

[DG98] R. A. Dirckze and L. Gruenwald. A Toggle Transaction Management Technique for Mobile Multidatabases. In G. Gardarin, J. French, N. Pissinou, K. Makki, and L. Bougamin, editors, *Proc. of the 7th ACM CIKM Int. Conf. on Information and Knowledge Management, November 3–7, 1998, Bethesda, Maryland, USA*, pages 371–377. ACM Press, New York, 1998.

[DHB97] M. H. Dunham, A. Helal, and S. Balakrishnan. A Mobile Transaction Model That Captures Both the Data and Movement Behavior. *Mobile Networks and Applications (MONET)*, 2(2):149–162, 1997.

[DK99] M. H. Dunham and V. Kumar. Impact of Mobility on Transaction Management. In *Proc. of the ACM Int. Workshop on Data Engineering for Wireless and Mobile Access, August 20, 1999, Seattle, WA, USA*, pages 14–21, ACM Press, 1999.

[ELLR90] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, editors, *Proc. of the 16th Int. Conf. on Very Large Data Bases, VLDB'90, Brisbane, Australia, August 13–16, 1990*, pages 507–518. Morgan Kaufmann Publishers, Palo Alto, CA, 1990.

[Fra96] M. J. Franklin, editor. *IEEE Technical Committee on Data Engineering: Special Issue on Data Dissemination*, September 1996.

[Fra01] M. J. Franklin. Challenges in Ubiquitous Data Management. In R. Wilhelm, editor, *Informatics – 10 Years Back 10 Years Ahead*, Lecture Notes in Computer Science, Vol. 2000, pages 24–33. Springer-Verlag, Berlin, 2001.

[GC89] C. G. Gray and D. R. Cheriton. Leases: an Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proc. of the 12th ACM Symposium on Operating Systems Principles, ACM Operating Systems Review*, Vol. 23, No. 5, pages 202–210, 1989.

[GHOS96] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In H. V. Jagadish and I. S. Mumick, editors, *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada*, ACM SIGMOD Record, Vol. 25, No. 2, pages 173–182, ACM Press, June 1996.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Gra78] J. Gray. Notes on Data Base Operating Systems. In R. Bayer, R. M. Graham, and G. Seegmüller, editors, *Operating Systems, An Advanced Course*, Lecture Notes in Computer Science, Vol. 60, pages 393–481. Springer-Verlag, Berlin, 1978.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On Serializability of Multidatabase Transactions through Forced Local Conflicts. In N. Cercone and M. Tsuchiya, editors, *Proc. of the 7th IEEE Int. Conf. on Data Engineering, ICDE'91, April 8–12, 1991, Kobe, Japan*, pages 314–323. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[GSC96] M. Garcia-Solaco, F. Saltor, and M. Castellanos. Semantic Heterogeneity in Multidatabase Systems. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, chapter 5, pages 129–202, Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[HAE00] J. Holliday, D. Agrawal, and A. El Abbadi. Exploiting Planned Disconnections in Mobile Environments. In *RIDE 2000, Proc. of the 10th Int. Workshop on Research Issues in Data Engineering: Middleware for Mobile Business Applications and E-Commerce, February 27–28, 2000, San Diego, CA, USA*, pages 25–30, IEEE Computer Society Press, 2000.

[IB94] T. Imielinski and B. R. Badrinath. Mobile Wireless Computing: Challenges in Data Management. *Communications of the ACM*, 37(10):18–28, 1994.

[IBM00] IBM Corporation. *IBM DB2 Universal Database: SQL Reference, Version 7*, 2000.

[IBM01] IBM. *IBM DB2 Everyplace Sync Server Administration Guide Version 7 Release 2*. IBM Corp., 3 edition, 2001.

[Inf99] Informix Software, Inc. *Informix Cloudscape Application Synchronization*, November 1999.

[JHE99] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2):117–157, June 1999.

[KCGS95] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems*, chapter 26, pages 521–550, ACM Press, New York, NJ, 1995.

[KK00] K.-I. Ku and Y.-S Kim. Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems. In *RIDE 2000, Proc. of the 10th Int. Workshop on Research Issues in Data Engineering: Middleware for Mobile Business Applications and E-Commerce, February 27–28, 2000, San Diego, CA, USA*, pages 39–46, IEEE Computer Society Press, 2000.

[KS91] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Proc. of the ACM Symposium on Operating System Principles, October 13–16, 1991, Asilomar, Pacific Groove, CA*, pages 213–225, ACM Press, 1991.

[Kum00] V. Kumar. A Timeout-Based Mobile Transaction Commitment Protocol. In J. Stuller, J. Pokorny, B. Thalheim, and Y. Masunaga, editors, *Current Issues in Databases and Information Systems, Proc. of the ADBIS-DASFAA 2000 Symposium, Prague, Czech Republic, September 5–8, 2000*, Lecture Notes in Computer Science, Vol. 1884, pages 331–338. Springer-Verlag, Berlin, 2000.

[LS94] Q. Lu and M. Satyanarayanan. Isolation-Only Transactions for Mobile Computing. *ACM Operating Systems Review*, 28(2):81–87, 1994.

