PEMOCO: AN INFRASTRUCTURE FOR PERSONAL MOBILE E-COMMERCE FOR JAVA-ENABLED SMART PHONES

By

TAPAN DIVEKAR

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

I dedicate this thesis to my family.

ACKNOWLEDGMENTS

This thesis is a result of contribution and support provided by many individuals. I would like to thank all of them for helping me to reach an important milestone of my career. First I would like to thank Dr. Abdelsalam Helal, my advisor, whose innovative ideas and constant motivation helped me to accomplish the thesis. I also wish to thank Dr. Joachim Hammer and Dr. Sanguthevar Rajasekaran for serving on my thesis committee.

I am grateful to Harris Communication Laboratory and my friends who provided constant support. I am also grateful to the coffee pot, which gave me resistance throughout the days and nights. Last but not the least I would thank my family for their constant encouragement and for directing me toward a successful career.

TABLE OF CONTENTS

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

PEMOCO: AN INFRASTRUCTURE FOR PERSONAL MOBILE E-COMMERCE
FOR JAVA-ENABLED SMART PHONES

By

Tapan Divekar

December 2001

Chairman: Dr. Abdelsalam (Sumi) Helal
Major Department: Computer and Information Science and Engineering

The birth of mobile computing has certainly given new dimensions to the concept of computing. The emerging portable mobile devices have given the power of ubiquity to users, with anytime, anywhere access in all situations. These devices have brought computing and information closer to ordinary users, which will enable them to transact on e-commerce applications right from their personal devices.

Mobile Commerce (M-Commerce) is an emerging field incorporating mobile devices, appropriate middleware, and the use of wireless networks and applications. Location-aware applications seem to play an important role in differentiating services from one another. Discovering services for a user in a new location or any specified position is one of the key requirements for M-Commerce.

This thesis proposes an infrastructure for the development of end-to-end M-Commerce applications on java-enabled Smart Phone platforms. This infrastructure shall aid in development of location-aware M-Commerce applications. We focused on buying,

selling and auctioning of tickets as a main application domain. The mobile user is equipped with a smart phone, through which he can sell or buy a ticket. A major part of this infrastructure is a small version of an HTTP server built as a proxy to the phone. The server handles HTTP requests and supports both push and pull mechanisms for selling and buying tickets.

The thesis also proposes an XML based representation for tickets, which will allow efficient representation of tickets and also would be easy to understand. The ticket structure features the name of the game, participating sides, venue date, venue time, auction information and the price of the ticket.

To demonstrate the end-to-end depth of our infrastructure, we have prototyped a multi-broker system in the fixed network. The brokering system is responsible for having matchmaking between sellers and buyers with “location” as an attribute.

Finally, we provide a case study of location-based ticket transactions using our system. We studied the scalability of our infrastructure in the context of this case study and measured the scalability of the micro HTTP server in terms of the number of simultaneous requests it can handle. We also studied the search time on the storage structures. We also compared the push vs. pull approaches in terms of performance.

CHAPTER 1
INTRODUCTION

Background

The future of business environment is aimed toward providing services even to people on the move. With the rapid growth in mobile computing, mobile devices play an important role in the lifestyles of many. In addition, the growth of e-commerce in the recent years is also considerable. The above two factors lead to the growth of Mobile Commerce.

Mobile Commerce [1] is an emerging field incorporating mobile devices, appropriate middleware, and use of wireless networks and applications, which are built using the above tools. Location-aware applications help differentiate services from one another. Discovering services for a user in a new location or any specified position is one of the key requirements for M-Commerce.

Development of Ideas

In this new era of mobile computing users with handheld devices are constantly on the move. In some situations the user's request might depend on their physical location. There could also be situations where the user is aware of his current location and is looking for services in a particular area [2, 3].

Thus location could be an important ingredient in Mobile Commerce applications. The services that we developed as a part of this thesis are ticketing applications. With the

help of a mobile device, in our case the smart phone, we supported the following 3 options:

- Buying
- Selling
- Auctioning

To support the above applications, we built three applications on the phone. The auctioneer application is the one through which the clients can access the *buy* or *sell* options. The µHTTP server is a smaller version of a normal HTTP server, to handle simple requests. This server is used in push-type of buyer applications as explained above.

The publishing client is the interface through which the seller can make various formats to specify the details of his ticket. The representation used is XML [4] (Extended Markup Language) with predefined tags. The publishing client has options to insert or delete tags. The architecture consists of a collection of brokers in specific areas, normally closer to the base station. A request from the client is forwarded to the appropriate broker, which checks its internal tables and queries the appropriate brokers.

The seller first makes the representation of the item he wishes to sell using the publishing client. After the seller makes this attribute file, he gets on the auctioneer application and then sells his item. Every smart phone is registered to a particular broker depending on the physical location of the smart phone. Thus the item the seller wants to sell gets registered with the broker. The broker maintains appropriate data structures to store the items. Whenever the buyer makes a query the broker checks to see if there is a match and then queries the clients for data.

The buyer first makes his choice of naming scheme and then specifies attributes like location in the form of city, street, type of ticket and the participants (players). The buyer has a choice of finding closer people by querying people associated with the broker. This gives the approximate degree of closeness and facilitates a location-based query. This query is sent to the local broker, who forwards it to the appropriate brokers and sends the reply back to the buyer.

We also built an auctioning scheme, in which the seller can specify his requirements of price and time. The buyers can then vote their price for the item. The intermediate broker calculates the best buyer among the bidders and notifies the winner of the auction.

Chapter 2 focuses on the technologies that are required for this thesis and how they are essential building blocks for the infrastructure. Chapter 3 describes the architecture of our infrastructure in the network. Chapter 4 explains the implementation details of the architecture. Chapter 5 deals with performance issues. Chapter 6 includes conclusions derived from this thesis and also talks about the highway to the future.

CHAPTER 2
RELATED WORK

This architecture is targeted to exploit location-based ticketing applications for smart phone customers. This section covers some of the components required for this thesis.

Pervasive Computing

The growth of mobile devices has also unearthed certain applications that are not up to the expectations of pervasive computing [5, 6]. People have different views for the exact meaning of pervasive computing. Pervasive computing deals with how people view mobile devices, and their use in specific environments. It also deals with the way the applications are created and the way they are deployed to accomplish their task. Third, pervasive computing talks about the changes of the environment due to the emergence of new functionality.

Mobile Commerce

With rapidly expanding development of wireless infrastructures, mobile computing has become an essence in the day-to-day life of a common man. The development of electronic commerce has also been substantial over the previous few years. The merger of mobile computing and electronic commerce resulted in the birth of a new field called as M-Commerce.

Several definitions have been proposed for M-Commerce. The Mobile Commerce report at Durlacher Research laboratory [1] defines Mobile Commerce as "*any transaction with monetary value that is conducted via a mobile telecommunications network.*" This definition is a subset of e-commerce applications in the B2B and in the B2C area. The definition tries to define a boundary between mobile applications and Mobile Commerce applications. One application may be an M-Commerce application in one context and not in another context where it uses some services that make it fall into the Mobile Commerce category. For example, The SMS messaging system will definitely not be M-Commerce where simple SMS messages are from person to person, but SMS messages from an information service provider that are charged at a premium rate will represent M-Commerce. Thus services make a distinction between mobile applications and Mobile Commerce applications. Find services in a new location is one of the critical requirements in M-Commerce.

Another view of Mobile Commerce was presented by John Fallon and Guy Singh of Baltimore Technologies [7] in which they have classified the following applications that could fall within the domain of M-Commerce into four broad categories as shown below:

Entertainment. A user pays to be able to access some entertainment items (downloading a song or movie clip or playing games like chess).

Communications. Communications applications include Short Messaging (e.g., SMS), Unified Messaging, E Mail, Chat Rooms and Conferencing. Users will be willing to pay for these services if they the service provider is willing to make a commitment that a consistent level of quality, reliability and security will always be maintained.

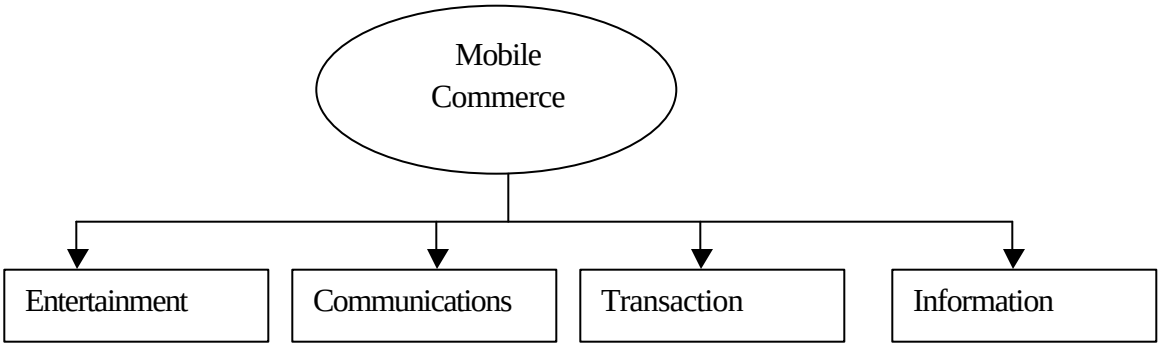

Figure 2-1. Mobile Commerce

Transactions. Some examples of applications involving transactions are Banking, Broking, Shopping, Auctions, Betting, Booking and Reservations, and the Mobile Wallet.

Information Services. Mobile information services are very useful for mobile users. Getting timely and accurate information about a an essential item could be so useful or critical that the user may even be willing to pay for it, if information provide can be held responsible for the timeliness and accuracy of that information. Information Services include News, City Guides, Directory Services, Maps, Traffic and Weather, Corporate Information and Market Data.

According to Aphrodite Tsalgatidou of the University of Jyvaskyla, Finland, a mobile e-commerce transaction is any type of transaction of an economic value that is conducted through a mobile terminal that uses a wireless telecommunications network for communication with the e-commerce infrastructure. Mobile Electronic Commerce refers to e-commerce activities relying solely or partially on mobile e-commerce transactions.

Mobile Commerce applications that combine the advantages of mobile communications and e-service would certainly be successful but would need additional

services, which would fully explore its potential applications. The attributes that make up today's Mobile Commerce applications are as follows:

Ubiquitous Access. It is said that people primarily work in a world of shared situations and technological skills. Computers today are isolated and isolating from the overall situation and is an important entity in our daily work. Comparing with other tools that disappear from our awareness, the computer is such a tool that often remains the focus of attention.

Ubiquitous computing aims at creating an environment where many computers are available, but making them effectively invisible to the user. Ubiquitous computing impacts all areas of computer science, including hardware components (e.g., chips), network protocols, interaction substrates (e.g., software for screens and pens), applications, privacy, and computational methods. Ubiquitous computing [8] envisions a world of fully connected devices, with cheap wireless networks everywhere. It postulates that you need not carry anything with you, since information will be always easily accessible everywhere.

There are numerous challenges proposed for ubiquitous computing like the machine address, which will change in different networks. Also, the underlying infrastructure to handle a wider bandwidth for wireless networks poses a significant challenge.

Today's mobile phones have made ubiquity a reality. They are used in M-Commerce applications to give an added advantage of ubiquity. This mobile device can fulfill requirements for real-time information and communication everywhere, independent of user's location.

Reachability. With the handy mobile phone people can be reached anywhere anytime. They can chose to be available anytime anywhere.

Security. Security is one of the key issues in any kind of network. Security for mobile networks is also emerging in the form of SSL (Secure Socket Layer). Use of smart cards in the mobile device provides authentication of the user and provides security better than that provided by fixed Internet environment

Ease and Convenience. Data at one's palm gives the freedom of convenience. There are certain characteristics, which could be essential for the future of Mobile Commerce:

Location. Adding a new attribute of physical location will add a new dimension to applications for M-Commerce. The attribute-location will help in giving relevant services to a user. A prominent example would be for a visitor who would like to eat in the closest restaurant. The attribute of location would help the associated application to better serve him.

Personalization. Though personalization has been developed there are certain aspects, which still need improvement. The new needs for payment mechanisms along with the availability of personalized information may have some uses. There could be a personal profile in which user specifies his interests and information about himself. This could be beneficial in providing relevant services to the user. Sharing only the information of the user's choice provides some kind of privacy.

Internet Connectivity. The growth of GPRS has enabled fast connection to the Internet. WAP enables having a micro browser on the phone to connect to the Internet.

Service Discovery

Services could be termed as facilities available to a computing device [9]. Being on a network, a workstation has access to nearby printer and scanner. If this terminal is connected to the Internet the domain of services extends to having online services like shopping for books, tickets. Being composed of enormous heterogeneous services, the massive Internet compels a service discovery to be an efficient one.

Service discovery in the context of Mobile Commerce could be best explained with an example: Imagine being on I-75 driving up to Tallahassee for a Gator-Seminole encounter. You are equipped with a smart phone and on board PC, which communicates, with PCs in another cars to have important information look up and distribution. Information sharing amongst all spectators could be could be made effective by passing information regarding nearby gas stations, traffic information.

The idea is to have information in the form of services, which could be static (in the form of a desktop PC) or moving (in the form of mobile devices). The difficulty lies in locating such a service. The evolution of dynamic service discovery in such of situation has been beneficial.

A service would be selected automatically for a job, taking into consideration its physical location, previous history and various other semantic information. Some of the important characteristics of a service discovery protocol could be summarized as follows:

- Efficiency with respect to time required to detect the service and the cost associated.
- Mobile communication has just evolved and there might be disconnections. Thus a stronger robust protocol is necessary.
- Service lifetime management: In a dynamic environment services might be available only for a particular amount of time.
- The services should be secured and not intervened by a third party.

