Mobile Collaborative Information System using Distributed Database Architecture

by

Duane Cato

in fulfillment of the requirements for the degree of
Master of Science in Information Systems.

Athabasca University

October, 2009

Document Ref: *009174*

Version: 1.3

ATHABASCA UNIVERSITY

Project Committee Acceptance

The undersigned certify that they have read and recommend for acceptance the project **Mobile Collaborative Information System using Distributed Database Architecture** submitted by **Duane Cato** in fulfillment of the requirements for the degree of **MASTER OF SCIENCE** in **INFORMATION SYSTEMS**.

Dr. Mahmoud F. Abaza	Associate Professor School of Computing and Information Systems (**Thesis Project Supervisor**)	Signature:	_____
Dr. Fuhua (Oscar) Lin	Associate Professor/MSIS Program Coordinator, School of Computing and Information Systems	Signature:	_____
Mr. Richard Huntrods	Academic Coordinator, School of Computing and Information Systems	Signature:	_____
Mrs. Clover Barnett, FCCA	Audit Director, Canadian Tire Ltd. (**Thesis Project Sponsor**)	Signature:	_____

DEDICATION

Dedicated to my parents, who nurtured the scientist in me.

ABSTRACT

This thesis examines the feasibility of communal information sharing between mobile devices using a distributed architecture for the underlying database topology, through research aimed at satisfying two primary objectives:

i. Examination and review of available technologies and products currently supporting distributed mobile database functionality, and

ii. Development of a prototype groupware solution utilising distributed database synchronisation to implement information-sharing functions.

ACKNOWLEDGEMENTS

I would like to acknowledge the support and help of a number of people, without whom this thesis project would have never been completed. Specifically, Mrs. Clover Barnett-Cobb, who was an ever-present source of inspiration and critique. Dr. Mahmoud Abaza, who was a firm supervisor, with significant and useful guidance in presentation and discourse. And especially, my wife, Karen Chen, who was patient and supportive of my studies, providing the emotional, material and intellectual backing for my efforts.

Distribution

Copy No.	Name	Location
1	Athabasca University MSIS Programme library	Programme Library
2	Thesis Supervisor	
3	Thesis Sponsor	
4	Project Committee members	

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I -
BACKGROUND

This thesis examines the feasibility of communal information sharing between mobile devices using a distributed architecture for the underlying database topology. For the purposes of this project, the following definitions are presented to clarify the overall project objectives:

A distributed database can be defined as a database that is not stored at any one single physical location, but rather is dispersed across a network of interconnected computers or devices. For this project, the network of supporting computers will be mobile devices (e.g., cellular telephones), i.e., a **mobile distributed database (MDD)** system *(Tanenbaum, Van Steen, 2002)*.

A **homogeneous distributed database** is a distributed database system that implements each participating device (or **node**) as an instance of the same underlying database infrastructure. All conclusions and solutions derived from this research effort will be predicated upon the use of a homogeneous distributed database architecture[i].

The research planned for this project has two primary objectives:

iii. Examination and review of available technologies and products currently supporting distributed mobile database functionality, and

iv. Development of a prototype groupware solution utilising distributed database synchronisation to implement information-sharing functions.

It is hoped that the dual goals above will help present a more detailed and complete perspective on current mobile distributed database solutions, while also identifying potential directions for improvement and innovation.

Objective

As indicated in the preceding background, this research effort examines the current state of research in mobile device distributed database (MDD) technology, with particular focus on areas of performance, reliability and usability *(Tomasic, Garcia-Molina, 1996)*, with the eventual deliverable being a report comparing mobile distributed database solutions along several critical dimensions. The resulting analysis will be used to supplement design and deployment of a prototype mobile groupware system utilizing distributed database technology for data storage and management.

Significance

Although a great deal of work has been completed in the area of distributed database architecture, the specific field of mobile distributed database management continues to be a burgeoning area for research, due in large part to the very rapid changes that have occurred in mobile device capability over the past few years. Mobile devices now are capable of rapid processing, high-speed communication and support high-level programming primitives (e.g., using Java Platform, Micro Edition - J2ME) and are thus perfect candidates for empowering the average user with improved data accessibility and collaboration, within a mobile context[ii]. This can potentially provide benefits in increased usefulness, timeliness, and availability of on-time, real-time information to the everyday mobile user.

It is expected that research in the areas identified for this project will become more important, and more prevalent in the near future, as the processing power, data management capability and communications flexibility of commodity mobile computing devices increases. This thesis attempts to provide additional information and context for the inevitable discussions which will be necessary to fully utilize and monetize solutions based upon this technology.

Comparison of Mobile Distributed Databases

The existing MDD comparison solutions report provides details of characteristics and metrics for a number of identified candidate products. These dimensions include:

- feature-set,
- availability,
- operating environment/platform, and

- cost and usability *(V. Kumar, 2006)*.

These factors all impact the deliverability and usability of the solution in developing, deploying and operating solutions based on the particular MDD technology.

Prototype MDD Groupware Solution

The proposed groupware system focuses on distribution and sharing of questions, answers and comments between members of a 'study-group', utilising mobile devices to handle the tasks of posting, updating and most importantly, *storing* communication between group constituents. The design is unusual in that it includes no centralised storage database envisaged in the architecture of this particular solution; all persistent and session data generated and utilised in the course of operation exists as the sum total of information within component participating mobile database nodes *(Zondervan and Lee, 1999)*. Future enhancement directions may include the possibility of implementing some sort of offload or external backup mechanism, in order to provide long-term persistent storage or archival capability.

It should also be noted that, as a prototype, the primary goal of this solution was to illustrate existing design and operational morphologies of the specific MDD identified from the evaluation phase of the project, as well as potentially identify improvements and innovations in existing infrastructure and design, that could lead to performance, reliability or functional improvements in the MDD arena.

[i] K. M. Hanna, B. N. Levine, and R. Manmatha. Mobile distributed information retrieval for highly-partitioned networks. In IEEE ICNP, Nov 2003.
[ii] Lim, J. B. and Hurson, A. R. 2001. Transaction Processing in a Mobile, Multi-Database Environment. *Multimedia Tools Appl.* 15, 2 (Nov. 2001), 161-185. DOI= http://dx.doi.org/10.1023/A:1011646626868

CHAPTER II -
MDD PRODUCTS AND DEVELOPMENTS

This chapter provides some background on the usage and uptake of mobile distributed databases, from both a commercial and non-commercial (e.g., academic) perspective.

The commercial environment for MDD products is largely undeveloped at this time, primarily due to the relative immaturity of most of the products that are available for building distributed mobile information management solutions. One of the more interesting details discovered in this research is that mobile databases are usually deployed as extensions of centralized data management systems, which is an approach very much at odds with the philosophy of a distributed peer-based database architecture. The most popular mobile database as of this writing is actually Sybase SQL Anywhere which has approximately 68% of the mobile database market, and is largely deployed as a front-end to “big-iron” database products., e.g., Sybase, SQL Server, DB2 and Oracle. Additionally, there are other mobile database products in the space, such as Oracle 10G/9i Lite, IBM DB2 Everywhere and Borland Jdatastore, however, all of these products remain completely bound to centralized database systems for back-end data persistence, management and processing. The most well-known commercially available distributed database capable of mobile operation, would be Perst, followed closely by db4o (which is not mobile-capable without significant back-end centralized database synchronisation support).

The use of mobile distributed database architectures does not appear at this point, to be a significant driver for mobile database information management systems; however, the flexibility offered by the use of the distributed approach for fault-tolerance, replication, synchronization and data persistence, has made it a highly active field of study in academia. Of note in this regard, are products such as J2MEMicroDB from Universitat Politècnica de Catalunya, Spain, which continues to make significant inroads into improving robustness and functionality, without sacrificing the advantages of the MDD paradigm for database operations. This study presents more detail on the usability, and readiness for commercial use of many of the products mentioned above, as well as outlines some of the pitfalls and difficulties with utilising MDD-based architetcures for information management solution delivery .

CHAPTER III -
METHODOLOGY

The two objectives of this research project, being interrelated, will be satisfied by the following approaches:

i. The ‘current-state’ analysis of the capabilities, opportunities and efficacy of differing mobile distributed database solutions, will be developed through the identification, review, and qualitative/ quantitative comparison of candidates in the MDD solution space.

ii. Design and development of a prototype groupware solution over a P2P-based distributed database architecture, using one of the preferred MDD candidates identified above, in combination with development of any required extensions or custom distributed data management functionality not already provided by the chosen MDD infrastructure.

Distributed Database Infrastructure

In order to satisfy the unique needs of data management in a distributed environment, distributed information retrieval (DIR) techniques are more appropriate than the centralized methods common to monolithic stand-alone databases. In a DIR, all participating nodes in the distributed environment are indexed, to identify those that are likely candidates for locating the particular information desired; only those that meet the search criteria are included in a final list of search hosts. The assumption here is that each participating node in the database, indexes its own subset of data, and thus can answer the question of what information is contained therein. Also, this mechanism presupposes the availability of all the hosts in the database: disconnected of nodes in the database will lead to skewed, or even incorrect search results.

The improvement of the underlying fault-tolerance of the databases' network connectivity will lead to a concomitant increase in the reliability and accuracy of search and data management operations from the overall information management system. Mechanisms for increasing the availability and recoverability within a mobile distributed context are limited by a variety of operational parameters (e.g., infrastructure cost, data transmission cost, network latency, bandwidth , underlying connection protocol artifacts). For purposes of this analysis, we focus principally on reducing data management and transmission costs, through the use of enhanced data transmission protocols and node selection schemes. Candidate methodologies for DIR node interaction include:

i. Communication through a centralized server, which manages the process of data synchronization between the nodes in the mobile distributed database. Issues related to this method include synchronization and federation update consistency, as well as performance bottlenecks and single-point failure concerns.

ii. Communication in an ad hoc manner as necessary for synchronization between individual nodes in the distributed database. This remedies the single-point failure issues identified in (i), but only changes the nature and cause of performance and synchronization concerns.

iii. Communication between peers in the distributed database, using an enhanced protocol and associated topology to avoid failure sensitivity and performance issues associated with options (i) and (ii) above. Synchronization issues continue to require creative management, particularly in light of the more complex interaction now occurring between peers.