[Mad98] S. K. Madria. Transaction Models for Mobile Computing. In A. L. Ananda, H. K. Pung, and W. Wang, editors, *Networks'98, Proc. of the 6th IEEE Singapore Int. Conf. on Networks, SICON'98, June 30 – July 3, 1998*, World Scientific Publishing, 1998.

[MB01] S. K. Madria and B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing. *Distributed and Parallel Databases: An International Journal*, 2001.

[Mic01] Microsoft Corp. *http://msdn.microsoft.com/library/*, 2001.

[Mos85] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, Cambridge, MA, 1985.

[MRKS92] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A Transaction Model for Multidatabase Systems. In *Proc. of the 12th Int. Conf. on Distributed Computing Systems (ICDCS), June, Yokohama, Japan*, pages 56–63. IEEE Computer Society Press, Washington, DC, 1992.

[O'N86] P. O'Neil. The Escrow Transactional Method. *ACM Transactions on Database Systems*, 11(4):405–430, December 1986.

[Ora00] Oracle Corporation. *Oracle8i Lite - Oracle Lite Replication Guide*, 2000.

[PB94] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In N. R. Adam, B. Bhargava, and Y. Yesha, editors, *Proc. of the 3rd Int. Conf. on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, November 29 – December 2, 1994*, pages 371–378, ACM Press, 1994.

[PB95] E. Pitoura and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. In *Proc. of the 15th Int. Conf. on Distributed Computing Systems*, pages 404–413, IEEE Computer Society Press, 1995.

[PB99] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(6):896–915, November/December 1999.

[Pit96] E. Pitoura. A Replication Schema to Support Weak Connectivity in Mobile Information Systems. In R. R. Wagner and H. Thoma, editors, *Database and Expert System Applications, Proc. of the 7th Int. Conf., DEXA'96, Zurich, Switzerland, September 1996*, Lecture Notes in Computer Science, Vol. 1134, pages 510–520. Springer-Verlag, Berlin, 1996.

[PKH88] C. Pu, G. Kaiser, and N. Hutchinson. Split-Transactions for Open-Ended Activities. In F. Bancilhon and D. J. Dewitt, editors, *Proc. of the 14th Int. Conf. on Very Large Data Bases, VLDB'88, Los Angeles, CA, USA, August 29 – September 1, 1988*, pages 26–37. Morgan Kaufmann Publishers, Palo Alto, CA, 1988.

[PS99] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1999.

[SKK+90] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SRA01] P. Serrano-Alvarado, C. L. Roncancio, and M. Adiba. Analyzing Mobile Transaction Support for DBMS. In *Proc. of the 4th Int. Work. on Mobility in Databases and Distributed Systems, MDDS'2001, September 3–7, 2001, Munich, Germany*, IEEE Computer Society Press, 2001.

[ST97] K. Schwarz and C. Türker. Investigating Advanced Transaction Models for Federated Database Systems. Preprint No. 7, Fakultät für Informatik, Universität Magdeburg, 1997.

[Syb00a] Sybase Inc. *Adaptive Server Anywhere User's Guide*, November 2000.

[Syb00b] Sybase Inc. *UltraLite Developer's Guide*, November 2000.

[Syn00] SyncML Consortium. *SyncML Sync Protocol*, 1999–2000.

[WC95] G. D. Walborn and P. K. Chrysanthis. Supporting Semantics-Based Transaction Processing in Mobile Database Applications. In *Proc. of the 14th IEEE Symposium on Reliable Distributed Systems*, pages 31–40, IEEE Computer Society Press, 1995.

[WC97] G. D. Walborn and P. K. Chrysanthis. PRO-MOTION : Management of Mobile Transactions. In *Proc. of the 1997 ACM Symposium on Applied Computing, February 28 – March 1, 1997, San Jose, CA, USA*, pages 101–108, ACM Press, 1997.

[WC99] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PRO-MOTION. In *Proc. of the 1999 ACM Symposium on Applied Computing, February 28 – March 2, 1999, San Antonio, Texas, USA*, pages 389–398, ACM Press, 1999.

[WC01] N. Wyatt and A. Carlson. *Cloudscape 3.6: A Technical Overview*, March 2001. www.informix.com.

[Wei91] G. Weikum. Principles and Realization Strategies of Multilevel Transactions Management. *ACM Transactions on Database Systems*, 16(1):132–180, March 1991.

[Wei93] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):94–104, 1993.

[WS92] G. Weikum and H.-J. Schek. Concepts and Applications of Multi-level Transactions and Open Nested Transactions. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 515–553, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[YZ94a] L. H. Yeo and A. Zaslavsky. Layered Approach to Transaction Management in Multidatabase Systems. In *Proc. of the 5th Int. Hong Kong Computer Society Database Workshop: Next Generation Database Systems*, pages 179–189, 1994.

[YZ94b] L. H. Yeo and A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. In *Proc. of the 14th IEEE Int. Conf. on Distributed Computing Systems, June 21–24, 1994 Poznan, Poland*, pages 372–379, IEEE Computer Society Press, 1994.