Jini: Java-Based Service Discovery

Jini [10] is a distributed service-oriented architecture developed by Sun Microsystems. Jini network technology provides a simple infrastructure for delivering services in a network and for creating communication among devices/ programs, which may be implemented in either hardware or software. A Jini federation could be defined as a collection of JINI services. Jini is designed to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

Jini Lookup Service (JLS) is the central co-ordination point between end users and service providers, which maintains dynamic information about the available services in a Jini federation. A service enters a Jini federation by registering itself with one or more look up service. A JLS could be located using a multicast (if its address is unknown) or using unicast (if its address is known). Group names may be associated with JLS so that a service that is registered making it easier for services which are registered with one or more JLSs. When a Jini service wants to join a Jini federation, it first discovers one or many Jini Lookup Service from the local or remote networks. The service then uploads its service proxy (i.e., a set of Java classes) to the Jini Lookup Service. The clients download this proxy and invoke the appropriate (service) methods from it. A service client can invoke print requests to a PostScript printing service even if it does not have any knowledge about the PostScript language.

A user searching for a service in the network multicasts a query to locate the JLS. A remote object is downloaded to the user's machine if a JLS is found. The user then uses this object along with attribute matching to find out its required service. The proxy

of the queried service is downloaded to the client, through which he calls the appropriate interfaces.

The existing service discovery protocols use typical attribute or interface matching to compare existing services in the network. Service Location Protocol (SLP) assumes the existence of an underlying Internet Protocol based communication mechanism and uses User Datagram Protocol (UDP) to communicate.

Jini and SLP are based on centralized look up schemes. This improves the scalability of the two but has the single point of failure problems. Salutation, another discovery protocol, in contrast to Jini is a lightweight protocol and makes the least assumption of the underlying protocol stack and computing resources making it easy to port to handheld devices. Jini, which is Java based requires considerable computing resources to function properly.

Drawbacks of Service-Discovery Protocols

Inability of Rich Representation. M-Commerce defines various kinds of services, which may be heterogeneous in nature. The inability of service discovery protocols to represent these services properly creates difficulties in attribute matching, especially in terms of account distance, disconnections and other performance, efficiency related parameters, which may come into account when mobile devices are used as clients.

Lack of Inexact Matching. The Jini architecture defines service functionalities and capabilities in Java object interface types. Service capability matching is processed in the object-level without taking into account certain parameters. The following example illustrates the fact. The generic Jini Lookup and other discovery protocols allow a service client to find a printing service that supports color printing, but the protocols are not

powerful enough to find a geographically closest printing service that has the shortest print queue.

Location Service

The rapid growth in mobile computing requires corresponding change in the applications built. Having the ability of moving from one place to another, the mobile devices have added a new dimension in the form of physical location, bringing about what is known as “location awareness” [11, 12]. Location awareness has a characteristic of privacy, which may be significant in different applicative domains. It is thus essential to build a model, which takes into account privacy and functionality. Location awareness is incomplete without an entity that specifies the physical location of the object, and also grants access rights. Such an entity—location service supports location awareness. Following are some of the location-based systems:

Active Badge

Active badge is useful for a fixed environment, which is populated by an array of sensors. A customer wearing an active badge moves through the environment, whereby his badge transmits code representing his identity. The sensors collect this information and record it. Applications wanting to query the user location can query the location database. Active badge has a disadvantage that it can be used only in fixed environments. Olivetti research group has built a location aware service using active badges. Based on the client-server approach, the functionalities are divided between different servers:

- Location servers collect badge sightings from different sensor networks. It maintains a list of sensor ID, badge and a timestamp.
- A name server-mapping badge ID to the name of the person.
- Message server co-ordinates message delivery to all the badges.
- An exchange server covering issues related to security and access control.

Location Awareness in HP's CoolTown Project

CoolTown [13, 14] offers a web model for supporting nomadic users; based on the convergence of web technology, wireless networks and portable devices. The basic idea in CoolTown is to tie web resources (in the form of URLs) to physical objects and places, and how users interact with resources using the portable appliances like laptops to watches. Enabling the automatic discovery of URLs from our physical surroundings, and using localized web servers for directories, they create location-aware but ubiquitous systems.

Customized services are provided based on the knowledge of the user's location. CoolTown defines two types of location [14]:

Semantic location specifies the position of an entity within a larger construct called as space or a region. As explained earlier this is a contextual representation, which gives some semantic issues in addition to location and thus helps in find appropriate services. An example of a space is a conference room, or a shopping mall, or a bus stop. This space carries information about the local environment and resources. A space is represented by a Web page.

Physical location specifies the absolute representation of an object, either in the form of coordinate based system like GPS or cell phone triangulation. An object location may be given by a set of coordinates such as a (latitude, longitude) pair. This location can be provided with a varying degree of precision. The object location can be used in connection with global services that personalize information based upon the location. All objects in CoolTown may have both Semantic and Physical location information associated with them. The Semantic or Physical Location Information might be used depending on the context of use by the service provider. A Yellow Page service requires

the user's current zip code or city name whereas another service might need characteristics of the space in which the user is.

The CoolTown project [15] also has certain location agents, which translate requests. For example: a user in a conference room requesting for a pizza. The location agent translates the physical location of the user in the form of region or area of the person, namely the ZIP code.

Beacons are one mechanism for providing the semantic location of a space. An IR or RF emitting device sends out a beacon, which consists of a URL. A Personal Access Devices (PAD), such as a WAP enabled cellular phone, which is in range of such a beacon, will be able to access the Web page pointed to by the URL. This Web page provides access to location descriptors identifying the semantic and possibly the physical location of the space and associated services.

CoolTown also deals with the issue of privacy. Clients in CoolTown can access services without knowing where they are there. A service might release information about itself.

Location Models

We define location space as a data model that can adequately represent locations of fixed and mobile objects. We can have two ways of naming this model: One is an n dimensional co-ordinate system-geometric and another could be one where we could specify relationships—symbolic.

Ideally the symbolic and geometric systems have to be used independent from each other. The advantages of either could be improved by combining the above two approaches.

Symbolic Model

Symbolic model uses an abstract way to represent objects. Common way of representing position could be “Harris Lab,” “Hub,” etc. Since the objects bear an idea of “contained within” this could be represented in mathematical form in the form of sets. The areas or regions defined using this approach could be overlapping or non-overlapping.

The advantages of using symbolic naming scheme would be due to its ability to access locations by name. This would facilitate location awareness and access control. The hierarchical representation could be beneficial in terms of scalability and manageability.

The drawback of this approach could be extra processing that is required due to an additional layer of indirection and management of such models, which might be cumbersome.

Geometric Model

In this model, locations are represented as points, areas or volume within the co-ordinate systems. Such locations are described by sets of co-ordinate tuples. Geometric models are advantageous as far as accuracy of information unless there is loss of data during conversion from one co-ordinate system to another. The co-ordinate based system provides flexible way of retrieval and are typically re-usable. The only difficulty with it is being weakly structured efficient design is difficult. It may be necessary to convert information from one system to another. Additional information might be needed to extract useful data from the co-ordinate systems.

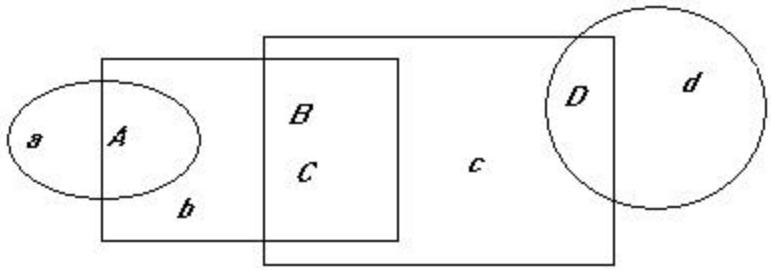

Figure 2-2. The Zone and the Cell Location Model

These models are one way of representing symbolic models. The cells depict one way to represent location in terms of well-defined geographical area known as cells. A to D are the cells in figure 2-2. The three shapes of figures show three different types of sensor systems used. As shown above the cells depict overlapping areas since the size and shape of the area covered by one type of sensor system could be different.

Zones are defined as the non-overlapping areas in the above cells. Thus *a* to *d* depict zones which are non-overlapping. These zones might cover more than one cell. The zone is advantageous as it gives better accuracy and therefore space computations can be distributed. However the shortcoming of this model is that there is no notion of abstraction or multi resolution processing and zone space for one located object might be different from another's if both are visible to different location sensor systems.

The location model is a model, which can be ordered with respect to other location domains. This is ordered by the "contains" relationship.

The location of an object could be a hierarchy, which would be represented in the form of a tree. The model utilizes predefined set of domains to represent locations of

predefined objects. If a location object is a member of a particular domain then it is a member of the domain of each of its parent.

It can be inferred from figure 2-3 that the UF domain includes all the domains lying below it and, the ECE and CISE departments share the NEB and also the zones that lie below it. The figure also depicts inheritance kind of relationship. UF inherits all the zones of NEB and CISE.

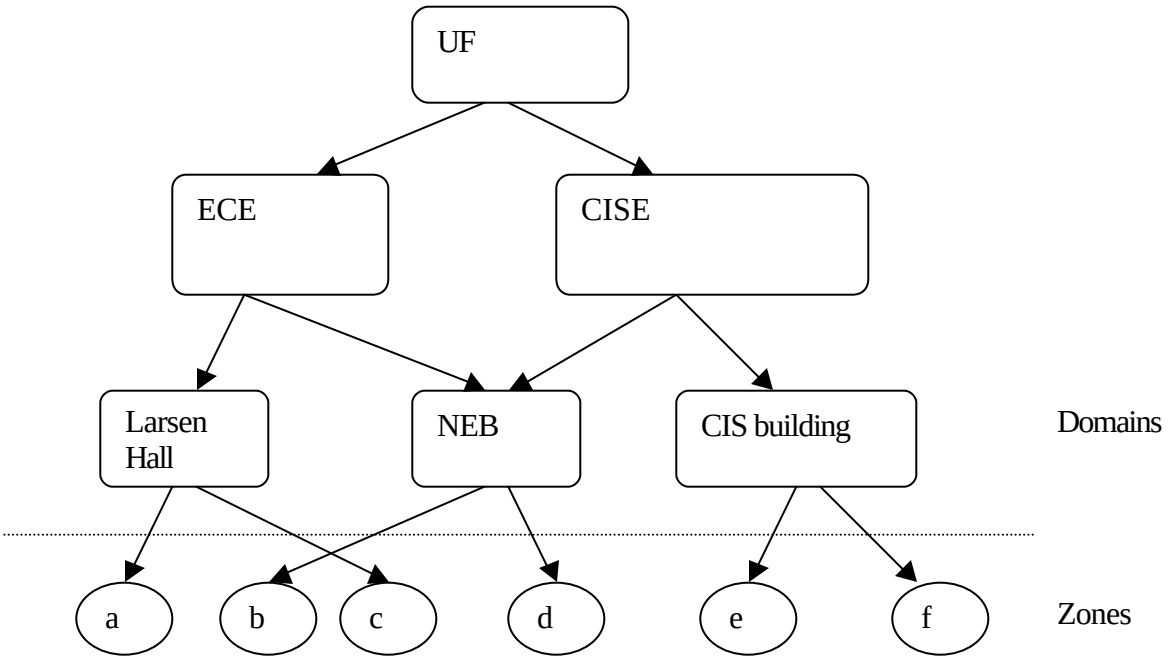

Figure 2-3. Location Model

Notation for Location Information

It is essential to develop a notation [11] for the textual representation of the above addressed naming schemes.

The notations [16] defined shall be consistent and be easily understood by a common user. The naming schemes should be hierarchical, consistent to represent mobile

objects, which are changing locations frequently. Also they should be able to represent abstract locations and geometric positions. We can use the following expressions to represent simple locations.

The symbolic naming schemes could be specified in the form of a well-known position or either hierarchically. Geometric position can be specified in the form of well-defined co-ordinates. Common examples are E309 @Gainesville/UF/CIS, <5m@Gainesville/UF or <1m@WGS: 84(0.04 W, 51.3 N, 0).

Location-Area @ Position
Area-fixed point | zone | radius
Position-Symbolic |Geometric
Symbolic-Well known position | Hierarchical

Figure 2-4. Naming Scheme

Extended Markup Language (XML)

According to the W3C Recommendation [4] “XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them.” XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO8879].

Each XML document consist of a collection of entities, which contain either parsed or unparsed data. The parsed data consists of data or markup in the form of characters. Markup is basically a tagged structure, which encodes a description of the document's storage layout and logical structure.

XML was developed with the following design goals:

- It shall be made as a component of the internet.
- It should be compatible with SGML.
- The domain of applications for XML shall be large.
- Ability to represent items efficiently and in a manner which is easily understandable.
- XML documents shall be faster to develop.

The general set of rules for an XML document's tags and attributes (i.e., the structure) is defined in a Document Type Definition (DTD). An XML document without a DTD is a “well formed” document, if the basic tag constraints are followed (e.g., every start tag has an end tag, etc.) With a DTD, validity of an XML document can be checked and it helps create a consistent structure for the type of document to be displayed

Tags identify the type of the element being described . The tag-value identify the attribute specifications of a particular type of element: A tag is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. The string “xml,” or any string which would match (('X'|'x') ('M'|'m') ('L'|'l')), are reserved for standardization.

A regular expression to specify XML data is (<tag>value </tag>)+ where value describes the data.