Validation of the approaches indicated above, requires a quantitative determination of cost for implementation, management and system resource utilization. Assuming that external costs of management and implementation remain consistent between the three options, the varying cost becomes that required for ongoing system resource utilization. A methodology for evaluating this cost as it applies to multiply synchronized devices is discussed in the following section.

Device Data Synchronization

An additional factor to be considered in any approach for managing data across a distributed system, is the identification of data to be synchronized across participating nodes. Any algorithm designed should support synchronization of multiple replicas of a distributed database, ensuring consistency of data between all copies of the database. Issues which come to fore include:

- Storage constraints on portable devices precludes working with the full dataset on the device; i.e., participating nodes may not necessarily posess the entire data set of the database, but only an operationally (or geographically) relevant subset.

- For disparate data platforms, data will have to be translated or mapped between types. For our research, we have limited ourselves to a consistent database platform across the procured research devices, despite potential differences in

underlying hardware topology, in the interests of reducing variability in the evaluation parameters.

Zondervan and Lee, (1999), indicate a preference for using an ID Mapping Table (IMT) to manage data translation between desktop or server databases and mobile device replicas. This was achieved within the context of the above restrictions, by storing the IMT on the main server, and referencing it for translation of data store-relevant documents between device and server. In the multiple-mobile device scenario we envision, using an IMT is less of an issue, as a consistent data platform between the individual devices makes an IMT of limited value (since there is reduced requirement for translation of values between nodes). We can assume therefore that the algorithm is operating as if a 1-1 imaginary IMT mapping exists between all items in a particular device node and any other mobile device against which we want to replicate.

CHAPTER IV -
COSTS DETERMINATION

In order to develop and validate a realistic methodology for determination of the run-time execution costs of a distributed mobile database, the construct can be viewed in terms of its overarching distributed system characteristics. These characteristics would of necessity, encompass both the attributes and designed behavior of the system, specifically:

- internal node and distributed data constructs and structures
- number of cpu nodes and communication bus paths
- data allocation algorithm (static vs. dynamic)
- Distributed dictionary inverted indexing
- Distributed dictionary hardware infrastructure

One approach that has had good success is the use of inverted indices in distributed dictionary/text-retrieval systems, presented as either a multi-cpu or multi-db retrieval problem. In our scenario, this is relevant as it affects the manner in which we deploy our data-management facilities across the distributed database system, i.e., optimization and performance data indexed inversely across multiple data resources will vary depending on the utilization and number of cpu nodes, degree of inter-cpu I/O, as well as amount and quanta of cpu-data resource communication. In this research environment, use of a distributed dictionary can be viewed as an extrapolation of a simplified multi-host data solution, where each participating host (node) has its own data, cpu and I/O path. This is probably the simplest manner in which data can be fragmented across multiple participating resources (without engaging exotic data management structures or algorithms). As per *Tomasic, A. Garcia-Molina, H. 1993 p. 10*,

“The documents reside in a uniformly distributed manner across all disks *d* in the system
(*d = Hosts * I/O BusesPerHost * DisksPer I/O Bus*). Let the storage components be numbered from *0* to *d - 1* as in the Table below:

Table 1: Host Bus path architecture comparison

Parameter	Value	Description
Hosts	*4*	*Number of Hosts*
I/O Buses Per Host	*4*	*Controllers and I/O Buses per Host*
Stores Per I/O Bus	*2*	*Stores Per 1/O bus*

From research using the host scenario above, Tomasic and Garcia-Molina concluded that although relatively simple, this is a highly effective design for text retrieval, with ready application over parallel architectures. Their research identified a number of performance characteristics of the text-retrieval mechanism above:

- the choice of an index organization depends heavily on the access time of the storage device and the bandwidth of interprocessor communication. As the size of a query increases, its response time may drop; more complex prefetch optimizations were often less effective.

- results indicate that the host index organization is a good choice, as it uses system resources effectively and can lead to high query throughputs in many cases. While it does not perform the best, it is not very far off from the best strategy.

- results also indicate that the system-based organization, even with the prefetch organization, is not good unless disk and network bandwidth utilization is high.

There are a number of factors that may be unsupportive of this approach:

- If the documents were stored on the same devices as the indexes, then storage utilizations would be higher. This would make the system organization more attractive since it reduces the I/O load.

- Not modeling pipelining of I/O and CPU processing within a query can reduce query response time, and would be more

beneficial to the system organization since it deals with longer inverted lists.

- If the inverted lists are in sorted order, the intersection algorithm can (in some cases) terminate having read only a fraction of the inverted lists.

Despite optimizations derived from the reduced complexity for this analysis, it is necessary to identify the resource costs related to the management of data objects across the distributed nodes of the database. These costs can be segregated into object management costs, communications costs and I/O costs, and calculated using any of a variety of methodologies. For our purposes, the approach outlined by *Huang and Wolfson, (1994)* will suffice, particularly due to its specificity for determination of object allocation and access costs. Their method analyzes the cost of distributed object management algorithms in stationary and mobile computing environments. As a precursor to the discussion of their methodology, we include a few definitions:

- An execution schedule is a sequence of requests, with its own associated execution set (reads and writes).

- A saving-read is a read operation that results in saving the data object locally rather than to a remote node

- An allocation schedule is an execution schedule where some of the read requests are saving-reads. At the end of the allocation schedule the data object is stored in the local databases of the participating nodes. A legal allocation schedule is one in which the execution set for every read request contains a reference to a valid (in the network) node or processor; i.e., a node with the latest version of the data object in its local database.

- Allocation scheme for a request is the set of nodes (or processors) that have the latest version of the data object in the local database, just before execution.

- A distributed object management algorithm (DOM) is defined as an algorithm which generates a legal allocation schedule based on an initial allocation schedule ψ.

In their approach, Huang and Wolfson outlined a methodology for comparison of the competitiveness and costliness (in object resources) of differing distributed object management algorithms (DOMs). Here, competitiveness is a measure of the performance of a particular algorithm, while costliness refers to the resource usage in terms of memory and disk requirements. The approach assumes that there is a constant α (termed the competitiveness factor), at which, for a

sequence of read-write operations on a DOM (**Ạ**), the cost of algorithm **Ạ** is < *α x (best-case cost for a DOM*). Another algorithm, **Ḅ**, is less competitive when the cost of **Ḅ** is > *α x (best-case cost for a DOM*). That is, the object-allocation cost of Ḅ is greater than the object-allocation cost of Ạ. Thus, a distributed object management (DOM) algorithm is regarded as competitive if the ratio of the cost of the algorithm to the optimal cost is at most α, for a random sequence of read-write requests. This helps us to compare quantitatively the relative competitiveness of algorithm designs from the candidate distributed database products, as part of our evaluation process.

The difference between the cost quantification for distributed terrestrial vs. distributed mobile environments is that in mobile computing, because of wireless communication charges, local I/O cost is of lesser significance. In a stationary computing scenario, the I/O cost makes up the bulk of read-write requests. A distributed object management algorithm can thus be defined in terms of two dimensions:

i. Information level of the algorithm,

ii. Approach of the allocation scheme during processing.

For the first dimension, the object management algorithm can either determine the requirements of all read-write requests prior to processing (common in online connected environments), or in the second instance, service a request and determine an object allocation scheme without knowing the resource requirements of future requests (an offline, disconnected algorithm scenario).

The second dimension is a variation of the allocation scheme. The traditional, read-one-write-all, static allocation (SA) algorithm does not vary the allocation scheme while processing reads and writes, whereas in a dynamic allocation algorithm (which saves a copy of objects in its local database) the allocation approach varies in response to context changes due to I/O requirements.

The method of Huang and Wolfson, describing the performance and costs of the two approaches, identifies the ratio of the cost of transmitting a control message to the cost of inputting/outputting the object to the local database on secondary storage as c_c, and the ratio of the cost of transmitting the object between two processors to the I/O cost as c_d. In the stationary computing model, the static allocation algorithm is regarded as *tightly competitive with respect to* $(1+c_c+c_d)$, while the dynamic allocation algorithm is *competitive according to* $(2+2xc_c,)$ when $c_d < 1$, and *competitive according to* $(2+c_c,)$ when $c_d > 1$. These results are summarized in Figures 1 and 2.

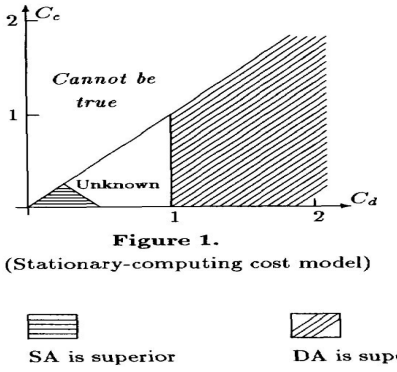

Figure 1. Stationary-computing cost model

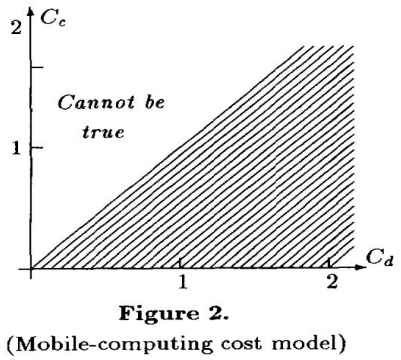

Figure 2. Mobile-computing cost model

It should be noted that the costs here are not directly related to a monetary allocation , but are used to allow a determination of the resources required (from a system perspective) to satisfy the underlying requirements of the distributed system. These costs can be related to actual financial expenditures through standard present/future value evaluations, if so desired.

The authors determined that the cost of a request in a stationary context can be treated as the simplest case for a mobile distributed environment, resulting in a formula for determination of both this as well as the cost of resource usage in a mobile distributed environment:

$$COST(o^i) = \begin{cases} (|X| - 1) \cdot (c_c + c_d) \\ \qquad \text{if } o \text{ is a read, and } i \in X \\ |X| \cdot (c_c + c_d) \\ \qquad \text{if } o \text{ is a read, and } i \notin X \\ |Y \setminus X| \cdot c_c + (|X| - 1) \cdot c_d \\ \qquad \text{if } o \text{ is a write, and } i \in X \\ |Y \setminus X \setminus \{i\}| \cdot c_c + |X| \cdot c_d \\ \qquad \text{if } o \text{ is a write, and } i \notin X \end{cases} \quad \textbf{(1)}$$

where o^i is a transaction request, Y is the allocation scheme at o^i, and X is the corresponding execution set. The cost of a particular request $\boldsymbol{o}^i$ is denoted by $COST(o^i)$.