Smart Phone

Mobile phones have evolved over the past few years reaching across almost all of the individuals across the planet. The drawback of mobile phones has been their limited computational capability. Recently the evolution of smart phones has overcome this drawback by coupling the powers of mobile phones and Personal Digital Assistants (PDAs).

Mobile phones are no longer just phones, but they are mobile communication devices. By providing wireless Internet access, and limited computing capability, Smart Phones open up a realm of new possibilities. Thus Smart Phones are nothing but devices with the combined features of Mobile Phone + PDA + Handheld PC. Smart Phones coupled with Smart Web services will enable people to have a single mobile device instead of having so many devices to carry as in today with each device doing its specific function.

Java ™ 2 Platform, Micro Edition (J2ME™)

Seeing the scope of Java for a wide variety of devices, which differ in size and structures, Sun has developed three types of Java editions: Micro (J2ME) [17], Standard (J2SE), and Enterprise (J2EE). Each edition is a developer treasure chest of tools and supplies that can be used with a particular product:

J2ME specifically addresses the vast consumer space, which covers the range of extremely tiny commodities such as smart cards or a pager all the way up to the set-top box. Following are the characteristics of J2ME. Many of the characteristics are in accordance with Java.

- Consistent across wide variety of products.
- Portability of the code.
- Leveraging of the same Java programming language.
- Upward scalability with J2SE and J2EE.

J2ME enables device manufacturers, service providers, and content creators to gain a competitive advantage and capitalize on new revenue streams by rapidly and cost-effectively developing and deploying compelling new applications and services to their customers worldwide.

Why J2ME?

- Information Appliances and extensive growing Wireless Revolution.
- Everything Connected, Ubiquitous computing.
- Customizable, Personal Services, which will allow users to download new applications such as interactive games, banking and ticketing applications.

J2ME ™ Software Layers

In order to support the kind of flexibility and customizable deployment demanded by the consumer and the embedded marketplace in general, the J2ME architecture is designed to be modular and scalable. There are three layers of software built for the J2ME.

Java Virtual Machine Layer is an implementation of a Java virtual machine that is customized for a particular device's host operating system and supports a particular J2ME ™ configuration.

Configuration Layer defines the minimum set of Java virtual machine features and Java class libraries available on a particular category.

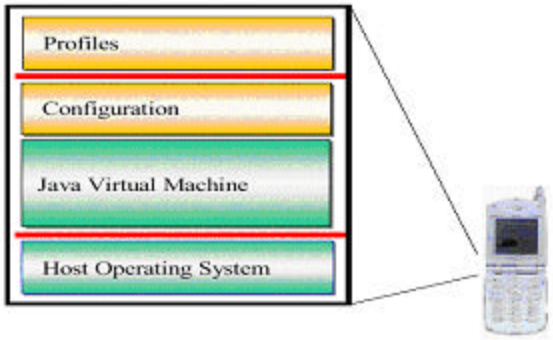

Figure 2-5. J2ME Software Layer Stack. Courtesy: Java 2 Micro Edition Technology for Creating Mobile Devices, White Paper, Sun Microsystems

Profile Layer defines the minimum set of Application Programming Interfaces (APIs) available on a particular group of devices, which are basically developed on the underlying configuration. In general a device can support multiple profiles on which different applications are built.

Java ™ 2 Platform Micro Edition (J2ME ™) Devices

J2ME specifically addresses the large, rapidly growing consumer space, which covers a range of devices (figure 2.6) from tiny commodities, such as pagers and TV set-top box, which is almost as powerful as a desktop computer.

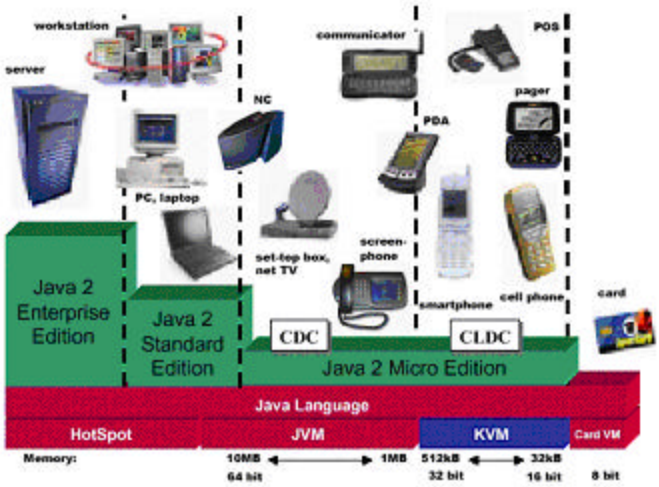

Figure 2-6. Applications of Java. Courtesy: White paper "CDC and Foundation Profile" at Sun Microsystems

Thus J2ME was developed to serve 2 kinds of products for the CDC and CLDC configurations. The difference between the above two categories is based more by the memory, bandwidth considerations, battery power consumption, and physical screen size of the device, rather than by its specific functionality or type of connectivity.

J2ME Building Blocks: Configurations and Profiles

To address diversity between different devices, modularity and customizability are two important features for J2ME.

Configuration and Profiles are two essential concepts are defined by J2ME for increasing customizability and extensibility. These configurations and profiles are defined through the Java Community Process (JCP).

J2ME Profiles

Application portability is a key benefit of Java technology in the desktop and enterprise server markets and is also a critical element of the J2ME ™ value proposition for consumer devices.

Profiles can serve the following two distinct portability requirements:

It provides the foundation to build a wide variety of applications like pagers, set-top box, cell phone, washing machine, or interactive electronic toy.

A profile may also be created to support a significant, coherent group of applications that might be hosted on several categories of devices. For example, in addition to having device specific profiles, it might be useful to have personal profile, which would keep some kind of personal information management or home-banking applications could be portable to each of these devices.

It is possible for a single device to support several profiles. Some of these profiles will be very device-specific, while others will be more application-specific.

J2ME Configurations

A profile is based on a configuration. A configuration basically specifies the Java programming language features supported, the Java virtual machine features supported, and also the basic Java libraries and APIs supported.

A configuration could be defined as the interface between the profile and the JVM of the particular device.

The intercommunication between certain devices (for cell phones, washing machines, and intercommunicating toys) would most likely be built upon the same configuration, the CLDC. To avoid fragmentation, there will be a very limited number of J2ME configurations. Currently there are two configurations:

Connected Limited Device Configuration (CLDC)

The market consisting of personal, mobile, connected information devices are served by the CLDC [18]. This configuration includes some new classes, not drawn from the J2SE APIs, designed specifically to fit the needs of small-footprint devices.

The CLDC configuration was built with the following objectives:

- To define a standard Java platform for small, resource-constrained, connected devices and moreover allow dynamic delivery of Java applications and content to those devices.
- Enable developers to develop applications on these devices.

The requirements of CLDC are:

- To run on a wide variety of small devices.
- To make minimal assumptions about the native system software available in CLDC devices.
- To define a minimum layer of Java technology, which shall be applicable to a wide variety of mobile devices.
- To guarantee portability and interoperability of profile-level code between various kinds of mobile (CLDC) devices.

The CLDC also aims to have a small memory footprint of no more 128 kilobytes for its implementation. It also assumes that applications could run in as little as 32 kilobytes of Java heap space.

The CLDC specification covers Java language and virtual machine features, core Java libraries (java.lang. *, java.util. *), input/output, networking, security and internationalization and does not include application life-cycle management (application installation, launching, user interface, event handling and high-level application model (the interaction between the user and the application).

Cell phones, pagers and personal organizers are examples of devices in this category and have very simple user interfaces compared to desktop computer systems, 128 kb of memory, and low bandwidth, intermittent network connections.

Connected Device Configuration (CDC)

The Connected Device Configuration (CDC) covers the market consisting of shared, fixed, connected information devices. CDC is a subset of CLDC and thus guarantees upward compatibility.

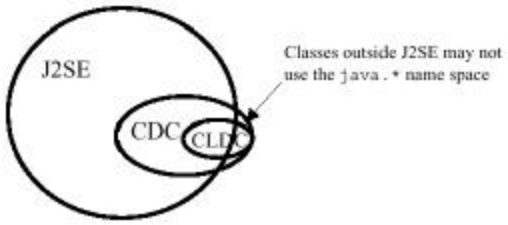

Figure 2-7. Relationship between Java and J2ME Configurations. Courtesy: Java 2 Micro Edition Technology for Creating Mobile Devices, White Paper, Sun Microsystems

Figure 2.7 illustrates the relationship between CLDC, CDC and Java 2 Standard Edition (J2SE). As shown in figure 2.7, CLDC and CDC inherit majority of their

functionalities from J2SE. Also there might be device specific functionalities, which might be handled by CDC and CLDC, which are not included in J2SE.

Specification of Java Platform Micro Edition (J2ME), Sun Microsystems, Inc.

Configurations and Java virtual machines are very closely related and are rather complex pieces of software. To incorporate a large number of configurations implies a large number of modifications to the internal design of a JVM. This would involve a huge amount of maintenance. Having a small number of configurations means that a relatively small number of Java virtual machine implementations can serve the needs of both a large number of profiles and a large number different device hardware types.

KVM Technology

The KVM technology [18, 19] is a compact, portable Java virtual machine specifically designed from the ground up for small, resource-constrained devices. The main goal in designing the KVM was to have the smallest nearly complete JVM, which would run in a constraint memory environment with about few hundred-kilo bytes of memory. The KVM was also designed to be highly portable, well commented, modular and customizable.

KVM is suitable for 16/32-bit RISC/CISC microprocessors and is applicable to digital cellular phones, pagers, personal organizers, and small retail payment terminals. The minimum total memory budget required by a KVM implementation is about 128 kB, including the virtual machine, the minimum Java class libraries specified by the configuration, and some heap space for running Java applications.

The functions of KVM differ as per the implementations on which it is used. Some implementations require KVM technology to give the ability to download and run dynamic, interactive, secure Java content on the device. In other implementations, the

KVM technology is used at a lower level to also implement the lower-level system software and applications of the device in the Java programming language.

Presently, the CLDC technology runs only on top of KVM technology, and CLDC technology is the only configuration supported by KVM technology. In future it is expected that CLDC technology will run on other J2ME virtual machine implementations. Also KVM technology may perhaps support other configurations as they are defined.

Mobile Information Device Profile (MIDP)

The Mobile Information Device Profile (MIDP) is a set of Java APIs, which together with the Connected Limited Device Configuration (CLDC), provides a complete J2ME application runtime environment targeted at mobile information devices, such as cellular phones and two-way pagers. MIDP covers issues related to interface, persistence storage, networking, and application model.

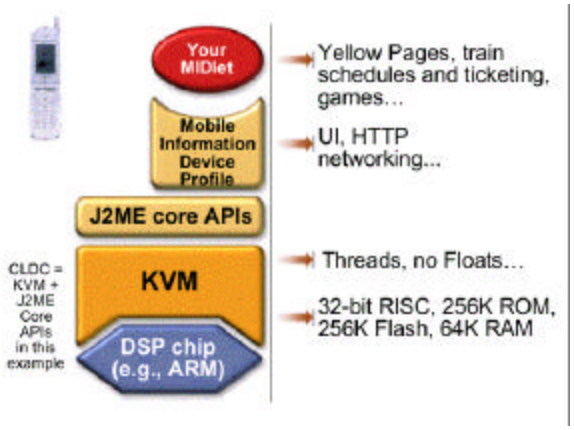

Figure 2-8. Wireless Device Stack. Courtesy: "Developing wireless applications using J2ME" by Bill Day, Sun Microsystems

The MID Profile provides a standard runtime environment that allows new applications and services to be dynamically deployed on the end user devices.

Connected Device Configuration (CDC) and the C Virtual Machine (CVM)

The Connected Device Configuration (CDC) is mainly developed for higher-end, emerging, next generation devices. Typically, these devices run a 32-bit microprocessor/controller and have more than 2.0 Mb of total memory for the storage of the C virtual machine and libraries. The main component of the CDC is the C Virtual machine (CVM). This virtual machine is intended for devices needing the functionality of Java 2 VM. CDC and CVM technologies are targeted for consumer electronic and embedded devices.

Foundation Profile

The Foundation Profile is a set of Java APIs, which, together with the Connected Device environment targeted at consumer electronics and embedded devices such as residential gateways. Connected Device Configuration (CDC), provides a complete J2ME application runtime; emerging, next-generation smart phones and communicators; and two-way pagers. The standard runtime environment allows new applications and services to be dynamically deployed on end-user devices.

The J2ME™ Wireless Toolkit is a set of tools that provides Java developers with the emulation environment, documentation and examples needed to develop CLDC/MIDP compliant applications. This chapter built a foundation for our design and implementation, which is discussed in more detail in the following chapters.

CHAPTER 3
ARCHITECTURE

The previous sections saw how the field of M-Commerce has evolved over the last few years. We also looked into the various technologies [20, 21] that form an essential ingredient in building our infrastructure. This section gathers all these technical tools to build a scalable infrastructure for buying and selling tickets over the smart phone.

Conceptual Framework

Figure 3-1 shows the conceptual framework for PEMOCO. The framework contains clients, which are residing on smart phones. The basic components are the clients (buyers and sellers) on smart phone, a network of brokers and the underlying network. The sellers make their ticket using the web-publishing client and then upload their tickets to the local broker for sale. The buyers make a query based on the location of the choice and can thus obtain a ticket. The micro HTTP Server built on the clients help in keeping buyers updated about information relating to tickets of their choice.

Overall Architecture

Figure 3-2 shows the overall architecture of PEMOCO. The client contains the Web publishing client, the micro HTTP Server that is built over CLDC. The client also contains persistent storage for ticket and *MyTicket*.

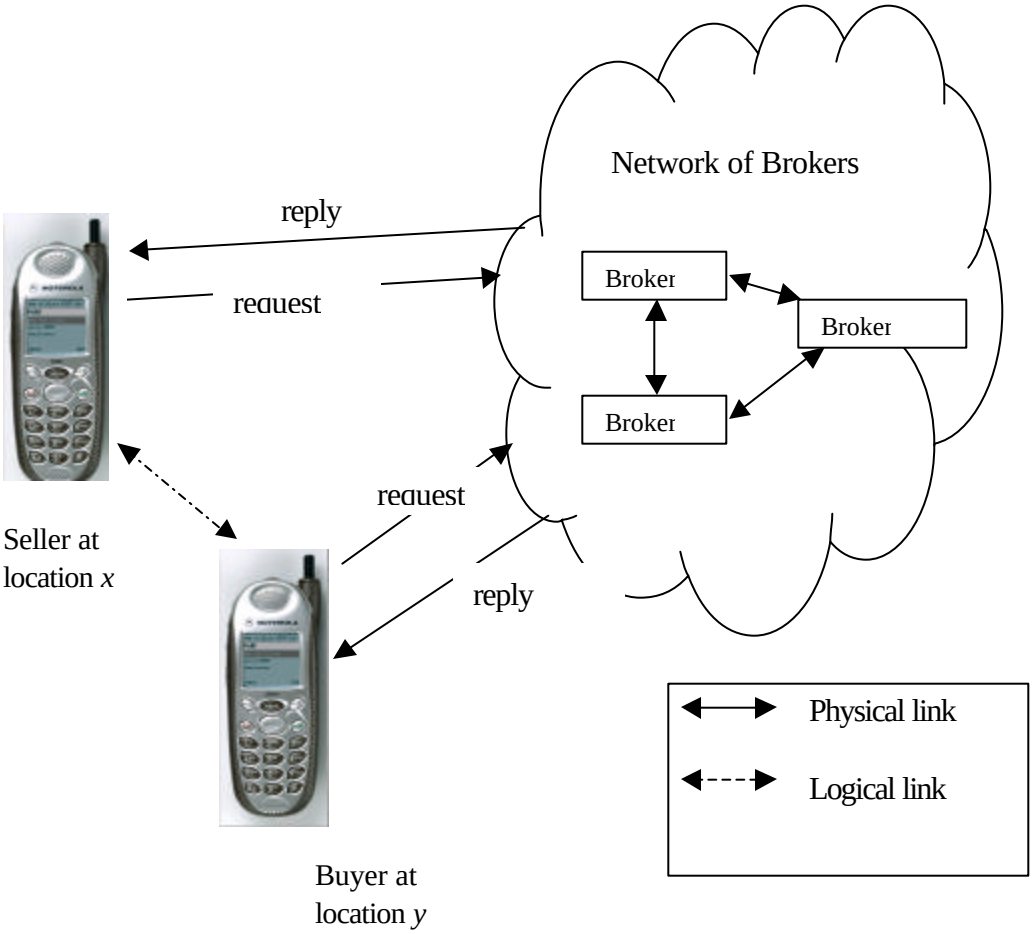

Figure 3-1. Conceptual Framework

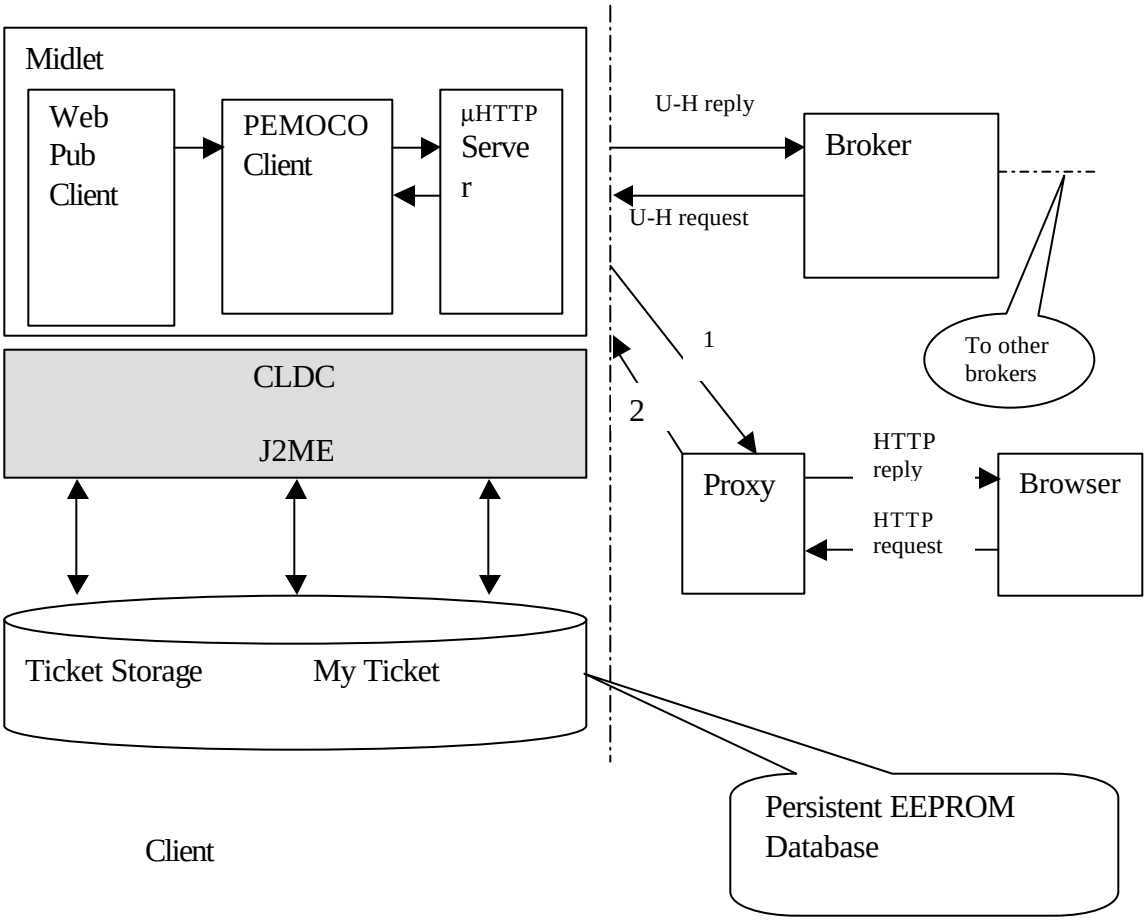

Figure 3-2. PEMOCO Overall Architecture (1 is U-H reply, 2 is U-H Request)

Micro HTTP Server Design

We designed a small HTTP server on the smart phone. The J2ME implementation supports the following API for networking.

We present here a hierarchy of the classes used. The generic connection framework is as follows:

- The connector class is a placeholder for the static methods used to create all the connection objects.
- The Datagram connection is derived from the Connection class and contains implementation for 'passive sockets' (Server Side Sockets) and also for 'active sockets'.
- Http Connection, which is lower in hierarchy, also inherits methods from Connection.

The 'Connector' class also contains methods to open a TCP socket but the current implementation of i85 [22] doesn't support server side sockets.

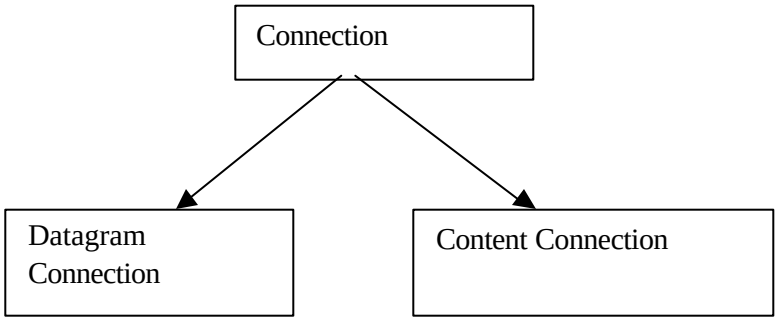

Figure 3-3. Connection Class Hierarchy

This absence of server side sockets has spawned a need to develop a new kind of protocol, which is explained in detail. Before we discuss about the protocol there are some key issues of networking specific to i85 phones:

- The implementation of *javax.microedition.io* and *java.io* restricts an application to 2 simultaneous connects.

- The timeout period for TCP implementation on the i85s is 180 seconds. The timeouts normally occur while attempting to open a connection to an unknown server or if server terminates abnormally.
- The maximum payload for outgoing UDP packets is 1472 bytes and incoming is 2944 bytes.

As shown in figure 3-4 HTTP [23] 'protocol' works on the TCP layer. The implementation of a HTTP server requires a TCP server socket, which is continuously listening for requests at the server. Server side TCP sockets could catch a HTTP request from a browser.

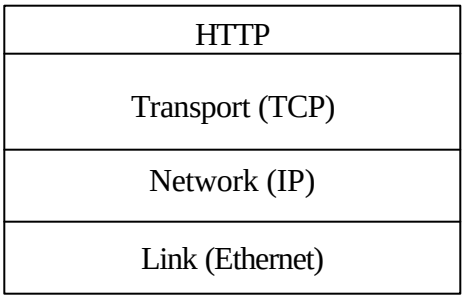

Figure 3-4. Protocol Stack for TCP/IP

As discussed earlier, the i85 phones do not support server side TCP sockets, which is one of the key requirements to obtain HTTP kind of requests. We built our HTTP protocol over UDP (HTTP-UDP) to support HTTP kind of requests. Basically each HTTP header is embedded inside a UDP packet. As shown in figure 3-2, we have two ways of querying the server, one through the proxy and other directly to the phone.

When the client is using the browser, the requests are first sent to the proxy, which is basically a servlet, running on a predefined host. The client side browser essentially should have the capability to specify the proxy server's address. Any request that the user makes through the browser essentially goes through the proxy. This is an

HTTP GET request. The proxy server then contacts the phone using the U-H protocol, which we developed. This protocol is UDP-UDP communication, which has HTTP packets. The reply from the server then goes to the proxy server, back to the client.

The second kind of architecture is for peer-peer communication between the phones. A client on a phone can contact another phone, both of which contain µHTTP. The requests and replies both are U-H. More details about packets and protocols are covered in Chapter 4.

Location Awareness and Naming Schemes

As explained in the previous chapter naming is an important ingredient in the domain of location aware applications. We make use of different notations to represent names. In the following paragraphs, we discuss some of the defined naming schemes.

An Internet DNS uses path names in the format like *US/Gainesville/University_Avenue* identifies University Avenue in Gainesville, US. Though they are easy to name like "river," "room8" these are abstract locations, which are difficult to implement due to the lack of knowledge about them. Here we require definition in the form of *f(x)* where x is the location.

When we define location we should have fixed terminologies in defining them. We identify 2 types of granularities-abstract granularity refers to granularity where information is limited to one geographical scope. This limits the user visibility to a single geographical unit. Relative granularity is the other type, which bears the notation of "contained within." This increases user flexibility. In this schema units are related hierarchically to each other. For example 34th Street in Gainesville.

We have the following structure when we try to define locations. Figure 3-5 shows how the symbolic naming could be mapped.

The nodes of tree can be used to define a particular area of a region. How big is this region? The level of granularity can be a town, a county or a state depending on the application. In such a tree the child is a subset of the parent, meaning the child bears scope for only a small component (area) inside the parent.

For example considering our granularity as a county ->Alachua, FL (ABCD in figure), then we could have AB as Gainesville and CD as Ocala.

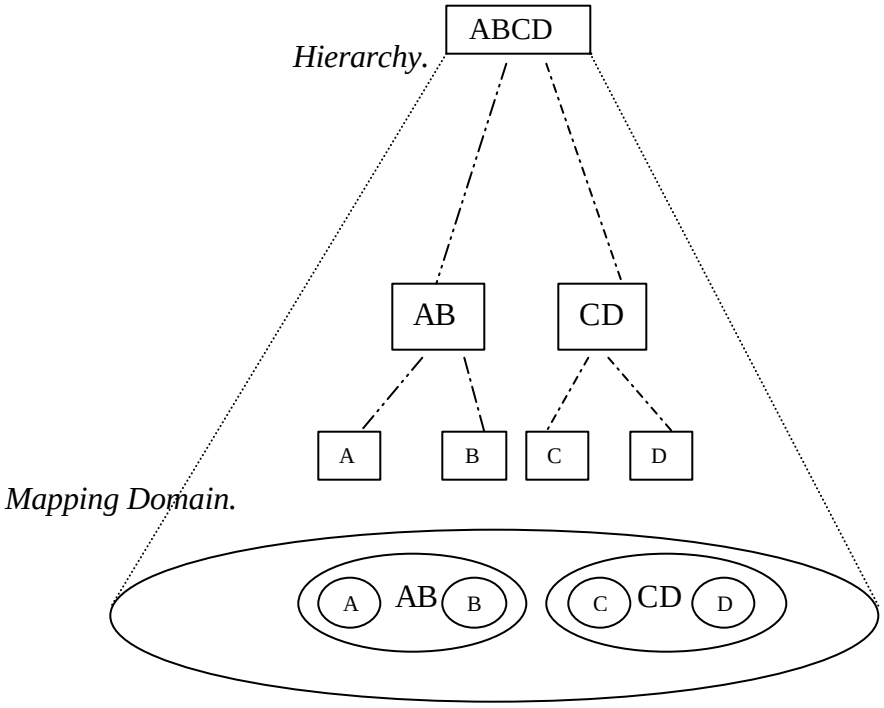

Figure 3-5. Symbolic Naming Scheme

'A' and 'B' could point to North Gainesville and South Gainesville respectively or divide them into "zip codes" of a location. This hierarchy shall prove useful in one of

the naming schemes we shall use. A query like *select all buyers from North Gainesville* or *select buyers from zip 32611* could be easily solved using this structure.