It was possible to identify basic metrics for the candidate distributed object products surveyed for this research effort, and thus determine values of COST(o^i) for each one. The process used to evaluate the execution statistics and performance of each candidate product is presented in the following section

(MDD Product Cost Analysis), which will outline the methodology and resulting performance statistics based on the execution profile of the various libraries.

Distributed Information Management System Architecture

This project proposes to use a distributed P2P data-management solution using mobile nodes for storage of information in the system. Data will be replicated across the system nodes to provide redundancy, fault-tolerance, rapid access and recovery. This system is designed to reduce network communication costs, as well as reduce latency in communication between data storage resources in satisfying system queries.

The rationale for use of a P2P-based information retrieval system (IR) is supported by the research of *Hanna, Levine, Manmatha, Nov 2003*, where they indicate some of the advantages of non-ad hoc protocols for data communication in distributed mobile system context:

- Connectivity of multiple nodes to information without need for a central server
- No need for a single authority to perform indexing or central data management tasks to respond to system queries
- Ensures delivery of data without need for dedicated communication routes to any node; i.e., data can be delivered to participating mobile devices, regardless of physical partitions in the network.

The following operational characteristics of the mobile distributed system have been identified as relevant to this study:

- Each device participating in the system stores only a portion of the total dataset, and any node will query only its own collection and collections of neighbours that can be directly contacted.
- The number and set of connected neighbours can change dynamically - the criteria for determining what constitutes a direct connection between nodes will depend upon the underlying network topology. For partitioned radio LANs, direct connectivity criteria are much more clearly defined than for devices connected using a WAN or telephony network for data communication.

By comparing the effectiveness and cost metrics of a centralized server topology vs. the distributed mobile P2P data transport mechanism, it was shown that measurable viability and performance advantages may be realised from the latter

approach to satisfying mobile data retrieval and reliability requirements. The methodology used to determine this quantitative difference in meeting fault-tolerance performance criteria utilises a basic cost model comparison of the transaction requirements for performing basic I/O operations in the mobile context. This is the basis for the assumptions and formulae used in the cost-model comparisons for the candidate products of this research paper.

CHAPTER V -
PRODUCT COST ANALYSIS

In order to comprehensively evaluate the MDD products in this study, both raw performance characteristics, as well as derived metrics, were compared. Criteria for distributed database performance evaluation included I/O, node response-time (database ACK), and real-time node resource usage (memory, disk).The following methodology was used to test and evaluate the candidate set of products:

- Identified performance characteristics data and literature for the candidate products.
- Created a distributed database instance for each product for use with the test infrastructure.
- Implemented a test harness for the MDD product libraries, suitable for evaluating operational and performance characteristics of data transactions against the MDD instances above.
- Performed a set of quantifiable updates and searches against the databases, capturing statistics of performance, as indicated by expressions in previous section.
- Generated cost comparison matrix for the candidate products, using both experimentally derived and literature data above.

As indicated in the “Tools and Instrumentation” section later in this document, the candidate J2ME database platforms under consideration in this study were as follows:

i. **Perst**

ii. **Berkeley DB Java Edition**

iii. **db4o**

iv. **J2MEMicroDB**

MDD Evaluation Performance Benchmarks

Table 2: Candidate MDD Product Benchmarks.

Candidate product	Mobile Benchmark	Delete record time ms	Insert record time ms	Search record time ms	Scan/update record time ms
Perst	Android G1 phone	37811	18214	15743	9747
Berkeley DB Java Edition	Mobile Java product is not distributed, stand-alone only				
db4o	J2ME version requires server components due to missing base libraries				
J2MEMicroDB	Tungsten C (running IBM J9)		7710	2000	6610

The following section presents the product cost methodology and publicly available benchmark and performance data for the candidate MDD libraries.

CHAPTER VI -
PERFORMANCE EVALUATION METHODOLOGY

Comparison Approach

Although primarily technology-focused, this research effort was also concerned with examining how actual person-related goals may be further advanced through the use of MDD solutions. It attempted to determine real-world scenarios in which current and future advances in MDD can improve communications and information transformation activities between individuals, social groups and corporate bodies. To this end, comparison candidate database solutions were compared on the following criteria:

i. Mobile database availability – this refers to the presence of an actual database product that runs on mobile devices. Since it has been identified that the preferred solution will be J2ME-compatible (allowing for greater cross-platform compatibility), only databases supporting this will be included.

ii. Distributed data synchronization and update – specifically refers to the ability of multiple participating nodes to share, update and maintain consistent, replicated data with each other.

iii. Object-based architecture – this allows the storage and manipulation of discrete information elements within the distributed database, increasing algorithm and system design simplicity, as well as enhancing opportunities for reuse of data objects in other areas or development endeavours.

iv. Availability and licensing options – that is, is the database solution encumbered by commercial patents or copyright restrictions preventing ready research and investigation.

In addition to the criteria indicated above, this research project attempted to draw clear conclusions on the state of mobile distributed database solutions by considering the following related perspectives and questions:

i. What impact does the mobile distributed database field have at the social and technological interface?

ii. How does the use of this technology affect the overall aspect of human social, corporate or intimate communications?

iii. Both quantitative and qualitative aspects of this research were evaluated. Specifically, it was desired that a comparison of the

following measures enables a picture to be modeled of the technology and its social impact:

- MDD performance criteria (Corwin, B. N. and Braddock, R. L. 1992)[iii],
- MDD operating parameters[iv],
- Project usage and breakdown,
- Market and platform compatibility, and
- Existing solution deployment vs. future projected take-up

All MDD evaluation criteria will be sourced from relevant and current trade, system design and research publications. Quantitative data will be collated and analysed using statistical and comparative methods (e.g. ratio, categorical, interval), while qualitative analyses will include commentary from both developers and users of the various MDD packages available.

Product Cost Calculation

As part of the overall cost determination for the above products, a factor γ that represents the degree of customization needed to fully satisfy MDD functional capability was derived as part of the evaluation criteria for the study.

This necessitated some modification to the formulae provided earlier for the cost of a particular request *COST(o^i), i.e., a customization effort factor for a product* γ^p.

Determination of the customization factor γ^p can be accomplished through the use of a simple ratio between the number of satisfied MDD requirements (r_{sat}) against the total number of MDD characteristics (r_{tot}) . Thus:

$$\gamma^p = \frac{r_{sat}}{r_{tot}} \qquad \textbf{(2)}$$

[iii] Corwin, B. N. and Braddock, R. L. 1992. Operational performance metrics in a distributed system. Part I.: Strategy. In Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's (Kansas City, Missouri, United States). H. Berghel, G. Hedrick, E. Deaton, D. Roach, and R. Wainwright, Eds. SAC '92. ACM Press, New York, NY, 867-872. DOI= http://doi.acm.org/10.1145/130069.130101

[iv] Yee, W. G., Donahoo, M. J., Omiecinski, E., and Navathe, S. B. 2001. Scaling replica maintenance in intermittently synchronized mobile databases. In *Proceedings of the Tenth international Conference on information and Knowledge Management* (Atlanta, Georgia, USA, October 05 - 10, 2001). H. Paques, L. Liu, and D. Grossman, Eds. CIKM '01. ACM Press, New York, NY, 450-457. DOI= http://doi.acm.org/10.1145/502585.502661

Since the cost of all requests of $COST(o^i)$ can be used to determine total average cost for a product (based on number of requests **n**), we see that:

$$COST(p_a) = \frac{\Sigma_0^n \; COST(o^n)}{\mathbf{n}} \qquad \textbf{(3)}$$

giving the execution cost of a particular candidate product as:

$$COST(p) = \gamma^p \cdot COST(p_a) \qquad \textbf{(4)}$$

As part of the process of determining the product execution cost, candidate MDD performance and I/O metrics were collected from vendor literature and independent evaluation results (see performance benchmarks table). As well, product and usage information provided further details of the MDD capabilities for each product, allowing the determination of the customization factor (see following table). The product execution cost was then calculated, and is included in the table.

Cost Evaluation Assumptions

In order to quantitatively account for the overall cost of customization of the candidate MDD products in the comparison effort, the cost expressions derived above were optimized using the following assumptions and observations: about the project's reduced distributed node environment:

- For any set of read/write operations, the number of nodes that end up with the latest data will be equivalent to the number of nodes in our set. Running in a simulated environment, this will be equal to the number of unique instances we have intercommunicating, ie., this will be the number of actual devices running a MDD instance for the application.

- *X* = An allocation schedule is an execution schedule where some of the read requests are saving-reads. At the end of the allocation schedule the data object is stored in the local databases of the participating nodes. Then, we can say that *X=number of transactions*.

- *Y* = Allocation scheme for a request is the set of nodes (or processors) that have the latest version of the data object in the local database, just before execution. Hence, *Y=number of mobile devices*.

- Knowing read time, write time from the benchmarks for each product; the average time for any read or write operation

against the local device was calculated. Thus, *average read/write time ≈ local operation time*.

That is, it was assumed that all transactions make up the allocation schedule, and the allocation scheme is the set of all devices. It was further assumed that any instance is completely updated after all operations (read & write) have completed.

- Finally, we further assume that network ping times for benchmark test networks, taken as a fraction of the network communication cost are a good approximation for control message transmission time across the network.

Benchmark network type (e.g., Bluetooth, GPRS, Simulated)	**Fractional network control communication time/txn (network ping) ms**
TCP/Bluetooth ($tcp_{bluetooth}$)	37.5000
TCP/Simulated ($tcp_{simulated}$)	0.0980

Thus,

$t_{i/o}$ = total time to transmit both data and control (request) message from one device to another.

t_c = time for transmitting control (request) message from one device to another.

t_d = time for transmitting data message from one device to another.

t_{local} = time to store data message on local device.

Allowing the following relations to be derived:

$t_c = tcp_{bluetooth}$

t_{local} = average read-write time from benchmark values

t_d, the transmission time for data block between devices can be worked out by calculating **(5)** below:

$$t_d = \text{data transmission time} = \text{ping transmission time} \times \frac{\text{data block size}}{\text{ping block size}}$$

Using the following data block parameters from the benchmark data:

transaction data block size (bytes) = Integer (8) + String(255)=263 bytes

Since ping control message = 32 bytes, and benchmark data blocks were 263 bytes,

t_d = *(37.5000 x 263)/32*

= *308.2 ms*

$t_{i/o}$ = t_c+t_d = *37.5 + 308.2*

= *345.7 ms*

Calculating c_c and c_d using the above network communication and benchmark times **(6):**

$$c_c = \frac{\text{ratio of the cost of transmitting a control message}}{\text{cost of I/O for the object to the local database on secondary storage}}$$

$$c_d = \frac{\text{ratio of the cost of transmitting the object between two processors}}{\text{I/O cost}}$$

and,
$c_c = t_c/t_{local}$
$c_d = t_d/t_{i/o}$

Finally, the determination of the customization factor allowed the analysis to take into account the degree of source code modification that would be required to provide the candidate product with full MDD capabilities.

CHAPTER VII -
PRODUCT PERFORMANCE COMPARISON

In this section, we present comparisons based on qualitative review of the candidate product literature, along with quantitative performance data from the database product comparisons results.

The following table outlines the determination of the product customization factor.

Table 3: Product customization factor determination.

	Perst	**Berkeley DB Java Edition**	**db4o**	**J2MEMicroDB**
MDD requirements				
Mobile context	1			1
J2ME support	1			1
Local Autonomy	1	1	1	1
No Reliance on a Central Site				1
Continuous Operation	1	1	1	1
Data Location Independence		1		
Data Fragmentation Independence			1	
Data Replication Independence	1	1	1	
Distributed Query Processing		1	1	
Distributed Transaction Management			1	1
Hardware Independence	1			1
Operating System Independence	1			1
Network Independence	1			1
Database Independence	1			1
Total Score	9	6	6	10
Customization factor γ^{p}	**0.69**	**0.31**	**0.46**	**0.77**

The table below outlines the summarized results of the performance evaluation and comparison costs of the candidate products:

Table 4: Derived candidate product comparison costs.

Candidate product	Write time (ms)	Read time (ms)	# transactions	Avg. read/write time (t_{local}) ms	Customization factor γ^{p}	Cost(p)
Perst	65772	25490	10000	4.56	0.69	49731.13
Berkeley DB Java Edition	0	0			0.31	0
db4o	0	0			0.46	0
J2MEMicroDB	14320	8610	1000	11.47	0.77	2115.65

The first candidate product reviewed, db4o, although a Java-based mobile database, does not support distributed synchronization between mobile instances, without the use of a number of traditional database server components (i.e., installations of a “big-iron” RDBMS such as Oracle or MySQL), as well as the db4o proprietary distributed synchronization manager, dRS. For these reasons, db4o is an unlikely candidate for easy customisation in a purely MDD environment.

Similarly, Berkely DB does not have a mobile Java-based product which provides a distributed capability; this suggests a high customization requirement to port the solution to a mobile J2ME context from the available J2SE framework. This expectation is borne out by the customizability factor determined previously. Additionally, the lack of verifiable public benchmark data for these two products in a mobile context, disqualifies them from further consideration as sufficiently viable mobile distributed database solutions (using the criteria defined for this project).

The other candidate products examined in this part of the thesis project, were Perst and J2MEMicroDB, both of which are J2ME capable, support multiple node capability and are significantly customizable, due to their open-source licensing regimes. However, a number of items differentiate the two products, particularly from the standpoints of customizability and product maturity. Perst is a well-known, mature product in the distributed database market space, having been first introduced in 2003, and possesses a significant installed base. J2MEMicroDB is a newer product, and does not have the existing uptake that is exhibited by Perst; this may be a result of it's academic origins, as it is not heavily promoted as a mobile database solution commercially.

Of further impact, is the large disparity in performance between these two J2ME local databases, without including any distributed capabilities. Perst shows an almost 2-fold order of magnitude speed differential with J2MEMicroD in basic mobile database read/write/update operations (see the tables in the previous section). This is a significant factor, particularly considering the added impact of communication time for database transactions in a distributed context is taken

into account. Both Perst and J2MEMicroDB have significant support mechanisms and regular maintenance updates, indicating a vibrant development culture around both. A point of interest is the customization approaches for these products are significantly different, since they have quite distinct approaches in handling database concerns in the limited-resource, distributed environments under consideration. Neither of the products appears to compromise in pursuing highly customizable, developer-friendly usage patterns in the codebases, which bodes well for the MDD development space on a whole. All of these quantitative and qualitative characteristics point to Perst being a significantly more suitable MDD solution candidate for information management problems with a distributed mobile component than any of the other reviewed products. J2MEMicroDB, by virtue of it's high level of support, ease-of use, and portability, does come a close second in our evaluation.

CHAPTER VIII -
EVALUATION CONCLUSIONS

The conclusion of this project thesis indicates that, of our candidate product set of mobile distributed database solutions, Perst is the most capable MDD solution candidate for distributed information management solutions, as a result of it's high level of support, ease-of use, portability, customizability and satisfaction of MDD functional criteria. In particular, Perst, though not a distributed database solution readily capable of multi-nodal input, showed itself to be easily customizable for that purpose, and in fact, was used in the secondary portion of this project thesis, as the base for the MDD prototype application.

The analysis and research presented in this paper, outlines the primary criteria, characteristics and factors that affect the uptake and usage of mobile distributed databases as an information management solution component. From the results of our product comparison and performance examination, it is clear that the field is still in a maturing stage, but there are some clear leaders, which are being used in highly significant applications for organizational and competitive benefit. It seems likely that the MDD product space will expand through both continuing open-source and proprietary development of existing solutions, as well as from inroads and participation by mainstream database vendors, such as Oracle (with Berkely DB) and db4o (which supports multiple database vendors). At this juncture, it would be premature to suggest that any one approach to handling problems in the mobile database space can be more appropriately handled using MDD solutions; what is clear, is that MDD solutions can and will continue to be a very real component in satisfying the increasing information-sharing needs of our society.

CHAPTER IX -
PROTOTYPE BACKGROUND

The prototype distributed information system was intended to provide both a distributed database infrastructure and application framework for the sharing of questions, answers and commentary between participating individuals via a mobile network *(Kam-yiu Lam, 2000)*. It was designed and developed using an object-oriented design methodology, with a plugin-architecture for actual database access. This was necessary due to the diverse nature of database APIs that were available during the assessment phase of the project; the final database access API chosen was largely dependent on the particular MDD solution selected for the final prototype design.

This portion of the project was included to serve both an exploratory and confirmatory purpose; it is expected that it will help examine and illustrate the issues and compromises which are required to implement a distributed database system over a mobile connectivity framework *(Mao, Z. and Douligeris, C. 2004)*, as well as investigate potential ways in which improvements may be made in these same mechanisms. As such, there has been significant emphasis on qualitative measurement of the prototype's performance, operation and usability. To this end, the high-level approach and operating requirements have been identified for the prototype's operation, and are presented in the subsequent sections.

CHAPTER X -
PROTOTYPE FUNCTIONAL REQUIREMENTS

The prototype application presents a simplified single-page interface, displaying a scrolling list of the most recent posts in the group discussion session which the mobile device is currently monitoring. User/mobile device access is authenticated against a master list for the system; however, there has not been a rigorous application of security protocols in this project. As authenticated devices sign into the system, they will receive the list of current discussion groups, from which one may be chosen to continue communication. Subsequent posts and messages will be maintained within this group, until the user transfers to another available group (or starts a new one).

It should be reiterated here, that this group discussion system used only participating mobile devices as the backing database store for all operations; there was no "central" database or external persistent store, although future developments may include provision for an archival mechanism implemented in that manner *(Lewis, L. F. and Keleman, K. S. 1988)*.

Requirements for Prototype

As an example of an MDD-based solution, this prototype illustrates the capture of data from multiple nodes (mobile phones), and the synchronization of that data between nodes on an ongoing basis, without need for a centralized server. Some of th basic requirements can be itemized as follows:

- User authentication – users should be authenticated against the database for security
- Database node registration – lacking a central database, nodes are registered with each other manually. In a real-world scenario, this would probably be handled using an advertising service component of the application.
- Multi-node data synchronization – data must be replicated/synchronized between all registered mobile devices using the application.
- J2ME (Java Mobile Edition) capable – the application should be packaged and distributed as a portable Java midlet, to illustrate use in multiple device types and environments.

It should be noted that as a prototype, there are a number of assumptions that have been made about the operation and context of the application. These are outlined in the Assumptions section highlighted below.

Assumptions

i. The application does not attempt to enforce data validation. Invalid input can crash the program, since the application does not enforce **real-world** restrictions and checks.

ii. The prototype currently is deployed expecting J2ME HTTP/TCP communication. It has not been designed to communicate over non TCP-based mobile contexts.

iii. Lacking a centralised node tracking database, al nodes have to manually register with each other to support synchronization. In a real-world scenario, a simple solution to this would involve advertising new nodes in a TCP broadcast, or use of a centralised registry.

CHAPTER XI -
PROTOTYPE HIGH-LEVEL APPROACH

This section of the research project used one of the reviewed MDD candidates as a suitable base to develop a prototype groupware information system utilising existing and custom-developed functionality. The prototype was designed to illustrate the following characteristics *(Motzkin, D. 1991)* of distributed database systems:

i. Distributed data synchronization and update – specifically refers to the ability of multiple participating nodes to share, update and maintain consistent, replicated data with each other. This capability may require some further development to create a working prototype, depending on the database solution used for the research.

ii. Management of node failure through fault-tolerance and fail-over mechanisms. This would include use of previously mentioned redundancy arrangements as a base upon which to build.

iii. Secure internal database communications over inherently insecure medium (e.g., public mobile network or Internet).

iv. Increased availability through multiple, ubiquitous node participation.

v. Persistence of data through use of back-end storage and backup to non-mobile site.

vi. Scalability (increased ability to service data requests), through greater node participation in the distributed database system.

The prototype application presents a simplified single-page interface, displaying a scrolling list of the most recent posts in the group discussion session which the mobile device is currently monitoring. User/mobile device access is authenticated against a master list for the system; however, there has not been a rigorous application of security protocols in this project. As authenticated devices sign into the system, they receive the list of current discussion groups, from which one may be chosen to continue communication.