An ideal namespace is one that takes into account all of the ways of naming a location. We describe the notations we used with an example:

1. Gainesville/ University_Avenue, takes into account all locations in University Avenue.
2. Within x miles from 34th Street, Gainesville.
3. North Gainesville.
4. All locations in Gainesville.

We define here our scope of locations and naming infrastructures used. We define a location scheme for Gainesville, FL. Other schemes shall be based on the same architecture. Suppose we have a query of the type 2 miles from 34th Street, Gainesville. We systematically cut the map into rows and columns based on avenues and streets[1]. A hash table such as the one shown in figure 3-6 maps beginning of an address to a particular index in the table.

We assume that our element of granularity is 2 miles. When a user specifies within 2 miles we have to look for adjacent columns or rows and intersecting rows and columns.

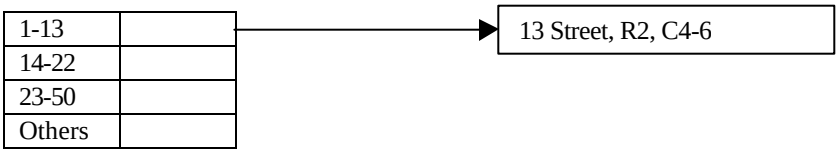

Figure 3-6. Data Structures Used in Naming Structures

[1] Avenues run perpendicular to Streets thus simulate a grid.

For example, for 34th Street we look for all places in R1 and also those appearing in columns 4 to 6. This can be inferred from the table 3-1.

We shall also have a similar type of structure for columns. Each of the leaf nodes of the above tree points to a base table, which contains metadata information as shown in table 3-2.

Table 3-1. Address of a Location and Mapping Index

Row	From Column	To Column	Index
1	2	4	2
2	5	7	1

Table 3-2. Metadata Information

Row	Column	City	Address	Associated Broker	IP
2	4-6	Gainesville	34th Street	Broker A	127.0.0.1
3,4	5	Gainesville	16th Avenue	Broker B	127.228.115.112

Table 3-2 is the base table, which is used in the look up of addresses. For the "within" naming convention we also have factors defining the degree of closeness. The degree of closeness is defined by two factors: match and unmatch. The weighted mean of these two factors helps in finding the closest brokers in the area. The details shall be covered in the chapter for implementation.

Client-Broker Communication

The client with his smart phone first contacts the local broker and downloads a form of his choice. This form is specific to the naming system. Four types of forms are supported currently as shown in figure 3.8.

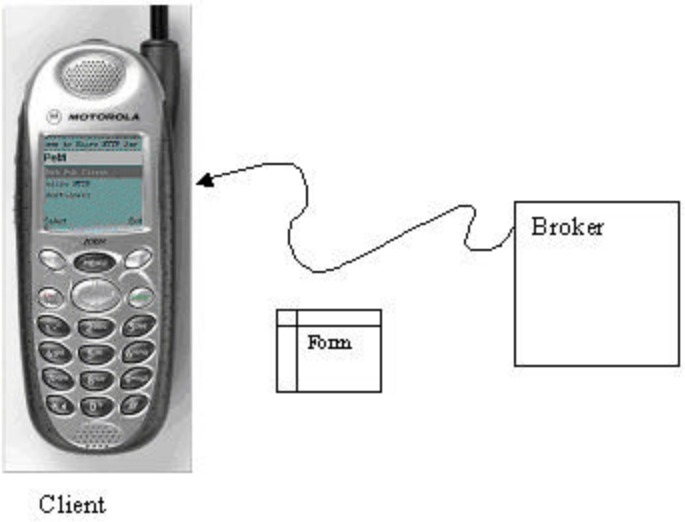

Figure 3-7. Client-Broker Communication

	Ticket type: Football Teams: Gators Location Parameters: City: Gainesville Area: University_Avenue

Form 1

	Ticket Type: Football Teams: Gators Location Parameters City: Gainesville Area: 4 miles of 34th Street

Form 2

	Ticket type: Football Teams: Gators Location parameters: City: Gainesville Area: North

Form 3

	Ticket Type: Football Teams: Gators Location Parameters City: Gainesville Default all locations

Form 4

Figure 3-8. Forms for Client

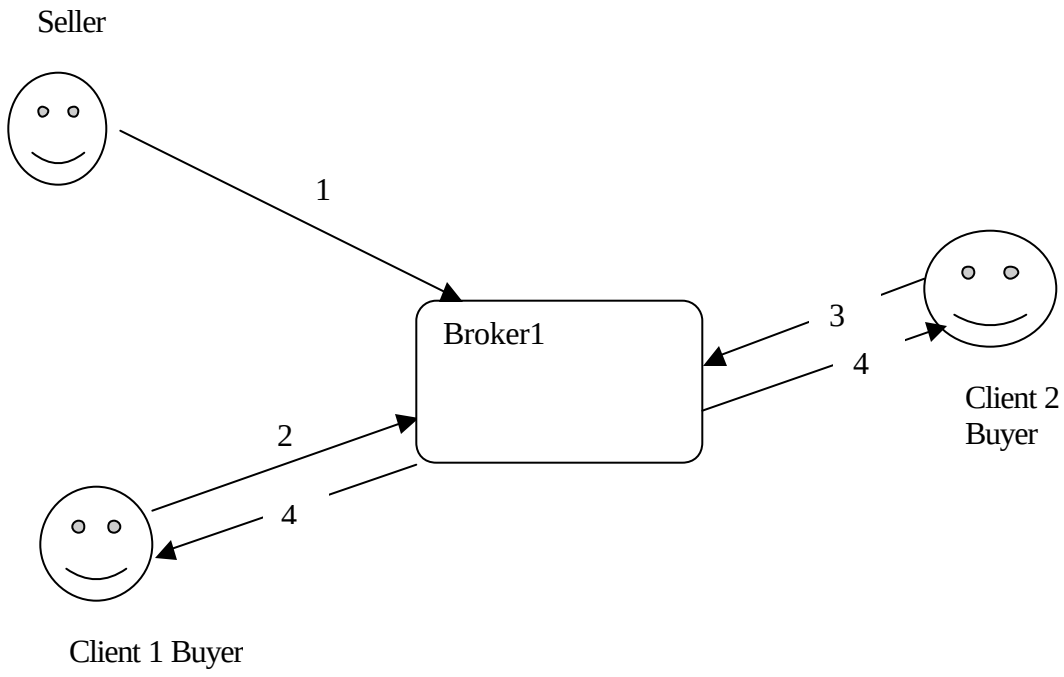

Figure 3-9. Auctioning

<u>Auction</u>

Sequence of events for Auctioning (Figure 3-9)

1. Seller registers his ticket to auction.

2. Client 1 requests for a ticket, which the seller is selling. Client1 gives his estimate for auctioning.

3. Client 2 requests for a ticket, which the seller is selling.

4. Broker announces the winner/loser result.

Storage of Data

We shall consider the following representation of an ideal ticket: User can make this page using his web-publishing client. This form, filled by the seller basically consists of attributes of a ticket (described later). The broker has a choice of data structures to choose from, to store data. We shall consider the following representation of an ideal ticket: User can make this page using his web-publishing client.

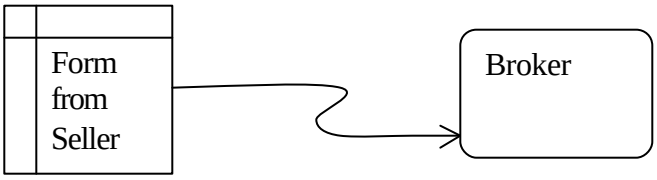

Figure 3-10. Seller-Broker Communication

```
<title> Ticket </title>
<ticket>
<type> BasketBall </type>
<players>Gator vs Seminoles</players>
<main>
 <date> 31st Aug,2001 </date>
<place>Gainesville, Fl</place>
<price>$50</price>
</ticket>
```

Figure 3-11. Ticket

The above figure depicts a typical structure of a ticket. The ticket is basically in XML format, using which attributes can be described using *tags*. The ticket has a title tag with the type of the ticket between the <title> tags. The type of the ticket is specified in

the <type> tags. The above figure is one for a basketball game. The next tag is for the participants of the game. These attributes are used while querying a ticket. Other tags are day, place and price. Following are different storage strategies [24] we surveyed for storing the ticket.

Edge Approach

The edge approach is basically a table approach. It represents the hierarchy in a tabular format. The index is based on [Tag, Data] for faster access.

Table 3-2. Edge Approach

Source ID	Tag	Child #	Target ID	Data
1	ticket	1	2	-
2	type	0	0	Basketball
3	main	1	6	-
4	game	2	0	"GvsS"
5	money	3	7	-
6	date	1	0	"31st Aug,2001"
7	place	2	0	"Gainesville, Fl"
8	auction	3	0	"Yes"
9	price	4	0	"$50"

Object Based Storage

Store XML elements collectively as an object inside an object or may be inside a file. They can be accessed using the offset from the current position. The Object based approach is shown in the figure 3-3.

Table 3-3. Object Based Approach

Offset	Information
0	Length=40, type, parent=nill, prev=nill, next=nill, first_child=40, last_child= 120, Attribute=basketball”
40	Length=20, main, parent=nill, prev=nill, next=“game”, first_child=100, last_child= Attribute=“NULL”
60	Length=20, game,parent=nill, prev=40, next=“money”, first_child=nill, Last_child=nill, Attribute=“UFvsSeminole”
80	Length=20, money, parent=nill, prev=40, next=nill, first_child=nill, Last_child= , Attribute=“NULL”
100	Length=20, date, parent=40, prev=nill, next=120, first_child= nill, Last_child= nill, Attribute=“31st Aug 2001”
120	Length=20, qty, parent=40, prev=100, next=140, first_child= nill, Last_child= nill, Attribute=“1”
140	Length=20, place, parent=40, prev=120, next=nill, first_child= nill, Last_child= nill, Attribute=“Gainesville, FL”
160	Length=20, auction, parent=80, prev=nill, next=180, first_child= nill, Last_child= nill, Attribute=“Yes”
180	Length=20, price, parent=40, prev=160, next=nill, first_child= nill, Last_child= nill, Attribute=“$50”

B-Tree

Instead of having relational offsets, the B-Tree uses absolute ones. The access time of this structure has proved to be the shortest and hence in our implementation we used this approach. The detail of how we implemented this is covered in the next chapter.

Web Publishing Client

Simple tags like <ticket> help publishing meta-data information about the ticket. The ticket is in XML format with user-friendly tags. When the user starts his web-publishing client, he has an initial empty screen and then he can chose from the list of tags to choose from. When the user chooses the inbuilt <ticket> tag, the following ticket appears on the user’s screen.

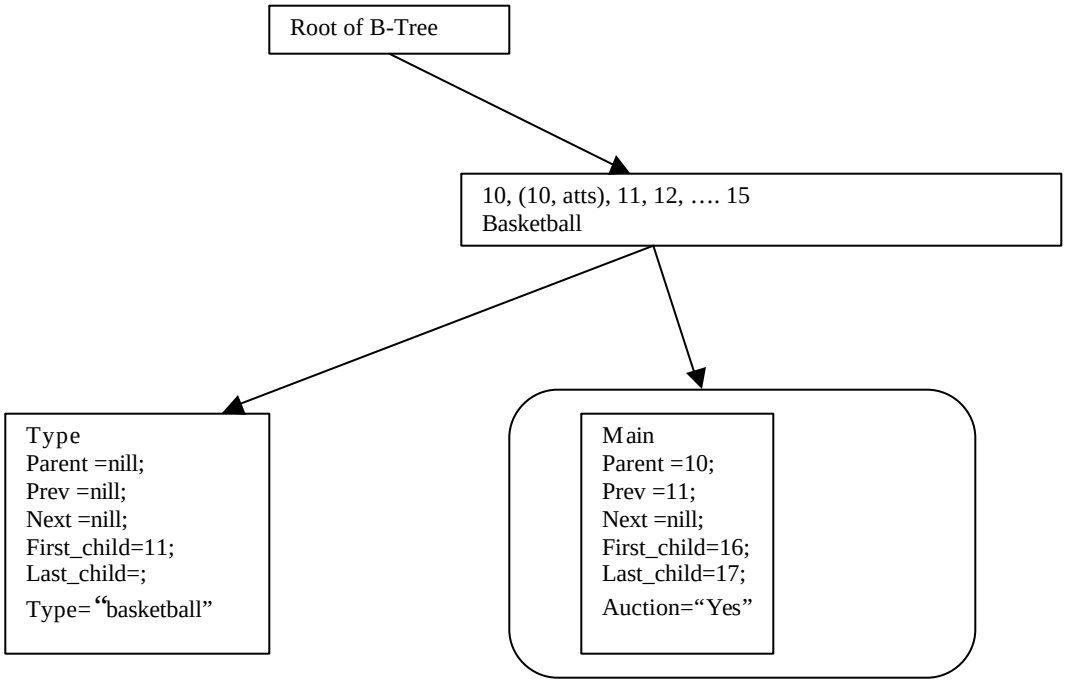

Figure 3-12. B-Tree

```
<title> Ticket </title>
<ticket>
<type> BasketBall </type>
<game>gators</game>
 <date> 31st Aug,2001 </date>
<place>Gainesville</place>
<price>$50</price>
</ticket>
```

Figure 3-15. Screen on the Web-Publishing Client

As a demonstration we showed Basket Ball as the commonly played game and hence the ticket to be quite common. Different tickets could be designed as default as per requirements.

Typing on the phone is not as easy as that on a keyboard. To incorporate efficiency and throughput in terms of time and user friendliness we take into consideration some minor details, which might prove useful. The parameters in the ticket have been so chosen taking into account the common requests from clients. Also the ticket metadata requires a "<" character, which is not currently supported on the phone's tiny keyboard. The client can then save the data giving it a certain filename. The seller could use this filename when he is trying to push his ticket data to the broker or if the broker needs to pull data from the seller.

In this chapter we saw the various design factors for our architecture. The next chapter deals with implementation details.

CHAPTER 4
IMPLEMENTATION

Micro HTTP Server Design

As discussed in section 3.1, the i85 phones do not support server side sockets, which made us come up with HTTP-UDP protocol. The idea is to embed HTTP headers inside UDP packets. The protocol basically adds TCP /HTTP kind of headers to UDP.

A proxy is needed if the client is a browser. This proxy would take in HTTP requests from the client (browser) and then use the new UDP/HTTP protocol to communicate with the server. If the client is a phone then it could make use of our UDP/HTTP protocol to communicate with the http server. Following figure depicts the 2 scenarios with different clients:

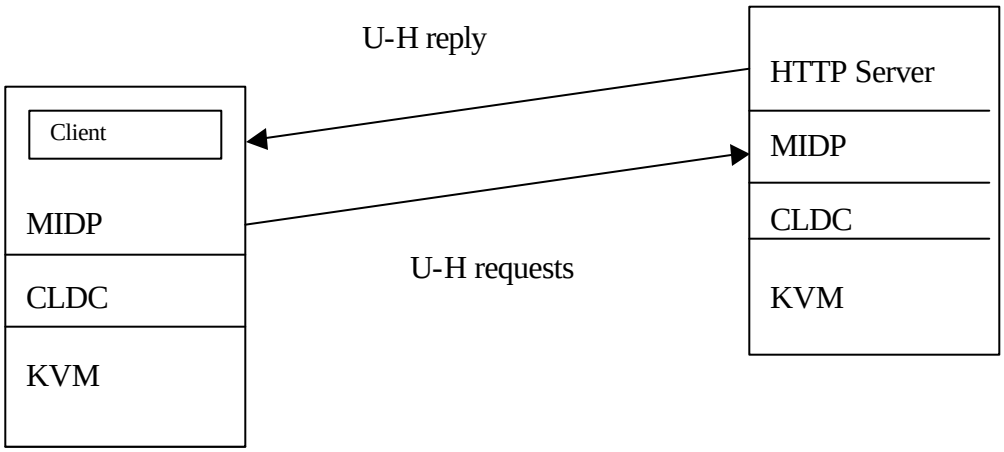

Figure 4-1. HTTP over UDP (Client Is Using a Smart Phone)

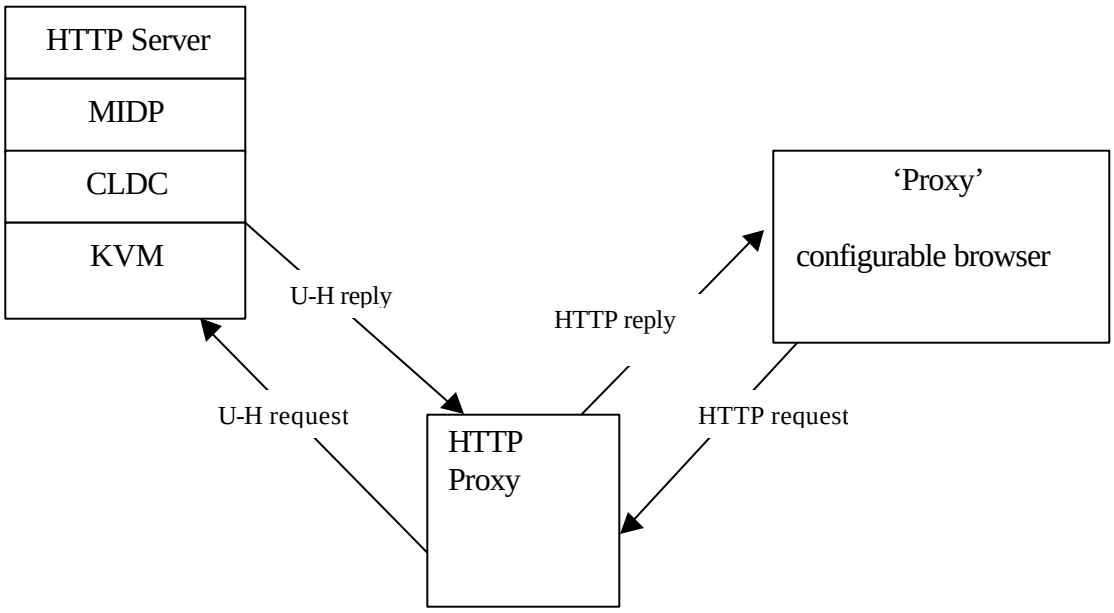

Figure 4-2. HTTP over UDP (Client Is Using a Browser. U-H stands for UDP-HTTP Protocol)

As shown in figure 4-1, the proxy takes in HTTP request parameters from the browser and then sends an UDP-HTTP request to the µHTTP server. The proxy is running on a specified server, which accepts GET and POST requests from the browser, parses these requests and sends it to the HTTP server. The server responds with UDP-HTTP replies, which the proxy parses and passes it in the form of HTTP packets to the browser.

The server listens on a specified port for any requests. This is basically a server side UDP socket through which it listens for requests. Requests could be a filename in which case the server opens the appropriate record storage, reads in contents of the file and sends it back to the requesting broker / proxy. Brokers contact the client (seller) through their (seller's) µHTTP server for any updates, like item being sold.

A stream showing this would look like this:

<item name, customer IP, customer name, price>.The following figure depicts the communication details [25].

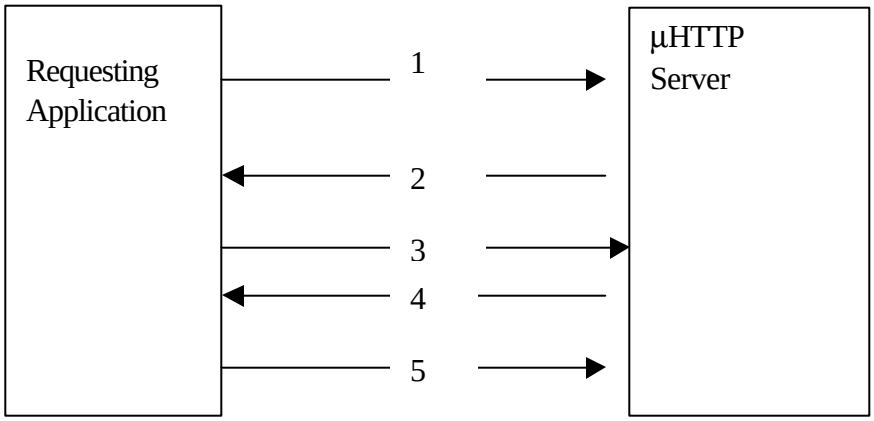

Figure 4-3. U-H Protocol Sequence

Sequence of operations:

- The requesting application makes an UDP request to the µHTTP server. This is the initial Connection-Request.
- The µHTTP replies with a Connection-Reply message.
- The application makes a Request-Session to the µHTTP server.
- The µHTTP replies with Response-Session back to the µHTTP server.
- The application makes a Finish request to the µHTTP server

Table 4.1 shows the opcodes for the communication.

Connection-Request. The header for a connection request looks like

10	Ticket length	Version no.	Optional

The opcode for connect is 10. The version number is used to maintain uniformity in the protocol. The current version is 1.0. Optional fields are used for future upgrades.

Table: 4.1. Code Table (Request Values)

Opcode	Definition	Meaning
10	Connect	Establish Connection
20	Reply	Reply connect
30	Get	Get parameters
40	Inform	Send data to the Server
50	Response	Response to the request.
60	Finish	Abort connection

20	OK/wait/version number less	Flag	Queue size

Connection-Reply. The opcode for Connection-Reply is 20. The second byte represents OK if the connection is accepted. If the queue size is full, the reply is WAIT. If the reply is a WAIT, then the Flag is set and Queue size contains the number of connections waiting / served. Flag 1 is otherwise set to 0.

Request-Session. A typical request packet looks like:

30	Packet length	HEADERS

The opcode for Request-Session is 30. The HEADER looks like:

Data	Name	Length	Time	HTTP	Requester	Connection ID

Name is the file requested or the data type the application tries to send to the server. Length is the length of this packet. Time is the time message was sent. HTTP is the HTTP version. Requestor is the Name of the requesting application. A Connection ID

identifies each request. Data contains additional information when the packet is of the inform type.

40	Length	Data

Inform. The inform message is used for instant messaging for push based applications, when the broker pushes some data on the server.

50	OK/FAIL	Length	Data

Reply-Session. If the request is type of a GET request then the third parameter in the reply is the OK or FAIL specifying whether the request is satisfied or not. The fourth parameter is filled only in case of GET requests. In case of the INFORM requests the field is left as <NULL>.

Finish. At the end of the communication the application sends a FINISH request to the µHTTP server to indicate the end of the session. A connection ID is sent along with the opcode to identify the connection.

60	Connection ID

Typical Communication Example

Connection Request

<operation = “CONNECT”>

<packet length =1234>

<version number =“1.0”>

Connection Reply

<connection reply opcode = “OK”>

Session Request

<opcode = “GET”>

filename = “sample”

Session Reply

<opcode = “Response”>

type = “OK”

Length =500

Data = “hello”

Client-Broker Protocol

The client sends the filled form to the locator. The address of the home locator is fixed and hard coded into the buyer. The message looks like this:

Opcode	City	Street/All/Zone	Distance	Game	Participating team

The Opcode identifies the type of form filled. Some of the other parameters vary depending on the opcode, that is depending on the type of form filled.

Table 4-2. Opcode for Client-Broker

Opcode	Form identifier
01	Absolute Form
02	Distance
03	Area
04	All

- Absolute Form. User specifies addresses in the form of Gainesville/UnivAvenue. The distance field is filled with 0 here.
- Within-Specifier Form. This type of request is typically of the form of "Within 5 miles of 34th Street, Gainesville." The third field is filled with the street / location address.
- Area Form. This type of request is of type "North Gainesville." The third field is filled with Zone identifier. Zone in the above example would be "North." Distance would be left to zero.
- All Form. In this form, the third parameter is "All" and distance is 0.
- Structures for Naming Scheme. The client chooses the appropriate form and sends his data to a local locator. The locator then checks his structure to find the associated brokers.

We have the following object to represent various naming scheme:

NamingStructure.java. This class defines the following classes:

RowMajor. Contains elements row, colfrom, colto and index into the metadata table

ColMajor. Like the RowMajor, this class contains elements col, rowfrom, rowto and index into the metadata table

MetaData. This contains a comprehensive consisting of row, col, city, address, broker and IP. The city field indicates the name of the city. The address field is the Street or area identifier, the broker contains the name of the broker for this area and IP is the IP address of the broker. It is understood that all brokers listen on the same port.

HashData. This contains index, which is the index into the metadata field. Position is a String representation for identifying the Row / Column major fields. The

row and column fields are required for the "contains within" address in which the user specifies the perimeter of a region.

Treenode. This class identifies each node of the hierarchical tree. This tree contains pointers to HashData, RowMajor and ColMajor classes respectively.

Any query first goes through the root node of the tree and then through its left OR right children finding the appropriate node. Once the node is located then its appropriate data structures could be found.

Degree of Locality. For the 'contains within' addressing scheme we used certain mathematical numbers to match closeness. This is better explained with the following schemas:

We define a *Vector* to show the perimeter. The elements of the vector are source row number, target row number, source column number and target column number. This defines the perimeter for the area defined.

Case 1:

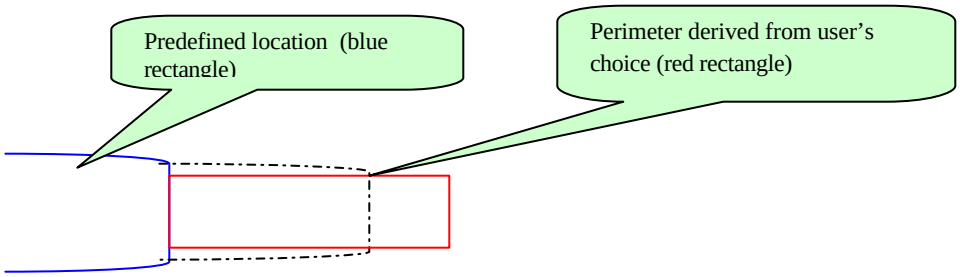

Figure 4-4. Left Side Inclusion

For example: Consider the red rectangle to enclose area from row 1, column 4 to row 1 column 8. The blue rectangle might depict 16th Avenue. It starts before column 4

and continues anywhere beyond column 4 to the right. The doted lines depict the extent of the boundary.

We define 2 factors to map the closeness. ‘match’ is a factor defined by the formula:

Match = Percentage of user derived rectangle covered by predefined location.

Match defines the degree of closeness. We also have absolute degree of ‘farness’ defined by:

Unmatch= End of column for red block – End of column for blue block.