The completed prototype allowed evaluation of the performance characteristics of a MDD-based information system *(Triantafillou, P. 1996)*, including measurement of metrics such as:

i. Node synchronization dissonance - that is, degree of differences between participating database nodes (devices)

ii. information propagation rates - the rate at which all participating nodes present the same view of the data

iii. Node/database failure rate - how often a device node loses connectivity (and thus a valid data view) with other nodes.

iv. Data loss, retransmit and error handling quantification - how much of actual data being transmitted between nodes is to satisfy error, loss or exceptional conditions

v. Database implementation cost - a base figure for the cost of delivery of this type of information system using MDD architecture, for comparison with cost using a centralised database infrastructure.

In addition, other indicators derived from the above metrics can be used to present a more comprehensive picture of the operating characteristics of the MDD chosen for the project.

Discussion Forum High-level Design

Database

As concluded in the first part of this project dissertation, only two candidate products satisfied the key requirements identified as necessary for a high-performance, solution-ready mobile distributed database. These were Perst and J2MEMicroDB, both of which are J2ME capable, support multiple node capability and are significantly customizable, due to their open-source licensing regimes. However, as concluded, a number of items mitigated against the selection of J2MEMicroDB as the preferred product in this analysis: the relative immaturity of the product, it's rather poor performance in benchmarks, as well as its very limited multi-node support make it less likely to immediately be a successful solution in mainstream MDD solution sets.

It should be noted that the above does not suggest that the selected product, Perst, does not have some limitations. During the development of this prototype, it was identified that the master-slave approach favoured by Perst for supporting

replication and distributed database synchronization, would not satisfactorily handle the requirements of the prototype The master-slave approach enforces read-only capability on all slave nodes, while allowing only the master node to handle updates (writes) to the database. This of course, runs counter to the primary goal of allowing data-entry from any node, with synchronization of all nodes periodically, without need for a centralized server.

Operation

In order to provide our prototype with the ability to support synchronization between the multiple nodes, the application design uses Perst as the underlying local database on each node, and adds the capability to "register" all participating nodes, with periodic synchronization of updates between all the database nodes. Thus, we have the following sequence of actions for the operation of any instance of the prototype application:

- Local mobile device starts MDDForum application
- Local Perst database is opened
- Register other device nodes with this instance
    i. Application gets names/addresses of previously registered database nodes
    ii. Application contacts all nodes, and updates itself with most recent data from them
- MDDForum application synchronizes local database with other nodes
- User authenticates against distributed database
- Present user authentication screen, and subsequent UI forms.
- Perform regular database operations for forum functionality (e.g., posts, reads, etc.). For purposes of this research, we propose a HTTP-based protocol for communication between database nodes.
- Periodically synchronize local updates with other database nodes.

Communication

The prototype application has been designed to utilise TCP/IP for network communication between database nodes, largely because of the ubiquity of

operating system support for that protocol in the mobile device space. In a real-world deployment,it is likely that additional flexibility and functionality would be derived from the use of TCP/IP, as this allows the participation of a more diverse set of devices , each of which may operate using differing hardware network interfaces, while participating in the same distributed database. For example, one mobile device may use a 802.11b (Wifi) mechanism for communication, while another participates through the use of a 802.15.1 (Bluetooth) connection. For development and demonstration purposes, this project utilised a wholly TCP/Bluetooth network for node-node communication.

The following diagrams (overleaf) further illustrate the design and underlying architecture decisions made to support the prototype operation:

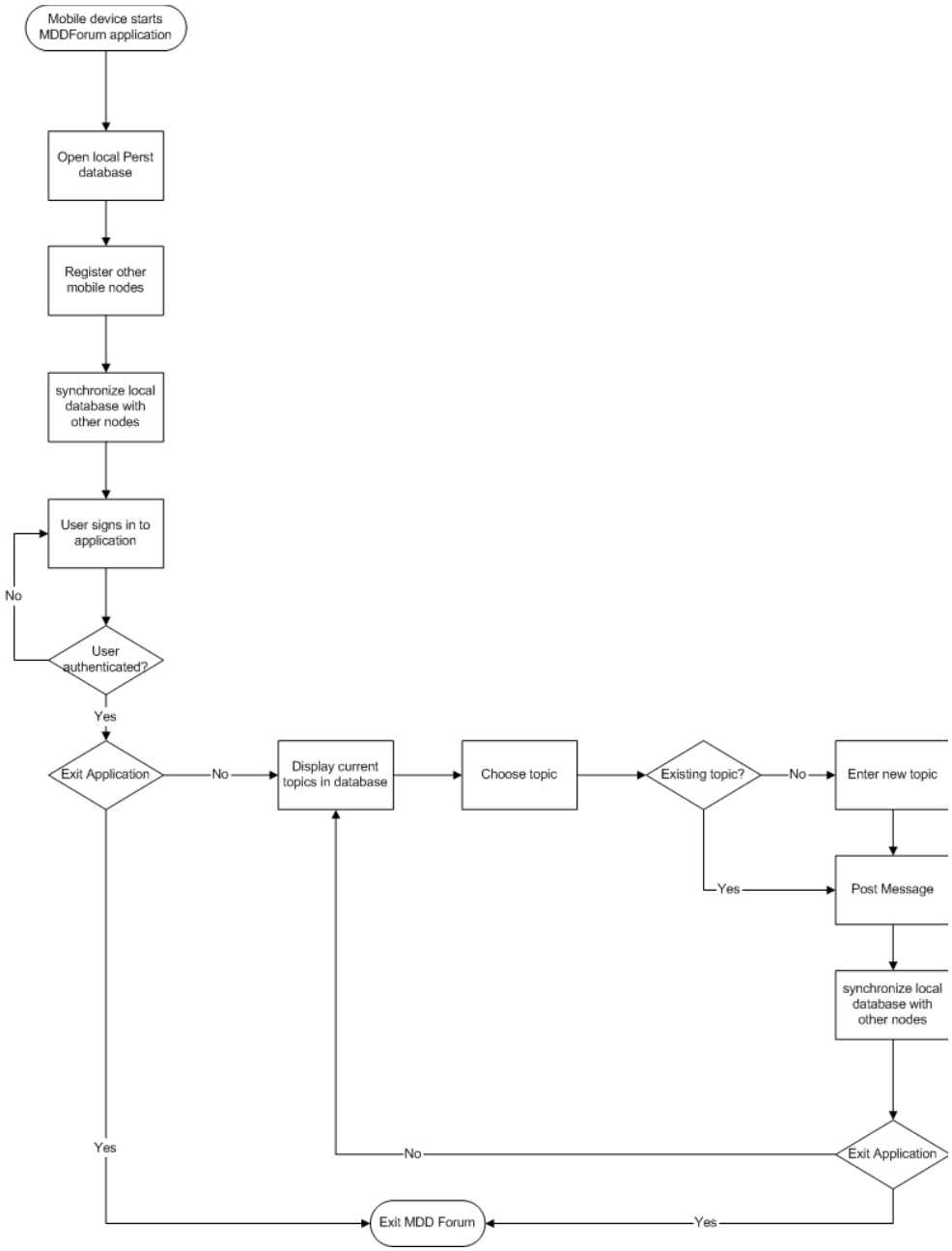

Figure 3: Flowchart illustrating prototype execution.

Prototype database architecture

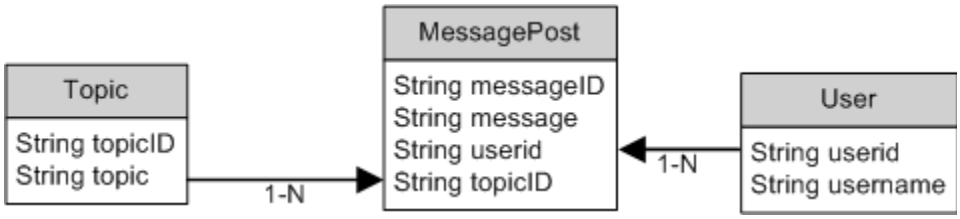

Figure 4: Diagram illustrating entity/table relationships in the prototype.

Tables used for the prototypes simple forum authentication and post mechanism are:

- MessagePost(String userID, String TopicID, MessageID, Message)
- User(UserID, Username)
- Topics(TopicID, Topic)

The diagram on the following page illustrates the overall architecture and infrastructure approach for the MDD prototype application.