As shown in figure 4-5 the right inclusion extends on or beyond the end of column for the red box. Consider the red rectangle to enclose area from row 1, column 4 to row 1, column 8. The blue block might depict 23rd Avenue. It starts anywhere before column 4 and continues anywhere beyond or on column 8 to the right.

The dotted lines depict the extent of the boundary. A weighted mean is taken considering 80% of ‘match’ and 20% of ‘unmatch’ factor. This helps us in defining the degree of closeness and also in contacting the brokers in the respective areas.

Case 2:

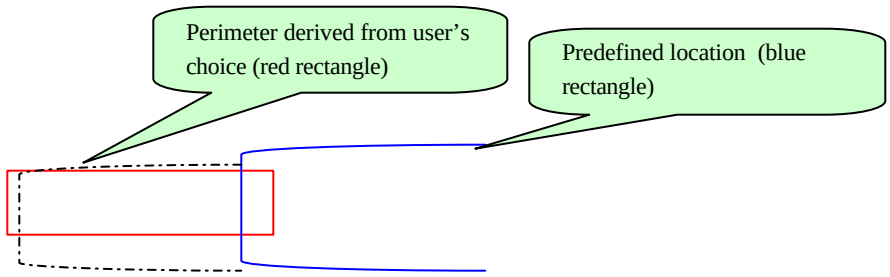

Figure 4-5. Right Side Inclusion

Storage Structure Issues

The storage structure used to store the seller's document has been implemented as a B-Tree. The first level of children gives the different type of tickets that are to be stored. The following section describes the B-Tree storage [26] structure used to store the "ticket."

The B-Tree structures have an identifier in the form of keys, which are used to locate data in the tree. There is an identifier table for commonly played games as shown below.

Table 4-3. Games

Element ID	Game
0	Basketball
1	Football
2	Soccer

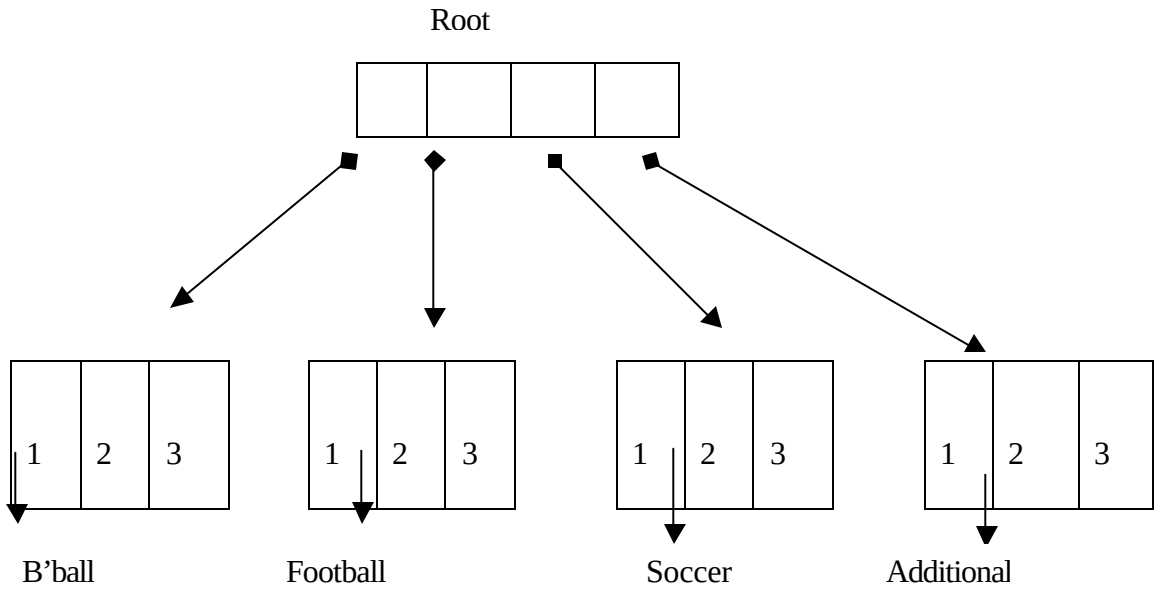

Figure 4-6. B-Tree Representation of the Storage Structure

Item-IDs are given depending on the type of ticket. For example basketball tickets shall start with ID “1.” The first ticket on sale shall be represented by ID “11.” This ID is helpful to uniquely identify tickets when a client makes a choice on his phone. The “TicketStructure” class is used to store the root node and the Type Nodes. This structure is initialized by the initialize () method call.

When a seller sends his request to sell an item the XML data is parsed [27] and kept in this storage structure. The structure used to represent is

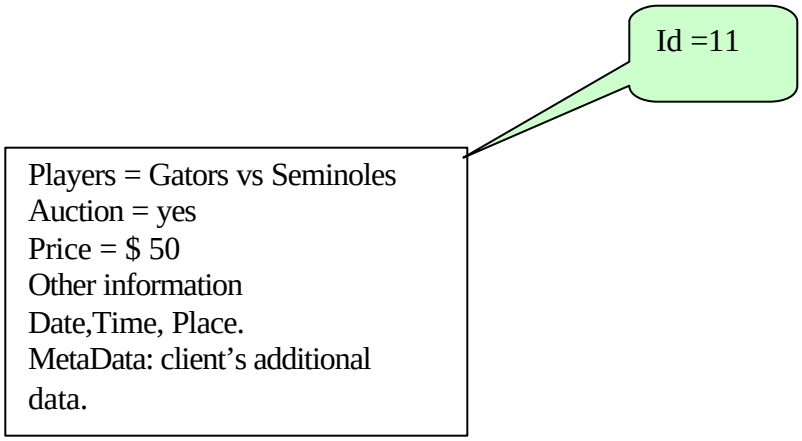

Figure 4-7. Structure to Store Ticket

Seller-Broker Protocol Implementation

The seller-broker protocol is implemented with the help of UDP packets. The seller sends the following packet to the broker:

Opcode	Ticket

When the seller wants to upload the information about his ticket to the broker, he makes use of the web-publishing client to publish information about his ticket. The seller specifies the file name of his ticket-data. When he chooses to sell his item, this file is opened and the contents of the file along with opcode “11” is sent to the broker, who then stores it in his B-tree storage structure. A unique <identifier> (referred earlier as element ID) is given to the item.

Broker-Broker Communication

21	<type>	Players	Additional parameters

Opcode 21 specifies that this is a broker-broker communication. Type is the identifier for the ticket and players is one of the matching identities. Normally it is one of the participating sides. This communication is basically used when the broker cannot find an entry for the location specified by the user. A client could also specify additional parameters with the help of the “additional parameters.”

Snapshots

Figure 4-8 shows some of the snapshots of the application. The previous 2 chapters talked in detail about the design issues and also about implementation. The next chapter speaks about performance related issues.

A B C D

Figure 4-8. Snapshots

A is the front screen of the PEMOCO.

B is the front screen of Auctioneer.

C is the screen where the user specifies the filename for auctioning along with time limit for auctioning.

D is the naming scheme used by the seller to make his choice for location.

CHAPTER 5
PERFORMANCE

This section covers the various performance issues [28, 29] to be considered. We start with the performance evaluation of the HTTP Server.

Micro HTTP Server Performance

We described the design and implementation of the micro HTTP Server. The micro HTTP Server could enable peer-to-peer communication between smart phones; also some of the smart phones could be looked on as a source for a huge chunk of information. Improving the performance of the server could help in quick and efficient response to the clients. The performance of the server could be scaled on the basis of the number of requests it can handle. This analysis helps in understanding scalability issues with the phone and then taking appropriate measures to improve them.

The micro HTTP Server's evaluation could be evaluated by simulating an environment consisting of multiple clients on different machines. There are many difficulties in simulating many clients on different machines. Moreover there are differences in delay between the delay and loss characteristics of the LANs and WANs used in test beds.

A strategy using minimal number of client machines to simulate actual web traffic is proposed, by having C clients on N machines. In this schema the client repeatedly sends requests to the server and then waits for a certain amount of time and then sends another. The factor N is kept as high as possible and C as small as possible. While

generating these requests from the client's end care must be taken such that the resource limitations on the client side do not affect the performance of the server. As the number of requests from a single machine increase the memory and CPU might be a bottleneck. There could be a stage in the performance evolution where the client and not the server could be a bottleneck. These client bottlenecks could be avoided by reducing the number of client processes / machine. Also the WAN based networks have longer delays than the ones based on LANs. Packet loss could be another source of problems in WAN based networks. If the clients making requests to the server are slow the server spends more time with one client and this affects the response times of the other servers.

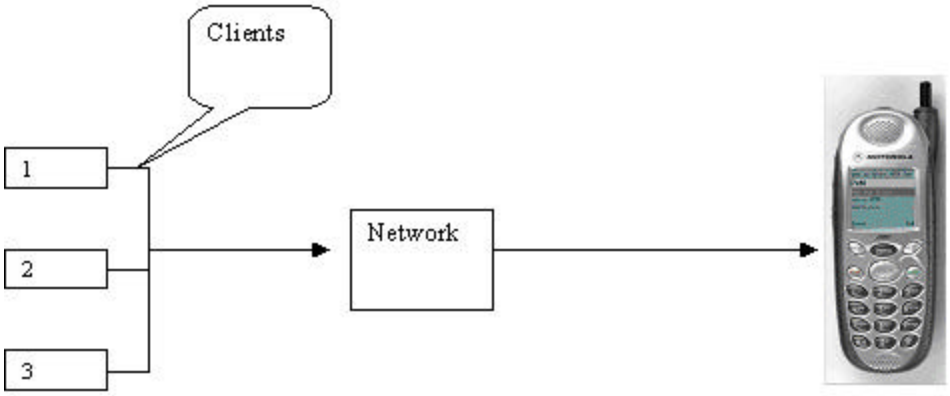

Figure 5-1. Performance Architecture

Server Design I

A typical web server listens for requests on a particular port (normally port 80 for HTTP protocol) and opens an active connection to communicate with every client. The web server has no limitations on the number of connections open, unless the resources of

memory run out of supporting each connection. The design for our µHTTP server has to be slightly changed since it supports only two active connections at a time.

Design I

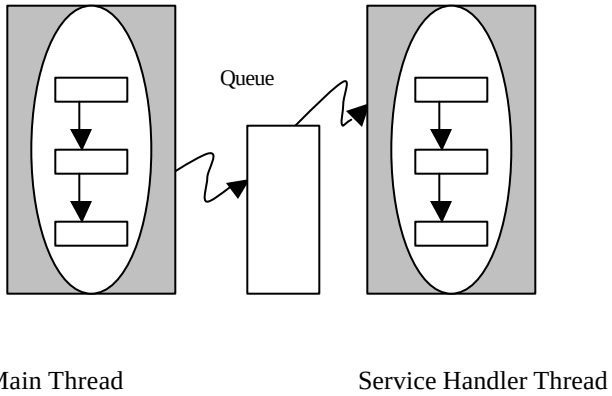

Figure 5-2. Server Design I

The µHTTP server has a main thread, which continuously listens for requests and puts them in a queue. The length of the queue is fixed. The service handler thread waits for messages to be put in the queue and then handles the handshake protocol with the client. If the size of the queue is full, then a negative acknowledgement is sent back to the client. This helps the client in knowing that its request cannot be served. The object Queue is shared between the Main Thread and the Service Handler Thread and the methods are synchronized to have serial update of the shared data.

Client was simulated on one machine with N threads and each producing M requests with different wait times. We simulated actual condition by having some kind of sleep in between requests. Following are the results we obtained:

Table 5-1. Server Test Results I
No. Of Threads (Clients): 2

No. of Requests	Sleep time between requests (ms)	Average Response Time (ms)
2	500	3400
5	500	3800

Server Design II

With more number of simultaneous requests we observed some packets being dropped and the Service Handler thread being unnecessarily waiting for packets from the client. This leaded to a deadlock state where the Service Handler thread could not serve any more requests. We thus had a modified Service Handler Thread as shown in design II below.

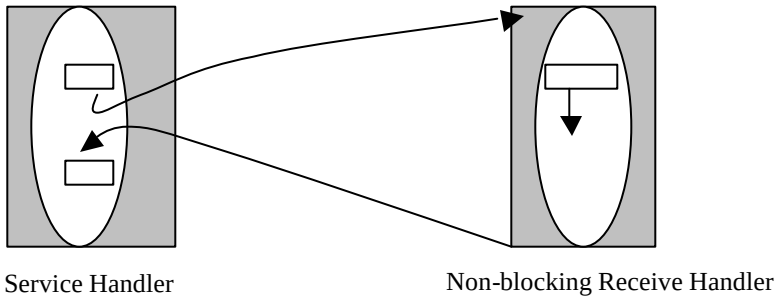

Figure 5-3. Server Design II (Main Thread Is Same as in Design I)

We designed the non-blocking receive handler which was spawned after the server had sent its acceptance to the request from the client and was now expecting to receive a query from the client (phase 2 of handshake protocol) from the client. The service handler thread would wait for a specified amount of time to wait for data to arrive. A special flag was set once new data arrived. After the delay, the service handler

would check the flag and if it is set then would continue with the rest of the handshake protocol, else would just continue with the next request in the queue. The above design was basically implementation of a non-blocking receive. The communication was much more reliable but at the cost of higher communication time. Following are some of the relevant results:

Table 5-2. Server Test Results II

No. of clients	Response times (average) in seconds
2	12.5
5	22
10	43.5

The response time increases proportionally to the number of the requests. The response time for the Nth request is more compared to that of the (N-1)th request. The graph is for 5 clients simulated on the same machine with 5 requests per client and an interval of 1second between them.