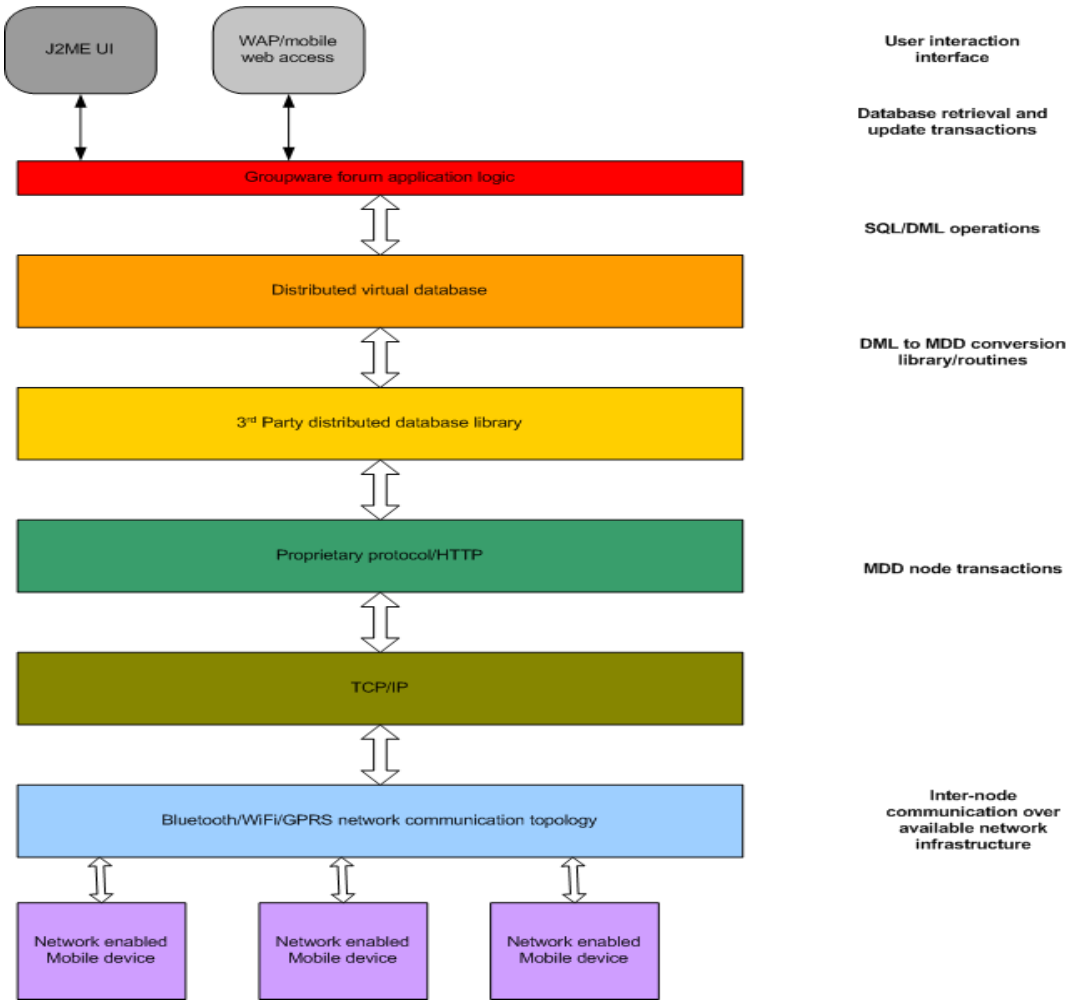

Figure 5: Prototype architecture technology stack.

Tools and Instrumentation

The following additional tools and devices were used to evaluate the various MDD products and develop custom code for the research project:

- J2ME-capable MDD database candidates:

i. **Perst** (http://www.mcobject.com/perst/), which is an Open Source J2ME/C# database for constrained devices. It does not have a pre-packaged distributed solution, but its GPL-based open source license makes it very amenable to adding new functionality as part of the project.

ii. **Berkeley DB Java Edition** (http://www.sleepycat.com/products/je.shtml), is an embedded all-Java solution; however, research has indicated that it does not have significant capability in the constrained J2ME environment. Additionally, it appears to have clear limitations on its distributed database capabilities, particularly in the mobile context, leading to significant effort required to add the specialised functionality needed to satisfy the project prototype aims, and disqualifying it from use in the prototype phase of the project.

iii. **db4o** (http://www.db4objects.com/) is also a multi-platform embedded database, with an all-Java option. It does have a version with limited functionality to run in a J2ME environment; however it differs from the other products considered in this project, in it's need for a server-based database component to handle advanced distributed database functionality. This limitation disqualified db4o from use as a prototype base for this project.

iv. **J2MEMicroDB** (http://morfeo.upc.es/crom/mod/wiki/view.php?id=16&name=dfwikipage&page=Introduction+to+J2MESDLIB#toc3) is an open source project developed by the Universitat Politècnica de Catalunya that provides J2ME developers with several APIs to manage a limited-environment relational database on a mobile device. It is a fairly new product, but shows great potential in satisfying a number of requirements for a mobile database. However, it does not currently provide any support for distributed database, replication, or synchronization.

- J2ME-capable mobile device(s), with local area network capability (e.g., Bluetooth). Project working devices were **Palm Treo 650 and Palm Centro**. Limiting total cost of hardware purchases was a factor in choosing devices for this research.

- Personal computer compatible USB Bluetooth devices for communication and configuration of communication with mobile devices in wireless LAN.

- Java development environment was JDeveloper, in addition to the Sun Java Micro Edition Java Wireless Toolkit 2.5.201 CLDC development application and libraries.

- Office suite and spreadsheet software for data collation, data analysis and presentation of results was OpenOffice 2.4 (**http://openoffice.org**).

- Version control and document management software used was CVS.

As much as possible, the project endeavoured to use free or open-source solutions and products to satisfy project tasks and deliverable requirements. This helped keep project research costs to a minimum, as well as reduced the likelihood of infringing upon patents or copyrights of third-party agencies or solution providers. To further reduce likelihood of such violations, all external material (not developed by this researcher) has been attributed clearly to its inventor and/or owner of record; any inaccuracies or oversights are unintentional and will be corrected immediately upon notification.

The population for the test-bed of the project included a total of 3 (three) mobile devices. This evaluation sample size permitted some degree of real-world impact studies, particularly in the areas of node interference, data synchronisation and data replication. The prototype evaluation scenario used a local area network topology of TCP/IP over Bluetooth for communication between devices *(Qusay H. Mahmoud, 2003)*, *(Michael Cymerman, 2001)*, although in the real-world, performance will be influenced by the speeds prevalent from mobile device data providers, possibly utilising GPRS or other packet-switched high-bandwidth solutions. In a real-world scenario, this difference would necessitate some comparison of topologies and their underlying performance characteristics, with a view to factoring this into the effect on the performance of any MDD application operating characteristics.

Product Selection Methodology

A critical comparison of the MDD requirements for the prototype, and the feature-set of the candidate products, allowed an initial ranking of the suitability of the candidates for the project objectives. The analysis scored the candidate database solutions, by taking into account several criteria.

From the candidate product evaluation portion of this project, the identified product which most appeared to most closely satisfy the key requirements for a competitive MDD, was Perst, primarily due to its performance, J2ME small-footprint support, distributed node capability and permissive licensing scheme. In addition, the value calculated for the customizability factor (measure of the effort required to improve the node-node read/write capability of the product), was significantly better for Perst than for it's nearest competitor, J2MEMicroDB. As a result, Perst was selected as the underlying database to be used for developing the MDD prototype.

CHAPTER XII -
PROTOTYPE UML DESIGN

The diagrams below provide UML diagrams illustrating the activity and class design of the prototype application. Note that low-level program details are not included in these summary diagrams for sake of brevity.

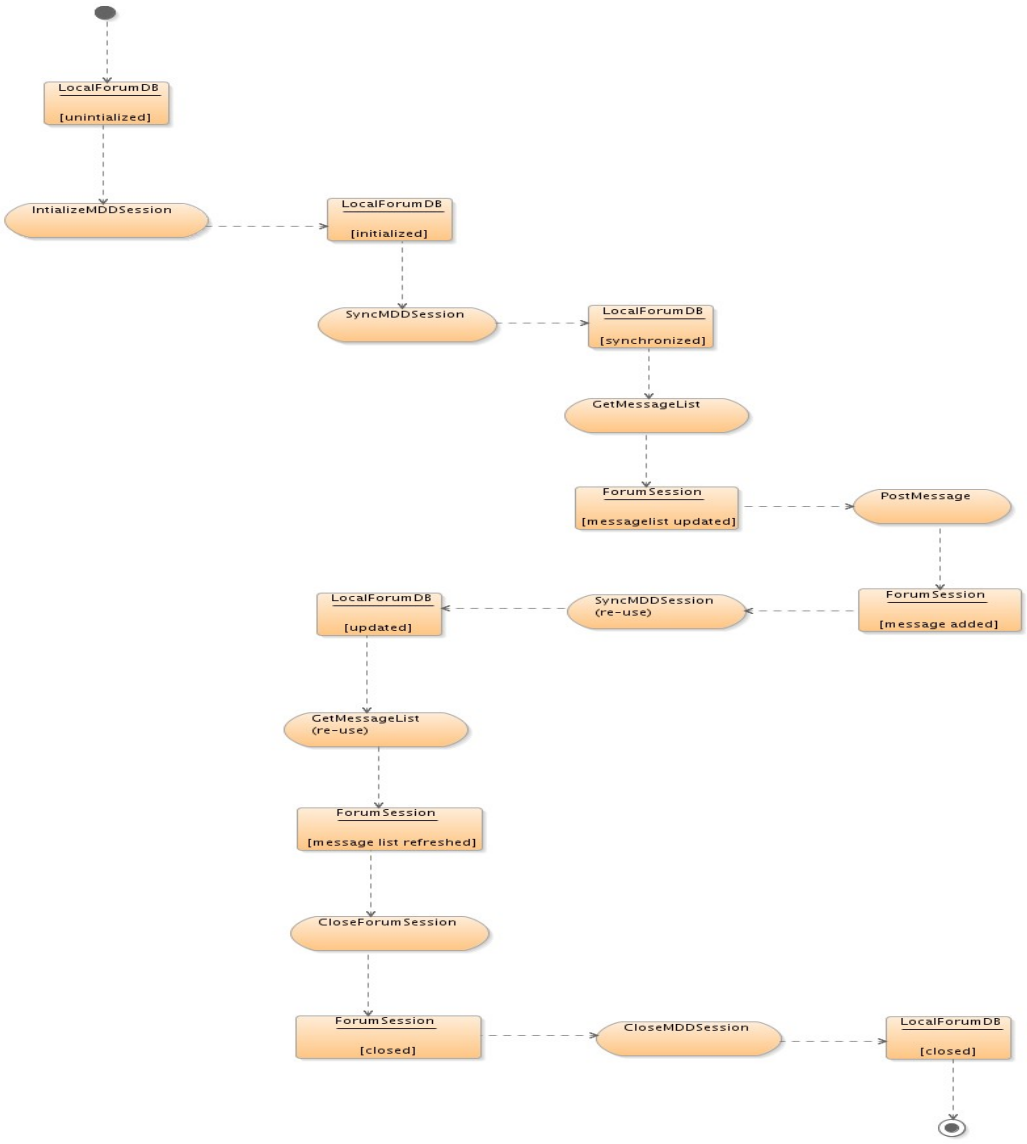

Figure 6: Prototype UML Activity diagram.

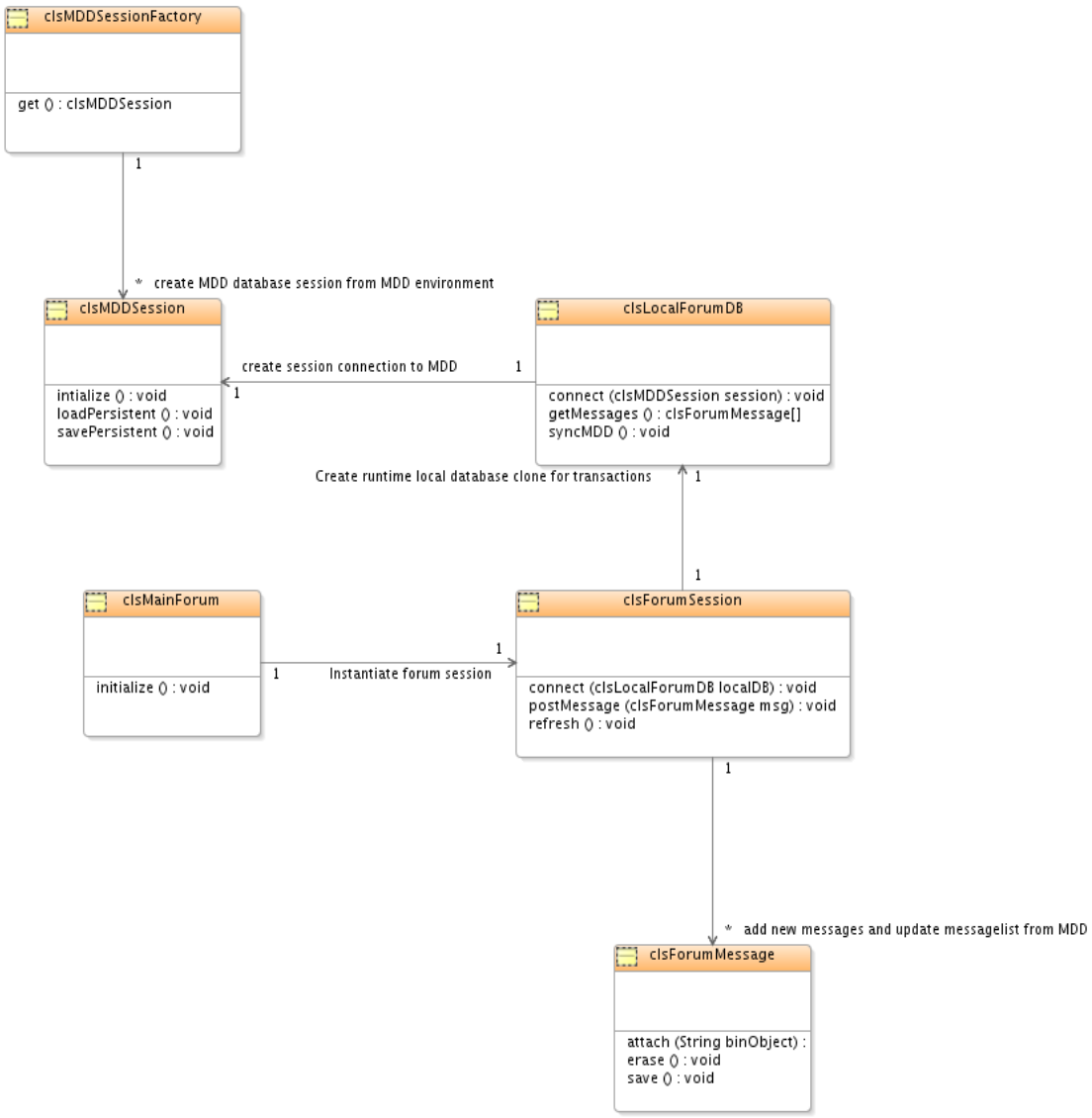

Figure 7: Prototype design high-level class diagram.

CHAPTER XIII -
PROTOTYPE DETAILED DESIGN

This section provides detailed code snippets on the portions of the prototype design relevant to this study, i.e., customizations added to support multi-node read/write capability and synchronization, as well as the distributed database node registry mechanism. Additionally, this section illustrates the operation of the prototype as it is executed, through a number of screenshots.

Below are screen shots of the prototype running on a Palm Centro with IBM Java J9.2.2.14/ARM.

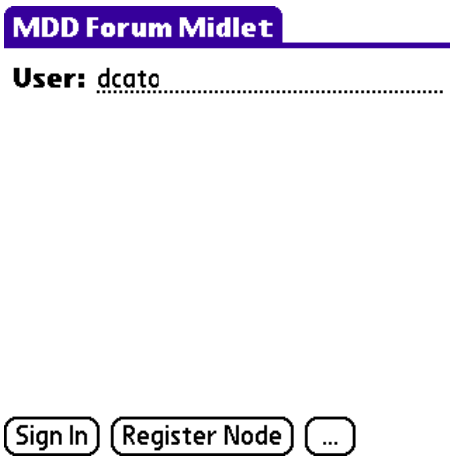

Figure 8: User sign in screen

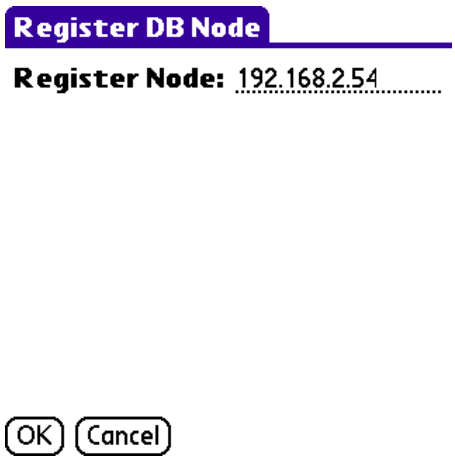

Figure 9: Registration screen

Figure 10: Topic selection screen

Figure 11: Message Post screen

The following snippets of code illustrate the distributed node registry mechanism, as well as the approach used to keep all registered databases against a particular device synchronized:

Device Registry

```
    /**
     * provides form for entering the hostname/ip for another device
     * with an instance of the prototype database.
     * Calls the MMD registration function to be used for
     * later synchronization
     */
    public void regDBNode(){
        System.err.println("Register database node Command pressed..");
        fRegDB = new Form("Register DB Node");        //display new form to register database
        regDBField = new TextField("Register Node:", "",30, TextField.ANY);
        fRegDB.append(regDBField);
       //add the command buttons to post and exit
        fRegDB.addCommand(regDBOKCommand);
       fRegDB.addCommand(regDBCancelCommand);
       fRegDB.setCommandListener(this);
        display.setCurrent(fRegDB);
    }
```

```
public class MDDatabase {
Storage db;
Hashtable databaseInstances=new Hashtable();
String nodeID;
String hostname;
.
.
.
    /**
     * Registers a database instance node with this instance, so that it can be
     * synchronized after updates
     * @param nodeID - identifier that uniquely identifies an instance of the database
     * usually use a hash of the hostname of the device running the node, but can be any unique id
     * @param dbNode - identifies the node instance
     */
    public void registerDatabaseNode(String nodeID, String dbNode){
        System.err.println("Registering database node: "+nodeID+"/"+dbNode);
        //add the default Perst replication port
        databaseInstances.put(nodeID,dbNode+":"+ getReplicatePort());
    }
.
.
```

Device Synchronization

```
    /**
     * Synchronizes all database instances registered with this instance, so they have
     * same set of transaction data
     * using Perst dynamic replication to move entire database
     * This requires finding out if remote db has dirty pages, and if so, we accept
     * it's update. All nodes are responsible for updating themselves.
     */
    public void synchronize(){
        System.err.println("Synchronizing database nodes...");
        //array of all known nodes
        int i=0;
        String[] slaveNodes = new String[databaseInstances.size()];

        this.db.close(); // close main db storage for sync process
        //get the array of all known nodes to be updated from this instance as slaves
        for (Enumeration e=databaseInstances.elements();e.hasMoreElements();)
            slaveNodes[i++] = (String)e.nextElement();
        //this is the master - we know by checking the flag set by checking
        //our list of nodes.
        if (this.isMaster()) {
            ReplicationMasterStorage db =
               StorageFactory.getInstance().createReplicationMasterStorage(
            // port at which master will accept connections of new
            // replicas,-1 means that connections of new replicas are not supported
            -1,
            // list of slave node addresses
            slaveNodes,
            false ? asyncBufSize : 0); // size of asynchronous buffer
            // Disable synchronous flushing of disk buffers because
            // in case of fault database can be recovered from slave node
            db.setProperty("perst.file.noflush", Boolean.TRUE);
            // set replication mode
            db.setProperty("perst.replication.ack", Boolean.FALSE);
            db.open(dbName, pagePoolSize); // open master storage
            sendUpdates();  // ... Work with the database
            db.close(); // close master storage
        }
        else  {     //this is a node to get replicated updates
            System.out.println("started slave replication");
            // Create replica which accepts master connection at the
            // specified port
            ReplicationSlaveStorage db =
            StorageFactory.getInstance().createReplicationSlaveStorage(getReplicatePort());
            // Disable synchronous flushing of disk buffers because
            // in case of fault database can be recovered from slave node
            db.setProperty("perst.file.noflush", Boolean.TRUE);
            // set replication mode
            db.setProperty("perst.replication.ack", Boolean.FALSE);
            db.open(dbName, pagePoolSize); // open slave storage
            // Slave node receives modifications from master in separate
            // thread. Concurrently it can execute its own read-only
            // transactions. But to perform some processing at slave node
            // only when some data is changed (transaction is committed by
            // master node)then the
            // ReplicationSlaveStorage.waitForModifications() method can be
            // used to wait for update of the database.
            while (db.isConnected()) { // while master is alive
                  // Wait until master commits new transaction
                  db.waitForModification();
                  // Start special read-only transaction at replica
                  db.beginThreadTransaction(
                       Storage.REPLICATION_SLAVE_TRANSACTION);
```

```
                applynewUpdates(); // Do some processing of the database
                db.endThreadTransaction();
            }
        db.close(); // close slave storage
        }
        this.db.open(dbName, PAGE_POOL_SIZE);      //re-open the database, ready to do more work
    }
```

CHAPTER XIV -
PROTOTYPE DESIGN DECISIONS

This chapter of the research provides more detail on the planned operation of the prototype, and some of the design caveats, exceptions and assumptions.

A number of design decisions were made to support the objectives of the prototype, i.e., to demonstrate distributed database functionality at work, and provide some illustration of the challenges likely in this type of application environment. Many of the decisions made about the design approach were geared towards limiting the overall scope of the project, primarily for reasons of effort and time, with the knowledge that these compromises should and would not compromise the overall usability of the prototype evaluation. Because of the highly complex nature of distributed database topologies and communication, it was decided early on to ensure that only the simplest case of a distributed network would be considered in this project, i.e., a two-node network.