As seen from figure 5-4 the response time increases with the increase in the request number. The first request is immediately satisfied since the queue is empty. Thereafter as the queue fills up the response time also increases.

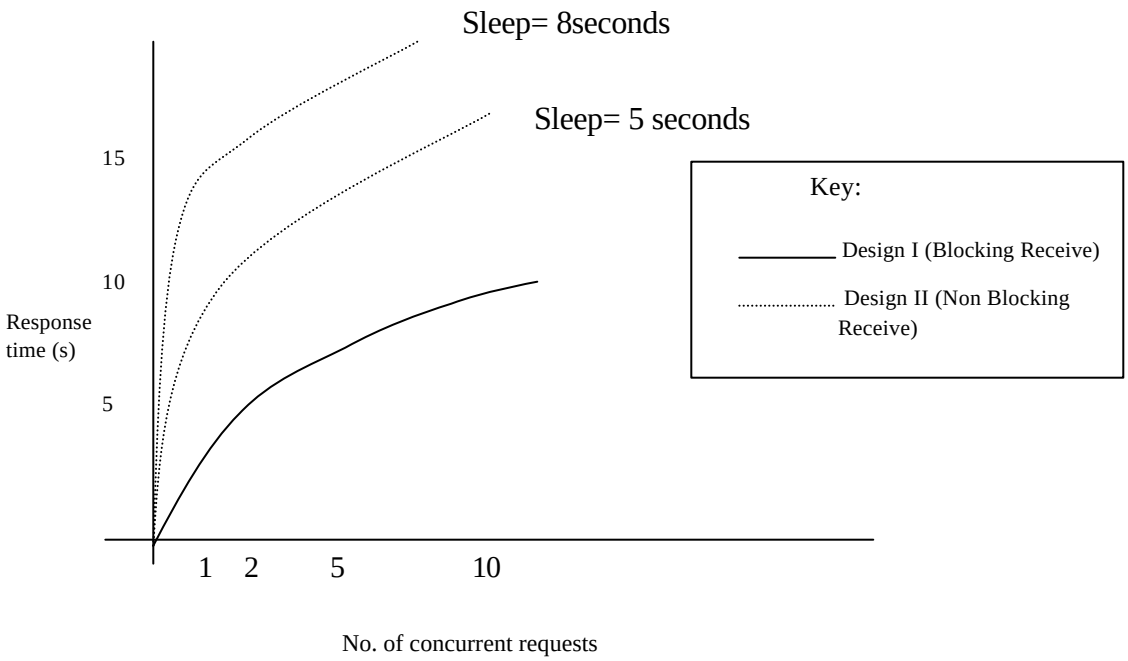

Figure 5-4. Response Curves for the Two Designs

Storage Structures

Currently the storage structures used with the phones are B-Trees. We developed a small test program to analyze the search time for items in the B-Trees. First we make N records for different types of games and then search for the items sequentially, alternately or randomly.

Table 5-3 shows that the time required to access every record takes the least amount of time compared to the other two. The ‘Alternate Record’ takes the maximum amount of time. The ‘Random’ method being a mix of the two takes time in between the other two methods. These results were generated based on generating results for N records and then calculating the time required for each record.

Table 5.3. Storage Structure Test Results. No. of records 700

Type	Time / record (ms)
Sequential record	0.006
Alternate record	0.011
Random record	0.018

No. of records: 2000

Type	Time / record (ms)
Sequential record	0.007
Alternate record	0.022
Random record	0.010

Push vs. Pull Approaches

This [30] is one of the major factors to be considered during transaction. A typical push application would require the seller to push information regarding his ticket to the associated broker. Whenever the buyers query for some ticket, the brokers need not contact the sellers (mobile phone) but would query their own database for the required data. A pull-based approach is one where the clients would contact the server periodically for any data they want. We shall study about the factors affecting performance.

Coherency

To maintain coherency of the cached data, the seller shall be updated with the latest item on sale. We assume that a user specifies a temporal coherency requirement *tc* for each item of interest. The parameter decides the number of communications involved

and is thus very crucial especially for the mobile phone since communication involves more consumption compared to the others.

Communication Overheads

In general, the number of messages generated over the network can be anticipated based on the *tr* value, which is decided by the user. In this way user specified temporal coherency is maintained.

The number of communications in a pull-based approach is based on the user's estimate of how frequently the data is changing. If the data actually changes at a slower rate, then polling might be done more frequently than necessary. This might impose a larger load on network. A push-based approach may also push information to clients who are no longer interested generating unnecessary message overheads.

Computational Overheads

Computational overheads for a pull-based server results from the need to deal with individual pull requests. As far as the push-based approach is concerned the broker has to check if the data is suitable for the particular client and if the *tr* value is surpassed. The number of times this check is to be performed is directly proportional to the arrival of new data. In this aspect push is more demanding to pull. Pull might increase additional overhead from the broker (server) point of view, which may receive multiple simultaneous requests.

Space Complexity

A push-based broker (server) could be classified as one maintaining the state of every client. This state must be maintained throughout client activity, due to which the number of clients the broker is handling might be limited. A pull-based approach is a stateless one.

Failures

Failures in push-based approaches will lead to a loss in the state information in the broker. The client has to detect this server failure and re-register its *tr* requirements. Consequently, the client's coherency requirements will not be met until the proxy detects the failure and re-registers the *tcr* requirements with the server.

The push approach is one in which the client does not query the broker, but the broker pushes data onto the buyer. The seller has his micro HTTP server always running through which he can receive and reply to requests.

We tried to compare the push and pull approaches with respect to storage, communication, computation and time. Communication is an important factor with respect to the power consumption of the phones. UDP /TCP communication consumes most part of the power and is thus an important factor.

CHAPTER 6
CONCLUSION AND FUTURE WORK

Conclusion

This thesis defined an architecture for personal M-Commerce by making use of location based services and other useful tools which will be useful in today's as well as tomorrow's mobile computing domain. First we made a survey of the existing service discovery protocols and then proposed our architecture. We also studied how location awareness would serve as a parameter in this domain. We surveyed some current means of determining location like the GPS and also some already available applications like HP's Cool Town. We built some tools on the phone like the micro HTTP server that was based on the U-H (UDP-HTTP) protocol and handled requests like a web server. Several limitations on the mobile device made the architecture and protocol different and we came up with a handshake protocol and then even evaluated the web servers by having it serve many requests. We also built a web-publishing client, which facilitates users to make web pages on the fly. We studied the fact that the use of keys on the phone was not comfortable and tried to minimize the client's efforts by having a simple to use tag, which would insert the appropriate text in the file.

The *Auctioneer* application witnessed the role of location awareness in the Mobile Commerce domain. A buyer would have the ability to choose sellers based on location. We demonstrated the concept of naming schemes by showing four different naming techniques.

Future Work

As mentioned before the emerging field of mobile computing has opened the doors for a wide area of research and applications. In this application we made a contribution in the mobile commerce area.

The future is moving towards peer-peer computing [31]. JXTA [32], which is a research project started at Sun Microsystems, Inc. explores some new styles of distributed computing. Mobile Commerce sees a new ray of hope with the emergence of JXTA. In our application we proposed the micro HTTP Server, which could be important for having peer-peer kind of applications.

Another area that shares borders with *location* is context awareness [33]. Context Awareness. Not only is the physical location of the user important, but also the environment in which the user is computing can be taken as an attribute to give better quality of services. This parameter will add more semantics to any query submitted by the user.

Lastly, artificial intelligence is another area of research, which could focus on the concept of *intuition* helping on discovering what kind of services the user may need. This could begin on the note of having user profile and then noting changes in the profile as the user makes request. The path of requests could be surveyed to give better services. The above features, if incorporated will certainly make a mark in tomorrow's world and will truly justify the evolution of mobile computing.

LIST OF REFERENCES

[1] F. Müller-Veerse: "Mobile Commerce Report, (pdf)," Durlacher Corp., London. Appeared in October 2000 issue of IEEE Computer.

[2] Car Guides. http://www.cars.com. Accessed May 2001.

[3] Digital City. http://www.digitalcity.com. Accessed May 2001.

[4] T. Bray, J. Paoli and C.M. Sperberg-Mc Queen: "Extensible Markup language (XML) 1.0," W3 recommendation. http://www.w3.org/TR/REC-xml, February 1998.

[5] T. Agoston, IBM Global Services, USA; T. Ueda, IBM Global Services, Japan; Y. Nishimura, Nishimura Marketing Services, Japan: "Pervasive computing in a Networked World," Proceedings of INET,Yokohama, Japan, July 2000.

[6] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman and D. Zukowski: "Challenges: An Application Model for Pervasive Computing," Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, August 2000.

[7] J. Fallon, G. Singh: "Mobile Internet Security," Baltimore Technologies. http://www.morganstanley.com/waprap/presentations/000601.pdf. Accessed July 2001.

[8] M. Weiser, "Hot Topics: Ubiquitous Computing," IEEE Computer. Page 71-72. October 1993.

[9] H. Chen, D. Chakraborty: "Service Discovery in the Future for Mobile Commerce," ACM Crossroads. Page 18-24. Issue 7.2, Winter 2000.

[10] Sun's JINI Technology. http://www.sun.com/jini. Accessed June 2001.

[11] U. Leonhardt, "Supporting Location Awareness in Open Distributed systems," Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 1998.

[12] Y. Charlie Hu, D. Rodney and P. Druschel "Design and Scalability of NLS, a Scalable Naming and Location Service," Computer Science Department, Rice

University, TX, USA, Submitted for publication, Sigmetrics 2001, November 2000.

[13] Hewlett Packard Cool Town project. http://www.cooltown.hp.com. Accessed June 2001.

[14] V. Krishnan, "Position Paper," Hewlett Packard Laboratories. http://www.w3.org/Mobile/posdep/HPw3cwapref.html. Accessed June 2001.

[15] T. Kindberg and J. Barton, "A Web-Based Nomadic Computing System," Internet and Mobile Systems Laboratory, Hewlett Packard Laboratories, August 2001.

[16] Corporation for National Research Initiatives: "The Handle Naming System: The general-purpose Global Name System Enabling Secure Name Resolution over the Internet." http://www.handle.net/. May 2001.

[17] Sun Microsystems: J2ME. http://java.sun.com/j2me/docs. Accessed June 2001.

[18] Sun Microsystems: CLDC and K virtual machine. http://java.sun.com/products/cldc/. Accessed June 2001.

[19] R. Evans Web Site: "KVM on the Palm Pilot," http://webdev.apl.jhu.edu/~rbe/kvm/. February 2001.

[20] H. Chen, A. Joshi, T. Finin: "Dynamic Service Discovery with Mobile Computing: Intelligent Agents meet Jini in the Aether," Department of Computer Science and Electrical Engineering, University of Maryland at Baltimore County, February 2000.

[21] Reticular Systems, Incorporate; "Using Intelligent Agents for Wireless E-Commerce Applications: The Yellow Stone Project," Reticular Systems Incorporated, May 2001.

[22] Motorola Developer Community Web Site: http://www.idendev.com. Accessed April 2001.

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hyper Text Transfer Protocol-HTTP/1.1," RFC 2068, January 1997.

[24] F. Tian, D. DeWitt, J. Chen, C. Zhang: "The Design and Performance Evolution of Alternative XML Storage Strategies," Department of Computer Science, University of Wisconsin-Madison, WI, June 2001.

[25] Counterpoint Systems Foundry Inc, Microsoft Corporation "IrDA Object Exchange Protocol," IrOBEX Version 1.2, March 1999.

[26] P. Neubauer: "B-trees Balanced Data Structures." http://www.public.asu.edu/~peterjn/btree/, May 1999.

[27] S. Sheth: "QT4XML: A Query Tool for XML Documents and Databases," Masters Thesis, Department of Computer Science, The University of Georgia, July 1999.

[28] G. Bagga and P. Druschel: "Measuring the capacity of a Web Server," Department of Computer Science, Rice University, Houston, Texas. http://www.cs.rice.edu/CS/Systems/Web-measurement/paper/paper.html. Accessed June 2001.

[29] A. Iyengar, E. Nair and T. Nyugen. "An Analysis of Web Server Performance," IBM Research Division, T. J. Watson Research Center, NY. http://www.cs.berkeley.edu/~eanders/notes/papers/web-server-capacity97.html. Accessed June 2001.

[30] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, P. Shenoy: "Adaptive Push-Pull: Disseminating Dynamic Web Data," Pages 265-274, ACM Publications, July 2001.

[31] G. Bolcer, M. Gorlick, A. Hitomi, P. Kammer, B. Morrow, P. Oreizy, R. Taylor. "Peer-to-Peer Architectures and the Magi ™ Open-Source Infrastructure," Endeavors Technology, Inc. http://www.endeavors.com, 2001.

[32] Sun Microsystems: Project JXTA. http://www.jxta.org. Accessed July 2001

[33] G. Abowd, A. Dey, G. Abowd, R. Orr and J. Brotherton, "Context-Awareness in Wearable and Ubiquitous Computing," Graphic, Visualization and Usability Center, Georgia Institute of Technology. http://www.cc.gatech.edu/fce/pubs/iswc97/wear.html. Accessed July 2001.

BIOGRAPHICAL SKETCH

Tapan Divekar received his Bachelor of Engineering degree in Computer Science and Engineering from the University of Pune, Poona India in 1998. He won a scholarship to Sharp Corporation Japan for a year from August 1998 to August 1999. He graduated in August 2001 with Master of Science degree in Computer Science. His research interests include mobile computing, pervasive computing, computer networks and distributed computing.