To additionally reduce likelihood of being sidetracked by performance and behaviour artifacts in the prototype execution, the application was designed to perform only the barest minimum of validations and exception-handling, with the understanding that a real-world application will have a definite increment in resource effort and time to ensure robust, consistent operation in a variety of executing environments. Additional assumptions and challenges are presented below.

Discussion Forum High-level Design

The performance of the MDD prototype was relatively good, from a purely qualitative standpoint; of greater concern was the multiple occurrences of data consistency and update errors between nodes, due to concurrency and multi-update issues. Because the underlying Perst database software is designed for replication between databases using a single master to handle updates to many slaves, it was particularly difficult to design an approach that allowed for usage of multiple masters in the distributed database.

In the prototype, a simple list of the nodes registered with a device, were treated as slaves to be updated by this master. This means that at any one time, multiple nodes could be attempt to be the master updated in the database node cluster. To avoid this, a fair election mechanism has to be developed; unfortunately, the review and testing of such a master-selection mechanism is out of the scope of this paper.

Discussion Forum Design Challenges

Some of the challenges faced in the development of this prototype included:

- The use of J2ME as the development platform for the Prototype required learning details of an API and event management schema that is quite different from the traditional J2SE AWT/JFC approach. The J2ME API, being Form-based, is much more simplified, and offers less flexibility in handling UI components than the AWT.

- Learning both the API for the underlying Perst database, as well as adding functionality to support unique functionality proved to be significant hurdles.

- The mechanism for handling synchronization between participating nodes in the distributed database, proved to be non-intuitive and more complex than expected, due to the vendor-specific API compromises that had to be made to work-around the master-slave update limitations of the product.

- A little-known difference between the J2ME and J2SE Java development environments is in the networking capability. Many of the classes and utility functions available in the J2SE java.net package are unavailable in the J2ME environment; this includes, classes for hostname identification, many socket connection routines, and socket to stream functionality. As a result, this prototype limited all network operations to the simplest available in the J2ME environment – it should be understood that a production-quality J2ME distributed database application will have to face this challenge through either custom network management classes or the use of an alternate architecture which offloads the network communication burden onto a centralized server (this appears to be the approach taken by db4o).

CHAPTER XV -
PROTOTYPE CONCLUSIONS

The clearest conclusion to be drawn from the design, development, and performance of this prototype, is that there remains a great deal of work to be done in improving the reliability, flexibility and consistency of distributed databases, especially in the mobile context. Despite the use of a well-known, well-regarded, and highly customizable product as the base for prototype application development (Perst), there were still a myriad of challenges and issues which mitigated against the provision of a robust solution to even the limited scope identified by this two-part project. Our research prototype application performed successfully as designed, with data input and sharing between mobile nodes occurring as expected, within the limits of the parameters set for the application execution, and the defined research examination criteria.

As indicated in the previous section on the prototype design, a number of design decisions were made to reduce the use of certain exception handling and network management operations which might be considered the norm in a non-distributed, immotile environment: the relative immaturity of the underlying technology (J2ME and Perst, turned out to be a significant counterpoint to the achieved aims of code portability and device independence.

Additionally, the requirement to handle the bulk of the distributed database node replication and synchronization logic (as opposed to making use of a database-level capability), significantly impacted the reliability and consistency of the prototype application. As indicated earlier, it appears that for a well-behaved mobile distributed database application, a fairly robust synchronization and node management mechanism has to be developed or delivered with the underlying database.

However, the conclusions of this portion of research on distributed database in mobile devices, is not entirely negative, since it was possible to evaluate multiple products for suitability as infrastructure components in our prototype solution. Further, it was possible to design and develop an actual application that executed on multiple mobile devices and shared data between them, albeit with limited consistency. Finally, this researcher was able to develop custom database node registration and synchronization routines that enhanced the underlying mobile database to support multi-nodal input and replication. It is clear however, that there are a few areas that pose challenges to the significant uptake and usage of mobile distributed database solutions:

- Underlying platform technology and capability must be improved – this speaks directly to the library, connectivity and functionality limitations of environments such as J2ME.

- Distributed database solutions must become much more reliable, in order to provide better capability for managing real-world scenarios of multi-site data input and data update consistency.

- Customizability of products must be balanced with robust behaviour out of the box – consistency in limited-resource or mobile distributed environments should not require significant development to implement.

It appears likely that we are at the cusp of much more significant development in the mobile distributed database space; personal devices are becoming ever more powerful, with greater connectivity options, making them highly desirable targets for business, commercial and entertainment applications. With this in mind, and based on the limited success achieved with this project's simplified prototype application, it seems highly probable that there will be significant development and improvement in some (if not all) of the products examined in this paper, particularly Perst and J2MEMicroDB. Both of these products are already production-ready, assuming significant custom development; what remains is to reduce the barrier to entry for MDD application developers to make use of these tools in solving future information management problems.

CHAPTER XVI -
BIBLIOGRAPHY

References

1. Andrew S. Tanenbaum, Marten Van Steen, 2002*, Distributed Systems Principles and Paradigms*

2. Tomasic, A. and Garcia-Molina, H. 1996. *Performance issues in distributed shared-nothing information-retrieval systems. Inf. Process. Manage. 32, 6 (Nov. 1996), 647-665.* DOI= http://dx.doi.org/10.1016/S0306-4573(96)00019-2

3. Vijay Kumar (2006), *Mobile Database Systems (Wiley Series on Parallel and Distributed Computing) (Hardcover) by Vijay Kumar (2006)*. ISBN-10: 0471467928

4. Quinton Zondervan and Alexandre Lee, 1999*, Data Synchronization of Portable Mobile Devices in a Distributed Database System, Quinton Zondervan and Alexandre Lee, Lotus Development Corporation, 1999* (http://domino.watson.ibm.com/cambridge/research.nsf/0/c71ebac11ec6e54f8525661600797829/$FILE/mobile.pdf)

5. Motzkin, D. 1991. *Distributed database design—optimization vs feasibility. Inf. Syst. 15, 6 (Jan. 1991), 615-625.* DOI= http://dx.doi.org/10.1016/0306-4379(90)90064-V

6. Mao, Z. and Douligeris, C. 2004. *A distributed database architecture for global roaming in next-generation mobile networks. IEEE/ACM Trans. Netw. 12, 1 (Feb. 2004), 146-160.* DOI= http://dx.doi.org/10.1109/TNET.2003.820435

7. Kam-yiu Lam, 2000, *Transaction Processing in Mobile Distributed Real-time Database Systems, Kam-yiu Lam, Department of Computer Science, City University of Hong Kong, 2000* (http://ipdps.cc.gatech.edu/1998/wpdrts/kylam.pdf)

8. Huang, Y. and Wolfson, O. 1994. *Object Allocation in Distributed Databases and Mobile Computers. In Proceedings of the Tenth international Conference on Data Engineering (February 14 - 18, 1994). IEEE Computer Society, Washington, DC, 20-29.* (http://citeseer.ist.psu.edu/ACMLINK/http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=655255)

9. Byun, S. and Moon, S. 1998. *Fault-tolerant quorum consensus scheme for replication control in mobile distributed database systems: FTQC. J. Database Manage. 9, 3 (Jun. 1998), 16-24*.

10. Lewis, L. F. and Keleman, K. S. 1988. *Issues on group decision support system (GDSS) design. J. Inf. Sci. 14, 6 (Jul. 1988), 347-354.* DOI= http://dx.doi.org/10.1177/016555158801400605

11. Triantafillou, P. 1996. *Availability and performance limitations in multidatabases. Inf. Syst. 21, 7 (Nov. 1996), 577-593.* DOI= **http://dx.doi.org/10.1016/S0306-4379(96)00029-4**

12. Michael Cymerman, 2001, *Device programming with MIDP, Part 1 The concepts behind MIDP APIs and J2ME. By Michael Cymerman, JavaWorld.com, 01/05/01* (**http://www.javaworld.com/javaworld/jw-01-2001/jw-0105-midp.html**)

13. Qusay H. Mahmoud, 2003, *Wireless Application Programming with J2ME and Bluetooth by Qusay H. Mahmoud February 2003* (**http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/**)

14. Qusay H. Mahmoud 2003, *J2ME Low-Level Network Programming with MIDP 2.0 by Qusay H. Mahmoud April 2003* (**http://developers.sun.com/techtopics/mobility/midp/articles/midp2network/**)

15. Alier, M.; Casado, P.; Casany, M.J., 2007, *J2MEMicroDB: a new Open Source lightweight Database Engine for J2ME Mobile Devices by Alier, M.; Casado, P.; Casany, M.J., Multimedia and Ubiquitous Engineering, 2007. MUE apos;07. International Conference on Volume , Issue , 26-28 April 2007 Page(s):247 – 252.*

16. Philipp Bolliger and Marc Langheinrich, *Distributed Persistence for Limited Devices, Philipp Bolliger and Marc Langheinrich, Inst. for Pervasive Computing, ETH Zurich, Switzerland*

17. Hassan Artail, Manal Shihab, Haidar Safa 2008, *A distributed mobile database implementation on Pocket PC mobile devices communicating over Bluetooth, Hassan Artail, Manal Shihab, Haidar Safa, Department of Electrical and Computer Engineering, American University of Beirut, P.O. Box 11-0236, Riad El-Solh 1107 2020, Beirut, Lebanon, April 2008*

18. Eric Falsken , 2008, *Enabling the Mobile Enterprise with db4o, By Eric Falsken, db4objects Inc., 2008,* **http://www.db4o.com/about/productinformation/whitepapers/db4o Whitepaper - Enabling the Mobile Enterprise with db4o.pdf**

19. db4objects Inc., 2008, *db4o: Java & .NET Object Database - Benchmarks: Performance advantages to store complex object structures, db4objects Inc., 2008*, **http://www.db4o.com/about/productinformation/benchmarks/**

20. *McObject Benchmarks Embedded Databases on Android Smartphone*, **http://www.mcobject.com/march9/2009**

21. Database Options for the Mobile Application Developer by Bryan Morgan, Jul 19, 2001, **http://www.informit.com/articles/article.aspx?p=22285&seqNum=4**